
SoftPython

Introductive guide to data cleaning and analysis with Python 3

David Leoni, Alessio Zamboni, Marco Caresia

Aug 26, 2023

Copyright © 2023 by David Leoni, Alessio Zamboni, Marco Caresia.

SoftPython is available under the Creative Commons Attribution 4.0 International License, granting you the right to copy, redistribute, modify, and sell it, so long as you attribute the original to David Leoni, Alessio Zamboni, Marco Caresia and identify any changes that you have made. Full terms of the license are available at:

<http://creativecommons.org/licenses/by/4.0/>

The complete book can be found online for free at:

<https://en.softpython.org>

CONTENTS

Preface	1
News	1
1 Overview	3
1.1 Intended audience	3
1.2 Contents	3
1.3 Author	6
1.4 License	6
1.5 Acknowledgments	7
2 Overview	9
2.1 Chapters	9
2.2 Why Python?	10
2.3 Approach and goals	11
2.4 Doesn't work, what should I do?	11
2.5 Installation and tools	12
2.6 Let's start !	12
3 Installation	13
3.1 Installing Python	14
3.2 Installing packages	17
3.3 Jupyter Notebook	18
3.4 Projects with virtual environments	22
3.5 Further readings	24
4 A - Foundations	25
4.1 Quick introduction to Python	25
4.2 Tools and scripts	64
5 A1 Data Types	85
5.1 Basics	85
5.2 Strings	140
5.3 Lists	220
5.4 Tuples	324
5.5 Sets	347
5.6 Dictionaries	376
6 A2 Control Flow	453
6.1 If command	453
6.2 For loops	480
6.3 While loops	573
6.4 Sequences	618

7 A3 Basic Algorithms	647
7.1 Functions, error handling and testing	647
7.2 Matrices of lists	753
7.3 Mixed structures	841
7.4 Numpy matrices	854
8 B - Data Analysis	929
8.1 Data formats	929
8.2 Visualization	982
8.3 Pandas	1074
8.4 Relational data	1163
9 C - Applications	1277
9.1 Database	1277
10 D - Projects	1305
10.1 Text data	1305
10.2 Tabular data	1322
10.3 Relational data	1468
11 E - Appendix	1559
11.1 Commandments	1559
11.2 Revisions	1565
11.3 References	1566

Preface

Introductive guide to coding, data cleaning and analysis for Python 3, with many worked exercises.

Nowadays, more and more decisions are taken upon factual and objective data. All disciplines, from engineering to social sciences, require to elaborate data and extract actionable information by analysing heterogenous sources. This book of practical exercises gives an introduction to coding and data processing using [Python](#)¹, a programming language popular both in the industry and in research environments.

News

23 August 2023

- restyling!
- restructured *analytics with pandas*:
 - separated notebooks into 1. intro and 2. advanced (grouping, merging, geopandas)
 - moved exercise notebook 2 (eures) to *worked projects section*
 - renamed dataset into astropi.csv reming ROW_ID column and substituted with time_stamp
 - added paragraphs to first notebook
 - added meteo pressure intervals exercise
- matrices-lists1: added *visiting with style paragraph*
- lists3: added copy/deepcopy paragraph
- Python tutor now always show data structures as non-nested
- strings1: added paragraph on f-strings
- sets1: added paragraph ‘What can we search?’
- formats2-csv: swapped ‘with’ when reading and writing

Old news: [link](#)

¹ <https://www.python.org>

**CHAPTER
ONE**

OVERVIEW

1.1 Intended audience

This book can be useful for both novices who never really programmed before, and for students with more technical background, who desire to know about data extraction, cleaning, analysis and visualization (among used frameworks there are Pandas, Numpy and Jupyter editor). Data is going to be processed in a practical way, without delving into more advanced considerations about algorithmic complexity and data structures. To overcome issues and guarantee concrete didactical results, step-by-step tutorials are presented.

1.2 Contents

- *Overview*: Approach and goals

1.2.1 A - Foundations

1. *Installation*
2. *Quick Python intro* (if you already have programming skills)
3. *Tools and scripts* (if you are a beginner)

1.2.2 A.1 Data Types

1. Basics: 1. *variables and integers* 2. *booleans* 3. *real numbers* 4. *challenges*
2. Strings: 1. *intro* 2. *operators* 3. *basic methods* 4. *search methods* 5. *challenges*
3. Lists: 1. *intro* 2. *operators* 3. *basic methods* 4. *search methods* 5. *challenges*
4. Tuples: 1. *intro* 2. *challenges*
5. Sets: 1. *intro* 2. *challenges*
6. Dictionaries: 1. *intro* 2. *operators* 3. *methods* 4. *special classes* 5. *challenges*

1.2.3 A.2 Control Flow

1. If conditionals: [1. intro](#) [2. challenges](#)
2. For loops: [1. intro](#) [2. strings](#) [3. lists](#) [4. tuples](#) [5. sets](#) [6. dictionaries](#)
[7. nested for](#) [8. challenges](#)
1. While loops [1. intro](#) [2. challenges](#)
2. Sequences and comprehensions: [1. intro](#) [2. challenges](#)

1.2.4 A.3 Basic Algorithms

1. Functions: [1. intro](#) [2. error handling and testing](#)
[3. strings](#) [4. lists](#) [5. tuples](#) [6. sets](#)
2. Matrices - list of lists: [1. intro](#) [2. other exercises](#) [3. challenges](#)
3. Mixed structures: [1. intro](#) [2. challenges](#)
4. Matrices - numpy: [1. intro](#) [2. exercises](#)

1.2.5 B - Data Analysis

1. Data formats: [1. line files](#) [2. CSV files](#) [3. JSON files](#) [4. challenges](#)
2. Visualization (matplotlib,: [1. intro](#) [2. challenges](#) [images](#)
3. Analytics with Pandas: [1. intro](#) [2. exercises](#) [3. challenge](#)
4. Relational data: [1. intro](#) [2. binary relations](#) [3. simple statistics](#) [4. challenge](#)

1.2.6 C - Applications

1. *Database integration*: executing simple SQL queries to extract data from a database, loading into Pandas

1.2.7 D - Projects

1.2.8 Worked projects

Projects as exercises (with solutions), involving some raw data preprocessing, simple analysis and final chart display. Some are about serious topics, some are light-hearted, others come from daily work scenarios: pick your choice!

Note that since the purpose of the book is to introduce to computational thinking, we preferred following the no-magic approach of using basic Python data structures and modules instead of more advanced libraries like numpy or pandas, even when they could dramatically ease the task and improve performances.

Text data worked projects

- *Phone calls*
- *Music Sequencer*

Tabular data worked projects

- *Bus Speed*
- *Town Events*
- *What's your business?*
- *Zoom Surveillance*
- *I CHING Divination*
- *Witchcraft*
- *Galactic Love*
- *University staff*
- *ITEA Public housing*
- *Public price catalog*
- *EURES job offers*

Relational data worked projects

- *Trans-Atlantic Slave Trade*
- *Bud Spencer and Terence Hill movies*
- *Bus Network*
- *Wikispeedia*
- *Mexican Drug Wars*
- *Wordnet*
- *MetaMath*

1.2.9 E - Appendix

- *Commandments*
- *References*

1.3 Author

David Leoni: Software engineer specialized in data integration and semantic web, has made applications in open data and medical in Italy and abroad. He frequently collaborates with University of Trento for teaching activities in various departments. Since 2019 is president of CoderDolomiti Association, where along with Marco Caresia manages volunteering movement CoderDojo Trento to teach creative coding to kids. Email: david.leoni@unitn.it Website: davidleoni.it²

1.3.1 Contributors

Marco Caresia (2017 Autumn Edition assistent @DISI, University of Trento): He has been informatics teacher at Scuola Professionale Einaudi of Bolzano. He is president of the Trentino Alto Adige Südtirol delegatioon of the Associazione Italiana Formatori and vicepresident of CoderDolomiti Association.

Alessio Zamboni (2018 March Edition assistent @Sociology Department, University of Trento): Data scientist and software engineer with experience in NLP, GIS and knowledge management. Has collaborated to numerous research projects, collecting experinces in Europe and Asia. He strongly believes that '*Programming is a work of art*'.

Luca Bosotti (2020 Data Science Summerschool assistant, 2021 seminars @Sociology Department, University of Trento): Developer, scientist and professor. Believes the world is getting more and more complicated and interesting, so we must study it by taking advantage of all the available potential and reasoning. He thought to youngsters of all ages, from elementary schools up till university level and got impressed by the diversity of people who approach programming.

Massimiliano Luca (2019 summer edition teacher @Sociology Department, University of Trento): Loves learning new technologies each day. Particularly interested in knowledge representation, data integration, data modeling and computational social science. Firmly believes it is vital to introduce youngsters to computer science, and has been mentoring at Coder Dojo DISI Master.

Others: We also wish to thank the students Ludovico Maria Valenti and Ioana Doleanu for the improvements to the numpy page, and Stefano Moro for the numerous reports.

1.4 License

The making of this website and related courses was funded by Department of Information Engineering and Computer Science (DISI)³, University of Trento, and also Sociology⁴ and Mathematics⁵ departments.



UNIVERSITY
OF TRENTO

Department of Information
Engineering and Computer Science



² <https://davidleoni.it>

³ <https://www.disi.unitn.it>

⁴ <https://www.sociologia.unitn.it/en>

⁵ <https://www.maths.unitn.it/en>

Unless otherwise noted, the material in this website is original and distributed with license CC-BY 4.0 International Attribution <https://creativecommons.org/licenses/by/4.0/deed.en>. Basically, you can freely redistribute and modify the written content, just remember to cite University of Trento and the authors⁶

Datasets from *Data analysis* and *Worked projects* sections might have some restrictions, sources are citated in the pages where they are used. Other third party resources are listed in third-party-licences.txt

Technical notes: all website pages are easily modifiable Jupyter notebooks, that were converted to web pages using NB-Sphinx⁷ and Jupman⁸ template. Text sources are on Github at <https://github.com/DavidLeoni/softpython-en>

1.5 Acknowledgments

We thank in particular professor Alberto Montresor of Department of Information Engineering and Computer Science, University of Trento to have allowed the making of first courses from which this material was born from, and the project Trentino Open Data (dati.trentino.it⁹) for the numerous datasets provided.



⁶ <https://en.softpython.org/index.html#Author>

⁷ <https://nbsphinx.readthedocs.io>

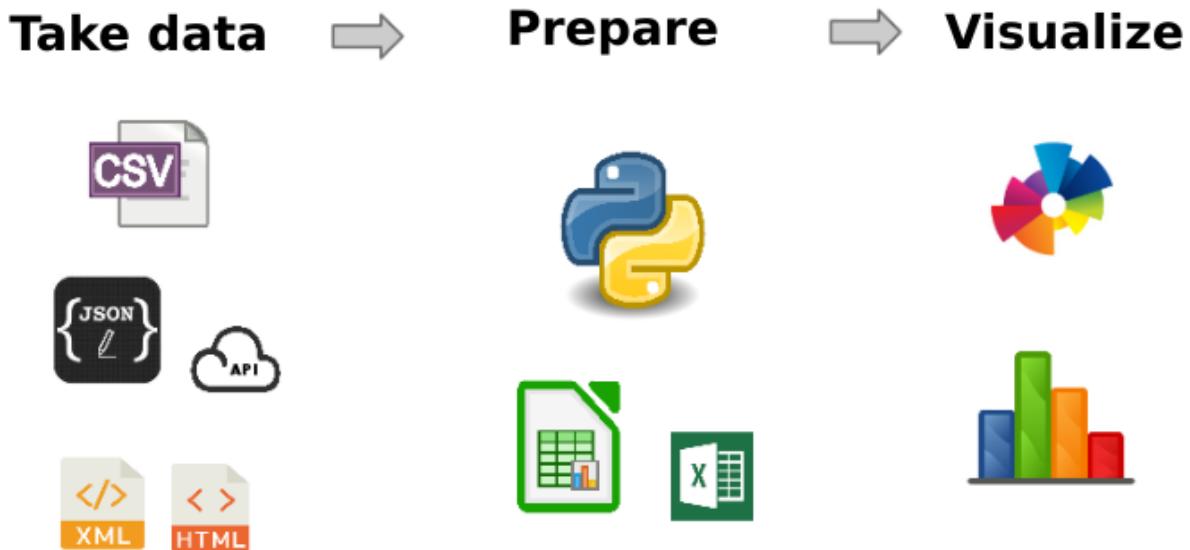
⁸ <https://jupman.softpython.org>

⁹ <https://dati.trentino.it>

OVERVIEW

To start with we will spend a couple of words on the approach and the goals of the book, then we will deep dive into the code.

WHAT ARE WE GOING TO DO?



2.1 Chapters

The tutorials mostly deal with fundamentals of Python 3, data analysis (more like raw data processing rather than statistics) and some applications (dashboards, databases, ..)

What are *not* about:

- object oriented programming theory
- algorithms, computational complexity
- performance
 - no terabytes of data ...
- advanced debugging (pdb)

- testing is only mentioned
- machine learning
- web development is only mentioned

2.2 Why Python?



- **Easy** enough to start with
- **Versatile**, very much used for
 - scientific calculus
 - web applications
 - scripting
- **widespread** both in the industry and research environments
 - Tiobe¹⁰ Index
 - popularity on Github¹¹
- **Licence** open source & business friendly¹²
 - translated: you can sell commercial products based on Python without paying royalties to its authors

¹⁰ <https://www.tiobe.com/tiobe-index/>

¹¹ https://madnight.github.io/github/#/pull_requests/2020/1

¹² <https://docs.python.org/3/license.html>

2.3 Approach and goals

If you have troubles with programming basics:

- **Exercise difficulty:**  , 
- Read SoftPython - Part A - Foundations¹³

If you already have some programming skills:

- **Exercise difficulty:** , 
- Read Python Quick Intro¹⁴ and then go directly to Part B - Data Analysis¹⁵

Other guides: you can find links to further material in *References* page

2.4 Doesn't work, what should I do?

While programming you will surely encounter problems, and you will stare at mysterious error messages on the screen. The purpose of this book is not to give a series of recipes to learn by heart and that always work, as much as guide you moving first steps in Python world with some ease. So, if something goes wrong, do not panic and try following this list of steps that might help you. Try following the proposed order:

1. If in class, ask professor (if not in class, see last two points).
2. If in class, ask the classmate who knows more
3. Try finding the error message on Google
 - remove names or parts too specific of your program, like line numbers, file names, variable names
 - Stack overflow¹⁶ is your best friend
4. Look at Appendix A - Debug from the book Think Python¹⁷, by Allen B. Downey:
 - Syntax errors¹⁸
 - I keep making changes and it makes no difference.¹⁹
 - Runtime errors²⁰
 - My program does absolutely nothing.²¹
 - My program hangs.²²
 - Infinite Loop²³
 - Infinite Recursion²⁴
 - Flow of Execution²⁵

¹³ <https://en.softpython.org/index.html#foundations>

¹⁴ <https://en.softpython.org/quick-intro/quick-intro-sol.html>

¹⁵ <https://en.softpython.org/index.html#data-analysis>

¹⁶ <https://stackoverflow.com>

¹⁷ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html>

¹⁸ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#235>

¹⁹ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec236>

²⁰ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec237>

²¹ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec238>

²² <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec239>

²³ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec240>

²⁴ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec2241>

²⁵ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec242>

- When I run the program I get an exception²⁶
- I added so many print statements I get inundated with output²⁷
- Semantic errors²⁸
 - My program doesn't work²⁹
 - ve got a big hairy expression and it doesn't do what I expect.³⁰
 - ve got a function that doesn't return what I expect.³¹
 - I'm really, really stuck and I need help.³²
 - No, I really need help.³³

5. Gather some courage and ask on a public forum, like Stack overflow or python-forum.io - see [how to ask questions](#).

2.4.1 How to ask questions

IMPORTANT

If you want to ask written questions on public chat/forums (i.e. like [python-forum.io](#)³⁴) DO FIRST READ the forum rules - see [How to ask Smart Questions](#)³⁵

In substance, you are always asked to clearly express the problem circumstances, putting an explicative title to the post /mail and showing you spent some time (at least 10 min) trying a solution on your own. If you followed the above rules, and by misfortune you still find people who use harsh tones, just ignore them.

2.5 Installation and tools

- If you still haven't installed Python3 and Jupyter, have a look at [Installation](#)

2.6 Let's start !

- **If you already have some programming skill:** you can look [Quick Python intro](#)
- **If you don't have programming skills:** go to [Tools and scripts](#)³⁶

²⁶ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec243>

²⁷ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec244>

²⁸ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec245>

²⁹ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec246>

³⁰ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec247>

³¹ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec248>

³² <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec249>

³³ <http://greenteapress.com/thinkpython2/html/thinkpython2021.html#sec250>

³⁴ <https://python-forum.io/index.php>

³⁵ <https://python-forum.io/misc.php?action=help&hid=19>

³⁶ <https://en.softpython.org/tools/tools-sol.html>

CHAPTER THREE

INSTALLATION

We will see whether and how to install Python, additional Python libraries, Jupyter notebook and finally how to manage virtual environments.

Sometimes you don't even need to install!

If you want, you can also directly program online with the following services

NOTE 1: if you want to try one, always remember to check it is using Python 3 !

NOTE 2: As for any online service, whenever it is freely offered do not abuse it. If you try processing a terabyte of data per day without paying a subscription, you risk a denial of service.

Python 3 on repl.it³⁷: allows to edit Python code collaboratively with other users, and also supports libraries such as Matplotlib

Python Tutor³⁸: allows to execute one instruction at a time while offering a very useful visualization of what is happening ‘under the hood’

Google Colab³⁹: allows editing collaboratively Jupyter notebooks and save them to Google Drive.

- NOTE 1: it might be you won't be able to access with university accounts (i.e. ``@studenti.unitn.it``). In that case, use personal accounts such as @gmail.com
- NOTE 2: the ‘collaborative’ aspect of Colab changed over time, **be very careful at what happens when working in two people over the same document**. Once (2017) changes performed by one were immediately seen by other users, but lately (2019) they seem only visibly when saving - even worse, they overwrite changes others could have done in the meanwhile.

Online Jupyter demo⁴⁰: sometimes it works but it is not always available. If you manage to access, remember to select from the menu *Kernel->Change kernel->Python 3*

³⁷ <https://repl.it/languages/python3>

³⁸ <http://pythontutor.com/visualize.html#py=3>

³⁹ <https://colab.research.google.com>

⁴⁰ <http://try.jupyter.org>

3.1 Installing Python

There are various ways to install Python 3 and its modules: there is the official ‘plain’ Python distribution but also package managers (i.e. Anaconda) or preset environments (i.e. Python(x,y)) which give you Python plus many packages. Once completed the installation, Python 3 contains a command `pip` (sometimes called `pip3` in Python 3), which allows to install afterwards other packages you may need.

The best way to choose what to install depends upon which operating system you have and what you intend to do with it. In this book we will use Python 3 and scientific packages, so we will try to create an environment to support this scenario.

Attention: before installing random stuff from the internet, read carefully this guide

We tried to make it generic enough, but we couldn’t test all various cases so problems may arise depending on your particular configuration.

Attention: do not mix different Python distribution for the same version !

Given the wide variety of installation methods and the fact Python is available in already many programs, it might be you already have installed Python without even knowing it, maybe in version 2, but we need the 3! Overlaying several Python environments with the same version may cause problems, so in case of doubt ask somebody who knows more!

3.1.1 Windows installation

For Windows, we suggest to install the distribution [Anaconda for Python 3.8⁴¹](#) or greater, which, along with the native Python package manager `pip`, also offers the more generic command line package manager `conda`.

Once installed, verify it is working like this:

1. click on the Windows icon in the lower left corner and search for ‘Anaconda Prompt’. It should appear a console where to insert commands, with written something like `C :\Users\David>`. NOTE: to launch Anaconda commands, only use this special console. If you use the default Windows console (`cmd`), Windows will not be able to find Python.
2. In Anaconda console, type:

```
conda list
```

It should appear a list of installed packages, like

```
# packages in environment at C:\Users\Jane\AppData\Local\Continuum\Anaconda3:  
#  
alabaster          0.7.7           py35_0  
anaconda          4.0.0           np110py35_0  
anaconda-client    1.4.0           py35_0  
...  
numexpr            2.5             np110py35_0  
numpy              1.10.4          py35_0  
odo                0.4.2           py35_0  
...  
yaml               0.1.6           0  
zeromq            4.1.3           0  
zlib               1.2.8           0
```

⁴¹ <https://www.anaconda.com/download/#windows>

3. Try Python3 by typing in the Anaconda console:

```
C:> python
```

It should appear something like:

```
Python 3.6.3 (default, Sep 14 2017, 22:51:06)
MSC v.1900 64 bit (Intel) [GCC 5.4.0 20160609] on win64
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Attention: with Anaconda, you must write `python` instead of `python3` !

If you installed Anaconda for Python3, it will automatically use the correct Python version by simply writing `python`. If you write `python3` you will receive an error of file not found !

Attention: if you have Anaconda, always use `conda` to install Python modules ! So if in next tutorials you see written `pip3 install whatever`, you will instead have to use `conda install whatever`

3.1.2 Mac installation

To best manage installed app on Mac independently from Python, usually it is convenient to install a so called *package manager*. There are various, and one of the most popular is [Homebrew⁴²](#). So we suggest to first install Homebrew and then with it you can install Python 3, plus eventually other components you might need. As a reference, for installation we took and simplified this [guide by Digital Ocean⁴³](#)

Attention: check if you already have a package manager !

If you already have installed a package manager like for example Conda (in *Anaconda* distribution), *Rudix*, *Nix*, *Pkgsrc*, *Fink*, or *MacPorts*, maybe Homebrew is not needed and it's better to use what you already have. In these cases, it may be worth asking somebody who knows more ! If you already have *Conda/Anaconda*, it can be ok as long as it is for Python 3.

— 1 Open the Terminal

MacOS terminal is an application you can use to access command line. As any other application, it's available in *Finder*, navigation in *Applications* folder, and the in the folder *Accessories*. From there, double click on the *Terminal* to open it as any other app. As an alternative, you can use *Spotlight* by pressing *Command* and *Space* to find the Terminal typing the name in the bar that appears.

— 2 Install Homebrew by executing in the terminal this command:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

— 3 Add `/usr/local/bin` to PATH

In this passage with an unsettling name, once Homebrew installation is completed, you will make sure that apps installed with Homebrew shall always be used instead of those Mac OS X may automatically select.

⁴² <https://brew.sh/>

⁴³ <https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-local-programming-environment-on-macos>

- 3.1 Open a new Terminal.
- 3.2 From within the terminal, digit the command

```
ls -a
```

You will see the list of all files present in the home folder. In these files, verify if a file exists with the following name: `.profile` (note the dot at the beginning):

- If it exists, go to following step
- If it doesn't exist, to create it type the following command:

```
touch $HOME/.profile
```

- 3.3 Open with text edit the just created file `.profile` giving the command:

```
open -e $HOME/.profile
```

- 3.4 In text edit, add to the end of the file the following line:

```
export PATH=/usr/local/bin:$PATH
```

- 3.5 Save and close both Text Edit and the Terminal

- 4 Verify Homebrew is correctly installed, by typing in a new Terminal:

```
brew doctor
```

If there aren't updates to do, the Terminal should show:

```
Your system is ready to brew.
```

Otherwise, you might see a warning which suggest to execute another command like `brew update` to ensure the Homebrew installation is updated.

- 5. Install python3 (Remember the '3' !):

```
brew install python3
```

Along with python 3, Homebrew will also install the internal package manager of Python `pip3` which we will use in the following.

- 6 Verify Python3 is correctly installed. By executing this command the writing `/usr/local/bin/python3` should appear:

```
which python3
```

After this, try to launch

```
python3
```

You should see something similar:

```
Python 3.6.3 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on mac
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

To exit Python, type `exit()` and press Enter.

3.1.3 Linux installation

Luckily, all Linux distributions are already shipped with package managers to easily install applications.

- If you have Ubuntu:
 1. follow the guide of [Dive into Python 3, chapter 0 - Installare Python⁴⁴](#) in particular by going to the subsection [installing in Ubuntu Linux⁴⁵](#)
 2. after completing the guide, install also `python3-venv`:

```
sudo apt-get install python3-venv
```

- If you *don't* have Ubuntu, [read this note⁴⁶](#) and/or ask somebody who knows more.

To verify the installation, try to run from the terminal

```
python3
```

You should see something like this:

```
Python 3.6.3 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3.2 Installing packages

You can extend Python by installing several free packages. The best way to do it varies according to the operating system and the installed package manager.

ATTENTION: We will be using *system commands*. If you see `>>>` in the command line, it means you are inside Python interpreter and you must first exit: to do it, type `exit()` and press Enter.

In what follows, to check if everything is working, you can substitute `PACKAGENAME` with `requests` which is a module for the web.

If you have Anaconda:

- click on Windows icon in the lower left corner and search `Anaconda Prompt`. A console should appear where to insert commands, with something written like `C:\Users\David>`. (NOTE: to run commands in Anaconda, use only this special console. If you use the default Windows console (`cmd`), Windows will not be able to find Python)
- In the console type `conda install PACKAGENAME`

If you have Linux/Mac open the Terminal and give this command (`--user` install in your home):

- `python3 -m pip install --user PACKAGENAME`
- **NOTE:** If you receive errors which tell you the command `python3` is not found, remove the `3` after `python`

⁴⁴ <https://diveintopython3.problemsolving.io/installing-python.html>

⁴⁵ <https://diveintopython3.problemsolving.io/installing-python.html#ubuntu>

⁴⁶ <https://diveintopython3.problemsolving.io/installing-python.html#other>

INFO: there is also a system command pip (or pip3 according to your system). You can directl call it with pip install --user PACKAGENAME

Instead, we install instead with commands like python3 -m pip install --user PACKAGENAME for uniformity and to be sure to install packages for Python 3 version

3.3 Jupyter Notebook

3.3.1 Run Jupyter notebook

A handy editor you can use for Python is [Jupyter](#)⁴⁷:

- If you installed Anaconda, you should already find it in the system menu and also in the Anaconda Navigator.
- If you didn't install Anaconda, try searching in the system menu anyway, maybe by chance it was already installed
- If you can't find it in the system menu, you may anyway from command line

Try this:

```
jupyter notebook
```

or, as alternative,

```
python3 -m notebook
```

ATTENTION: jupyter is NOT a Python command, it is a *system* command.

If you see written >>> on command line it means you must first exit Python insterpreter by writing 'exit()' and pressing Enter !

ATTENTION: If Jupyter is not installed you will see error messages, in this case don't panic and [go to installation](#).

A browser should automatically open with Jupyter, and in the console you should see messages like the following ones. In the browser you should see the files of the folders from which you ran Jupyter.

If no browser starts but you see a message like the one here, then copy the address you see in an internet browser, preferably Chrome, Safari or Firefox.

```
$ jupyter notebook
[I 18:18:14.669 NotebookApp] Serving notebooks from local directory: /home/da/Da/prj/
↳softpython/prj
[I 18:18:14.669 NotebookApp] 0 active kernels
[I 18:18:14.669 NotebookApp] The Jupyter Notebook is running at: http://localhost:
↳8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888
[I 18:18:14.669 NotebookApp] Use Control-C to stop this server and shut down all
↳kernels (twice to skip confirmation).
[C 18:18:14.670 NotebookApp]
```

(continues on next page)

⁴⁷ <http://jupyter.org/>

(continued from previous page)

Copy/paste this URL into your browser when you connect for the first time, to login with a token:
<http://localhost:8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888>

ATTENTION 1: in this case the address is <http://localhost:8888/?token=49d4394bac446e291c6ddaf349c9dbffcd2cdc8c848eb888>, but yours will surely be different!

ATTENTION 2: While Jupyter server is active, you can't put commands in the terminal !

In the console you see the server output of Jupyter, which is active and in certain sense 'it has taken control' of the terminal. This means that if you write some commands inside the terminal, these **will not** be executed!

3.3.2 Saving Jupyter notebooks

You can save the current notebook in Jupyter by pressing Control-S while in the browser.

ATTENTION: DO NOT OPEN THE SAME DOCUMENT IN MANY TABS !!

Be careful to not open the same notebook in more than one tab, as modifications in different tabs may overwrite at random ! To avoid these awful situations, make sure to have only one tab per document. If you accidentally open the same notebook in different tabs, just close the additional tab.

Automated savings

Notebook changes are automatically saved every few minutes.

3.3.3 Turning off Jupyter server

Before closing Jupyter server, remember to save in the browser the notebooks you modified so far.

To correctly close Jupyter, *do not* brutally close the terminal. Instead, from the terminal where you ran Jupyter, hit Control-c, a question should appear to which you should answer y (if you don't answer in 5 seconds, you will have to hit control-c again).

```
Shutdown this notebook server (y/[n])? y
[C 11:05:03.062 NotebookApp] Shutdown confirmed
[I 11:05:03.064 NotebookApp] Shutting down kernels
```

3.3.4 Navigating notebooks

(Optional) To improve navigation experience in Jupyter notebooks, you may want to install some Jupyter extension, like `toc2` which shows paragraph headers in the sidebar. To install:

Install the Jupyter contrib extensions⁴⁸:

1a. If you have Anaconda: Open Anaconda Prompt (or Terminal if on Mac/Linux), and type:

```
conda install -c conda-forge jupyter_contrib_nbextensions
```

1b. If you don't have Anaconda: Open the terminal and type:

```
python3 -m pip install --user jupyter_contrib_nbextensions
```

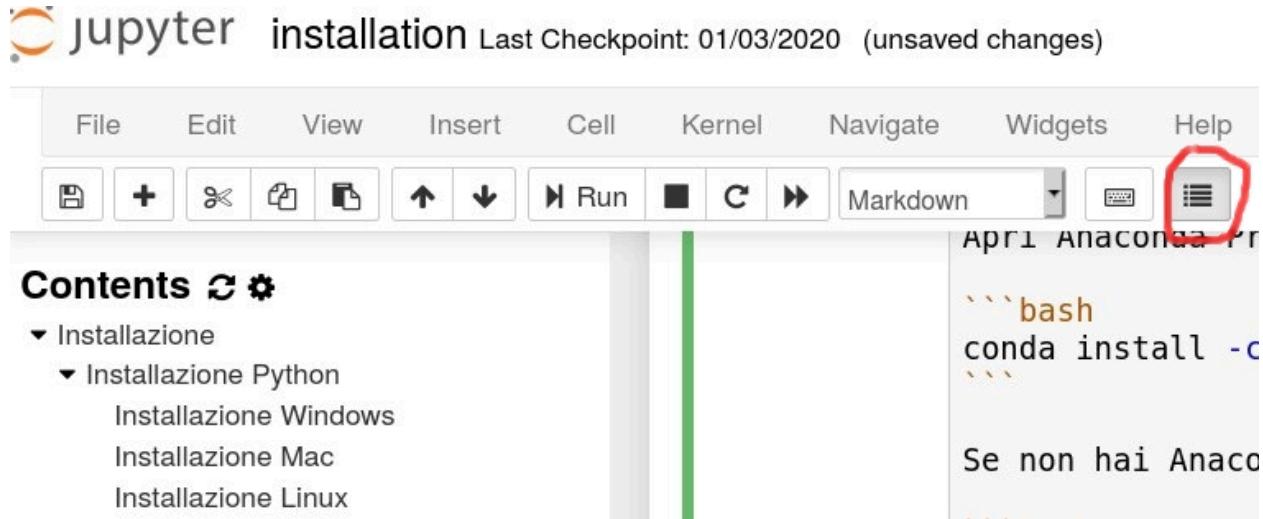
2. Install in Jupyter:

```
jupyter contrib nbextension install --user
```

3. Enable extensions:

```
jupyter nbextension enable toc2/main
```

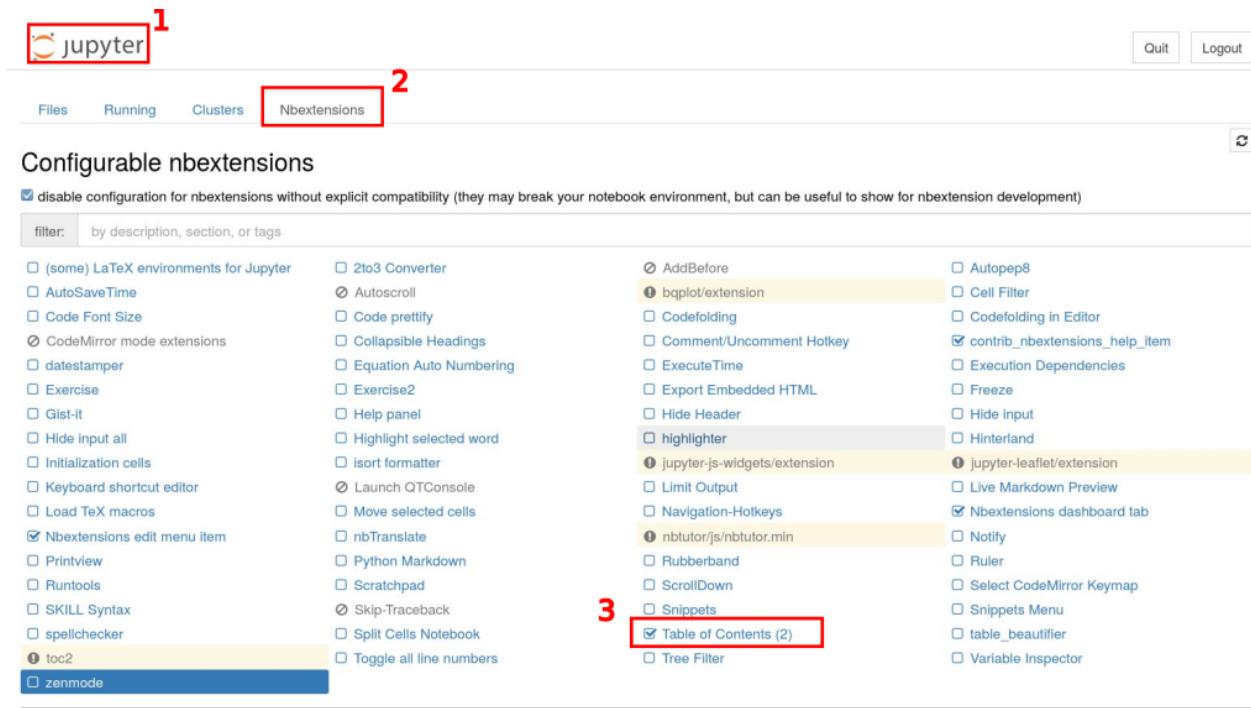
Once installed: To see table of contents in a document you will have to press a list button on the right side of the toolbar:



If by chance you don't see the button:

1. go to main Jupyter interface
2. check Nbextensions tab
3. make sure Table of Contents (2) is enabled
4. Close Jupyter, reopen it, go to a notebook, you should finally see the button

⁴⁸ https://github.com/ipython-contrib/jupyter_contrib_nbextensions



3.3.5 Installing Jupyter notebook - all operating systems

If you didn't manage to [find and/or start Jupyter](#), probably it means we need to install it!

You may try installing Jupyter with pip (the native package manager of Python)

To install, run this command:

```
python3 -m pip install --user jupyter -U
```

Once installed, follow the section

Una volta installato, segui la sezione [Run Jupyter Notebook](#)

ATTENTION: you DON'T need to install Jupyter inside [virtual environments](#). You can consider Jupyter as a system-level application, which should be independent from virtual environments. If you are inside a virtual environment (i.e. the command line begins with a writing in parenthesis like `(myprj)`), exit the environment by typeing `deactivate`

HELP: if you have trouble installing Jupyter, while waiting for help you can always try the [online demo version](#)⁴⁹ (note: it's not always available) or [Google Colab](#)⁵⁰

⁴⁹ <https://try.jupyter.org/>

⁵⁰ <http://colab.research.google.com/>

3.4 Projects with virtual environments

WARNING: If these are your first steps with Python, you can skip this section.

You should read it if you have already done personal projects with Python that you want to avoid compromising, or when you want to make a project to ship to somebody.

When we start a new project with Python, we usually notice quickly that we need to extend Python with particular libraries, like for example to draw charts. Not only that, we might also want to install Python programs which are not written by us and they might as well need their peculiar libraries to work.

Now, we could install all these extra libraries in a unique cauldron for the whole computer, but each project may require its specific versions of each library, and sometimes it might not like versions already installed by other projects. Even worse, it might automatically update packages used by old projects, preventing old code from working anymore. So it is PRACTICALLY NECESSARY to separate well each project and its dependencies from those of other projects: for this purpose you can create a so-called *virtual environment*.

3.4.1 Creating virtual environments

- **If you installed Anaconda**, to create virtual environments you can use its package manager `conda`. Supposing we want to call our project `myprj` (but it could be any name), to put into a folder with the same name `myprj`, we can use this command to create a virtual environment:

```
conda create -n myprj
```

The command might require you to download packages, you can safely confirm.

- **If you *don't have* Anaconda installed**, to create virtual environments it's best to use the native Python module `venv`:

```
python3 -m venv myprj
```

Both methods create the folder `myprj` and fill it with all required Python files to have a project completely isolated from the rest of the computer. But now, how can we tell Python we want to work right with that project? We must *activate* the environment as follows.

3.4.2 Activate a virtual environment

To activate the virtual environment, we must use different commands according to our operating system (but always from the terminal)

Activate environment in Windows with Anaconda:

```
activate myprj
```

Linux & Mac (without Anaconda):

```
source myprj/bin/activate
```

Once the environment is active, in the command prompt we should see the name of that environment (in this case `myprj`) between round parenthesis at the beginning of the row:

```
(myprj) some/current/folder >
```

The prefix lets us know that the environment `myprj` is currently active, so Python commands we will use all use the settings and libraries of that environment.

Note: inside the virtual environment, we can use the command `python` instead of `python3` and `pip` instead of `pip3`

Deactivate an environment:

Write in the console the command `deactivate`. Once the environment is deactivated, the environment name (`myprj`) at the beginning of the prompt should disappear.

3.4.3 Executing environments inside Jupyter

As we said before, Jupyter is a system-level application, so there should be one and only one Jupyter. Nevertheless, during Jupyter execution, we might want to execute our Python commands in a particular Python environment. To do so, we must configure Jupyter so to use the desired environment. In Jupyter terminology, the configurations are called *kernel*: they define the programs launched by Jupyter (be they Python versions or also other languages like R). The current kernel for a notebook is visible in the right-upper corner. To select a desired kernel, there are several ways:

With Anaconda

Jupyter should be available in the Navigator. If in the Navigator you enable an environment (like for example Python 3), when you then launch Jupyter and create a notebook you should have the desired environment active, or at least be able to select a kernel with that environment.

Without Anaconda

In this case, the procedure is a little more complex:

- 1 From the terminal [activate your environment](#Activate-a-virtual-environment)
- 2 Create a Jupyter kernel:

```
python3 -m ipykernel install --user --name myprj
```

NOTE: here `myprj` is the name of the *Jupyter kernel*. We use the same name of the environment only for practical reasons.

- 3 Deactivate your environment, by launching

```
deactivate
```

From now on, every time you run Jupyter, if everything went well under the `Kernel` menu in the notebook you should be able to select the kernel just created (in this example, it should have the name `myprj`)

NOTE: the passage to create the kernel must be done only once per project

NOTE: you don't need to activate the environment before running Jupyter!

During the execution of Jupyter simply select the desired kernel. Nevertheless, it is convenient to execute Jupyter from the folder of our virtual environment, so we will see all the project files in the Jupyter home.

3.5 Further readings

Go on with the page [Tools and scripts⁵¹](#) to learn how to use other editors and Python architecture.

⁵¹ <https://en.softpython.org/tools/tools-sol.html>

A - FOUNDATIONS

4.1 Quick introduction to Python

4.1.1 Download exercises zip

Browse files online⁵²

REQUIREMENTS:

- **THIS WORKSHEET IS INTENDED FOR PEOPLE WHO ALREADY HAVE PROGRAMMING SKILLS**, and in 3-4h course want to rapidly get an idea of Python
- **Having installed Python 3 and Jupyter**: if you haven't already, have a look at [Installation](#)⁵³

IF YOU ARE A BEGINNER:

Skip this worksheet and do instead the tutorials you find in the section [Foundations](#)⁵⁴, starting from [Tools and scripts](#)⁵⁵

What to do

- extract the zip in a folder

The notebook file MUST be in the extracted folder.

Otherwise it won't be properly visualized!

you should obtain something like this:

```
quick-intro
  quick-intro.ipynb
  quick-intro-sol.ipynb
  jupman.py
```

⁵² <https://github.com/DavidLeoni/softpython-it/tree/master/quick-intro>

⁵³ <https://en.softpython.org/installation.html>

⁵⁴ <https://en.softpython.org/index.html#foundations>

⁵⁵ <https://en.softpython.org/tools/tools-sol.html>

ONLY USE PYTHON 3 in this book

- If by chance you get surprising behaviours, check you are actually using Python 3 and not the 2.
- If by issuing `python` your operating system by chance runs python 2, try executing instead the command `python3`

- open Jupyter Notebook in that folder. Two things should open, first a console and then a browser. The browser should show a list of files: browser the list and open the notebook `quick-intro.ipynb`
- Keep reading the exercises file, every now and then you will find inside **EXERCISE** headers, which will ask you to write Python commands in the following cells. The exercises are marked by difficulty, from \oplus to $\oplus\oplus\oplus\oplus$ stars

Always remember to execute the first cell inside the notebook.

It contains instructions like `import jupman` which tell Python which modules are needed and how to find them. To execute it, see the following shortcuts

Keyboard shortcuts for Windows and Linux users:

- To execute Python code inside a Jupyter cell, hit `Ctrl+Enter`
- To execute Python code inside a Jupyter cell AND select the following cell, hit `Shift+Enter`
- To execute Python code inside a cell AND create a new cell right afterwards, hit `Alt+Enter`
- If by chance the Notebook looks stuck, try selecting `Kernel -> Restart`

If you are a Mac user, substitute above keys with the following:

- `Ctrl -> Command key ⌘`
- `Shift -> Shift ⇧`
- `Alt -> Option ⌥`

4.1.2 Let's try Jupyter

Let's briefly have a look at how Jupyter notebooks work.

EXERCISE: Try inserting a Python command: write in the cell below $3 + 5$, and while you are in the cell hit the special keys `Control+Enter`. As a result, you should see the number 8

[]:

EXERCISE: in Python we can write comments by starting a row with a crosshair `#`. As before, write in the cell below $3 + 5$ but this time write in the row below the writing `# write here`:

[2]: # write here

EXERCISE: for each cell Jupyter shows the result only in the last executed row in that cell. Try inserting this code in the cell below and execute by hitting `Control+Enter`. What is the result?

```
3 + 5  
1 + 1
```

[3] : # write here

EXERCISE: Let's try creating a new cell

- While you have the cursor in this cell, hit Alt+Enter. A new cell should open after the current one.
- In the newly created cell, insert `2 + 3` and then hit Shift+Enter. What happens to the cursor? Try the differences with Control+Enter. If you don't understand the difference, try hitting Shift+Enter repeatedly and see what happens.

4.1.3 Main types of data in Python

Since the book's theme is data processing, to start with we will focus on data types in Python.

References:

- Foundations - Data Types⁵⁶

Whenever we read data from an external data source like a file, we will inevitably be forced to embed the data we read into some combination of these types:

Type	Ex- emple 1	Exemple 2	Exemple 3
int	0	3	-5
float (floating point number)	0.0	3.7	-2.3
bool	False	True	
string	" "	"Good morning"	'How are you?'
list	[]	[5, 8, 10]	["give me", 5, "something"]
dict	{ }	{'key 1': 'value 1', 'key 2': 'value 2'}	{5: 'a string value', 'some string key': 7}

Sometimes we will use more complex types, for example we could store temporal values into the type `datetime` which can also hold the timezone.

In what follows, we will provide some brief example about what we can do on various data types, putting references to more detailed explanations in the book.

4.1.4 Integer and floating point numbers

We put here a couple of brief notes.

References:

- Foundations - A.1 Data types - Basics⁵⁷

In Python we have integer numbers:

[4] : 3 + 5

⁵⁶ <https://en.softpython.org/#data-types>

⁵⁷ <https://en.softpython.org/#basics>

[4]: 8

The sum between integers obviously gives us an integer:

[5]: type(8)

[5]: int

What if we divide integers? We will find the floating point type float:

[6]: 3 / 4

[6]: 0.75

[7]: type(0.75)

[7]: float

BEWARE of the dot !

Might be in your country you are used to express decimals with a comma ,

In Python and in many data formats, you always have to use the English dot . format.

⊕ EXERCISE: Try writing down here 3 . 14 with the dot, then 3 , 14 with a comma and execute with Ctrl+Enter. What happens in the two cases?

[8]: # write here with the dot

[9]: # write here with the comma

⊕ EXERCISE: Try writing down here 3 + 1 . 0 and execute with Ctrl+Enter. What is the result type? Check also using the command type.

[10]: # write here the commands

⊕ EXERCISE: Some math professor must have surely warned you to never divide by zero. Python doesn't like it neither. Try writing in the cell below 1 / 0 and then hit Ctrl+Enter to execute the cell, note Python will show the row where the error happened:

[11]: # write here the code

4.1.5 Booleans - bool

Booleans represent true and false values, and we can use them to verify when some condition happens.

References

- Basics - booleans⁵⁸
- Control flow - if⁵⁹

To denote booleans, Python provides two constants `True` and `False`. What can we do with them?

and operator

We might use them to save in variables whether or not a certain fact happened, for example to start the day we might make a program which tells us we can exit home only if we both had breakfast and cleaned teeth:

```
[12]: breakfast = True
cleaned_teeth = True

if breakfast and cleaned_teeth:
    print("done everything !")
    print("can exit home")
else:
    print("CAN'T exit home")

done everything !
can exit home
```

⊕ EXERCISE: try to manually write down here the program from the previous cell, and execute it with Ctrl+Enter. Try changing the values from `True` to `False` and see what happens.

Make sure to try all these cases:

- True True
- True False
- False True
- False False

WARNING: Remember the `:` at the end of the `if` row !!!!

```
[13]: # write here
```

You can also place an `if` inside another, obtaining a so called *nested if*. For example, this program works exactly as the previous one:

```
[14]: breakfast = True
cleaned_teeth = True
```

(continues on next page)

⁵⁸ <https://en.softpython.org/basics/basics2-bools-sol.html>

⁵⁹ <https://en.softpython.org/#if>

(continued from previous page)

```
if breakfast:
    if cleaned_teeth:
        print("done everything !")
        print("can exit home")
    else:
        print("CAN'T exit home")
else:
    print("CAN'T exit home")

done everything !
can exit home
```

⊗ EXERCISE: Try modifying the previous program to report the state of the various actions that were executed. We list here the possible cases and expected results:

- True False

```
had breakfast
didn't clean teeth
CAN'T exit home
```

- False True
- False False

```
didn't have breakfast
CAN'T exit home
```

- True True

```
had breakfast
cleaned teeth
done everything !
can exit home !
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[15]:

```
# write here

breakfast = True
cleaned_teeth = True

if breakfast:
    print("had breakfast")

    if cleaned_teeth:
        print("cleaned teeth")                      # NOTE: This if block is indented
        print("done everything !")                  #
        print("can exit home!")                    #       w.r.t the      if breakfast
    else:
        print("didn't clean teeth")
        print("CAN'T exit home")

else:
    print("didn't have breakfast")
    print("CAN'T exit home")
```

```
had breakfast
cleaned teeth
done everything !
can exit home!
```

</div>

[15]:
write here

or operator

To verify if *at least one* of the two conditions has occurred, we can use the `or` operator. For example, let's say that in order to have breakfast we need some whole or skimmed milk. Of course, we should have breakfast also when we have both!

[16]:
have_whole_milk = **True**
have_skimmed_milk = **False**

if have_whole_milk **or** have_skimmed_milk:
 print("can have breakfast !")
else:
 print("CAN'T have breakfast :-(")

can have breakfast !

⊗ EXERCISE: try to manually write down here the program from the previous cell, and execute with Ctrl+Enter. Try changing values from `True` and `False` and see what happens:

Be sure to try all cases:

- True True
- True False
- False True
- False False

[17]: # write here

⊗⊗ EXERCISE: try writing a program which tells you can exit home only if you had breakfast (so you need at least one kind of milk) and cleaned teeth. Make sure to try the program with all the possible combinations.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[18]:
have_whole_milk = **False**
have_skimmed_milk = **True**

cleaned_teeth = **False**

write here

(continues on next page)

(continued from previous page)

```
if have_whole_milk or have_skimmed_milk:  
    print("can have breakfast !")  
    had_breakfast = True  
else:  
    print("CAN'T have breakfast :-(")  
    had_breakfast = False  
  
if had_breakfast and cleaned_teeth:  
    print("can exit home")  
else:  
    print("CAN'T exit home")
```

```
can have breakfast !  
CAN'T exit home
```

</div>

```
[18]: have_whole_milk = False  
have_skimmed_milk = True  
  
cleaned_teeth = False
```

```
# write here
```



```
can have breakfast !  
CAN'T exit home
```

not operator

For negations, you can use the `not` operator:

```
[19]: not True
```

```
[19]: False
```

```
[20]: not False
```

```
[20]: True
```

```
[21]: had_breakfast = False
```

```
if not had_breakfast:  
    print("I'm hungry !")  
else:  
    print("the cereals were so good")
```

```
I'm hungry !
```

```
[22]: had_breakfast = True

if not had_breakfast:
    print("I' hungry !")
else:
    print("the cereals were so good")
the cereals were so good
```

⊗⊗ EXERCISE: try writing a program which tells you can swim if you DIDN'T have breakfast AND you have a life jacket

Be sure to try all cases:

- True True
- True False
- False True
- False False

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[23]: have_life_jacket = True
had_breakfast = True

# write here

if have_life_jacket and not had_breakfast:
    print("you can swim")
else:
    print("you CAN'T swim")
you CAN'T swim
```

</div>

```
[23]: have_life_jacket = True
had_breakfast = True

# write here

you CAN'T swim
```

Beyond True and False

BEWARE of booleans different from True and False !

In Python, the number 0 and other ‘null’ objects (like the object None, the empty string "" and the empty list []) are considered False, and everything which is not ‘null’ is considered True!

Let's make an example with regular booleans:

```
[24]: if True:  
    print("this")  
    print("will")  
    print("be printed")  
else:  
    print("this other one")  
    print("won't be printed")  
  
this  
will  
be printed
```

Everything which is not null is considered True, so let's try using the string "hello" instead of True:

```
[25]: if "hello":  
    print("this also")  
    print("will be printed!!")  
else:  
    print("this won't")  
  
this also  
will be printed!!
```

```
[26]: if False:  
    print("I won't be printed")  
else:  
    print("I will")  
  
I will
```

```
[27]: if 0:  
    print("this also won't be printed")  
else:  
    print("I will")  
  
I will
```

```
[28]: if None:  
    print("this won't be printed either")  
else:  
    print("I will")  
  
I will
```

```
[29]: if "": # empty string  
    print("Not even this one will be printed!!!!")  
else:  
    print("I will")
```

```
I will
```

⊕ EXERCISE: Copy down here the `if` with a space " " inside the `if` condition. What will happen?

- try also to place an empty list `[]`, what happens?

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[30]:

```
# write here the if

if " ": # space
    print("No print!")
else:
    print("I will be printed")

if []: # empty list
    print("No print")
else:
    print("I will be printed")
```

```
No print!
I will be printed
```

</div>

[30]:

```
# write here the if
```

4.1.6 Strings - str

Strings are *immutable* sequences of characters.

References:

- strings 1 - introduction⁶⁰
- strings 2 - operators⁶¹
- strings 3 - basic methods⁶²
- strings 4 - search methods⁶³

Concatenating strings

One of the most frequent things you do is concatenating strings:

[31]:

```
"hello " + "world"
```

[31]:

```
'hello world'
```

⁶⁰ <https://en.softpython.org/strings/strings1-sol.html>

⁶¹ <https://en.softpython.org/strings/strings2-sol.html>

⁶² <https://en.softpython.org/strings/strings3-sol.html>

⁶³ <https://en.softpython.org/strings/strings4-sol.html>

Note that when we concatenate a string and a number, Python gets angry:

```
"hello " + 5
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-38-e219e8205f7d> in <module>()
----> 1 "hello " + 5

TypeError: Can't convert 'int' object to str implicitly
```

This happens because PYthon wants us to explicitly convert the number 5 into a string. It will also similarly complain about other data types. So, whenever you concatenate objects which are not strings, to avoid problems you can enclose the object to convert within an `str` function call like here:

```
[32]: "hello " + str(7)
[32]: 'hello 7'
```

An alternative and faster way is by using the formatting percentage operator `%`, which substitutes the occurrences of the placeholder `%s` with whatever you place after the string:

```
[33]: "hello %s" % 7
[33]: 'hello 7'
```

Even better, the `%s` can stay withing the string and be repeated. For each occurrence you can pass a different object to substitute, like for example in the tuple `("nice", "Python")` (a tuple is simply an immutable sequence of elements separated by commas within round parenthesis).

```
[34]: "It's so %s I'm finally learning %s" % ("nice", "Python")
[34]: "It's so nice I'm finally learning Python"
```

⊗ **EXERCISE:** the placeholder `%s` works with strings as well as any other data type, for example intergers. Write below here the command above, placing a `%s` at the end of the string, and adding at the tuple's end the number 3 (separated by a comma).

Question: Can you place many `%s` one after another without spaces? Try.

```
[35]: # write here
```

Using object methods

Almost everything in Python is an object, here we make a rapid introduction to just give an idea.

References

- Think in Python, Chapter 15, Classes and objects⁶⁴
- Think in Python, Chapter 16, Classes and functions⁶⁵
- Think in Python, Chapter 17, Classes and methods⁶⁶

⁶⁴ <https://greenteapress.com/thinkpython2/html/thinkpython2016.html>

⁶⁵ <https://greenteapress.com/thinkpython2/html/thinkpython2017.html>

⁶⁶ <https://greenteapress.com/thinkpython2/html/thinkpython2018.html>

Almost everything in Python is an object. For example, the strings are objects. Every object type has actions called *methods* we can execute on that object. For example, we might want strings which represent names to have the first letter as capitalized: we can try finding a string method that already performs this action. Let's try the existing method "capitaliz`e()`" on the *string* "trento" (note the string is all lowercase.):

```
[36]: "trento".capitalize()
[36]: 'Trento'
```

Python just did the courtesy of upcasing the first letter.

⊕ **EXERCISE:** Write in the cell below "trento". and press TAB: Jupyter should show the available methods for the string. Try the methods `upper()` and `count("t")`

```
[37]: # write here
```

4.1.7 Lists - list

A list in Python is a mutable sequence of elements of possibly different types, into which we can place any element we want.

References:

- Lists 1 - introduction⁶⁷
- Lists 2 - operators⁶⁸
- Lists 3 - basic methods⁶⁹
- Lists 4 - search methods⁷⁰

Let's create a list of strings:

```
[38]: x = ["hello", "soft", "python"]
[39]: x
[39]: ['hello', 'soft', 'python']
```

Lists are sequences of possibly heterogenous elements, so inside you can place anything, integers, strings, dictionaries ...:

```
[40]: x = ["hello", 123, {"a": "b"}]
[41]: x
[41]: ['hello', 123, {'a': 'b'}]
```

To access a particular element in a list, you can use an index among square brackets to denote the position:

```
[42]: # first element
x[0]
```

⁶⁷ <https://en.softpython.org/lists/lists1-sol.html>

⁶⁸ <https://en.softpython.org/lists/lists2-sol.html>

⁶⁹ <https://en.softpython.org/lists/lists3-sol.html>

⁷⁰ <https://en.softpython.org/lists/lists4-sol.html>

```
[42]: 'hello'
```

```
[43]: # second element
```

```
x[1]
```

```
[43]: 123
```

```
[44]: # third element
```

```
x[2]
```

```
[44]: {'a': 'b'}
```

In a list we can change the elements by assignment:

```
[45]: # We change the *second* element:
```

```
x[1] = "soft"
```

```
[46]: x
```

```
[46]: ['hello', 'soft', {'a': 'b'}]
```

```
[47]: x[2] = "python"
```

```
[48]: x
```

```
[48]: ['hello', 'soft', 'python']
```

To obtain the list length, we can use `len`:

```
[49]: x = ["hello", "soft", "python"]
```

```
len(x)
```

```
[49]: 3
```

⊕ **EXERCISE:** try accessing an element outside the list, and see what happens.

- is `x[3]` inside or outside the list?
- Is there some list `x` by which we can write `x[len(x)]` without problems?
- if you use negative indexes, what happens? Try -1, -2, -3, -4 ...

```
[50]: # write here
```

We can add elements to the end of the list by using the method `append`:

```
[51]: x = []
```

```
[52]: x
```

```
[52]: []
```

```
[53]: x.append("hello")
```

```
[54]: x
```

```
[54]: ['hello']
```

```
[55]: x.append("soft")
```

```
[56]: x
```

```
[56]: ['hello', 'soft']
```

```
[57]: x.append("python")
```

```
[58]: x
```

```
[58]: ['hello', 'soft', 'python']
```

Ordering lists

We can comfortably sort lists with the method `.sort`, which works on all the sortable objects. For example, we can order numbers:

IMPORTANT: `.sort()` modifies the list on which it is called, it *doesn't* generate a new one!

```
[59]: x = [8, 2, 4]
```

```
x.sort()
```

```
[60]: x
```

```
[60]: [2, 4, 8]
```

As another example, we can order strings:

```
[61]: x = ['python', 'world', 'hello']
```

```
x.sort()
```

```
x
```

```
[61]: ['hello', 'python', 'world']
```

If we don't want to modify the original list and we want to generate a new one instead, we can use the function `sorted()`.

NOTE: `sorted` is a *function*, not a *method*:

```
[62]: x = ['python', 'world', 'hello']
```

```
sorted(x)
```

```
[62]: ['hello', 'python', 'world']
```

```
[63]: # original x was not changed:
```

```
x
```

```
[63]: ['python', 'world', 'hello']
```

⊕ **EXERCISE:** What happens if you order strings which contain the same characters but uppercase instead of lowercase? How are they sorted? Do some test.

```
[64]: # write here
```

⊕ **EXERCISE:** What happens if in the same list you place both string and numbers, and try to sort it? Do some test.

```
[65]: # write here
```

Reversed order

Suppose we want to sort the list in reversed order by using `sorted` function. To do so we can tell Python the boolean parameter `reverse` and its value, which in this case will be `True`. In other words, Python allows to specify optional parameters *by name*:

```
[66]: sorted(['mondo', 'python', 'ciao'], reverse=True)
```

```
[66]: ['python', 'mondo', 'ciao']
```

⊕ **EXERCISE:** To find info about `sorted` function, we could have asked some help to Python. To do so Python provides a handy function called `help`, which you could use like so `help(sorted)`. Try executing it in the cell below. Sometimes the help can be quite complex, and it is upon us to find the interesting parameters.

```
[67]: # write here
```

Reverse unordered lists

What if we wanted to reverse a list as it is, without any sorting, for example to pass from `[6, 2, 4]` to `[2, 4, 6]`? By searching the Python library, we can see there is a handy function `reversed()` which takes as parameter the list we want to reverse and generates a new reversed one.

⊕ **EXERCISE:** Try executing `reversed([6, 2, 4])` in the cell below, and see the result. Is it what you expect? In general, and especially in Python 3, whenever we expect a list and instead we get an object called `iterator`, we can solve the issue by passing the result to the function `list()`

```
[68]: # write here the code
```

4.1.8 Dictionaries - dict

Dictionaries are mutable containers which allow us to associate so-called *keys* to *values*. We will make a brief example to give the idea.

References:

1. Dictionaries 1 - intro⁷¹
2. Dictionaries 2 - operators⁷²

⁷¹ <https://en.softpython.org/dictionaries/dictionaries1-sol.html>

⁷² <https://en.softpython.org/dictionaries/dictionaries2-sol.html>

3. Dictionaries 3 - methods⁷³

We can create a dictionary with curly brackets {}, separating the keys from values with a colon :, and separating associations with the comma ,:

```
[69]: d = { 'key 1':'value 1',
           'key 2':'value 2'}
```

To access values, we can use keys among square brackets:

```
[70]: d['key 1']
```

```
[70]: 'value 1'
```

```
[71]: d['key 2']
```

```
[71]: 'value 2'
```

Values: dictionaries can hold whatever we want as values: numbers, strings, tuples, lists, other dictionaries ..

```
[72]: d['key 3'] = 123
```

```
[73]: d
```

```
[73]: {'key 1': 'value 1', 'key 2': 'value 2', 'key 3': 123}
```

```
[74]: d['key 4'] = ('I', 'am', 'a', 'tuple')
```

```
[75]: d
```

```
[75]: {'key 1': 'value 1',
       'key 2': 'value 2',
       'key 3': 123,
       'key 4': ('I', 'am', 'a', 'tuple')}
```

⊕ **EXERCISE:** try inserting into the dictionary some key/value couples with strings as keys and as values other lists and dictionaries

```
[76]: # write here:
```

Keys: Keys have the important restriction: they must be an *immutable* type. We've already put strings so we know they must be immutable. Numbers also are immutable:

```
[77]: d[123] = 'value 3'
```

```
[78]: d
```

```
[78]: {'key 1': 'value 1',
       'key 2': 'value 2',
       'key 3': 123,
       'key 4': ('I', 'am', 'a', 'tuple'),
       123: 'value 3'}
```

Tuples are an immutable sequence so we can use them as keys:

⁷³ <https://en.softpython.org/dictionaries/dictionaries3-sol.html>

```
[79]: d[('I', 'am', 'a', 'tuple')] = 'value 4'
```

```
[80]: d
```

```
[80]: {'key 1': 'value 1',
       'key 2': 'value 2',
       'key 3': 123,
       'key 4': ('I', 'am', 'a', 'tuple'),
       123: 'value 3',
       ('I', 'am', 'a', 'tuple'): 'value 4'}
```

WARNING: Not all types are viable as keys. Without going into details, in general you cannot insert into dictionaries as keys types which can be modified after they were created.

⊗ EXERCISE:

- Try inserting in a dictionary a list like ['a', 'b'] as key, and put any value you like as value. Python should complain, and you should see the writing `TypeError: unhashable type: 'list'`
- try also to insert a dictionary as key (for example the empty dictionary {}). Which result do you obtain?

```
[81]: # write here the code
```

4.1.9 Visualizing execution with Python Tutor

We've seen the main data types. Before going on, let's discover the best tools to understand what happens when we execute code. [Python tutor](#)⁷⁴ is a very good website to visualize Python code execution, which allows to progress forward and even *backwards* in the code execution. Take advantage of it as much as you can, it should work with most code you will see in the book. Let's see some example.

Python tutor 1/4

Go to the website pythontutor.com⁷⁵ and select *Python 3*

⁷⁴ <http://pythontutor.com/>

⁷⁵ <http://pythontutor.com/>

pythontutor.com

VISUALIZE Python, Java, JavaScript, TypeScript, Ruby, C, and C++

Python Tutor, created by [Philip Guo \(@pgbovine\)](#), helps people overcome a fundamental barrier to learning programming: understanding what happens as the computer runs each line of source code.

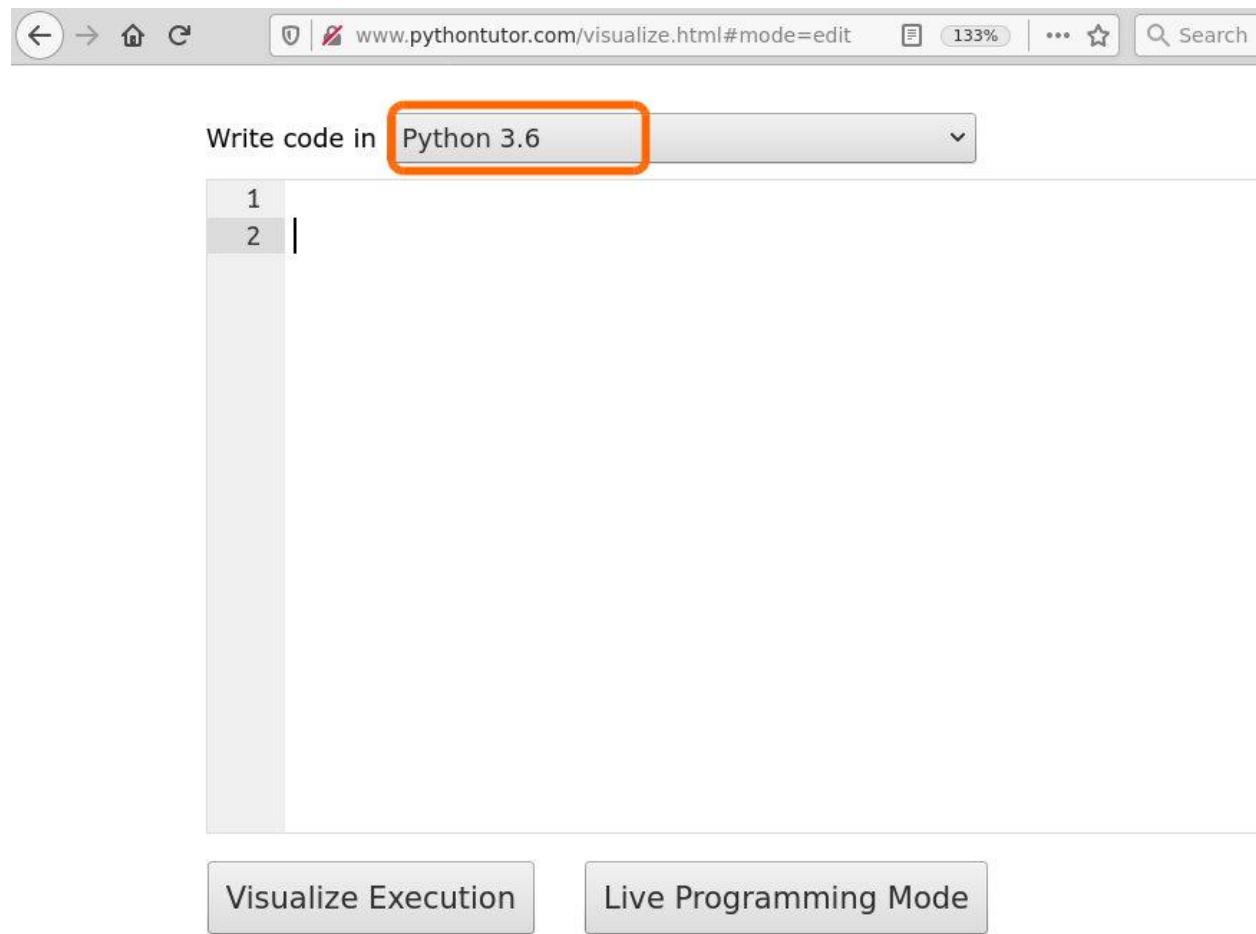
Using this tool, you can write [Python 2](#), [Python 3](#), [Java](#), [JavaScript](#), [TypeScript](#), [Ruby](#), [C](#), and [C++](#) code in your web browser and visualize what the computer is doing step-by-step as it executes.

Over 3.5 million people in over 180 countries have used Python Tutor to visualize over 50 million pieces of code, often as a supplement to textbooks, lectures, and online tutorials.

[Start visualizing your code now](#) (or try [live programming](#))

Python tutor 2/4

Make sure at least Python 3.6 is selected:



Python tutor 3/4

Try inserting:

```
x = 5  
y = 7  
z = x + y
```

Write code in Python 3.6

```

1 x = 5
2 y = 7
3 z = x + y
4

```

Visualize Execution Live Programming Mode

Python tutor 4/4

By clicking on Next, you will see the changes in Python memory

Python 3.6
(known limitations)

```

1 x = 5
2 y = 7
3 z = x + y

```

[Edit this code](#)

line that just executed
next line to execute

<< First < Prev **Next >** Last >>
Step 3 of 3

Customize visualization (NEW!)

unsupported features

Generate permanent link

Frames	Objects
Global frame	x 5 y 7

Debugging code in Jupyter

Python Tutor is fantastic, but when you execute code in Jupyter and it doesn't work, what can you do? To inspect the execution, the editor usually makes available a tool called *debugger*, which allows to execute instructions one by one. At present (August 2018), the Jupyter debugger is called `pdb`⁷⁶ and it is extremely limited. To overcome its limitations, in this book we invented a custom solution based on Python Tutor.

If you insert Python code in a cell, and then **at the cell end** you write the instruction `jupman.pytut()`, the preceding code will be visualized inside Jupyter notebook with Python Tutor, as if by magic.

WARNING: `jupman` is a collection of support functions we created just for this book.

Whenever you see commands which start with `jupman`, to make them work you need first to execute the cell at the beginning of the document. For convenience we report here that cell. If you already didn't, execute it now.

```
[82]: # Remember to execute this cell with Control+Enter
# These commands tell Python where to find the file jupman.py
import jupman;
```

Now we are ready to try Python Tutor with the magic function `jupman.pytut()`:

```
[83]: x = 5
y = 7
z = x + y

jupman.pytut()

[83]: <IPython.core.display.HTML object>
```

Python Tutor : Limitation 1

Python Tutor is handy, but there are important limitations:

ATTENTION: Python Tutor only looks inside one cell!

Whenever you use Python Tutor inside Jupyter, the only code Python tutors considers is the one inside the cell containing the command `jupman.pytut()`

So for example in the two following cells, only `print(w)` will appear inside Python tutor without the `w = 3`. If you try clicking *Forward* in Python tutor, you will be warned that `w` was not defined.

```
[84]: w = 3

[85]: print(w)

jupman.pytut()

3

Traceback (most recent call last):
  File ".../jupman.py", line 2453, in _runscript
```

(continues on next page)

⁷⁶ <https://davidhamann.de/2017/04/22/debugging-jupyter-notebooks/>

(continued from previous page)

```
self.run(script_str, user_globals, user_globals)
File "/usr/lib/python3.7/bdb.py", line 578, in run
    exec(cmd, globals, locals)
File "<string>", line 2, in <module>
NameError: name 'w' is not defined
```

[85]: <IPython.core.display.HTML object>

To have it work in Python Tutor you must put ALL the code in the SAME cell:

```
[86]: w = 3
print(w)

jupman.pytut()

3
```

[86]: <IPython.core.display.HTML object>

Python Tutor : Limitation 2

WARNING: Python Tutor only uses functions from standard Python distribution

Python Tutor is good to inspect simple algorithms with basic Python functions, if you use libraries from third parties it will not work.

If you use some library like numpy, you can try **only online** to select Python 3.6 with Anaconda :

Write code in Python 3.6 with Anaconda (experimental) ▾

```
1  
2 import numpy as np  
3  
4 a = np.arange(15).reshape(3,5)  
5 print(a)|
```

Visualize Execution

4.1.10 Iteration

You will often need to perform actions on every element of a sequence.

References

- Control Flow - for loops⁷⁷
- Control Flow - while loops⁷⁸

For loops

Among the various ways to do it, there is the so called `for` loop:

```
[87]: animals = ['dogs', 'cats', 'squirrels', 'elks']

for animal in animals:
    print("In the list there are:")
    print(animal)
```

⁷⁷ <https://en.softpython.org/#for>

⁷⁸ <https://en.softpython.org/#while>

```
In the list there are:  
dogs  
In the list there are:  
cats  
In the list there are:  
squirrels  
In the list there are:  
elks
```

Here we defined the variable `animal` (we could have called it with any name, also `foo`). For every element in the list `animals`, all the instructions in the block are executed. Everytime they are executed, the variable `animal` becomes one of the values from the list `animals`

WARNING 1: REMEMBER THE TWO DOTS ``::`` AT THE END OF THE `for` LINE !!!

WARNING 2: ALWAYS use sequences of 4 white spaces to indent the code

Sequences of only 2 spaces are still allowed but not recommended.

WARNING 3: TAB behavior may vary depending on your editor.

According to the editor you are using, by hitting TAB you could obtain a sequence of white spaces (i.e. the recommended 4 spaces as it happens in Jupyter), or a special character of tabulation (to avoid)! As much as boring this distinction might look to you, please remember it because it might generate errors very difficult to spot.

[88]: # Let's see what happens with Python tutor:

```
animals = ['dogs', 'cats', 'squirrels', 'elks']

for animal in animals:
    print("In the list there are:")
    print(animal)

jupman.pytut()
```

```
In the list there are:  
dogs  
In the list there are:  
cats  
In the list there are:  
squirrels  
In the list there are:  
elks
```

[88]: <IPython.core.display.HTML object>

⊗ **EXERCISE:** Let's try to understand all the warnings above a bit better. Write down here the previous `for` with the `animals` (no copy and paste!), try if it works. Remember to use 4 spaces for the indentation.

- Try removing the colon at the end and check the error given by Python
- re-add the colon, and try now varying the indentation. Try placing two spaces at the beginning of both prints, check if it works

- try now placing two spaces before the first print and 4 spaces before the second, check if it executes

```
[89]: # write here - copy the above for
```

for in range

Another very common iteration is incrementing a counter at each cycle. Compared to other languages, Python offers a peculiar system based on the function `range(n)`, which returns a sequence with the first numbers from 0 *included* to n *excluded*. We can use it like this:

```
[90]: for index in range(3):  
    print(index)  
  
0  
1  
2
```

```
[91]: for index in range(6):  
    print(index)  
  
0  
1  
2  
3  
4  
5
```

Let's have a better look with Python tutor:

```
[92]: for index in range(6):  
    print(index)  
  
jupman.pytut()  
  
0  
1  
2  
3  
4  
5  
  
[92]: <IPython.core.display.HTML object>
```

As an alternative to list our animals, we can use this style like so:

```
[93]: animals = ['dogs', 'cats', 'squirrels']  
  
for index in range(3):  
    print("In the list there are:")  
    print(animals[index])  
  
In the list there are:  
dogs  
In the list there are:  
cats  
In the list there are:  
squirrels
```

Let's have a better look with Python tutor:

```
[94]: animals = ['dogs', 'cats', 'squirrels']

for index in range(3):
    print("In the list there are:")
    print(animals[index])

jupman.pytut()

In the list there are:
dogs
In the list there are:
cats
In the list there are:
squirrels

[94]: <IPython.core.display.HTML object>
```

4.1.11 Functions

A function takes some parameters and uses them to produce or report some result.

References

- SoftPython - Functions⁷⁹

To define a function, we can use the keyword `def`:

```
[95]: def my_print(x,y):    # REMEMBER THE COLON AT THE END OF THE ROW !!!
        print('We will now print the sum of two numbers')
        print('The sum is %s' % (x + y))
```

We can call the function like this:

```
[96]: my_print(3,5)

We will now print the sum of two numbers
The sum is 8
```

Let's have a better look with Python Tutor:

```
[97]: def my_print(x,y):    # REMEMBER THE COLON AT THE END OF THE ROW !!!
        print('We will now print the sum of two numbers')
        print('The sum is %s' % (x + y))

my_print(3,5)

jupman.pytut()

We will now print the sum of two numbers
The sum is 8

[97]: <IPython.core.display.HTML object>
```

The function we just declared prints some values, but returns nothing. To have a function which actually returns a value, we must use the keyword `return`.

⁷⁹ <https://en.softpython.org/#functions>

```
[98]: def my_sum(x,y):
    s = x + y
    return s
```

```
[99]: my_sum(3,5)
```

```
[99]: 8
```

```
[100]: # Let's have a better look with Python Tutor:
```

```
def my_sum(x,y):
    s = x + y
    return s

print(my_sum(3,5))

jupman.pytut()
```

```
8
```

```
[100]: <IPython.core.display.HTML object>
```

⊗ EXERCISE: If we try to assign to a variable `x` the return value of the function `my_print` which apparently returns nothing, what value will go into `x`? Try to understand it down here:

```
[101]: # write here
```

⊗ EXERCISE: Write down here a function `average` which calculates and return the average of two input numbers `x` and `y`.

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[102]: # write here
```

```
def average(x, y):
    return (x + y) / 2
```

```
</div>
```

```
[102]: # write here
```

⊗⊗ EXERCISE: Write down here a function called `startb` which takes a string `x` as input. If the string begins with the letter '`b`', for example '`bank`' the function prints the writing `bank begins with b`, otherwise prints it doesn't.

- To check whether the first character equals '`b`', use the operator `==` (WARNING: it's DOUBLE equal!)
- Do you envisage any problem if the string is empty? How could you solve them? (to separate more conditions in the `if`, use either the `and` / `or` operators according to the way you built the `if`).

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[103]: # write here

def startb(x):
    if len(x) != 0 and x[0] == 'b':
        print(x + ' starts with b')
    else:
        print(x + " doesn't start with b")

startb('bank')
startb('volley')
startb('')

bank starts with b
volley doesn't start with b
doesn't start with b

</div>
```

```
[103]: # write here
```

```
bank starts with b
volley doesn't start with b
doesn't start with b
```

Lambda functions

Python allows a variable to contain a function. For example, we know that `len("ciao")` gives us the length of the string "ciao"

```
[104]: len("ciao")
```

```
[104]: 4
```

Let's try creating a variable `my_variable` which points to the function `len`.

```
[105]: my_variable = len
```

NOTE: we *didn't* add parameters to `len`!

Now we can use `my_variable` exactly like we use the function `len`, which gives us the length of sequences like strings:

```
[106]: my_variable("ciao")
```

```
[106]: 4
```

We can also reassign `my_variable` to other functions, for example `sorted`. Let's see what happens:

```
[107]: my_variable = sorted
```

by calling `my_variable`, we expect to see the characters of "ciao" in alphabetical order:

```
[108]: my_variable("ciao")
```

```
[108]: ['a', 'c', 'i', 'o']
```

In Python we can define functions in one row with the so-called *lambda functions*:

```
[109]: my_f = lambda x: x + 1
```

What is the `my_f`? It takes a parameter `x` and returns the result of calculating the expression `x + 1`:

```
[110]: my_f(5)
```

```
[110]: 6
```

We can also pass two parameters:

```
[111]: my_sum = lambda x,y: x + y
```

```
[112]: my_sum(3,5)
```

```
[112]: 8
```

⊕ EXERCISE: try defining down here a lambda function to calculate the average between two numbers `x` and `y`, and assign it to the function `average`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[113]:
```

```
# write here
```

```
average = lambda x,y: (x + y) / 2
```

```
average(2,7)
```

```
[113]: 4.5
```

```
</div>
```

```
[113]:
```

```
# write here
```

4.1.12 List transformations

References:

- SoftPython - Sequences⁸⁰

Let's say we want to take the animals list and generate a new one in which all the names start with a capitalized character. As a matter of fact, we are creating a new list by operating a transformation on the previous one. There exist several ways to achieve this goal, the simplest being a `for` cycle like the following.

⁸⁰ <https://en.softpython.org/sequences/sequences1-sol.html>

Transformations with a for

```
[114]: animals = ['dogs', 'cats', 'squirrels', 'elks']

new_list = [] # at every cycle the variable 'animal' contains a name taken from the
             # list 'animals'
for animal in animals:
    new_list.append(animal.capitalize()) # we add the current animal name to the
                                         # new list,
                                         # with the first letter uppercased
new_list

#let's see what happens in Python Tutor
jupman.pytut()

[114]: <IPython.core.display.HTML object>
```

Important note: strings methods never modify the original string, they always generate a new string. So the original list `animals` will still contain the original strings without modifications:

```
[115]: animals
[115]: ['dogs', 'cats', 'squirrels', 'elks']
```

⊕ EXERCISE: Try writing down here a `for` loop (no copy and paste!) to parse the list of animal names and create another list we will call `m`, in which all the characters of animals names are uppercase (use `.upper()` method)

[Show solution](#)[Hide](#)

```
[116]: animals = ['dogs', 'cats', 'squirrels', 'elks']

# write here

m = []
for animal in animals: # at every cycle the variable 'animal' contains a name taken
                     # from the list 'animals'
    m.append(animal.upper()) # we add the current animal name to the new list, with
                           # the first letter uppercased
m

[116]: ['DOGS', 'CATS', 'SQUIRRELS', 'ELKS']
```

</div>

```
[116]: animals = ['dogs', 'cats', 'squirrels', 'elks']

# write here

[116]: ['DOGS', 'CATS', 'SQUIRRELS', 'ELKS']
```

Transformations with *list comprehensions*

References: SoftPython - Sequences⁸¹

The same identical above transformation could be performed with a so-called *list comprehension*, which allows to generate new lists by executing the same operation on all the elements of an existing starting list. The syntax is similar to lists, in fact they start and end with square brackets, but inside you will find a special `for` to cycle through the sequence:

```
[117]: animals = ['dogs', 'cats', 'squirrels', 'elks']

new_list = [animal.capitalize() for animal in animals]
```

```
[118]: new_list
```

```
[118]: ['Dogs', 'Cats', 'Squirrels', 'Elks']
```

Let's see what happens with Python Tutor:

```
[119]: animals = ['dogs', 'cats', 'squirrel', 'elks']

new_list = [animal.capitalize() for animal in animals]

jupman.pytut()

[119]: <IPython.core.display.HTML object>
```

⊕ EXERCISE: Try using a list comprehension to place all the characters as uppercase

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[120]: animals = ['dogs', 'cats', 'squirrels', 'elks']

# write here

new_list = [animal.upper() for animal in animals]
new_list

[120]: ['DOGS', 'CATS', 'SQUIRRELS', 'ELKS']
```

</div>

```
[120]: animals = ['dogs', 'cats', 'squirrels', 'elks']

# write here
```

Filtering with comprehensions:

If we want, we can also filter data by using a special `if` placed at the end of the comprehension. For example, we could select only the animals having a name length of 4 characters:

```
[121]: [animal.upper() for animal in animals if len(animal) == 4]
```

⁸¹ <https://en.softpython.org/sequences/sequences1-sol.html#List-comprehensions>

```
[121]: ['DOGS', 'CATS', 'ELKS']
```

Transformations with map

Yet another way to transform a list into a new one is by using the operation `map`, which given a list, generates another one by applying a function `f` we pass as parameter to each element of the original list. For example, to solve the same previous exercise we could create on the fly a function `f` with a `lambda` which places the first letter of a string as uppercase, then we could call `map` and pass the function we've just created:

```
[122]: animals = ['dogs', 'cats', 'squirrels', 'elks']

f = lambda animal: animal.capitalize()

map(f, animals)

[122]: <map at 0x7fa0cc3a46d0>
```

Sadly, the result is not yet what we wanted. The problem is Python 3 awaits to return a real list, preferring instead to give us an *iterator*. How comes? For efficiency reasons, Python 3 hopes we will never actually use any element from the new sequence, which would spare it the need to compute the function on all the elements of the original list.

We can force it to actually materialize a list by using the function `list`:

```
[123]: animals = ['dogs', 'cats', 'squirrels', 'elks']

f = lambda animal: animal.capitalize()

list(map(f, animals))

[123]: ['Dogs', 'Cats', 'Squirrels', 'Elks']
```

To get a completely equivalent example, we can assign the result to `new_list`:

```
[124]: animals = ['dogs', 'cats', 'squirrels', 'elks']

f = lambda animal: animal.capitalize()

new_list = list(map(f, animals))

[125]: new_list

[125]: ['Dogs', 'Cats', 'Squirrels', 'Elks']
```

A true Python hacker will probably prefer writing everything in one line, like this:

```
[126]: animals = ['dogs', 'cats', 'squirrels', 'elks']

new_list = list(map(lambda animal: animal.capitalize(), animals))

[127]: new_list

[127]: ['Dogs', 'Cats', 'Squirrels', 'Elks']
```

⊕ **EXERCISE:** Was the original list `animals` changed? Check it.

⊕ **EXERCISE:** Given a list `numbers = [3, 5, 2, 7]` try writing a `map` which generates a new list with the numbers doubled, like `[6, 10, 4, 14]`:

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[128]:

```
numbers = [3, 5, 2, 7]

# write here

list(map(lambda x: x * 2, numbers))
```

[128]:

```
[6, 10, 4, 14]
```

```
</div>
```

[128]:

```
numbers = [3, 5, 2, 7]

# write here
```

4.1.13 Matrices

Once we're done with the presentation, it's time to put some more effort. Let's briefly see lists of lists, for more details check the references.

References:

- matrices as lists of lists⁸²
- numpy matrices⁸³

⊕⊕ EXERCISE: Given two lists with animal names and the corresponding life expectancy in years:

```
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12, 14, 30, 6, 25]
```

write in the cell below some code to generate a list of lists of two elements, like so:

```
[
    ['dog', 12],
    ['cat', 14],
    ['pelican', 30],
    ['squirrel', 6],
    ['eagle', 25]
]
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[129]:

```
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12, 14, 30, 6, 25]

# write here
```

(continues on next page)

⁸² <https://en.softpython.org/matrices-lists/matrices-lists1-sol.html>

⁸³ <https://en.softpython.org/matrices-numpy/matrices-numpy1-sol.html>

(continued from previous page)

```

couples = []
for i in range(len(animals)):
    couples.append([animals[i], years[i]])
couples
[129]: [['dog', 12], ['cat', 14], ['pelican', 30], ['squirrel', 6], ['eagle', 25]]

```

</div>

```

[129]: 
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here

```

⊗⊗ **EXERCISE** modify the code of previous exercise with a regular `for` loop to filter only the species with life expectancy above 13 years, so to obtain this result:

```
[['cat', 14], ['pelican', 30], ['eagle', 25]]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```

[130]: 
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here
couples = []
for i in range(len(animals)):
    if years[i] > 13:
        couples.append([animals[i], years[i]])
couples
[130]: [['cat', 14], ['pelican', 30], ['eagle', 25]]

```

</div>

```

[130]: 
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here

```

EXERCISE: Write down here some code with a regular `for` loop so to filter only the species with life expectancy above 10 years and below 27, so to obtain this result:

```
[['dog', 12], ['cat', 14], ['eagle', 25]]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[131]: animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here

couples = []
for i in range(len(animals)):
    if years[i] > 10 and years[i] < 27 :
        couples.append([animals[i], years[i]])
couples
[131]: [['dog', 12], ['cat', 14], ['eagle', 25]]
```

</div>

```
[131]: animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here
```

Zip function

The `zip` function takes two lists and produces a new sequence, in which places couples of elements as tuples (we recall tuples are immutable sequences), coupling the first element from the first list with the first element from the second list, and so on and so forth:

```
[132]: list(zip(['a','b','c'], [5,2,7]))
[132]: [('a', 5), ('b', 2), ('c', 7)]
```

Why did we place also `list` in the example? Because `zip` has the same problem of `map`: it doesn't materialize a list right away as maybe we would like:

```
[133]: zip(['a','b','c'], [5,2,7])
[133]: <zip at 0x7fa0cc377c30>
```

⊕⊕⊕ **EXERCISE:** As you see with the `zip` we can obtain a result similar to that of previous exercise, but here we have tuples with round parenthesis instead of square brackets. Can you obtain the same identical result with a `list comprehension` or a `map` (without filtering, for now)?

- to convert a tuple into a list use the function `list`:

```
[134]: list( ('hello', 'soft', 'python') ) # we placed inside a tuple delimited by round
       ↴brackets
[134]: ['hello', 'soft', 'python']
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[135]: animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
```

(continues on next page)

(continued from previous page)

```

years = [12,14,30,6,25]

# write here - solution with list comprehension

[ list(c) for c in zip(animals, years) ]
[135]: [[['dog', 12], ['cat', 14], ['pelican', 30], ['squirrel', 6], ['eagle', 25]]]

```

</div>

```

[135]:
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here - solution with list comprehension

```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```

[136]:
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here - solution with map

list(map(list, zip(animals, years)))

```

```
[136]: [[['dog', 12], ['cat', 14], ['pelican', 30], ['squirrel', 6], ['eagle', 25]]]
```

</div>

```

[136]:
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here - solution with map

```

⊕⊕⊕ **EXERCISE:** carry out the previous exercise by filtering the animals with life expectancy above 13 years, using `zip` and a list comprehension

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```

[137]:
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12,14,30,6,25]

# write here
[ list(c) for c in zip(animals, years) if c[1] > 13 ]

```

```
[137]: [[['cat', 14], ['pelican', 30], ['eagle', 25]]]
```

</div>

[137]:

```
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12, 14, 30, 6, 25]

# write here
```

⊗⊗ **EXERCISE:** Given the two lists with animal names and the corresponding life expectancy as above, write in the cell below some code that with a regular `for` cycle generates a dictionary which associates each species to its life expectancy, like so:

```
{
    'dog': 12,
    'cat': 14,
    'pelican': 30,
    'squirrel': 6,
    'eagle': 25
}
```

WARNING: order doesn't matter

According to the exact Python version you have and how the dictionary is created, the fields order might differ from the example when printed. This is totally normal, as dictionary keys can be seen as members of a set without any particular order.

If you want to be sure about printing keys in the same insertion order, you have to use an `OrderedDict`⁸⁴

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[138]:

```
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12, 14, 30, 6, 25]

# write here
d = {}
for i in range(len(animals)):
    d[animals[i]] = years[i]
d

[138]: {'dog': 12, 'cat': 14, 'pelican': 30, 'squirrel': 6, 'eagle': 25}
```

```
</div>
```

[138]:

```
animals = ['dog', 'cat', 'pelican', 'squirrel', 'eagle']
years = [12, 14, 30, 6, 25]

# write here
```

To obtain the same result in only one line, it's possible to use the function `zip` as done in previous exercises, and then the function `dict` to create a dictionary starting from the list of element couples generated by the `zip`:

⁸⁴ <https://en.softpython.org/dictionaries/dictionaries4-sol.html#OrderedDict>

```
[139]: dict(zip(animals, years))
[139]: {'dog': 12, 'cat': 14, 'pelican': 30, 'squirrel': 6, 'eagle': 25}
```

⊕⊕ **EXERCISE:** Given a list of products containing lists each with a category, brand and quantity of sold packages:

```
sales = [
    ['tomatoes', 'Santini', 5],
    ['tomatoes', 'Cirio', 1],
    ['tomatoes', 'Mutti', 2],
    ['cereals', 'Kellogggs', 3],
    ['cereals', 'Choco Pops', 8],
    ['chocolate', 'Novi', 9],
    ['chocolate', 'Milka', 4],
]
```

Write some Python code in the cell below to create a dictionary in which the keys are the categories and values are the sum of sold packages for that category, like so:

```
{
    'tomatoes': 8,
    'cereals': 11,
    'chocolate': 13
}
```

- **USE** regular `for` cycle
- **HINT:** pay attention to the two cases, when the dictionary to create still doesn't hold the category extracted from the current list under examination, and when it already contains it.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[140]:
sales = [
    ['tomatoes', 'Santini', 5],
    ['tomatoes', 'Cirio', 1],
    ['tomatoes', 'Mutti', 2],
    ['cereals', 'Kellogggs', 3],
    ['cereals', 'Choco Pops', 8],
    ['chocolate', 'Novi', 9],
    ['chocolate', 'Milka', 4],
]

# write here

d = {}
for sale in sales:
    if sale[0] in d:
        d[sale[0]] += sale[2]
    else:
        d[sale[0]] = sale[2]
d

[140]: {'tomatoes': 8, 'cereals': 11, 'chocolate': 13}
```

</div>

[140]:

```
sales = [
    ['tomatoes', 'Santini', 5],
    ['tomatoes', 'Cirio', 1],
    ['tomatoes', 'Mutti', 2],
    ['cereals', 'Kellogg's', 3],
    ['cereals', 'Choco Pops', 8],
    ['chocolate', 'Novi', 9],
    ['chocolate', 'Milka', 4],
]

# write here
```

4.1.14 Furhter readings

Tools and scripts: If you want execute code in editors other than Jupyter or you're curious about Python architecture, we invite you to read [Tools and scripts⁸⁵](#) page.

Error handling and testing: To understand how to deal with error conditions you can look at the separate notebook [Error handling and testing⁸⁶](#), it's also useful to understand how to solve some exercises of [Part A - Foundations⁸⁷](#)

[]:

4.2 Tools and scripts

4.2.1 Download exercises zip

Browse files online⁸⁸

REQUISITES:

- **Having Python 3 and Jupyter installed:** if you haven't already, see [Installation⁸⁹](#)

4.2.2 Python interpreter

In these tutorials we will use extensively the notebook editor Jupyter, because it allows to comfortably execute Python code, display charts and take notes. But if we want only make calculations it is not mandatory at all!

The most immediate way (even if not very practical) to execute Python things is by using the *command line* interpreter in the so-called *interactive mode*, that is, having Python to wait commands which will be manually inserted one by one. This usage *does not* require Jupyter, you only need to have installed Python. Note that in Mac OS X and many linux systems like Ubuntu, Python is already installed by default, although sometimes it might not be version 3. Let's try to understand which version we have on our system.

⁸⁵ <https://en.softpython.org/tools/tools-sol.html>

⁸⁶ <https://en.softpython.org/functions/fun2-errors-and-testing-sol.html>

⁸⁷ <https://en.softpython.org/#foundations>

⁸⁸ <https://github.com/DavidLeoni/softpython-en/tree/master/tools>

⁸⁹ <https://en.softpython.org/installation.html>

Let's open system console

Open a console (in Windows: system menu -> Anaconda Prompt, in Mac OS X: run the Terminal)

In the console you find the so-called *prompt* of commands. In this *prompt* you can directly insert commands for the operating system.

WARNING: the commands you give in the prompt are commands in the language of the operating system you are using, **NOT** Python language !!!!

In Windows you should see something like this:

```
C:\Users\David>
```

In Mac / Linux it could be something like this:

```
david@my-computer:~$
```

Listing files and folders

In system console, try:

Windows: type the command `dir` and press Enter

Mac or Linux: type the command `ls` and press Enter.

A listing with all the files in the current folder should appear. In my case appears a list like this:

LET ME REPEAT: in this context `dir` and `ls` are commands of *the operating system*, **NOT** of Python !!

Windows:

```
C:\Users\David> dir
Arduino                  gotysc                      program.wav
a.txt                   index.html                  Public
MYFOLDER                java0.log                   RegDocente.pdf
backupsys               java1.log
BaseXData                java_error_in_IDEA_14362.log
```

Mac / Linux:

```
david@david-computer:~$ ls
Arduino                  gotysc                      program.wav
a.txt                   index.html                  Public
MYFOLDER                java0.log                   -
˓→ RegistroDocenteStandard(1).pdf
backupsys               java1.log                   -
˓→ RegistroDocenteStandard.pdf
BaseXData                java_error_in_IDEA_14362.log
```

Let's launch the Python interpreter

In the opened system console, simply type the command `python`:

WARNING: If Python does not run, try typing `python3` with the 3 at the end of `python`

```
C:\Users\David> python
```

You should see appearing something like this (most probably won't be exactly the same). Note that Python version is contained in the first row. If it begins with 2., then you are not using the right one for this book - in that case try exiting the interpreter (*see how to exit*) and then type `python3`

```
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on windows
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

CAREFUL about the triple greater-than `>>>` at the beginning!

The triple greater-than `>>>` at the start tells us that differently from before now the console is expecting commands *in Python language*. So, the system commands we used before (`cd`, `dir`, ...) will NOT work anymore, or will give different results!

Now the console is expecting Python commands, so try inserting `3 + 5` and press Enter:

WARNING DO NOT type `>>>`, only type the command which appears afterwards!

```
>>> 3 + 5
```

The writing 8 should appear:

```
8
```

Beyond calculations, we might tell PYthon to print something with the function `print("ciao")`

```
>>> print("ciao")
ciao
```

Exiting the interpreter

To get out from the Python interpreter and go back to system prompt (that is, the one which accepts `cd` and `dir/ls` commands), type the Python comand `exit()`

After you actually exited the Python interpreter, the triple `>>>` should be gone (you should see it at the start of the line)

In Windows, you should see something similar:

```
>>> exit()
C:\Users\David>
```

in Mac / Linux it could be like this:

```
>>> exit()
david@my-computer:~$
```

Now you might go back to execute commands for the operating system like `dir` and `cd`:

Windows:

```
C:\Users\David> dir
Arduino                      gotysc                  program.wav
a.txt                        index.html             Public
MYFOLDER                     java0.log                RegDocente.pdf
backupsys                    java1.log
BaseXData                    java_error_in_IDEA_14362.log
```

Mac / Linux:

```
david@david-computer:~$ ls
Arduino                      gotysc                  program.wav
a.txt                        index.html             Public
MYFOLDER                     java0.log
→ RegistroDocenteStandard(1).pdf
backupsys                    java1.log
→ RegistroDocenteStandard.pdf
BaseXData                    java_error_in_IDEA_14362.log
```

4.2.3 Modules

Python Modules are simply text files which have the extension `.py` (for example `my_script.py`). When you write code in an editor, as a matter of fact you are implementing the corresponding module.

In Jupyter we use notebook files with the extension `.ipynb`, but to edit them you necessarily need Jupyter.

With `.py` files (also said `script`) we can instead use any text editor, and we can then tell the interpreter to execute that file. Let's see how to do it.

Simple text editor

1. With a text editor (*Notepad* in Windows, or *TextEdit* in Mac Os X) creates a text file, and put inside this code

```
x = 3
y = 5
print(x + y)
```

2. Let's try to save it - it seems easy, but it is often definitely not, so read carefully!

WARNING: when you are saving the file, **make sure the file have the extension `.py` !!**

Let's suppose to create the file `my_script.py` inside a folder called `MYFOLDER`:

- **WINDOWS:** if you use *Notepad*, in the save window you have to set *Save as* to *All files* (otherwise the file will be wrongly saved like `my_script.py.txt` !)
- **MAC:** if you use *TextEdit*, before saving click *Format* and then *Convert to format Only text: if you forget this passage, TextEdit in the save window will not allow you to save in the right format and you will probably end up with a `.rtf` file which we're not interested in*

3. Open a console (in Windows: system menu -> Anaconda Prompt, in Mac OS X: run the Terminal) the console opens the so-called *commands prompt*. In this *prompt* you can directly enter commands for the operating system (see [previous paragraph](#))

WARNING: the commands you give in the prompt are commands in the language of the operating system you are using, **NOT** Python language !!!!

In Windows you should see something like this:

```
C:\Users\David>
```

In Mac / Linux it could be something like this:

```
david@my-computer:~$
```

Try for example to type the command `dir` (or `ls` for Mac / Linux) which shows all the files in the current folder. In my case a list like this appears:

LET ME REPEAT: in this context `dir` / `ls` are commands of the *operating system*, **NOT** Python.

```
C:\Users\David> dir
Arduino           gotysc           program.wav
a.txt            index.html       Public
MYFOLDER         java0.log        RegDocente.pdf
backupsys        java1.log        java_error_in_IDEA_14362.log
BaseXData
```

If you notice, in the list there is the name `MYFOLDER`, where I put `my_script.py`. To *enter* the folder in the *prompt*, you must first use the operating system command `cd` like this:

4. To enter a folder called `MYFOLDER`, type `cd MYFOLDER`:

```
C:\Users\David> cd MYFOLDER
C:\Users\David\MYFOLDER>
```

What if I get into the wrong folder?

If by chance you enter the wrong folder, like `DUMBTHINGS`, to go back of one folder, type `cd ..` (NOTE: `cd` is followed by one space and TWO dots .. *one after the other*)

```
C:\Users\David\DUMBTHINGS> cd ..
C:\Users\David\>
```

5. Make sure to be in the folder which contains `my_script.py`. If you aren't there, use commands `cd` and `cd ..` like above to navigate the folders.

Let's see what present in `MYFOLDER` with the system command `dir` (or `ls` if in Mac/Linux):

LET ME REPEAT: in this context `dir` (or `ls`) is a command of the *operating system*, **NOT** Python.

```
C:\Users\David\MYFOLDER> dir
my_script.py
```

dir is telling us that inside MYFOLDER there is our file my_script.py

6. From within MYFOLDER, type python my_script.py

```
C:\Users\David\MYFOLDER>python my_script.py
```

WARNING: if Python does not run, try typing python3 my_script.py with 3 at the end of python

If everything went fine, you should see

```
8
C:\Users\David\MYFOLDER>
```

WARNING: After executing a script this way, the console is awaiting new *system* commands, **NOT** Python commands (so, there shouldn't be any triple greater-than >>>)

IDE

In these tutorials we work on Jupyter notebooks with extension .ipynb, but to edit long .py files it's more convenient to use more traditional editors, also called IDE (*Integrated Development Environment*). For Python we can use Spyder⁹⁰, Visual Studio Code⁹¹ or PyCharm Community Edition⁹².

Differently from Jupyter, these editors allow more easily code *debugging* and *testing*.

Let's try Spyder, which is the easiest - if you have Anaconda, you find it available inside Anaconda Navigator.

INFO: Whenever you run Spyder, it might ask you to perform an upgrade, in these cases you can just click No.

In the upper-left corner of the editor there is the code of the file .py you are editing. Such files are also said *script*. In the lower-right corner there is the console with the IPython interpreter (which is the same at the heart of Jupyter, here in textual form). When you execute the script, it's like inserting commands in that interpreter.

- To execute the whole script: press F5
- To execute only the current line or the selection: press F9
- To clear memory: after many executions the variables in the memory of the interpreter might get values you don't expect. To clear the memory, click on the gear to the right of the console, and select *Restart kernel*

EXERCISE: do some test, taking the file my_script.py we created before:

```
x = 3
y = 5
print(x + y)
```

- once the code is in the script, hit F5

⁹⁰ <https://www.spyder-ide.org/>

⁹¹ <https://code.visualstudio.com/Download>

⁹² <https://www.jetbrains.com/pycharm/download/>

- select only `print (x+y)` and hit F9
 - select only `x=3` and hit F9
 - click on the gear the right of the console panel, and select *Restart kernel*, then select only `print (x+y)` and hit F9.
What happens?

Remember that if the memory of the interpreter has been cleared with *Restart kernel*, and then you try executing a code row with variables defined in lines which were not executed before, Python will not know which variables you are referring to and will show a `NameError`.

C:\Users\TUser\Documents\Spypy - Spyder (Python 3.6)

File Edit Search Source Run Debug Cell Help

Editor C:\Users\TUser\Downloads\16740156420fb76bb0ba67clee3aae+55140fec17fb7f956874bd8dbd95367816740156420fb76bb0ba67clee3aae+55140fec17fb7f956874bd8dbd953678

Project explorer

Spyder Data spyder .github spyder.recipe continuous_integration doc img_src requirement rope_profiling script editor

app tests _init_.py cli_options.py mac_stylesheet.qss measurement.py restart.py start.py tour.py config defaults fonts imagers locale plugins tests util widgets workers _init_.py dependencies.py interpreter.py spyderconfig.py pil_gif.py pyzmqcompat.py pyplot.py requirements.txt spyder.breakpoints spyder_dockspyder.spyder.spyder.spyder_profiler spyder_pylint checkpointer diocheck encodecopyright iconv_vim coverage gignore pep8speaks.yml project travis.yml Announcements.md zope-efvar.yml

Editor C:\Users\TUser\Downloads\16740156420fb76bb0ba67clee3aae+55140fec17fb7f956874bd8dbd95367816740156420fb76bb0ba67clee3aae+55140fec17fb7f956874bd8dbd953678

Outline

Variable explorer

Name Type Size Value

array_int8 int8 (2, 3) Min: -7 Max: 6

array_uint32 uint32 (2, 2, 3) Min: 1 Max: 1

bars container.BarContainer 20 BarContainer object of matplotlib.container

df DataFrame (3, 2) Column names: bools, ints

filename str 1 C:\ProgramData\Anaconda3\lib\site-pacakge

list_test list 2 [DataFrame, Numpy array]

nrows int 1 344

r float64 1 7.610082589334796

radii float64 (20,) Min: 0.4938463630535687 Max: 9.85884897442551

region tuple 2 (slice, slice)

rgb float64 (45, 45, 4) Min: 0.0 Max: 1.0

series Series (1,) Series object of pandas.core.series module

test_name NoneType 1 NoneType object of builtins module

ipython console Help File explorer Find in files Breakpoints Static code analysis Profiler Online help

Console 1/A Console 2/A Custom Name A 00:34:13

... ls = LightSource(270, 45)
... # To use a custom hillshading mode, override the built-in shading
... # in the 'rgb' colors of the shaded surface calculated from "shade".
... #rgb = ls.shade(z, cmap=cm.gist_earth, vert_exag=0.1, blend_mode='soft')
... surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, facecolors=rgb,
... linewidth=0, antialiased=False, shade=False)
...
... plt.show()

90° 45° 0° 180° 225° 315°

70° 60° 50° 40° 30° 20° 10°

670 650 630 610 590 570 550 530 510 490 470 450 430 410 390 370 350 330 310 290 270 250 230 210 190 170 150 130 110 90 70 50 30 10

In [12]:

iPython console End-of-lines LF Encoding UTF-8 Line: 26 Column: 4 Memory: 49% CPU: 15%

4.2.4 Jupyter

Jupyter is an editor that allows to work on so called *notebooks*, which are files ending with the extension .ipynb. They are documents divided in cells where in each cell you can insert commands and immediately see the respective output. Let's try opening this.

1. Unzip `exercises.zip` in a folder, you should obtain something like this:

```
tools  
    tools-sol.ipynb  
    tools.ipynb  
    jupman.py
```

WARNING: To correctly visualize the notebook, it **MUST** be in the unzipped folder.

2. open Jupyter Notebook. Two things should appear, first a console and then a browser. In the browser navigate the files to reach the unzipped folder, and open the notebook `tools.ipynb`

WARNING: DO NOT click Upload button in Jupyter

Just navigate until you reach the file.

WARNING: open the notebook WITHOUT the -sol at the end!

Seeing now the solutions is too easy ;-)

3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells. Exercises are graded by difficulty, from one star \oplus to four $\oplus\oplus\oplus\oplus$

WARNING: In this book we use ONLY PYTHON 3

If by chance you obtain weird behaviours, check you are using Python 3 and not 2. If by chance by typing `python` your operating system runs python 2, try executing the third by typing the command `python3`

If you don't find Jupyter / something doesn't work: have a look at [installation⁹³](#)

Useful shortcuts:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- when something seem wrong in computations, try to clean memory by running Kernel->Restart and Run all

EXERCISE: Let's try inserting a Python command: type in the cell below here `3 + 5`, then while in that cell press special keys Control+Enter. As a result, the number 8 should appear

[]:

EXERCISE: with Python we can write comments by starting a row with a sharp `#`. Like before, type in the next cell `3 + 5` but this time type it in the row under the writing `# write here`:

[2]: # write here

EXERCISE: In every cell Jupyter only shows the result of last executed row. Try inserting this code in the cell below and execute by pressing Control+Enter. Which result do you see?

```
3 + 5
1 + 1
```

[3]: # write here

EXERCISE: Let's try now to create a new cell.

⁹³ <https://en.softpython.org/installation.html#Jupyter-Notebook>

- While you are with cursor on the cell, press Alt+Enter. A new cell should be created after the current one.
- In the cell just created, insert `2 + 3` and press Shift+Enter. What happens to the cursor? Try the difference with Control+Enter. If you don't understand the difference, try pressing many times Shift+Enter and see what happens.

Printing an expression

Let's try to assign an expression to a variable:

```
[4]: coins = 3 + 2
```

Note the assignment by itself does not produce any output in the Jupyter cell. We can ask Jupyter the value of the variable by simply typing again the name in a cell:

```
[5]: coins
```

```
[5]: 5
```

The effect is (almost always) the same we would obtain by explicitly calling the function `print`:

```
[6]: print(coins)
```

```
5
```

What's the difference? For our convenience Jupyter will directly show the result of the last executed expression in the cell, but only the last one:

```
[7]: coins = 4  
2 + 5  
coins
```

```
[7]: 4
```

If we want to be sure to print both, we need to use the function `print`:

```
[8]: coins = 4  
print(2 + 5)  
print(coins)
```

```
7
```

```
4
```

Furthermore, the result of last expression is shown only in Jupyter notebooks, if you are writing a normal `.py` script and you want to see results you must in any case use `print`.

If we want to print more expressions in one row, we can pass them as different parameters to `print` by separating them with a comma:

```
[9]: coins = 4  
print(2+5, coins)
```

```
7 4
```

To print we can pass as many expressions as we want:

```
[10]: coins = 4  
print(2 + 5, coins, coins*3)
```

```
7 4 12
```

If we also want to show some text, we can write it by creating so-called *strings* between double quotes (we will see strings much more in detail in next chapters):

```
[11]: coins = 4
print("We have", coins, "golden coins, but we would like to have double:", coins * 2)
We have 4 golden coins, but we would like to have double: 8
```

QUESTION: Have a look at following expressions, and for each one of them try to guess the result it produces. Try verifying your guesses both in Jupyter and another editor of files .py like Spyder:

1.

```
x = 1
x
x
```

2.

```
x = 1
x = 2
print(x)
```

3.

```
x = 1
x = 2
x
```

4.

```
x = 1
print(x)
x = 2
print(x)
```

5.

```
print(zam)
print(zam)
zam = 1
zam = 2
```

6.

```
x = 5
print(x,x)
```

7.

```
x = 5
print(x)
print(x)
```

8.

```
carpets = 8
length = 5
print("If I have", carpets, "carpets in sequence I walk for",
      carpets * length, "meters.")
```

9.

```
carpets = 8
length = 5
print("If", "I", "have", carpets, "carpets", "in", "sequence",
      "I", "walk", "for", carpets * length, "meters.")
```

Exercise - Castles in the air

Given two variables

```
castles = 7  
dirigibles = 4
```

write some code to print:

```
I've built 7 castles in the air  
I have 4 steam dirigibles  
I want a dirigible parked at each castle  
So I will buy other 3 at the Steam Market
```

- **DO NOT** put numerical constants in your code like 7, 4 or 3! Write generic code which only uses the provided variables.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[12] :

```
castles = 7  
dirigibles = 4  
# write here  
print("I've built", castles, "castles in the air")  
print("I have", dirigibles, "steam dirigibles")  
print("I want a dirigible parked at each castle")  
print("So I will buy other", castles - dirigibles, "at the Steam Market")  
  
I've built 7 castles in the air  
I have 4 steam dirigibles  
I want a dirigible parked at each castle  
So I will buy other 3 at the Steam Market
```

</div>

[12] :

```
castles = 7  
dirigibles = 4  
# write here
```

4.2.5 Visualizing the execution with Python Tutor

We have seen some of the main data types. Before going further, let's see the right tools to understand what happens when we execute the code.

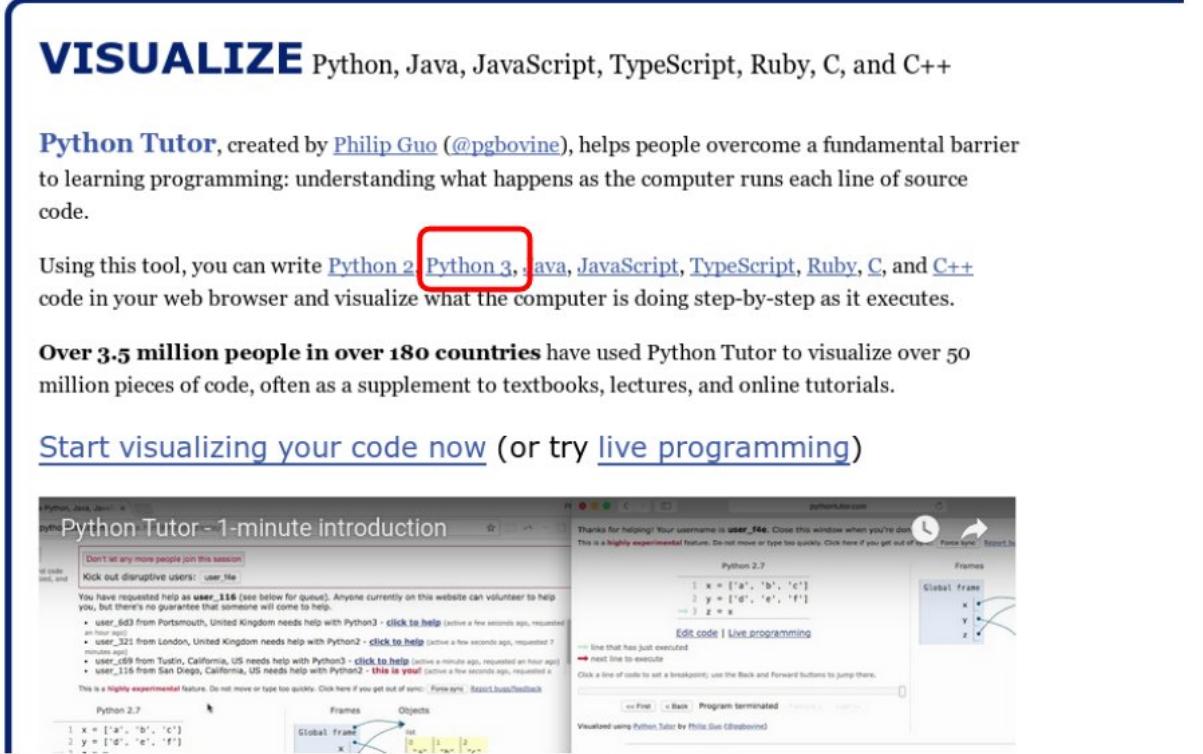
Python tutor⁹⁴ is a very good website to visualize online Python code execution, allowing to step forth and *back* in code flow. Exploit it as much as you can, it should work with many of the examples we shall see in the book. Let's now try an example.

Python tutor 1/4

Go to pythontutor.com⁹⁵ and select *Python 3*

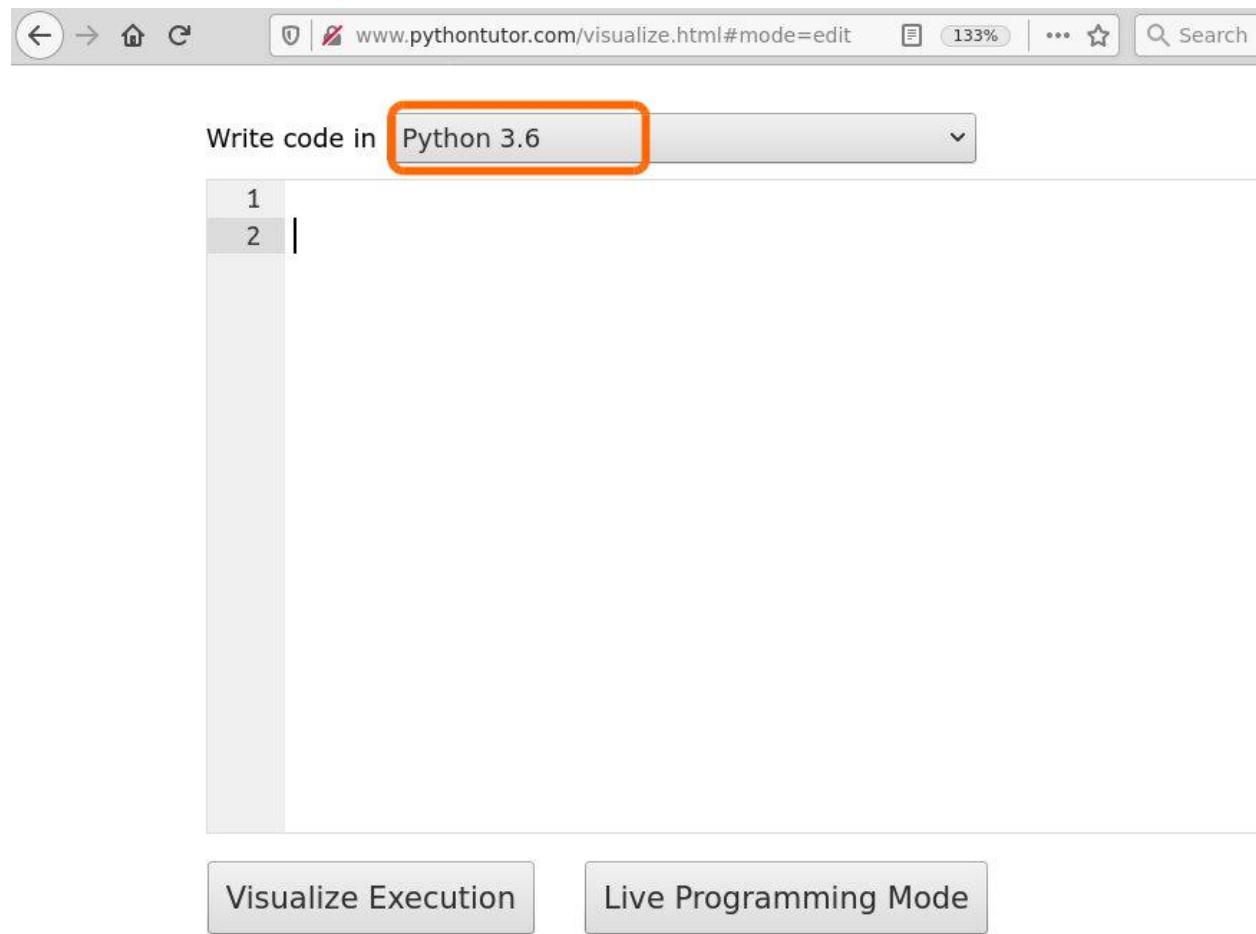
⁹⁴ <http://pythontutor.com/>

⁹⁵ <http://pythontutor.com/>

A screenshot of a web browser window displaying the Python Tutor website. The address bar shows "pythontutor.com". The main heading "VISUALIZE Python, Java, JavaScript, TypeScript, Ruby, C, and C++" is prominently displayed. Below it, a text block explains that Python Tutor, created by Philip Guo (@pgbovine), helps people overcome a fundamental barrier to learning programming: understanding what happens as the computer runs each line of source code. A red box highlights the word "Python" in the sentence "Using this tool, you can write [Python 2](#), [Python 3](#), [Java](#), [JavaScript](#), [TypeScript](#), [Ruby](#), [C](#), and [C++](#) code in your web browser and visualize what the computer is doing step-by-step as it executes." Another red box highlights the word "Python" in the sentence "Over 3.5 million people in over 180 countries have used Python Tutor to visualize over 50 million pieces of code, often as a supplement to textbooks, lectures, and online tutorials." A link "Start visualizing your code now (or try live programming)" is shown. The central part of the screenshot shows a code editor with Python 2.7 code: "x = ['a', 'b', 'c']", "y = ['d', 'e', 'f']", and "z = x". To the right, there's a visualization of variable frames and objects, showing the state of variables x, y, and z.

Python tutor 2/4

Make sure at least Python 3.6 is selected:



Python tutor 3/4

Try inserting:

```
x = 5  
y = 7  
z = x + y
```

Write code in Python 3.6

```

1 x = 5
2 y = 7
3 z = x + y
4

```

Visualize Execution Live Programming Mode

Python tutor 4/4

By clicking on Next, you will see the changes in Python memory

Python 3.6
(known limitations)

```

1 x = 5
2 y = 7
3 z = x + y

```

Edit this code

line that just executed
next line to execute

<< First < Prev Next > Last >>

Step 3 of 3

Customize visualization (NEW!)

unsupported features

Generate permanent link

Debugging code in Jupyter

Python Tutor is fantastic, but when you execute code in Jupyter and it doesn't work, what can you do? To inspect the execution, the editor usually makes available a tool called *debugger*, which allows to execute instructions one by one. At present (August 2018), the Jupyter debugger is called `pdb`⁹⁶ and it is extremely limited. To overcome its limitations, in this book we invented a custom solution which exploits Python Tutor.

If you insert Python code in a cell, and then **at the cell end** you write the instruction `jupman.pytut()`, the preceding code will be visualized inside Jupyter notebook with Python Tutor, as if by magic.

WARNING: `jupman` is a collection of support functions we created just for this book.

Whenever you see commands which start with `jupman`, to make them work you need first to execute the cell at the beginning of the document. For convenience we report here that cell. If you already didn't, execute it now.

```
[13]: # Remember to execute this cell with Control+Enter
# These commands tell Python where to find the file jupman.py
import jupman;
```

Now we are ready to try Python Tutor with the magic function `jupman.pytut()`:

```
[14]: x = 5
y = 7
z = x + y

jupman.pytut()

[14]: <IPython.core.display.HTML object>
```

Python Tutor : Limitation 1

Python Tutor is handy, but there are important limitations:

ATTENTION: Python Tutor only looks inside one cell!

Whenever you use Python Tutor inside Jupyter, the only code Python tutors considers is the one inside the cell containing the command `jupman.pytut()`

So for example in the two following cells, only `print(w)` will appear inside Python tutor without the `w = 3`. If you try clicking *Forward* in Python tutor, you will be warned that `w` was not defined.

```
[15]: w = 3

[16]: print(w)

jupman.pytut()

3

Traceback (most recent call last):
  File ".../jupman.py", line 2453, in _runscript
```

(continues on next page)

⁹⁶ <https://davidhamann.de/2017/04/22/debugging-jupyter-notebooks/>

(continued from previous page)

```
self.run(script_str, user_globals, user_globals)
File "/usr/lib/python3.7/bdb.py", line 578, in run
    exec(cmd, globals, locals)
File "<string>", line 2, in <module>
NameError: name 'w' is not defined
```

[16]: <IPython.core.display.HTML object>

To have it work in Python Tutor you must put ALL the code in the SAME cell:

```
[17]: w = 3
print(w)

jupman.pytut()

3
```

[17]: <IPython.core.display.HTML object>

Python Tutor : Limitation 2

WARNING: Python Tutor only uses functions from standard PYthon distribution

PYthon Tutor is good to inspect simple algorithms with basic Python functions, if you use libraries from third parties it will not work.

If you use some library like numpy, you can try **only online** to select Python 3.6 with Anaconda :

Write code in Python 3.6 with Anaconda (experimental) ▾

```
1
2 import numpy as np
3
4 a = np.arange(15).reshape(3,5)
5 print(a)|
```

Visualize Execution

Exercise - tavern

Given the variables

```
pirates = 10
each_wants = 5      # mugs of grog
kegs = 4
keg_capacity = 20  # mugs of grog
```

Try writing some code which prints:

```
In the tavern there are 10 pirates, each wants 5 mugs of grog
We have 4 kegs full of grog
From each keg we can take 20 mugs
Tonight the pirates will drink 50 mugs, and 30 will remain for tomorrow
```

- **DO NOT** use numerical constants in your code, instead try using the proposed variables
- To keep track of remaining kegs, make a variable `remaining_mugs`
- if you are using Jupyter, try using `jupman.pytut()` at the cell end to visualize execution

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: pirates = 10
each_wants = 5      # mugs of grog
kegs = 4
keg_capacity = 20  # mugs of grog

# write here
print("In the tavern there are", pirates, "pirates, each wants", each_wants, "mugs of grog")
print("We have", kegs, "kegs full of grog")
print("From each keg we can take", keg_capacity, "mugs")
remaining_mugs = kegs*keg_capacity - pirates*each_wants
print("Tonight the pirates will drink", pirates * each_wants, "mugs, and", remaining_mugs, "will remain for tomorrow")

#jupman.pytut()
```

In the tavern there are 10 pirates, each wants 5 mugs of grog
 We have 4 kegs full of grog
 From each keg we can take 20 mugs
 Tonight the pirates will drink 50 mugs, and 30 will remain for tomorrow

</div>

```
[18]: pirates = 10
each_wants = 5      # mugs of grog
kegs = 4
keg_capacity = 20  # mugs of grog

# write here

In the tavern there are 10 pirates, each wants 5 mugs of grog
We have 4 kegs full of grog
From each keg we can take 20 mugs
Tonight the pirates will drink 50 mugs, and 30 will remain for tomorrow
```

4.2.6 Python Architecture

While not strictly fundamental to understand the book, the following part is useful to understand what happens under the hood when you execute commands.

Let's go back to Jupyter: the notebook editor Jupyter is a very powerful tool and flexible, allows to execute Python code, not only that, also code written in other programming languages (R, Bash, etc) and formatting languages (HTML, Markdown, Latex, etc).

We must keep in mind that the Python code we insert in cells of Jupyter notebooks (the files with extension .ipynb) is not certainly magically understood by your computer. Under the hood, a lot of transformations are performed so to allow your computer processor to understand the instructions to be executed. We report here the main transformations which happen, from Jupyter to the processor (CPU):

Python is a high level language

Let's try to understand well what happens when you execute a cell:

1. **source code:** First Jupyter checks if you wrote some Python *source code* in the cell (it could also be other programming languages like R, Bash, or formatting like Markdown ...). By default Jupyter assumes your code is Python. Let's suppose there is the following code:

```
x = 3
y = 5
print(x + y)
```

EXERCISE: Without going into code details, try copy/pasting it into the cell below. Making sure to have the cursor in the cell, execute it with `Control + Enter`. When you execute it an 8 should appear as calculation result. The `# write down here` as all rows beginning with a sharp # is only a comment which will be ignored by Python

```
[19]: # write down here
```

If you managed to execute the code, you can congratulate Python! It allowed you to execute a program written in a quite comprehensible language *independently* from your operating system (Windows, Mac Os X, Linux ...) and from the processor of your computer (x86, ARM, ...)! Not only that, the notebook editor Jupyter also placed the result in your browser.

In detail, what happened? Let's see:

2. **bytecode:** When requesting the execution, Jupyter took the text written in the cell, and sent it to the so-called *Python compiler* which transformed it into *bytecode*. The *bytecode* is a longer sequence of instructions which is less intelligible for us humans (**this is only an example, there is no need to understand it !!**):

```
2      0 LOAD_CONST          1 (3)
      3 STORE_FAST           0 (x)

3      6 LOAD_CONST          2 (5)
      9 STORE_FAST           1 (y)

4     12 LOAD_GLOBAL         0 (print)
     15 LOAD_FAST            0 (x)
     18 LOAD_FAST            1 (y)
     21 BINARY_ADD
     22 CALL_FUNCTION        1 (1 positional, 0 keyword pair)
     25 POP_TOP
     26 LOAD_CONST           0 (None)
     29 RETURN_VALUE
```

3. **machine code:** The *Python interpreter* took the *bytecode* above one instruction per time, and converted it into *machine code* which can actually be understood by the processor (CPU) of your computer. To us the *machine code* may look even longer and uglier of *bytecode* but the processor is happy and by reading it produces the program results. Example of *machine code* (**it is just an example, you do not need to understand it !!**):

```
mult:
    push rbp
    mov rbp, rsp
    mov eax, 0
mult_loop:
    cmp edi, 0
    je mult_end
```

(continues on next page)

(continued from previous page)

```

add eax, esi
sub edi, 1
jmp mult_loop
mult_end:
    pop rbp
    ret

```

We report in a table what we said above. In the table we explicitly write the file extension in which we can write the various code formats

- The ones interesting for us are Jupyter notebooks `.ipynb` and Python source code files `.py`
- `.pyc` file may be generated by the compiler when reading `.py` files, but they are not interesting to us, we will never need to edit them,
- `.asm` machine code also doesn't matter for us

Tool	Language	File extension	Example
Jupyter Notebook	Python	<code>.ipynb</code>	
Python Compiler	Python source code	<code>.py</code>	<code>x = 3 y = 5 print(x + y)</code>
Python Interpreter	Python bytecode	<code>.pyc</code>	<code>0 LOAD_CONST 1 (3) 3 STORE_FAST 0 (x)</code>
Processor (CPU)	Machine code	<code>.asm</code>	<code>cmp edi, 0 je mult _end</code>

No that we now have an idea of what happens, we can maybe understand better the statement *Python is a high level language*, that is, it's positioned high in the above table: when we write Python code, we are not interested in the generated *bytecode* or *machine code*, we can **just focus on the program logic**. Besides, the Python code we write is **independent from the pc architecture**: if we have a Python interpreter installed on a computer, it will take care of converting the high-level code into the machine code of that particular architecture, which includes the operating system (Windows / Mac Os X / Linux) and processor (x86, ARM, PowerPC, etc).

Performance

Everything has a price. If we want to write programs focusing only on the *high level logic* without entering into the details of how it gets interpreted by the processor, we typically need to give up on *performance*. Since Python is an *interpreted* language has the downside of being slow. What if we really need efficiency? Luckily, Python can be extended with code written in *C language* which typically is much more performant. Actually, even if you won't notice it, many functions of Python under the hood are directly written in the fast C language. If you really need performance (not in this book!) it might be worth writing first a prototype in Python and, once established it works, compile it into *C language* by using *Cython compiler*⁹⁷ and manually optimize the generated code.

[]:

⁹⁷ <http://cython.org/>

A1 DATA TYPES

5.1 Basics

5.1.1 Python basics

Download exercises zip

Browse online files⁹⁸

PREREQUISITES:

- **Having installed Python 3 and Jupyter:** if you haven't already, look Installation⁹⁹
- **Having read Tools and scripts**¹⁰⁰

Jupyter

Jupyter is an editor that allows working on so called *notebooks*, which are files ending with the extension .ipynb. They are documents divided in cells where for each cell you can insert commands and immediately see the respective output. Let's try to open this.

1. Unzip `exercises.zip` in a folder, you should obtain something like this:

```
basics
    basics1-ints.ipynb
    basics1-ints-sol.ipynb
    basics2-bools.ipynb
    basics2-bools-sol.ipynb
    basics3-floats.ipynb
    basics3-floats-sol.ipynb
    basics4-chal.ipynb
    jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

⁹⁸ <https://github.com/DavidLeoni/softpython-en/tree/master/basics>

⁹⁹ <https://en.softpython.org/installation.html>

¹⁰⁰ <https://en.softpython.org/tools/tools-sol.html>

2. open Jupyter Notebook. Two things should appear, first a console and then a browser. In the browser navigate the files to reach the unzipped folder, and open the notebook `basics1-ints.ipynb`

WARNING: open the notebook WITHOUT the `-sol` at the end!

Seeing now the solutions is too easy ;-)

3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

WARNING: In this book we use ONLY PYTHON 3

If you obtain weird behaviours, check you are using Python 3 and not 2. If by typing `python` your operating system runs python 2, try executing `python3`

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

Objects

In Python everything is an object. Objects have **properties** (fields where to save values) and **methods** (things they can do). For example, an object `car` has the *properties* model, brand, color, numer of doors, etc ... and the *methods* turn right, turn left, accelerate, brake, shift gear ...

According to Python official documentation:

"Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects."

For now it's enough to know that Python objects have an **identifier** (like, their name), a **type** (numbers, text, collections, ...) and a **value** (the actual value represented by objects). Once the object has been created the *identifier* and the *type* never change, while the *value* may remain constant (**immutable objects**) or it may change (**mutable objects**).

Python provides these predefined types (*built-in*):

Type	Meaning	Domain	Mutable?
<code>bool</code>	Condition	<code>True, False</code>	no
<code>int</code>	Integer	\mathbb{Z}	no
<code>long</code>	Integer	\mathbb{Z}	no
<code>float</code>	Rational	\mathbb{Q} (more or less)	no
<code>str</code>	Text	Text	no
<code>list</code>	Sequence	Collezione di oggetti	yes
<code>tuple</code>	Sequence	Collezione di oggetti	no
<code>set</code>	Set	Collezione di oggetti	yes
<code>dict</code>	Mapping	Mapping between objects	yes

For now we will consider only the simplest ones, later in the book we will deep dive in each of them.

Variables

Variables are associations among names and objects (we can call them values).

Variables can be associated, or in a more technical term, *assigned* to objects by using the assignment operator =.

The instruction

```
[2]: diamonds = 4
```

may represent how many precious stones we keep in the safe. What happens when we execute it in Python?

- an object is created
- its type is set to `int` (an integer number)
- its value is set to 4
- a name `diamonds` is created in the environment and assigned to that object

Detect the type of a variable

When you see a variable or constant and you wonder what type it could have, you can use the predefined function `type`:

```
[3]: type(diamonds)
```

```
[3]: int
```

```
[4]: type(4)
```

```
[4]: int
```

```
[5]: type(4.0)
```

```
[5]: float
```

```
[6]: type("Hello")
```

```
[6]: str
```

Reassign a variable

Consider now the following code:

```
[7]: diamonds = 4
```

```
print(diamonds)
```

```
4
```

```
[8]: diamonds = 5
```

```
print(diamonds)
```

```
5
```

The value of `diamonds` variable has been changed from 4 to 5, but as reported in the previous table, the `int` type is **immutable**. Luckily, this didn't prevent us from changing the value `diamonds` from 4 to 5. What happened behind the scenes? When we executed the instructions `diamonds = 5`, a new object of type `int` was created (the integer 5) and made available with the same name `diamonds`

Reusing a variable

When you reassign a variable to another value, to calculate the new value you can freely reuse the old value of the variable you want to change. For example, suppose to have the variable

```
[9]: flowers = 4
```

and you want to augment the number of `flowers` by one. You can write like this:

```
[10]: flowers = flowers + 1
```

What happened? When Python encounters a command with `=`, FIRST it calculates the value of the expression it finds to the right of the `=`, and THEN assigns that value to the variable to the left of the `=`.

Given this order, FIRST in the expression on the right the old value is used (in this case 4) and 1 is summed so to obtain 5 which is THEN assigned to `flowers`.

```
[11]: flowers
```

```
[11]: 5
```

In a completely equivalent manner, we could rewrite the code like this, using a helper variable `x`. Let's try it in Python Tutor:

```
[12]: # WARNING: to use the following jupman.pytut() function,
# it is necessary first execute this cell with Shift+Enter
# it's enough to execute once, you can also find in all notebooks in the first cell.

import jupman
```

```
[13]:
flowers = 4

x = flowers + 1

flowers = x

jupman.pytut()
```

```
[13]: <IPython.core.display.HTML object>
```

You can execute a sum and do an assignment at the same time with the `+=` notation

```
[14]: flowers = 4
flowers += 1
print(flowers)

5
```

This notation is also valid for other arithmetic operators:

```
[15]: flowers = 5
flowers -= 1      # subtraction
print(flowers)

4
```

```
[16]: flowers *= 3      # multiplication
print(flowers)

12
```

```
[17]: flowers /= 2      # division
print(flowers)

6.0
```

Assignments - questions

QUESTION: Look at the following questions, and for each try to guess the result it produces (or if it gives an error). Try to verify your guess both in Jupyter and in another editor of .py files like Spyder:

1.

```
x = 1
x
x
```

2.

```
x = 1
x = 2
print(x)
```

3.

```
x = 1
x = 2
x
```

4.

```
x = 1
print(x)
x = 2
print(x)
```

5.

```
print(zam)
print(zam)
zam = 1
zam = 2
```

6.

```
x = 5
print(x, x)
```

7.

```
x = 5
print(x)
print(x)
```

8.

```
x = 3
print(x, x*x, x**x)
```

9. `3 + 5 = x
print(x)`

10. `3 + x = 1
print(x)`

11. `x + 3 = 2
print(x)`

12. `x = 2
x += 1
print(x)`

13. `x = 2
x = +1
print(x)`

14. `x = 2
x += 1
print(x)`

15. `x = 3
x *= 2
print(x)`

Exercise - exchange

⊕ Given two variables `a` and `b`:

```
a = 5  
b = 3
```

write some code that exchanges the two values, so that after your code it must result

```
>>> print(a)  
3  
>>> print(b)  
5
```

- are two variables enough? If they aren't, try introducing a third one.

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[18]:

```
a = 5  
b = 3  
  
# write here  
temp = a # associate 5 to temp variable, so we have a copy  
a = b # reassign a to the value of b, that is 3  
b = temp # reassign b to the value of temp, that is 5  
print(a)  
print(b)
```

```
3
5
```

</div>

[18] :

```
a = 5
b = 3

# write here
```

Exercise - cycling

⊕ Write a program that given three variables with numbers a,b,c, cycles the values, that is, puts the value of a in b, the value of b in c, and the value of c in a .

So if you begin like this:

```
a = 4
b = 7
c = 9
```

After the code that you will write, by running this:

```
print(a)
print(b)
print(c)
```

You should see:

```
9
4
7
```

There are various ways to do it, try to use **only one** temporary variable and be careful not to lose values !

HINT: to help yourself, try to write down in comments the state of the memory, and think which command to do

```
# a b c t      which command do I need?
# 4 7 9
# 4 7 9 7     t = b
#
#
#
```

[19] :

```
a = 4
b = 7
c = 9

# write code here

print(a)
print(b)
print(c)
```

```
4  
7  
9
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[20]: # SOLUTION
```

```
a = 4  
b = 7  
c = 9  
  
# a b c t which command do I need?  
# 4 7 9  
# 4 7 9 7 t = b  
# 4 4 9 7 b = a  
# 9 4 9 7 a = c  
# 9 4 7 7 c = t  
  
t = b  
b = a  
a = c  
c = t  
  
print(a)  
print(b)  
print(c)
```

```
9  
4  
7
```

```
</div>
```

```
[20]:
```

```
9  
4  
7
```

Changing type during execution

You can also change the type of a variable during the program execution but normally it is a **bad habit** because it makes harder to understand the code, and increases the probability to commit errors. Let's make an example:

```
[21]: diamonds = 4          # integer
```

```
[22]: diamonds + 2
```

```
[22]: 6
```

```
[23]: diamonds = "four"  # text
```

Now that `diamonds` became text, if by mistake we try to treat it as if it were a number we will get an error !!

```

diamonds + 2

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-9-6124a47997d7> in <module>
----> 1 diamonds + 2

TypeError: can only concatenate str (not "int") to str

```

Multiple commands on the same line

It is possible to put many commands on the same line (non only assignments) by separating them with a semi-colon ;

```
[24]: a = 10; print('So many!'); b = a + 1;
```

So many!

```
[25]: print(a,b)
```

10 11

NOTE: multiple commands on the same line are ‘not much pythonic’

Even if sometimes they may be useful and less verbose of explicit definitions, they are a style frowned upon by true Python ninjas.

Multiple initializations

Another thing are multiple initializations, separated by a comma , like:

```
[26]: x,y = 5,7
```

```
[27]: print(x)
```

5

```
[28]: print(y)
```

7

Differently from multiple commands, multiple assignments are a more acceptable style.

Exercise - exchange like a ninja

⊕ Try now to exchange the value of the two variables a and b in one row with multiple initialization

```
a,b = 5,3
```

After your code, it must result

```
>>> print(a)
3
>>> print(b)
5
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[29]: a,b = 5,3

```
# write here
a,b = b,a
#print(a)
#print(b)
```

</div>

[29]: a,b = 5,3

```
# write here
```

Names of variables

IMPORTANT NOTE:

You can chose the name that you like for your variables (we advise to pick something reminding their meaning), but you need to adhere to some simple rules:

1. Names can only contain upper/lower case digits (A–Z, a–z), numbers (0–9) or underscores _;
2. Names cannot start with a number;
3. Variable names should start with a lowercase letter
4. Names cannot be equal to reserved keywords:

Reserved words:

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

system functions: beyond reserved words (which are impossible to redefine), Python also offers several predefined system function:

- bool, int, float, tuple, str, list, set, dict
- max, min, sum
- next, iter

- `id, dir, vars, help`

Sadly, Python allows careless people to redefine them, but we **do not**:

V COMMANDMENT¹⁰¹ : You shall never ever redefine system functions

Never declare variables with such names !

Names of variables - questions

For each of the following names, try to guess if it is a valid *variable name* or not, then try to assign it in following cell

1. `my-variable`
2. `my_variable`
3. `theCount`
4. `the count`
5. `some@var`
6. `MacDonald`
7. `7channel`
8. `channel7`
9. `stand.by`
10. `channel45`
11. `maybe3maybe`
12. `"ciao"`
13. `'hello'`
14. `as` Please understand the difference between with the following two
15. `asino`
16. `As`
17. `lista` Please understand the difference between with the following two
18. `list` **DO NOT try assigning this one in the interpreter (like `list = 5`), doing so will basically break Python!**
19. `List`
20. `black&decker`
21. `black & decker`
22. `glab()`
23. `caffè` (notice the accented è !)
24. `) :-]`
25. `€zone` (notice the euro sign)
26. `some:pasta`

¹⁰¹ <https://en.softpython.org/commandments.html#V-COMMANDMENT>

27. aren't you bored yet
28. <angular>

```
[30]: # write the names here
```

Numerical types

We already mentioned that numbers are **immutable objects**. Python provides different numerical types: integers (`int`), reals (`float`), booleans, fractions and complex numbers.

It is possible to make arithmetic operations with the following operators, in precedence order:

Operator	Description
<code>**</code>	power
<code>+ -</code>	Unary plus and minus
<code>* / // %</code>	Multiplication, division, integer division, module
<code>+ -</code>	Addition and subtraction

There are also several predefined functions:

Function	Description
<code>min(x, y, ...)</code>	the minimum among given numbers
<code>max(x, y, ...)</code>	the maximum among given numbers
<code>abs(x)</code>	the absolute value

Others are available in the `math`¹⁰² module (remember that in order to use them you must first import the module `math` by typing `import math`):

Function	Description
<code>math.floor(x)</code>	round <code>x</code> to inferior integer
<code>math.ceil(x)</code>	round <code>x</code> to superior integer
<code>math.sqrt(x)</code>	the square root
<code>math.log(x)</code>	the natural logarithm of <code>x</code>
<code>math.log(x, b)</code>	the logarithm of <code>x</code> in base <code>b</code>

... plus many others we don't report here.

Integer numbers

The range of values that integer can have is only limited by available memory. To work with numbers, Python also provides these operators:

```
[31]: 7 + 4
```

```
[31]: 11
```

```
[32]: 7 - 4
```

¹⁰² <https://docs.python.org/3/library/math.html>

```
[32]: 3
```

```
[33]: 7 // 4
```

```
[33]: 1
```

NOTE: the following division among integers produces a **float** result, which uses a **dot** as separator for the decimals (we will see more details later):

```
[34]: 7 / 4
```

```
[34]: 1.75
```

```
[35]: type(7 / 4)
```

```
[35]: float
```

```
[36]: 7 * 4
```

```
[36]: 28
```

NOTE: in many programming languages the power operation is denoted with the cap ^, but in Python it is denoted with double asterisk **:

```
[37]: 7 ** 4    # power
```

```
[37]: 2401
```

Exercise - deadline 1

⊕ You are given a very important deadline in:

```
[38]: days = 4
hours = 13
minutes = 52
```

Write some code that prints the total minutes. By executing it, it should result:

```
In total there are 6592 minutes left.
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[39]:
```

```
days = 4
hours = 13
minutes = 52

# write here
print("In total there are", days*24*60 + hours*60 + minutes, "minutes left")

In total there are 6592 minutes left
```

```
</div>
```

[39]:

```
days = 4
hours = 13
minutes = 52

# write here
```

Modulo operator

To find the remainder of a division among integers, we can use the modulo operator which is denoted with %:

[40]: 5 % 3 # 5 divided by 3 gives 2 as remainder

[40]: 2

[41]: 5 % 4

[41]: 1

[42]: 5 % 5

[42]: 0

[43]: 5 % 6

[43]: 5

[44]: 5 % 7

[44]: 5

Exercise - deadline 2

⊕ For another super important deadline there are left:

```
tot_minutes = 5000
```

Write some code that prints:

```
There are left:
 3 days
 11 hours
 20 minutes
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[45]:

```
tot_minutes = 5000

# write here
print('There are left:')
```

(continues on next page)

(continued from previous page)

```
print(' ', tot_minutes // (60*24), 'days')
print(' ', (tot_minutes % (60*24)) // 60, 'hours')
print(' ', (tot_minutes % (60*24)) % 60, 'minutes')
```

There are left:

3 days
11 hours
20 minutes

</div>

[45]:

```
tot_minutes = 5000

# write here
```

min and max

The minimum among two numbers can be calculated with the function `min`:

[46]:

```
min(7, 3)
```

[46]:

3

and the maximum with the function `max`:

[47]:

```
max(2, 6)
```

[47]:

6

To `min` and `max` we can pass an arbitrary number of parameters, even negatives:

[48]:

```
min(2, 9, -3, 5)
```

[48]:

-3

[49]:

```
max(2, 9, -3, 5)
```

[49]:

9

V COMMANDMENT¹⁰³: You shall never ever redefine system functions like `min` and `max`

If you use `min` and `max` like they were variables, the corresponding functions will *literally* stop to work!

```
min = 4    # NOOOO !
max = 7    # DON'T DO IT !
```

QUESTION: given two numbers `a` and `b`, which of the following expressions are equivalent?

¹⁰³ <https://en.softpython.org/commandments.html#V-COMMANDMENT>

```
1. max(a,b)
2. max(min(a,b),b)
3. -min(-a,-b)
4. -max(-a,-b)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 1. and 3. are equivalent

</div>

Exercise - transportation

⊕⊕ A company has a truck that every day delivers products to its best client. The truck can at most transport 10 tons of material. Unfortunately, the roads it can drive through have bridges that limit the maximum weight a vehicle can have to pass. These limits are provided in 5 variables:

```
b1,b2,b3,b4,b5 = 7,2,4,3,6
```

The truck must always go through the bridge b1, then along the journey there are three possible itineraries available:

- In the first itinerary, the truck also drives through bridge b2
- In the second itinerary, the truck also drives through bridges b3 and b4
- In the third itinerary, the truck also drives though bridge b5

The company wants to know which are the maximum tons it can drive to destination in a single journey. Write some code to print this number.

NOTE: we do not want to know which is the best itinerary, we only need to find the greatest number of tons to ship.

Example - given:

```
b1,b2,b3,b4,b5 = 7,2,4,6,3
```

your code must print:

```
In a single journey we can transport at most 4 tons.
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[50]: b1,b2,b3,b4,b5 = 7,2,4,6,3 # 4
#b1,b2,b3,b4,b5 = 2,6,2,4,5 # 2
#b1,b2,b3,b4,b5 = 8,6,2,9,5 # 6
#b1,b2,b3,b4,b5 = 8,9,9,4,7 # 8
```

```
# write here
```

```
print('In a single journey we can transport at most',
      max(min(b1,b2), min(b1,b3,b4),min(b1,b5)),
      'tons')
```

```
In a single journey we can transport at most 4 tons
```

</div>

```
[50]: b1,b2,b3,b4,b5 = 7,2,4,6,3    # 4
#b1,b2,b3,b4,b5 = 2,6,2,4,5    # 2
#b1,b2,b3,b4,b5 = 8,6,2,9,5    # 6
#b1,b2,b3,b4,b5 = 8,9,9,4,7    # 8

# write here
```

In a single journey we can transport at most 4 tons

Exercise - armchairs

⊕⊕ The tycoon De Industrionis owns two factories of armchairs, one in Belluno city and one in Rovigo. To make an armchair three main components are needed: a mattress, a seatback and a cover. Each factory produces all required components, taking a certain time to produce each component:

```
[51]: b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 23,54,12,13,37,24
```

Belluno takes 23h to produce a mattress, 54h the seatback and 12h the cover. Rovigo, respectively, takes 13, 37 and 24 hours. When the 3 components are ready, assembling them in the finished armchair requires one hour.

Sometimes peculiar requests are made by filthy rich nobles, who pretend to be shipped in a few hours armchairs with extravagant features like seatback in solid platinum and other nonsense.

If the two factories start producing the components at the same time, De Industrionis wants to know in how much time the first armchair will be produced. Write some code to calculate that number.

- **NOTE 1:** we are not interested which factory will produce the armchair, we just want to know the shortest time in which we will get an armchair
- **NOTE 2:** suppose both factories **don't** have components in store
- **NOTE 3:** the two factories **do not** exchange components

Example 1 - given:

```
b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 23,54,12,13,37,24
```

your code must print:

The first armchair will be produced in 38 hours.

Example 2 - given:

```
b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 81,37,32,54,36,91
```

your code must print:

The first armchair will be produced in 82 hours.

Show solution<div class="jupman-sol" jupman-sol-code" style="display:none">

[52]:

```
b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 23,54,12,13,37,24    # 38
#b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 81,37,32,54,36,91    # 82
#b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 21,39,47,54,36,91    # 48

# write here

t = min(max(b_mat, b_bac, b_cov) + 1, max(r_mat, r_bac, r_cov) + 1)

print('The first armchair will be produced in', t,'hours.')
The first armchair will be produced in 38 hours.
```

</div>

[52]:

```
b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 23,54,12,13,37,24    # 38
#b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 81,37,32,54,36,91    # 82
#b_mat, b_bac, b_cov, r_mat, r_bac, r_cov = 21,39,47,54,36,91    # 48

# write here
```

The first armchair will be produced in 38 hours.

Continue

Go on with Basics 2: Booleans¹⁰⁴

5.1.2 Basics 2 - booleans

[Download exercises zip](#)

[Browse online files](#)¹⁰⁵

PREREQUISITES:

- Having read basics 1 integer variables¹⁰⁶

Booleans are used in boolean algebra and have the type `bool`.

Values of truth in Python are represented with the keywords `True` and `False`: a boolean object can only have the values `True` or `False`.

[2]:

```
x = True
```

[3]:

```
x
```

[3]:

```
True
```

¹⁰⁴ <https://en.softpython.org/basics/basics2-bools-sol.html>

¹⁰⁵ <https://github.com/DavidLeoni/softpython-en/tree/master/basics>

¹⁰⁶ <https://en.softpython.org/basics/basics1-ints-sol.html>

[4]: `type(x)`

[4]: `bool`

[5]: `y = False`

[6]: `type(y)`

[6]: `bool`

Boolean operators

We can operate on boolean values with the operators `not`, `and`, or:

`and`

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

`or`

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

`not`

a	not a
False	True
True	False

Questions with constants

QUESTION: For each of the following boolean expressions, try guessing the result (*before* guess, and *then* try them !):

1. `not (True and False)`

2. `(not True) or (not (True or False))`

3. `not (not True)`
4. `not (True and (False or True))`
5. `not (not (not False))`
6. `True and (not (not ((not False) and True)))`
7. `False or (False or ((True and True) and (True and False)))`

Questions with variables

QUESTION: For each of these expressions, for which values of `x` and `y` they give `True`? Try to think an answer before trying!

NOTE: there can be many combinations that produce `True`, find them all

1. `x or (not x)`
2. `(not x) and (not y)`
3. `x and (y or y)`
4. `x and (not y)`
5. `(not x) or y`
6. `y or not (y and x)`
7. `x and ((not x) or not (y))`
8. `(not (not x)) and not (x and y)`
9. `x and (x or (not (x) or not (not (x or not (x)))))`

QUESTION: For each of these expressions, for which values of `x`, `y` and `z` they give `False`?

NOTE: there can be many combinations that produce `False`, find them all

1. `x or ((not y) or z)`
2. `x or (not y) or (not z)`
3. `not (x and y and (not z))`
4. `not (x and (not y) and (x or z))`
5. `y or ((x or y) and (not z))`

De Morgan

There are a couple of laws that sometimes are useful:

Formula	Equivalent to
<code>x or y</code>	<code>not(not x and not y)</code>
<code>x and y</code>	<code>not(not x or not y)</code>

QUESTION: Look at following expressions, and try to rewrite them in equivalent ones by using De Morgan laws, simplifying the result wherever possible. Then verify the translation produces the same result as the original for all possible values of `x` and `y`.

1. `(not x) or y`

2. `(not x) and (not y)`

3. `(not x) and (not (x or y))`

Example:

```
x,y = False, False
#x,y = False, True
#x,y = True, False
#x,y = True, True

orig = x or y
trans = not((not x) and (not y))
print('orig=',orig)
print('trans=',trans)
```

[7]: # verify here

Conversion

We can convert booleans into integers with the predefined function `int`. Each integer can be converted into a boolean (and vice versa) with `bool`:

[8]: `bool(1)`

[8]: `True`

[9]: `bool(0)`

[9]: `False`

[10]: `bool(72)`

[10]: `True`

[11]: `bool(-5)`

```
[11]: True
```

```
[12]: int(True)
```

```
[12]: 1
```

```
[13]: int(False)
```

```
[13]: 0
```

Each integer is valued to True except 0. Note that truth values True and False behave respectively like integers 1 and 0.

Questions - what is a boolean?

QUESTION: For each of these expressions, which results it produces?

```
1. bool(True)
```

```
2. bool(False)
```

```
3. bool(2 + 4)
```

```
4. bool(4-3-1)
```

```
5. int(4-3-1)
```

```
6. True + True
```

```
7. True + False
```

```
8. True - True
```

```
9. True * True
```

Evaluation order

For efficiency reasons, during the evaluation of a boolean expression if Python discovers the possible result can only be one, it then avoids to calculate further expressions. For example, in this expression:

```
False and x
```

by reading from left to right, in the moment we encounter False we already know that the result of and operation will always be False independently from the value of x (convince yourself).

Instead, if while reading from left to right Python finds first True, it will continue the evaluation of following expressions and *as result of the whole "and" will return the evaluation of the *last* expression_*. If we are using booleans, we will not notice the differences, but by exchanging types we might get surprises:

```
[14]: True and 5
```

[14]: 5

[15]: 5 and True

[15]: True

[16]: False and 5

[16]: False

[17]: 5 and False

[17]: False

Let's think which order of evaluation Python might use for the `or` operator. Have a look at the expression:

`True or x`

By reading from left to right, as soon as we find the `True` we might conclude that the result of the whole `or` must be `True` independently from the value of `x` (convince yourself).

Instead, if the first value is `False`, Python will continue in the evaluation until it finds a logical value `True`, when this happens that value will be the result of the whole expression. We can notice it if we use different constants from `True` and `False`:

[18]: False or 5

[18]: 5

[19]: 7 or False

[19]: 7

[20]: 3 or True

[20]: 3

The numbers you see have always a logical result coherent with the operations we did, that is, if you see 0 the expression result is intended to have logical value `False` and if you see a number different from 0 the result is intended to be `True` (convince yourself).

QUESTION: Have a look at the following expressions, and for each of them try to guess which result it produces (or if it gives an error):

1. `0 and True`

2. `1 and 0`

3. `True and -1`

4. `0 and False`

5. `0 or False`

6. `0 or 1`

7. `False or -6`

8. `0 or True`

Evaluation errors

What happens if a boolean expression contains some code that would generate an error? According to intuition, the program should terminate, but it's not always like this.

Let's try to generate an error on purpose. During math lessons they surely told you many times that dividing a number by zero is an error because the result is not defined. So if we try to ask Python what the result of $1/0$ is we will (predictably) get complaints:

```
print(1/0)
print('after')

-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-51-9e1622b385b6> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

Notice that '`after`' is not printed because the program gets first interrupted.

What if we try to write like this?

[21]: `False and 1/0`

[21]: `False`

Python produces a result without complaining! Why? Evaluating from left to right it found a `False` and so it concluded before hand that the expression result must be `False`. Many times you will not be aware of these potential problems but it is good to understand them because there are indeed situations in which you can even exploit the execution order to prevent errors (for example in `if` and `while` instructions we will see later in the book).

QUESTION: Look at the following expression, and for each of them try to guess which result it produces (or if it gives an error):

1. `True and 1/0`

2. `1/0 and 1/0`

3. `False or 1/0`

4. `True or 1/0`

5. `1/0 or True`

6. `1/0 or 1/0`

7. `True or (1/0 and True)`

8. `(not False) or not 1/0`

9. `True and 1/0 and True`

10. `(not True) or 1/0 or True`

11. `True and (not True) and 1/0`

Comparison operators

Comparison operators allow to build *expressions* which return a boolean value:

Comparator	Description
<code>a == b</code>	True if and only if $a = b$
<code>a != b</code>	True if and only if $a \neq b$
<code>a < b</code>	True if and only if $a < b$
<code>a > b</code>	True if and only if $a > b$
<code>a <= b</code>	True if and only if $a \leq b$
<code>a >= b</code>	True if and only if $a \geq b$

[22]: `3 == 3`

[22]: `True`

[23]: `3 == 5`

[23]: `False`

[24]: `a,b = 3,5`

[25]: `a == a`

[25]: `True`

[26]: `a == b`

[26]: `False`

[27]: `a == b - 2`

[27]: `True`

[28]: `3 != 5 # 3 is different from 5 ?`

[28]: `True`

[29]: `3 != 3 # 3 is different from 3 ?`

[29]: `False`

[30]: `3 < 5`

```
[30]: True
```

```
[31]: 5 < 5
```

```
[31]: False
```

```
[32]: 5 <= 5
```

```
[32]: True
```

```
[33]: 8 > 5
```

```
[33]: True
```

```
[34]: 8 > 8
```

```
[34]: False
```

```
[35]: 8 >= 8
```

```
[35]: True
```

Since the comparison are expressions which produce booleans, we can also assign the result to a variable:

```
[36]: x = 5 > 3
```

```
[37]: print(x)
```

```
True
```

Joining comparisons

Given a couple of quantities:

```
[38]: x, y = 7, 5
```

If we ask ourselves whether both `x` and `y` are greater than zero, which boolean expression should we write?

The correct way (although a bit verbose) is the following:

```
[39]: x > 0 and y > 0
```

```
[39]: True
```

WARNING: the following code instead is **NOT** correct:

```
x and y > 0    # WRONG!
```

Why? Apparently on some inputs it seems to work:

```
[40]: x, y = 7, 5
x and y > 0    # WARNING!
```

```
[40]: True
```

```
[41]: x,y = 7, -6
x and y > 0 # WARNING!
```

[41]: False

Does it really work on all of them?

```
[42]: x,y = -3, 5
x and y > 0 # WARNING!
```

[42]: True

Mmm we found something wrong.... What's the reason? Implicitly, Python has considered the expression with these parenthesis:

```
[43]: x,y = -3,5
(x) and (y>0) # WARNING!
```

[43]: True

This way it's evident that on the left Python is considering `x` like it were a single boolean expression: as such, it tries to interpret the logical value of the integer `-3`: since it's a number different from zero, it's considered as `True`, hence the strange result.

QUESTION: What if we tried to place these parenthesis?

```
x,y = -3,5
(x and y) > 0 # WARNING!
```

Try thinking in depth all the steps Python will perform to calculate the expression.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

ANSWER: in this case both `x` and `y` are considered single boolean expressions, so Python internally will reduce the expression with these steps:

1. `(True and True) > 0`
2. `True > 0`
3. `1 > 0`
4. `True`

Note that at step 2) `True` is converted to the integer value `1`.

</div>

QUESTION: Look at the following expression, and for each of them try to guess which result it produces (or if it gives an error):

1. `x = 3 == 4`
`print(x)`
2. `x = False or True`
`print(x)`

```
3. True or False = x or False
   print(x)
```

```
4. x,y = 9,10
   z = x < y and x == 3**2
   print(z)
```

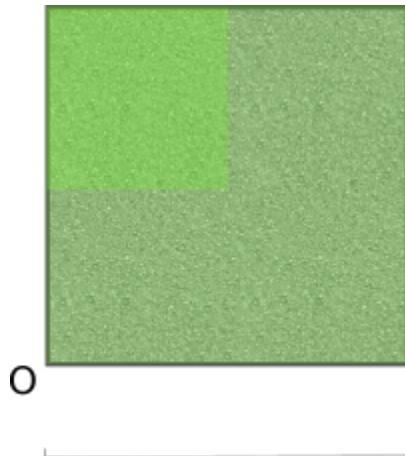
```
5. a,b = 7,6
   a = b
   x = a >= b + 1
   print(x)
```

```
6. x = 3^2
   y = 9
   print(x == y)
```

Exercise - The Lawnmower 1

Dr Angelo owns a squared lawn with a side of 100 meters, which every week is mowed by Jobe the gardener. Jobe always meticolously mows all the field area, but one day the doctor decides to add some variety to the garden and asks Jobe to mow only some zones. Alas, Jobe is not very good in geometry and so Dr Angelo invents a sensor to detect the position, linked to a led which only lights up when the lawnmower must be turned on. Write an expression which given two coordinates **x, y**, produces **True** whenever the lawnmower is in a greenlight grass, and **False** in dark zones.

- Note the origin of the coordinates is in the lower left corner
- **DO NOT use if commands**
- **WRITE** a generic formula by using **d** (so don't write 50...)



Show solutionHide</div>

[44] :

```
d = 100

x,y = 0, 0      # False
#x,y = 25, 25    # False
#x,y = 75, 75    # False
```

(continues on next page)

(continued from previous page)

```
#x,y = 75, 25      # False
#x,y = 25, 75      # True
#x,y = 100, 100    # False
#x,y = 10, 90      # True
#x,y = 60, 60      # False

# write here

x < d/2 and y > d/2
```

[44]: False

</div>

[44]: d = 100

```
x,y = 0, 0        # False
#x,y = 25, 25      # False
#x,y = 75, 75      # False
#x,y = 75, 25      # False
#x,y = 25, 75      # True
#x,y = 100, 100    # False
#x,y = 10, 90      # True
#x,y = 60, 60      # False

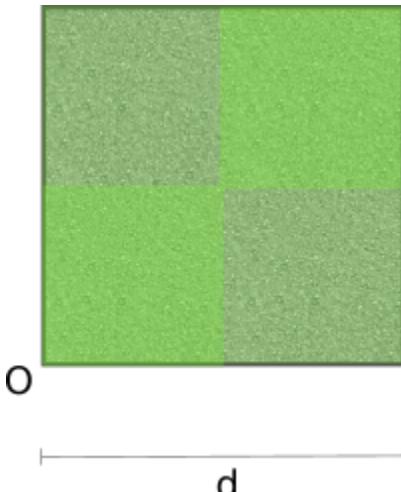
# write here
```

[44]: False

Exercise - The Lawnmower 2

Doctor Angelo now asks Jobe to mow more zones...

- DO NOT use if commands



Show solution</div><div class="jupman-sol jupman-sol-code" style="display:none">

```
[45]: d = 100

x,y = 0, 0      # True
#x,y = 25, 25    # True
#x,y = 75, 75    # True
#x,y = 75, 25    # False
#x,y = 25, 75    # False
#x,y = 100, 100   # True
#x,y = 10, 90     # False
#x,y = 60, 60     # True

# write here

(x < d/2 and y < d/2) or (x > d/2 and y > d/2)
```

```
[45]: True
```

</div>

```
[45]: d = 100

x,y = 0, 0      # True
#x,y = 25, 25    # True
#x,y = 75, 75    # True
#x,y = 75, 25    # False
#x,y = 25, 75    # False
#x,y = 100, 100   # True
#x,y = 10, 90     # False
#x,y = 60, 60     # True

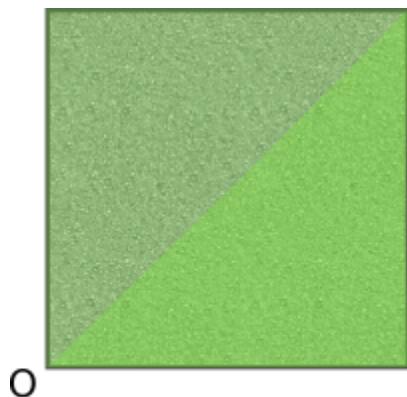
# write here
```

```
[45]: True
```

Exercise - The Lawnmower 3

Dr Angelo got tired of squared gardens, and now wants to split the garden with a diagonal.

- **DO NOT use if commands**



O

— d —

Show solution</div><div class="jupman-sol jupman-sol-code" style="display:none">

[46]: d = 100

```
x,y = 50, 10 # True
#x,y = 100,0 # True
#x,y = 75, 70 # True
#x,y = 25,75 # False
#x,y = 25, 5 # True
#x,y = 5, 80 # False
#x,y = 60, 70 # False

# write here
y < x
```

[46]: True

</div>

[46]: d = 100

```
x,y = 50, 10 # True
#x,y = 100,0 # True
#x,y = 75, 70 # True
#x,y = 25,75 # False
#x,y = 25, 5 # True
#x,y = 5, 80 # False
#x,y = 60, 70 # False

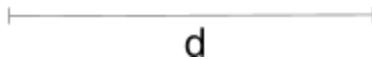
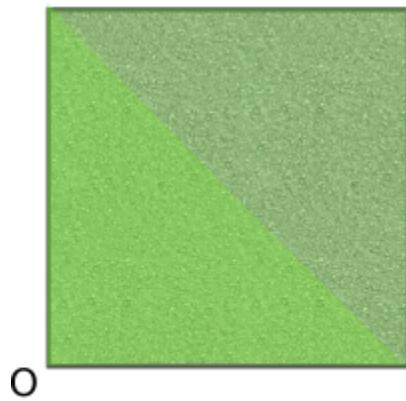
# write here
```

[46]: True

Exercise - The Lawnmower 4

Another day, another diagonal for Jobe..

- **DO NOT use if commands**
- **HINT:** if you don't remember the linear equation it's the time to look it up¹⁰⁷ :-)



Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[47]:

```
d = 100

x,y = 50, 10    # True
#x,y = 0,0      # True
#x,y = 75, 70   # False
#x,y = 25, 90   # False
#x,y = 25, 5    # True
#x,y = 80, 20   # False
#x,y = 25, 25   # True

# write here
y < -x + d
```

[47]:

```
True
```

[47]:

```
d = 100

x,y = 50, 10    # True
#x,y = 0,0      # True
#x,y = 75, 70   # False
#x,y = 25, 90   # False
#x,y = 25, 5    # True
#x,y = 80, 20   # False
#x,y = 25, 25   # True

# write here
```

(continues on next page)

¹⁰⁷ https://www.mathsisfun.com/equation_of_line.html

(continued from previous page)

[47] : True

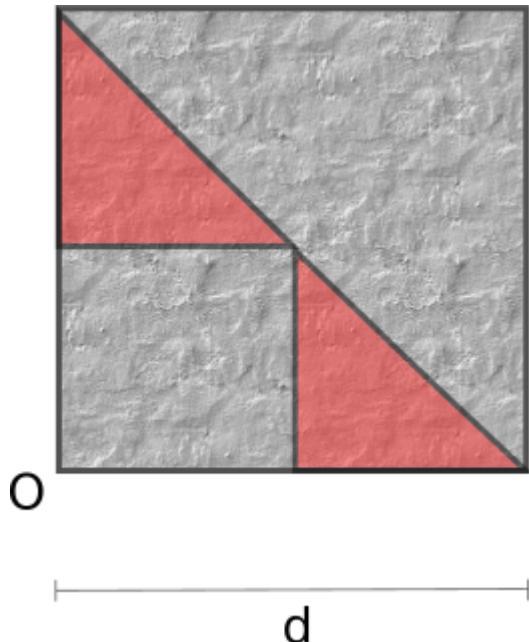
Exercise - The Lava Temple

During your studies you discover a map of an ancient temple, which hides marvelous treasures.

The temple measures $d=80$ meters each side, and is a labyrinth of corridors. You know for sure that some areas shown in red contain a fragile floor under which rivers of boiling lava are flowing: to warn about the danger while you're walking, you build a detector which will emit a sound whenever you enter red zones.

Write a boolean expression which gives back `True` if you are in a danger zone, and `False` otherwise.

- **DO NOT** use `if` instructions



Show solution

[48] :

```
d = 80

x,y = 0, 0      # False
#x,y = 20, 20  # False
#x,y = 60, 10  # True
#x,y = 10, 60  # True
#x,y = 20, 70  # False
#x,y = 70, 20  # False
#x,y = 70, 70  # False
#x,y = 0, 60  # True
#x,y = 60, 0   # True

# write here
```

(continues on next page)

(continued from previous page)

```
((x > d//2) or (y > d//2)) and (x < -y + d)
```

[48]: False

</div>

[48]:

```
d = 80

x,y = 0, 0      # False
#x,y = 20, 20  # False
#x,y = 60, 10  # True
#x,y = 10, 60  # True
#x,y = 20, 70  # False
#x,y = 70, 20  # False
#x,y = 70, 70  # False
#x,y = 0,60    # True
#x,y = 60,0    # True

# write here
```

[48]: False

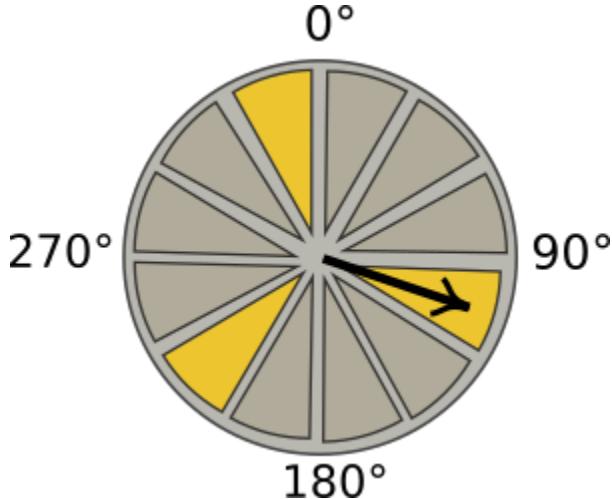
Exercise - The Tower of Gradius I

The hands of the clock on the first Gradius Tower has rotated so far of n degrees. Write some code which shows True if the hand is in the zones in evidence, False otherwise.

- DO NOT use `if` instructions
- n can be greater than 360

There two ways to solve the problem:

1. simple: write a long expressions with several `and`, `or` operators
2. harder: can you write a single short expression *without* `and` nor `or`?



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[49]: n = 20      # False
#n = 405      # False
#n = 70       # False
#n = 100      # True
#n = 460      # True
#n = 225      # True
#n = 182      # False
#n = 253      # False
#n = 275      # False
#n = 205      # False
#n = 350      # True
#n = 925      # False
#n = 815      # True
#n = 92       # True

# write here

# 1. LONG SOLUTION
# Looking at the figure, we see the zones are between 90° and 120°, 210° and 240°, ↵
# 330 and 360°

# A first problem to tackle is the fact the hand can have rotated more than 360°, ↵
# like 460° or 925° as in test.
# To solve this first problem, we can directly use the module operator like this
# to always get numbers between 0 and 359 included:

# m = n % 360

# This way we could simply solve the exercise with many and/or, like:

# (m >= 90 and m < 120) or (m >= 210 and m < 240) or (m >= 330 and m < 360)      #
# 'long' solution

# 2. SHORT SOLUTION

# As an alternative, consider this fact: if you take sequences of 4 segments of 30° ↵
# each,
# every sequence occupies 120° and we are interested to know when the hand is in the ↵
# last segment,
# between 90° and 120°. We can then creatively use the modulus operator to 'shorten' ↵
# the clock panel
# and ignore all the degrees after the 120th. Finally, we look whether or not m is ↵
# lying in the last segment:

# m = n % 120      # number between 0 and 119 included
# m > 3 * 30

n % 120 > 90      # 'short' solution

```

[49]: False

</div>

```
[49]: n = 20      # False
#n = 405      # False
```

(continues on next page)

(continued from previous page)

```
#n = 70    # False
#n = 100   # True
#n = 460   # True
#n = 225   # True
#n = 182   # False
#n = 253   # False
#n = 275   # False
#n = 205   # False
#n = 350   # True
#n = 925   # False
#n = 815   # True
#n = 92    # True

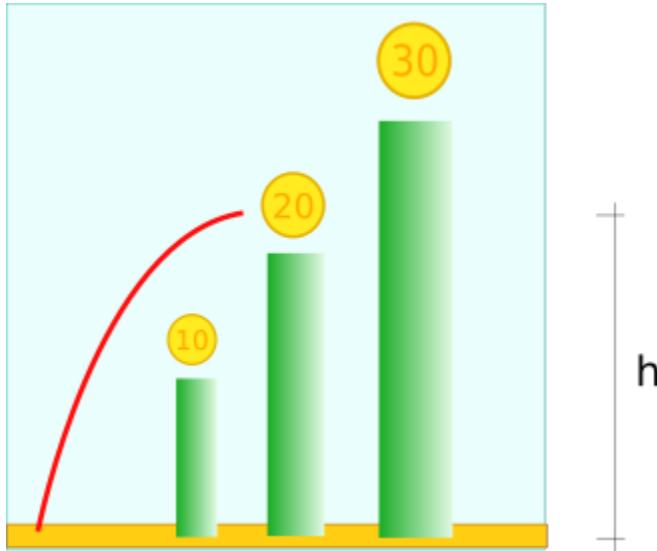
# write here
```

[49]: False

Exercise - The Pipe Jump

An Italian plumber is looking at 3 pipes of height t_1 , t_2 and t_3 , each having respectively 10, 20 and 30 coins above. Enthusiast, he makes a jump and reaches a height of h . Write some code which prints the number of coins taken (10, 20 or 30).

- **DO NOT use if instructions**
- **HINT:** If you don't know how to do it, check again the paragraph *Evaluation order* and try thinking how to produce numbers when only a certain condition is true ...



Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[50]:

```
t1,t2,t3 = 200,500,600
h=450    # 10
```

(continues on next page)

(continued from previous page)

```
#h=570 # 20
#h=610 # 30
#h=50 # 0

# write here

(h >= t3 and 30) or (h >= t2 and 20) or (h >= t1 and 10) or 0
```

[50]: 10

</div>

```
t1,t2,t3 = 200,500,600

h=450 # 10
#h=570 # 20
#h=610 # 30
#h=50 # 0

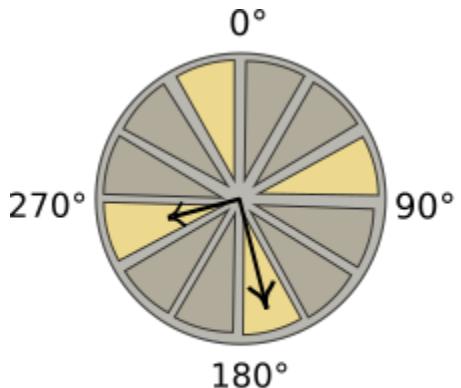
# write here
```

[50]: 10

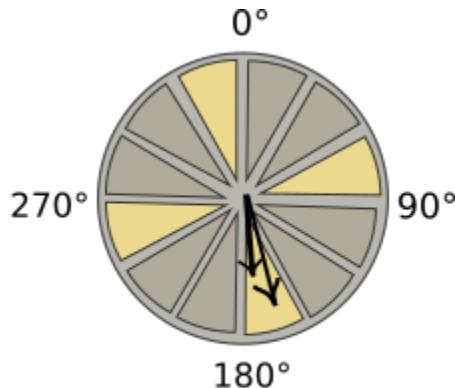
Exercise - The Tower of Gradius II

The hands of the clock on the second Gradius Tower have rotated so far of n and m degrees. Write some code which shows **True** if both hands are in the **same** zone among the highlighted ones, **False** otherwise.

- **DO NOT use `if` instructions**
- n and m can be greater than 360



False



True

Show solution
Hide

[51]:

n,m = 160,170 # True

(continues on next page)

(continued from previous page)

```
#n,m = 135, 140 # False
#n,m = 160,190 # False
#n,m = 70,170 # False
#n,m = 350,260 # False
#n,m = 350,340 # True
#n,m = 350,340 # True
#n,m = 430,530 # False
#n,m = 520,510 # True
#n,m = 730,740 # False

# write here
(n % 90 > 60) and ((n // 90) % 4 == (m // 90) % 4)
```

[51]: True

</div>

[51]:

```
n,m = 160,170 # True
#n,m = 135, 140 # False
#n,m = 160,190 # False
#n,m = 70,170 # False
#n,m = 350,260 # False
#n,m = 350,340 # True
#n,m = 350,340 # True
#n,m = 430,530 # False
#n,m = 520,510 # True
#n,m = 730,740 # False
```

write here

[51]: True

Continue

Go on with Basics 3 - float numbers¹⁰⁸

5.1.3 Python basics 3 - floats

[Download exercises zip](#)

Browse online files¹⁰⁹

PREREQUISITES:

- Having read basics 1 - integers¹¹⁰
- Having read basics 2 - booleans¹¹¹

¹⁰⁸ <https://en.softpython.org/basics/basics3-floats-sol.html>

¹⁰⁹ <https://github.com/DavidLeoni/softpython-en/tree/master/basics>

¹¹⁰ <https://en.softpython.org/basics/basics1-ints-sol.html>

¹¹¹ <https://en.softpython.org/basics/basics2-bools-sol.html>

Real numbers

Python saves the real numbers (floating point numbers) in 64 bit of information divided by sign, exponent and mantissa (also called significand). Let's see an example:

```
[2]: 3.14
```

```
[2]: 3.14
```

```
[3]: type(3.14)
```

```
[3]: float
```

WARNING: you must use the dot instead of comma!

So you will write `3.14` instead of `3,14`

Be very careful, whenever you copy numbers from documents in latin languages, they might contain very insidious commas!

Scientific notation

Whenever numbers are very big or very small, to avoid having to write too many zeros it is convenient to use scientific notation with the `e` like xen which multiplies the number x by 10^n

With this notation, in memory are only put the most significative digits (the *mantissa*) and the exponent, thus avoiding to waste space.

```
[4]: 75e1
```

```
[4]: 750.0
```

```
[5]: 75e2
```

```
[5]: 7500.0
```

```
[6]: 75e3
```

```
[6]: 75000.0
```

```
[7]: 75e123
```

```
[7]: 7.5e+124
```

```
[8]: 75e0
```

```
[8]: 75.0
```

```
[9]: 75e-1
```

```
[9]: 7.5
```

```
[10]: 75e-2
```

```
[10]: 0.75
```

```
[11]: 75e-123
```

```
[11]: 7.5e-122
```

QUESTION: Look at the following expressions, and try to find which result they produce (or if they give an error):

1. `print(1.000.000)`

2. `print(3,000,000.000)`

3. `print(2000000.000)`

4. `print(2000000.0)`

5. `print(0.000.123)`

6. `print(0.123)`

7. `print(0.-123)`

8. `print(3e0)`

9. `print(3.0e0)`

10. `print(7e-1)`

11. `print(3.0e2)`

12. `print(3.0e-2)`

13. `print(3.0-e2)`

14. `print(4e2-4e1)`

Too big or too small numbers

Sometimes calculations on very big or extra small numbers may give as a result `math.nan` (Not a Number) or `math.inf`. For the moment we just mention them, you can find a detailed description in the [Numpy page](#)¹¹²

¹¹² <https://en.softpython.org/matrices-numpy/matrices-numpy-sol.html#NaN-e-infinities>

Exercise - circle

⊕ Calculate the area of a circle at the center of a soccer ball (radius = 9.1m), remember that $area = pi * r^2$

Your code should print as result 263.02199094102605

Show solution<div class="jupman-sol" data-jupman-sol-code" style="display:none">

[12] : # SOLUTION

```
r = 9.15
pi = 3.1415926536
area = pi*(r**2)
print(area)
```

263.02199094102605

</div>

[12] :

263.02199094102605

Note that the parenthesis around the squared `r` are not necessary because the power operator has the precedence, but they may help in augmenting the code readability.

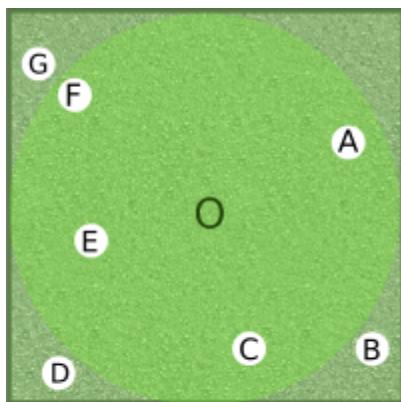
We recall here the operator precedence:

Operator	Description
<code>**</code>	Power (maximum precedence)
<code>+ -</code>	unary plus and minus
<code>* / // %</code>	Multiplication, division, integer division, modulo
<code>+ -</code>	Addition and subtraction
<code><= < > >=</code>	comparison operators
<code>== !=</code>	equality operators
<code>not or and</code>	Logical operators (minimum precedence)

Exercise - The golf club

Jobe the gardener is now so skilled that Dr Angelo decides to promote him to *paysagist* of the golf club where he is used to spend his weekends. But a new challenge awaits him: Jobe must perfectly mow the *green*, that is the circular area of radius `r` which encircles the hole. The LED on Jobe's lawnmower must light up when it's positioned on the greenlight zones: write an expression which given two coordinates `x, y` produces `True` in this case and `False` otherwise. With the only purpose to give some visual references, this time you can also see on the field some white balls marked with characters.

- **NOTE:** this time the origin is in **the square center**.
- **DO NOT** use the `if` command
- **HINT:** do you remember how to calculate the distance between two points? You will need the square root `math.sqrt...`



Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[13]: `import math`

```
r = 50

x,y = 35,20      # A: True
#x,y = 45,-40    # B: False
#x,y = 10,-40    # C: True
#x,y = -41,-46   # D: False
#x,y = -30,-10   # E: True
#x,y = -35,35    # F: True
#x,y = -37,37    # G: False

# write here
dist = math.sqrt(x**2 + y**2)
dist < r
```

[13]: `True`

</div>

[13]: `import math`

```
r = 50

x,y = 35,20      # A: True
#x,y = 45,-40    # B: False
#x,y = 10,-40    # C: True
#x,y = -41,-46   # D: False
#x,y = -30,-10   # E: True
#x,y = -35,35    # F: True
#x,y = -37,37    # G: False

# write here
```

[13]: `True`

Exercise - fractioning

⊕ Write some code to calculate the value of the following formula for $x = 0.000003$, you should obtain 2.753278226511882

$$-\frac{\sqrt{x+3}}{\frac{(x+2)^3}{\log x}}$$

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[14]: `x = 0.000003`

```
# write here
import math
- math.sqrt(x+3) / ((x+2)**3)/math.log(x)
```

[14]: `2.753278226511882`

</div>

[14]: `x = 0.000003`

```
# write here
```

[14]: `2.753278226511882`

Exercise - summation

Write some code to calculate the value of the following expression (don't use cycles, write down all calculations), you should obtain 20.53333333333333

$$\sum_{j=1}^3 \frac{j^4}{j+2}$$

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[15]: `# write here
((1**4) / (1+2)) + ((2**4) / (2+2)) + ((3**4) / (3+2))`

[15]: `20.53333333333333`

</div>

[15]: `# write here`

[15]: `20.53333333333333`

Reals - conversion

If we want to convert a real to an integer, several ways are available:

Function	Description	Mathematical symbol	Result
<code>math.floor(x)</code>	round x to inferior integer	$\lfloor 8.7 \rfloor$	8
<code>int(x)</code>	round x to inferior integer	$\lfloor 8.7 \rfloor$	8
<code>math.ceil(x)</code>	round x to superior integer	$\lceil 5.3 \rceil$	6
<code>round(x)</code>	round x to closest integer	$\lfloor 2.49 \rfloor$	2
		$\lfloor 2.51 \rfloor$	3

QUESTION: Look at the following expressions, and for each of them try to guess which result it produces (or if it gives an error).

1. `math.floor(2.3)`

2. `math.floor(-2.3)`

3. `round(3.49)`

4. `round(3.51)`

5. `round(-3.49)`

6. `round(-3.51)`

7. `math.ceil(8.1)`

8. `math.ceil(-8.1)`

QUESTION: Given a float x , the following formula is:

```
math.floor(math.ceil(x)) == math.ceil(math.floor(x))
```

1. always True
2. always False
3. sometimes True and sometimes False (give examples)

Show answer

>

ANSWER: 3: for integers like $x=2.0$ it is True, in other cases like $x=2.3$ it is False

</div>

QUESTION: Given a float x , the following formula is:

```
math.floor(x) == -math.ceil(-x)
```

1. always True
2. always False
3. sometimes True and sometimes False (give examples)

Show answer

>

ANSWER: 1.

</div>

Exercise - Invigorate

⊕ Excessive studies lead you search on internet recipes of energetic drinks. Luckily, a guru of nutrition just posted on her Instagram channel @HealthyDrink this recipe of a miracle drink:

Pour in a mixer 2 decilitres of kiwi juice, 4 decilitres of soy sauce, and 3 decilitres of shampoo of karité bio.
Mix vigorously and then pour half drink into a glass. Fill the glass until the superior deciliter. Swallow in one shot.

You run shopping the ingredients, and get ready for mixing them. You have a measuring cup with which you transfer the precious fluids, one by one. While transferring, you always pour a little bit more than necessary (but never more than 1 decilitre), and for each ingredient you then remove the excess.

- **DO NOT** use subtractions, try using only rounding operators

Example - given:

```
kiwi = 2.4
soia = 4.8
shampoo = 3.1
measuring_cup = 0.0
mixer = 0
glass = 0.0
```

Your code must print:

```
I pour into the measuring cup 2.4 dl of kiwi juice, then I remove excess until ↵keeping 2 dl
I transfer into the mixer, now it contains 2.0 dl
I pour into the measuring cup 4.8 dl of soia, then I remove excess until keeping 4 dl
I transfer into the mixer, now it contains 6.0 dl
I pour into the measuring cup 3.1 dl of shampoo, then I remove excess until keeping 3 ↵dl
I transfer into the mixer, now it contains 9.0 dl
I pour half of the mix ( 4.5 dl ) into the glass
I fill the glass until superior deciliter, now it contains: 5 dl
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[16]: import math

```
kiwi = 2.4
soy = 4.8
shampoo = 3.1
measuring_cup = 0.0
mixer = 0.0
glass = 0.0

# write here
print('I pour into the measuring cup', kiwi, 'dl of kiwi juice, then I remove excess' ↵
      'until keeping', int(kiwi), 'dl')
mixer += int(kiwi)
print('I transfer into the mixer, now it contains', mixer, 'dl')
print('I pour into the measuring cup', soy, 'dl of soia, then I remove excess until' ↵
      'keeping', int(soy), 'dl')
mixer += int(soy)
print('I transfer into the mixer, now it contains', mixer, 'dl')
print('I pour into the measuring cup', shampoo, 'dl of shampoo, then I remove excess' ↵
      'until keeping', int(shampoo), 'dl')
mixer += int(shampoo)
print('I transfer into the mixer, now it contains', mixer, 'dl')
glass = mixer/2
print('I pour half of the mix (', glass, 'dl ) into the glass')
print('I fill the glass until superior deciliter, now it contains:', math.ceil(glass), ↵
      'dl')
```

```
I pour into the measuring cup 2.4 dl of kiwi juice, then I remove excess until ↵
      keeping 2 dl
I transfer into the mixer, now it contains 2.0 dl
I pour into the measuring cup 4.8 dl of soia, then I remove excess until keeping 4 dl
I transfer into the mixer, now it contains 6.0 dl
I pour into the measuring cup 3.1 dl of shampoo, then I remove excess until keeping 3 ↵
      dl
I transfer into the mixer, now it contains 9.0 dl
I pour half of the mix ( 4.5 dl ) into the glass
I fill the glass until superior deciliter, now it contains: 5 dl
```

</div>

[16]: import math

(continues on next page)

(continued from previous page)

```

kiwi = 2.4
soy = 4.8
shampoo = 3.1
measuring_cup = 0.0
mixer = 0.0
glass = 0.0

```

write here

```

I pour into the measuring cup 2.4 dl of kiwi juice, then I remove excess until ↵
↪keeping 2 dl
I transfer into the mixer, now it contains 2.0 dl
I pour into the measuring cup 4.8 dl of soia, then I remove excess until keeping 4 dl
I transfer into the mixer, now it contains 6.0 dl
I pour into the measuring cup 3.1 dl of shampoo, then I remove excess until keeping 3 ↵
↪dl
I transfer into the mixer, now it contains 9.0 dl
I pour half of the mix ( 4.5 dl ) into the glass
I fill the glass until superior deciliter, now it contains: 5 dl

```

Exercise - roundminder

⊕ Write some code to calculate the value of the following formula for $x = -5.51$, you should obtain 41

$$|\lceil x \rceil| + \lfloor x \rfloor^2$$

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[17]: import math

```

x = -5.51    # 41
#x = -5.49   # 30

# write here
abs(math.ceil(x)) + round(x)**2

```

[17]: 41

</div>

[17]: import math

```

x = -5.51    # 41
#x = -5.49   # 30

# write here

```

[17]: 41

Reals - equality

WARNING: what follows is valid for *all* programming languages!

Some results will look weird but this is the way most processors (CPU) operates, independently from Python.

When floating point calculations are performed, the processor may introduce rounding errors due to limits of internal representation. Under the hood the numbers like floats are memorized in a sequence of binary code of 64 bits, according to *IEEE-754 floating point arithmetic* standard: this imposes a physical limit to the precision of numbers, and sometimes we get surprises due to conversion from decimal to binary. For example, let's try printing 4.1:

```
[18]: print(4.1)
```

```
4.1
```

For our convenience Python is showing us 4.1, but in reality a different number ended up in the processor memory! Which one? To discover what it hides, with `format` function we can explicitly format the number to, for example 55 digits of precision by using the `f` format specifier:

```
[19]: format(4.1, '.55f')
```

```
'4.09999999999996447286321199499070644378662109375000000'
```

We can then wonder what the result of this calculus might be:

```
[20]: print(7.9 - 3.8)
```

```
4.1000000000000005
```

We note the result is still different from the expected one! By investigating further, we notice Python is not even showing all the digits:

```
[21]: format(7.9 - 3.8, '.55f')
```

```
'4.1000000000000005329070518200751394033432006835937500000'
```

```
[ ]:
```

What if wanted to know if the two calculations with float produce the ‘same’ result?

WARNING: AVOID == WITH FLOATS!

To understand if the result between the two calculations with the floats is the same, **YOU CANNOT** use the `==` operator !

```
[22]: 7.9 - 3.8 == 4.1      # TROUBLE AHEAD!
```

```
[22]: False
```

Instead, you should prefer alternative that evaluate if a float number is *close* to another, like for example the handy function `math.isclose`¹¹³:

¹¹³ <https://docs.python.org/3/library/math.html#math.isclose>

```
[23]: import math
math.isclose(7.9 - 3.8, 4.1)    # MUCH BETTER
[23]: True
```

By default `math.isclose` uses a precision of $1e-9$, but, if needed, you can also pass a tolerance limit in which the difference of the numbers must be so to be considered equal:

```
[24]: math.isclose(7.9 - 3.8, 4.1, abs_tol=0.000001)
[24]: True
```

QUESTION: Can we perfectly represent the number $\sqrt{2}$ as a `float`?

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: $\sqrt{2}$ is irrational so there's no hope of a perfect representation, any calculation will always have a certain degree of imprecision.

</div>

QUESTION: Which of these expressions give the same result?

```
import math
print('a)', math.sqrt(3)**2 == 3.0)
print('b)', abs(math.sqrt(3)**2 - 3.0) < 0.0000001)
print('c)', math.isclose(math.sqrt(3)**2, 3.0, abs_tol=0.0000001))
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: b) and c) give `True`. a) gives `False`, because during floating point calculations rounding errors are made.

</div>

Exercise - quadratic

⊕ Write some code to calculate the zeroes of the equation $ax^2 - b = 0$

- Show numbers with **20 digits** of precision
- At the end check that by substituting the value obtained `x` into the equation you actually obtain zero.

Example - given:

```
a = 11.0
b = 3.3
```

after your code it must print:

```
11.0 * x**2 - 3.3 = 0 per x1 = 0.54772255750516607442
11.0 * x**2 - 3.3 = 0 per x2 = -0.54772255750516607442
Is 0.54772255750516607442 a solution? True
Is -0.54772255750516607442 a solution? True
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[25]:

```
a = 11.0
b = 3.3

# write here

import math

x1 = math.sqrt(b/a)
x2 = -math.sqrt(b/a)

print(a, "* x**2 -", b, "= 0 per x1 =", format(x1, '.20f'))
print(a, "* x**2 -", b, "= 0 per x2 =", format(x2, '.20f'))

# we need to change the default tolerance value
print("Is", format(x1, '.20f'), "a solution?", math.isclose(a*(x1**2) - b, 0, abs_
    tol=0.00001))
print("Is", format(x2, '.20f'), "a solution?", math.isclose(a*((x2)**2) - b, 0, abs_
    tol=0.00001))

11.0 * x**2 - 3.3 = 0 per x1 = 0.54772255750516607442
11.0 * x**2 - 3.3 = 0 per x2 = -0.54772255750516607442
Is 0.54772255750516607442 a solution? True
Is -0.54772255750516607442 a solution? True
```

</div>

[25]:

```
a = 11.0
b = 3.3

# write here
```

Exercise - trendy

⊗⊗ You are already thinking about next vacations, but there is a big problem: where do you go, if you don't have a *selfie-stick*? You cannot leave with this serious anxiety: to uniform yourself to this mass phenomena you must buy the stick which is most similar to others. You then conduct a rigourous statistical survey among turists obsessed by selfie sticks with the goal to find the most frequent brands of sticks, in other words, the *mode* of the frequencies. You obtain these results:

[26]:

```
b1,b2,b3,b4,b5 = 'TooManyLikes', 'Boombasticks', 'Timewasters Inc', 'Vanity 3.0',
    ↪'TrashTrend' # brand
f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.3 # frequencies (as percentages)
```

We deduce that masses love selfie-sticks of the brand 'Boombasticks' and TrashTrend, both in a tie with 30% turists each. Write some code which prints this result:

```
TooManyLikes is the most frequent? False ( 25.0 % )
Boombasticks is the most frequent? True ( 30.0 % )
Timewasters Inc is the most frequent? False ( 10.0 % )
Vanity 3.0 is the most frequent? False ( 5.0 % )
TrashTrend is the most frequent? True ( 30.0 % )
```

- **WARNING:** your code must work with **ANY** series of variables !!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[27]:

```
b1,b2,b3,b4,b5 = 'TooManyLikes', 'Boombasticks', 'Timewasters Inc', 'Vanity 3.0',
↪'TrashTrend'      # brand

f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.3 # frequencies (as percentages) False
↪True False False True
# CAREFUL, they look the same but it must work also with these!
#f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.1 + 0.2 # False True False False True

# write here
mx = max(f1,f2,f3,f4,f5)
print(b1, 'is the most frequent?', math.isclose(f1,mx), '(', format(f1*100.0, '.1f'), '
↪% )')
print(b2, 'is the most frequent?', math.isclose(f2,mx), '(', format(f2*100.0, '.1f'), '
↪% )')
print(b3, 'is the most frequent?', math.isclose(f3,mx), '(', format(f3*100.0, '.1f'), '
↪% )')
print(b4, 'is the most frequent?', math.isclose(f4,mx), '(', format(f4*100.0, '.1f'), '
↪% )')
print(b5, 'is the most frequent?', math.isclose(f5,mx), '(', format(f5*100.0, '.1f'), '
↪% )')
```

```
TooManyLikes is the most frequent? False ( 25.0 % )
Boombasticks is the most frequent? True ( 30.0 % )
Timewasters Inc is the most frequent? False ( 10.0 % )
Vanity 3.0 is the most frequent? False ( 5.0 % )
TrashTrend is the most frequent? True ( 30.0 % )
```

</div>

[27]:

```
b1,b2,b3,b4,b5 = 'TooManyLikes', 'Boombasticks', 'Timewasters Inc', 'Vanity 3.0',
↪'TrashTrend'      # brand

f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.3 # frequencies (as percentages) False
↪True False False True
# CAREFUL, they look the same but it must work also with these!
#f1,f2,f3,f4,f5 = 0.25, 0.3, 0.1, 0.05, 0.1 + 0.2 # False True False False True

# write here
```

Decimal numbers

For most applications float numbers are sufficient, if you are conscious of their limits of representation and equality. If you really need more precision and/or predictability, Python offers a dedicated numeric type called `Decimal`, which allows arbitrary precision. To use it, you must first import `decimal` library:

```
[28]: from decimal import Decimal
```

You can create a Decimal from a string:

```
[29]: Decimal('4.1')
```

```
[29]: Decimal('4.1')
```

WARNING: if you create a Decimal from a constant, use a string!

If you pass a `float` you risk losing the utility of Decimals:

```
[30]: Decimal(4.1) # this way I keep the problems of floats ...
```

```
[30]: Decimal('4.0999999999999996447286321199499070644378662109375')
```

Operations between Decimals produce other Decimals:

```
[31]: Decimal('7.9') - Decimal('3.8')
```

```
[31]: Decimal('4.1')
```

This time, we can freely use the equality operator and obtain the same result:

```
[32]: Decimal('4.1') == Decimal('7.9') - Decimal('3.8')
```

[32]: True

Some mathematical functions are also supported, and often they behave more predictably (note we are **not** using `math.sqrt`):

```
[33]: Decimal('2').sqrt()
```

```
[33]: Decimal('1.414213562373095048801688724')
```

Remember: computer memory is still finite!

Decimals can't be solved all problems in the universe: for example, $\sqrt{2}$ will never fit the memory of any computer! We can verify the limitations by squaring it:

```
[34]: Decimal('2').sqrt()**Decimal('2')
[34]: Decimal('1.999999999999999999999999999999')
```

The only thing we can have more with Decimals is more digits to represent numbers, which if we want we can increase at will¹¹⁴ until we fill our pc memory. In this book we won't talk anymore about Decimals because typically they are meant only for specific applications, for example, if you need to perform financial calculations you will probably want very exact digits!

¹¹⁴ <https://docs.python.org/3/library/decimal.html>

Continue

Go on with [the challenges](#)¹¹⁵

5.1.4 Python basics 4 - Challenges

Download exercises zip

Browse online files¹¹⁶

We now propose some exercises without solutions.

Try executing them both in Jupyter and a text editor such as Spyder or Visual Studio Code to get familiar with both environments.

Challenge - which booleans 1?

⊕ Find the row with values such that the final print prints True. Is there only one combination or many?

```
[2]: x = False; y = False
#x = False; y = True
#x = True; y = False
#x = True; y = True

print(x and y)

False
```

Challenge - which booleans 2?

⊕ Find the row in which by assigning values to x and y it prints True. Is there only one combinatin or many?

```
[3]: x = False; y = False; z = False
#x = False; y = True; z = False
#x = True; y = False; z = False
#x = True; y = True; z = False
#x = False; y = False; z = True
#x = False; y = True; z = True
#x = True; y = False; z = True
#x = True; y = True; z = True

print((x or y) and (not x and z))

False
```

¹¹⁵ <https://en.softpython.org/basics/basics4-chal.html>

¹¹⁶ <https://github.com/DavidLeoni/softpython-en/tree/master/basics>

Challenge - airport

⊕⊕ You finally decide to take a vacation and go to the airport, expecting to spend some time in several queues. Luckily, you only have carry-on bag, so you directly go to security checks, where you can choose among three rows of people sec1, sec2, sec3. Each person an average takes 4 *minutes* to be examined, you included, and obviously you choose the shortest queue. Afterwards you go to the gate, where you find two queues of ga1 and ga2 people, and you know that each person you included an average takes 20 *seconds* to pass: again you choose the shortest queue. Luckily the aircraft is next to the gate so you can directly choose whether to board at the queue at the head of the aircraft with bo1 people or at the queue at the tail of the plane with bo2 people. Each passenger you included takes an average 30 *seconds*, and you choose the shortest queue.

Write some code to calculate how much time you take in total to enter the plane, showing it in minutes and seconds.

Example - given:

```
sec1,sec2,sec3, ga1,ga2, bo1,bo2 = 4,5,8, 5,2, 7,6
```

your code must print:

```
24 minutes and 30 seconds
```

[4]:

```
sec1,sec2,sec3, ga1,ga2, bo1,bo2 = 4,5,8, 5,2, 7,6    # 24 minutes and 30 seconds
#sec1,sec2,sec3, ga1,ga2, bo1,bo2 = 9,7,1, 3,5, 2,9    # 10 minutes and 50 seconds

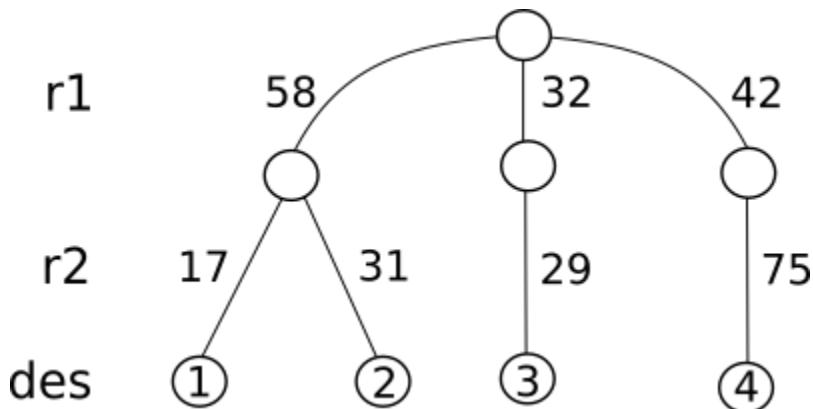
# write here
```

Challenge - Holiday trip

⊕⊕ While an holiday you are traveling by car, and in a particular day you want to visit one among 4 destinations. Each location requires to go through two roads r1 and r2. Roads are numbered with two digits numbers, for example to reach destination 1 you need to go to road 58 and road 17.

Write some code that given r1 and r2 roads shows the number of the destination.

- If the car goes to a road it shouldn't (i.e. road 666), put False in destination
- **DO NOT** use summations
- **IMPORTANT: DO NOT use if commands** (it's possible, think about it ;-)



Example 1 - given:

```
r1,r2 = 58,31
```

After your code it must print:

```
The destination is 2
```

Example 2 - given:

```
r1,r2 = 666,31
```

After your code it must print:

```
The destination is False
```

[5]:

```
r1,r2 = 58,17    # 1
r1,r2 = 58,31    # 2
r1,r2 = 32,29    # 3
r1,r2 = 42,75    # 4
r1,r2 = 666,31   # False
r1,r2 = 58,666   # False
r1,r2 = 32,999   # False

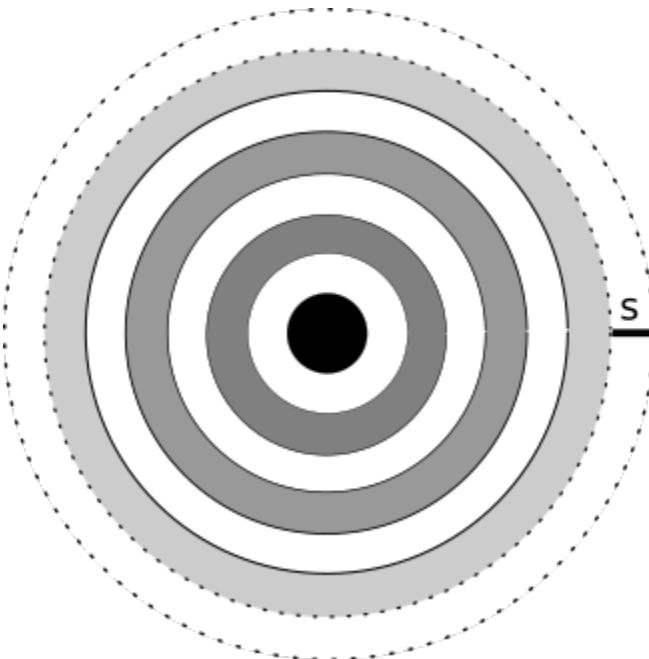
# write here
```

Challenge - The Tunnel of Time

Recently a spatio-temporal tunnel has been discovered which allows time travelling. To repair the tragic errors made in the past, humanity is struggling to send probes through the tunnel. Alas, the tunnel is perturbed by very strong magnetic-gravitational fields which might have sucked the probe into the folds of time (represented in shades of grey).

Write some code which given the `px, py` position of the probe, prints `True` if the probe went into a grey zone, and `False` if it went into a white zone.

- Origin 0,0 is at the center
- Each circle edge has $s=5$ distance from its inner circle
- edges are supposed to be infinitesimal and we assume the probe will never go exactly there - in such cases, your program behaviour is allowed to be undefined
- **DO NOT** use `if` instruction nor cycles
- **NOTE** the time tunnel is potentially infinite, so your code must also work for very big values of `px, py`



[6]:

```
s=5

px,py = 0,0      # True
#px,py = 2,0      # True
#px,py = 0,7      # False
#px,py = 3,3      # True
#px,py = 4,4      # False
#px,py = 6,7      # False
#px,py = 7,8      # True
#px,py = 15,9     # False
#px,py = 230,120   # False
#px,py = 1230,536  # True

# write here
```

5.2 Strings

5.2.1 Strings 1 - introduction

[Download exercises zip](#)

[Browse files online](#)¹¹⁷

Strings are *immutable* character sequences, and one of the basic Python types. In this notebook we will see how to manipulate them.

¹¹⁷ <https://github.com/DavidLeoni/softpython-en/tree/master/strings>

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
strings
  strings1.ipynb
  strings1-sol.ipynb
  strings2.ipynb
  strings2-sol.ipynb
  strings3.ipynb
  strings3-sol.ipynb
  strings4.ipynb
  strings4-sol.ipynb
  strings5-chal.ipynb
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `strings1.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

Creating strings

There are several ways to define a string.

Double quotes, in one line

```
[2]: a = "my first string, in double quotes"
```

```
[3]: print(a)
my first string, in double quotes
```

Single quotes, in one line

This way is equivalent to previous one.

```
[4]: b = 'my second string, in single quotes'
```

```
[5]: print(b)
my second string, in single quotes
```

Between double quotes, on many lines

```
[6]: c = """my third string  
in triple double quotes  
so I can put it  
  
on many rows"""
```

```
[7]: print(c)  
  
my third string  
in triple double quotes  
so I can put it  
  
on many rows
```

Three single quotes, many lines

```
[8]: d = '''my fourth string,  
in triple single quotes  
also can be put  
  
on many lines  
'''
```

```
[9]: print(d)  
  
my fourth string,  
in triple single quotes  
also can be put  
  
on many lines
```

Printing - the cells

To print a string we can use the function `print`:

```
[10]: print('hello')  
hello
```

Note that apices are *not* reported in printed output.

If we write the string without the `print`, we will see the apices indeed:

```
[11]: 'hello'  
[11]: 'hello'
```

What happens if we write the string with double quotes?

```
[12]: "hello"  
[12]: 'hello'
```

Notice that by default Jupyter shows single apices.

The same applies if we assign a string to a variable:

```
[13]: x = 'hello'
```

```
[14]: print(x)  
hello
```

```
[15]: x
```

```
[15]: 'hello'
```

```
[16]: y = "hello"
```

```
[17]: print(y)  
hello
```

```
[18]: y
```

```
[18]: 'hello'
```

The empty string

The string of zero length is represented with two double quotes "" or two single apices ''

Note that even if we write two double quotes, Jupyter shows a string beginning and ending with single apices:

```
[19]: ""
```

```
[19]: ''
```

The same applies if we associate an empty string to a variable:

```
[20]: x = ""
```

```
[21]: x
```

```
[21]: ''
```

Note that even if we ask Jupyter to use `print`, we won't see anything:

```
[22]: print("")
```

```
[23]: print('')
```

Printing many strings

For printing many strings on a single line there are different ways, let's start from the most simple with `print`:

```
[24]: x = "hello"  
y = "Python"  
  
print(x,y)    # note that in the printed characters Python inserted a space:  
hello Python
```

We can add to `print` as many parameters we want, which can also be mixed with other types like numbers:

```
[25]: x = "hello"  
y = "Python"  
z = 3  
  
print(x,y,z)  
hello Python 3
```

Length of a string

To obtain the length of a string (or any sequence in general), we can use the function `len`:

```
[26]: len("ciao")
```

```
[26]: 4
```

```
[27]: len("")    # empty string
```

```
[27]: 0
```

```
[28]: len('')    # empty string
```

```
[28]: 0
```

QUESTION: Can we write something like this?

```
"len"("hello")
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: no, "`len`" between quotes will be interpreted as a string, not as a function, so Python will complain telling us we cannot apply a string to another string. Try to see which error appears by rewriting the expression below:

```
</div>
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[29]: # write here  
  
#"len"("hello")
```

```
</div>
```

[29]: # write here

QUESTION: can we write something like this? What does it produce? an error? a number? which one?

```
len("len('hello')")
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: it returns the number 12: by putting the Python code `len('hello')` among double quotes, it became a string like any other. So by writing `len("len('hello')")` we count how long the string `"len('hello')"` is.

</div>

QUESTION: What do we obtain if we write like this?

```
len((((("ciao")))))
```

1. an error
2. the length of the string
3. something else

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: The second: "ciao" is an expression, as such we can enclose it in as many parenthesis as we want.

</div>

Counting escape sequences: Note that some particular sequences called *escape sequences* like for example `\t` occupy less space of what it seems (with `len` they count as 1), but if we print them they will occupy even more than 2 !!

Let's see an example (in the next paragraph we will delve into the details):

[30]: `len('a\tb')`

[30]: 3

[31]: `print('a\tb')`

a b

Printing - escape sequences

Some characters sequences called *escape sequences* are special because instead of showing characters, they force the printing to do particular things like line feed or inserting extra spaces. These sequences are always preceded by the *backslash* character \:

Description	Escape sequence
Linefeed	\n
Tabulation (<i>ASCII tab</i>)	\t

Example - line feed

```
[32]: print("hello\nworld")  
hello  
world
```

Note the line feed happens only when we use `print`, if instead we directly put the string into the cell we will see it verbatim:

```
[33]: "ciao\nmondo"  
[33]: 'ciao\nmondo'
```

In a string you can put as many escape sequences as you like:

```
[34]: print("Today is\na great day\nisn't it?")  
Today is  
a great day  
isn't it?
```

Example - tabulation

```
[35]: print("hello\tworld")  
hello    world
```

```
[36]: print("hello\tworld\twith\tmany\ttabs")  
hello    world    with    many    tabs
```

EXERCISE: Since *escape sequences* are special, we might ask ourselves how long they are. Use the function `len` to print the string length. Do you notice anything strange?

- 'ab\ncd'
- 'ab\tcd'

```
[37]: # write the code here
```

EXERCISE: Try selecting the character sequence printed in the previous cell with the mouse. What do you obtain? A space sequence, or a single tabulation character? Note this can vary according to the program that actually printed the string.

EXERCISE: find a SINGLE string which printed with `print` is shown as follows:

```
This    is  
an  
  
apparently    simple        challenge
```

- **USE ONLY** combinations of `\t` and `\n`
- **DON'T** use spaces
- start and end the string with a single apex

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[38]: # write here
print('This\tis\nan\nnapparently\tsimple\t\tchallenge')
This    is
an
apparently      simple          challenge
</div>
```

```
[38]: # write here
This    is
an
apparently      simple          challenge
```

EXERCISE: try to find a string which printed with print is shown as follows:

```
At  te
n
    t  ion
please!
```

- **USE ONLY** combinations of \t and \n
- **DON'T** use any space
- **DON'T** use triple quotes

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[39]: # write here
print("At\tte\nn\n\tt\tion\n\tplease!")
At      te
n
    t      ion
please!
</div>
```

```
[39]: # write here
At      te
n
    t      ion
please!
```

Special characters: if we want special characters like the single apex ' or double quotes " inside a string, we must create a so-called *escape sequence*, that is, we must first write the *backslash* character \ and then follow it with the special character we're interested in:

Description	Escape sequence	Printed result
Single apex	\ '	'
Double quote	\ "	"
Backslash	\ \	\

Example

Let's print a string containing a single apex ' and a double quote ":

```
[40]: my_string = "This way I put \'apices\' e \"double quotes\" in strings"
[41]: print(my_string)
This way I put 'apices' e "double quotes" in strings
```

If a string begins with double quotes, inside we can freely use single apices, even without *backslash* \:

```
[42]: print("There's no problem")
There's no problem
```

If the string begins with single apices, we can freely use double quotes even without the *backslash* \:

```
[43]: print('It Is So "If You Think So"')
It Is So "If You Think So"
```

EXERCISE: Find a string to print with `print` which shows the following sequence:

- the string MUST start and finish with single apices '

```
This "genius" of strings wants to /\\"/ trick me \/// with atrocious exercises O_o'
```

 data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[44]: # write here
print('This "genius" of strings wants to /\\"/ trick me \/// with atrocious_
      ↪exercises O_o\'')
This "genius" of strings wants to /\\"/ trick me \/// with atrocious exercises O_o'
```

```
</div>
[44]: # write here
This "genius" of strings wants to /\\"/ trick me \/// with atrocious exercises O_o'
```

Encodings

ASCII characters

When using strings in your daily programs you typically don't need to care much how characters are physically represented as bits in memory, but sometimes it does matter. The representation is called *encoding* and must be taken into account in particular when you read stuff from external sources such as files and websites.

The most famous and used character encoding is [ASCII¹¹⁸](#) (American Standard Code for Information Interchange), which offers 127 slots made by basic printable characters from English alphabet (a-z, A-Z, punctuation like . ; , ! and characters like (, @ ...) and control sequences (like \t, \n)

- See [Printable characters¹¹⁹](#) (Wikipedia)
- [ASCII Control codes¹²⁰](#) (Wikipedia)

Since original ASCII table lacks support for non-English languages (for example, it lacks Italian accented letters like è, à, ...), many extensions were made to support other languages, for examples see [Extended ASCII¹²¹](#) page on Wikipedia.

Unicode characters

Whenever we need particular characters like ☺ which are not available on the keyboard, we can look at Unicode characters. There are a lot¹²², and we can often use them in Python 3 by simple copy-pasting. For example, if you go to [this page¹²³](#) you can copy-paste the character ☺. In other cases it might be so special it can't even be correctly visualized, so in these cases you can use a more complex sequence in the format \uxxxx like this:

Description	Escape sequence	Printed result
Example star in a circle in format \uxxxx	\u272A	☺

EXERCISE: Search Google for *Unicode heart* and try printing a heart in Python, both by directly copy-pasting the character and by using the notation \uxxxx

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[45]: # write here

```
print("I ❤ Python, with copy-paste")
print("I \u2665 Python, also in format \\uxxxx")
```

```
I ❤ Python, with copy-paste
I ❤ Python, also in format \uxxxx
```

</div>

[45]: # write here

¹¹⁸ <https://en.wikipedia.org/wiki/ASCII>

¹¹⁹ https://en.wikipedia.org/wiki/ASCII#Printable_characters

¹²⁰ https://en.wikipedia.org/wiki/C0_and_C1_control_codes#Basic_ASCII_control_codes

¹²¹ https://en.wikipedia.org/wiki/Extended_ASCII

¹²² <http://www.fileformat.info/info/unicode/char/a.htm>

¹²³ <https://www.fileformat.info/info/unicode/char/272a/index.htm>

```
I ❤ Python, with copy-paste  
I ❤ Python, also in format \uxxxx
```

Unicode references: Unicode can be a complex topic we just mentioned, if you ever need to deal with complex character sets like Japanese or heterogenous text encodings here a couple of references you should read:

- first part on Unicode encoding from Strings chapter from book Dive into Python 3¹²⁴
- Python 3 Unicode¹²⁵ documentation

Strings are immutable

Strings are *immutable* objects, so once they are created you cannot change them anymore. This might appear restrictive, but it's not so tragic, because we still have available these alternatives:

- generate a new string composed from other strings
- if we have a variable to which we assigned a string, we can assign another string to that variable

Let's generate a new string starting from previous ones, for example by joining two of them with the operator +

```
[46]: x = 'hello'
```

```
[47]: y = x + 'world'
```

```
[48]: x
```

```
[48]: 'hello'
```

```
[49]: y
```

```
[49]: 'helloworld'
```

The + operation, when executed among strings, it joins them by creating a NEW string. This means that the association to x it didn't change at all, the only modification we can observe will be the variable y which is now associated to the string 'helloworld'. Try making sure of this in Python Tutor by repeatedly clicking on *Next* button:

```
[50]: # WARNING: before using the function jupman.pytut() which follows,  
# it is necessary to first execute this cell with Shift+Enter  
  
# it's sufficient to execute it only once, you find it also in all other notebooks in  
# the first cell  
  
import jupman
```

```
[51]: x = 'hello'  
y = x + 'world'  
  
print(x)  
print(y)  
  
jupman.pytut()  
  
hello  
helloworld
```

¹²⁴ <https://diveintopython3.net/strings.html>

¹²⁵ <https://docs.python.org/3/howto/unicode.html>

```
[51]: <IPython.core.display.HTML object>
```

Reassign variables

Other variations to memory state can be obtained by reassigning the variables, for example:

```
[52]: x = 'hello'
```

```
[53]: y = 'world'
```

```
[54]: x = y      # we assign to x the same string contained in y
```

```
[55]: x
```

```
[55]: 'world'
```

```
[56]: y
```

```
[56]: 'world'
```

If a string is created and at some point no variables point to it, Python automatically takes care to eliminate it from the memory. In the case above, the string `hello` is never actually changed: at some point no variable is associated with it anymore and so Python eliminates the string from the memory. Have a look at what happens in Python Tutor:

```
[57]: x = 'hello'
```

```
y = 'world'
```

```
x = y
```

```
jupman.pytut()
```

```
[57]: <IPython.core.display.HTML object>
```

Reassign a variable to itself

We may ask ourselves what happens when we write something like this:

```
[58]: x = 'hello'
```

```
x = x
```

```
[59]: print(x)
```

```
hello
```

No big changes, the assignment of `x` remained the same without alterations.

But what happens if to the right of the `=` we put a more complex formula?

```
[60]: x = 'hello'
```

```
x = x + 'world'
```

```
print(x)
```

```
helloworld
```

Let's try to carefully understand what happened.

In the first line, Python generated the string 'hello' and assigned it to the variable x. So far, nothing extraordinary.

Then, in the second line, Python did two things:

1. it calculated the result of the expression x + 'world', by generating a NEW string helloworld
2. it assigned the generated string helloworld to the variable x

It is fundamental to understand that whenever a reassignment is performed both passages occurs, so it's worth repeating them:

- FIRST the result of the expression to the right of = is calculated (so when the old value of x is still available)
- THEN the result is associated to the variable to the left of = symbol

If we check out what happens in Python Tutor, this double passage is executed in a single shot:

```
[61]: x = 'hello'  
       x = x + 'world'  
  
jupman.pytut()  
  
[61]: <IPython.core.display.HTML object>
```

EXERCISE: Write some code that changes memory state in such a way so that in the end the following is printed:

```
z = This  
w = was  
x = a problem  
y = was  
s = This was a problem
```

- to write the code, USE ONLY the symbols =,+,,z,w,x,y,s AND NOTHING ELSE
- feel free to use as many lines of code as you deem necessary
- feel free to use any symbol as many times you deem necessary

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[62]: # these variables are given  
  
z = "This"  
w = 'is'  
x = 'a problem'  
y = 'was'  
s = ''  
  
# write here the code  
  
w = y  
  
s = z + s + y + s + x  
  
</div>
```

```
[62]: # these variables are given

z = "This"
w = 'is'
x = 'a problem'
y = 'was'
s = ' '

# write here the code
```

```
[63]: print("z = ", z)
print("w = ", w)
print("x = ", x)
print("y = ", y)
print("s = ", s)
```

Strings and numbers

Python strings have the type `str`:

```
[64]: type("hello world")
[64]: str
```

In strings we can insert characters which represent digits:

```
[65]: print("The character 5 represents the digit five, the character 3 represents the
       ↪digit three")
[65]: The character 5 represents the digit five, the character 3 represents the digit three
```

Obviously, we can also substitute a sequence of digits, to obtain something which looks like a number:

```
[66]: print("The sequence of characters 7583 represents the number seven thousand five
       ↪hundred eighty-three")
[66]: The sequence of characters 7583 represents the number seven thousand five hundred
       ↪eighty-three
```

Having said that, we can ask ourselves how Python behaves when we have a *string* which contains *only* a sequence of characters which represents a number, like for example '`254`'

Can we use `254` (which we wrote like it were a string) also as if it were a number? For example, can we sum `3` to it?

```
'254' + 3
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-29-d39aa62a7e3d> in <module>
----> 1 "254" + 3

TypeError: can only concatenate str (not "int") to str
```

As you see, Python immediately complains, because we are trying to mix different types.

SO:

- by writing '254' between apices we create a *string* of type `str`
- by writing 254 we create a *number* of type `int`

```
[67]: type('254')
```

```
[67]: str
```

```
[68]: type(254)
```

```
[68]: int
```

BEWARE OF `print` !!

If you try to print a string which only contains digits, Python will show it without apices, and this might mislead you about its true nature !!

```
[69]: print('254')
```

```
254
```

```
[70]: print(254)
```

```
254
```

Only in Jupyter, to show constants, variables or results of calculations, as `print` alternative you can directly insert a formula in the cell. In this case we are simply showing a constant, and whenever it is a string you will see apices:

```
[71]: '254'
```

```
[71]: '254'
```

```
[72]: 254
```

```
[72]: 254
```

The same reasoning applies also to variables:

```
[73]: x = '254'
```

```
[74]: x
```

```
[74]: '254'
```

```
[75]: y = 254
```

```
[76]: y
```

```
[76]: 254
```

So, *only in Jupyter*, when you need to show a constant, a variable or a calculation often it's more convenient to directly write it in the cell without using `print`.

Conversions - from string to number

Let's go back to the problem of summing '254' + 3. The first one is a string, the second a number. If they were both numbers the sum would surely work:

```
[77]: 254 + 3
```

```
[77]: 257
```

So we can try to convert the string '254' into an authentic integer. To do it, we can use `int` as if it were a function, and pass as argument the string to be converted:

```
[78]: int('254') + 3
```

```
[78]: 257
```

WARNING: strings and numbers are immutable !!

This means that by writing `int('254')` a *new* number is generated without minimally affecting the string '254' from where we started from. Let's see an example:

```
[79]: x = '254'      # assign to variable x the string '254'
```

```
[80]: y = int(x)    # assign to variable y the number obtained by converting '254' in int
```

```
[81]: x           # variable x is now assigned to string '254'
```

```
[81]: '254'
```

```
[82]: y           # in y now there is a number instead (note we don't have apices here)
```

```
[82]: 254
```

It might be useful to see again the example in Python Tutor:

```
[83]: x = "254"
```

```
y = int(x)
```

```
print(y + 3)
```

```
jupman.pytut()
```

```
257
```

```
[83]: <IPython.core.display.HTML object>
```

EXERCISE: Try to convert a string which represents an ill-formed number (for example a number with inside a character: '43K12') into an `int`. What happens?

```
[84]: # write here
```

Conversions - from number to string

Any object can be converted to string by using `str` as if it were a function and by passing the object to convert. Let's try then to convert a number into a string.

```
[85]: str(5)
```

```
[85]: '5'
```

note the apices in the result, which show we actually obtained a string.

If by chance we want to obtain a string which is the concatenation of objects of different types we need to be careful:

```
x = 5
s = 'Workdays in a week are ' + x
print(s)

-----
TypeError                                     Traceback (most recent call last)
<ipython-input-154-5951bd3aa528> in <module>
      1 x = 5
----> 2 s = 'Workdays in a week are ' + x
      3 print(s)

TypeError: can only concatenate str (not "int") to str
```

A way to circumvent the problem (even if not the most convenient) is to convert into string each of the objects we're using in the concatenation:

```
[86]: x = 3
y = 1.6
s = "This week I've been jogging " + str(x) + " times running at an average speed of
     " + str(y) + " km/h"
print(s)
```

```
This week I've been jogging 3 times running at an average speed of 1.6 km/h
```

QUESTION: Having said that, after executing the code in previous cell, variable `x` is going to be associated to a *number* or a *string* ?

If you have doubts, use Python Tutor.

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: numbers, like strings, are immutable. So by calling the function `str(x)` it is impossible for the number 5 associated to `x` to be changed in any way. `str(x)` will simply generate a NEW string '`5`' which will then be used in the concatenation.

</div>

Formatting strings

Concatenating strings with plus sign like above is cumbersome and error prone. There are several better solutions, for a thorough review we refer to [Real Python](#)¹²⁶ website.

Formatting with %

Here we now see how to format strings with the % operator. This solution is not the best one, but it's widely used and supported in all Python versions, so we adopted it throughout the book:

```
[87]: x = 3
      "I jumped %s times" % x
[87]: 'I jumped 3 times'
```

Notice we put a so-called *place-holder* %s inside the string, which tells Python to replace it with a variable. To feed Python the variable, *after* the string we have to put a % symbol followed by the variable, in this case x.

If we want to place more than one variable, we just add more %s place-holders and after the external % we place the required variables in round parenthesis, separating them with commas:

```
[88]: x = 3
      y = 5
      "I jumped %s times and did %s sprints" % (x,y)
[88]: 'I jumped 3 times and did 5 sprints'
```

We can put as many variables as we want, also non-numerical ones:

```
[89]: x = 3
      y = 5
      prize = 'Best Athlet in Town'
      "I jumped %s times, did %s sprints and won the prize '%s'" % (x,y,prize)
[89]: 'I jumped 3 times, did 5 sprints and won the prize 'Best Athlet in Town''
```

Formatting with f-strings

f-strings allow to directly insert expressions between curly brackets {} into the string. To signal Python to calculate and convert the expressions into strings, the string must be preceded by the f letter. Note the moment you add the f your editor should show the expressions between curly brackets with a different color.

Warning: f-strings are only available since Python ≥ 3.6

```
[90]: title = "King of Great Britain"
      start = 1760
      end = 1801

      s1 = f"Giorgie III was {title.upper()} from {start} until {end}."
      print(s1)

      s2 = f"He ruled for {end - start} years."
      print(s2)
```

¹²⁶ <https://realpython.com/python-formatted-output/>

```
George III was KING OF GREAT BRITAIN from 1760 until 1801.  
He ruled for 41 years.
```

Exercise - supercars

You've got some money, so you decide to buy two models of supercars. Since you already know accidents are on the way, for each model you will buy as many cars as there are characters in each model name.

Write some code which stores in the string `s` the number of cars you will buy into the strings:

- `sa` formatted with `%s` placeholders
- `sb` formatted as f-string

Example - given:

```
car1 = 'Jaguar'  
car2 = 'Ferrari'
```

After your code, it should show:

```
>>> s1  
'I will buy 6 Jaguar and 7 Ferrari supercars'  
>>> s2  
'I will buy 6 Jaguar and 7 Ferrari supercars'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[91]:  
car1, car2 = 'Jaguar', 'Ferrari'      # I will buy 6 Jaguar and 7 Ferrari supercars  
#car1, car2 = 'Porsche', 'Lamborghini' # I will buy 7 Porsche and 11 Lamborghini  
→supercars  
  
# write here  
  
sa = "I will buy %s %s and %s %s supercars" % (len(car1), car1, len(car2), car2)  
sb = f"I will buy {len(car1)} {car1} and {len(car2)} {car2} supercars"  
print(sa)  
print(sb)  
  
I will buy 6 Jaguar and 7 Ferrari supercars  
I will buy 6 Jaguar and 7 Ferrari supercars
```

</div>

```
[91]:  
car1, car2 = 'Jaguar', 'Ferrari'      # I will buy 6 Jaguar and 7 Ferrari supercars  
#car1, car2 = 'Porsche', 'Lamborghini' # I will buy 7 Porsche and 11 Lamborghini  
→supercars  
  
# write here
```

Continue

Go on reading notebook Strings 2 - operators¹²⁷

[]:

5.2.2 Strings 2 - operators

Download exercises zip

Browse files online¹²⁸

Python offers several operators to work with strings:

Operator	Syntax	Result	Meaning
len	len(str)	int	Returns the length of the string
<i>indexing</i>	str [int]	str	Reads the character at the specified index
<i>concatenation</i>	str + str	str	Concatenate two strings
<i>inclusion</i>	str in str	bool	Checks whether a string is contained inside another one
<i>slice</i>	str [int :int]	str	Extracts a sub-string
<i>equality</i>	==, !=	bool	Checks whether strings are equal or different
<i>comparisons</i>	<, <=, >, >=	bool	Performs lexicographic comparison
ord	ord(str)	int	Returns the order of a character
chr	chr(int)	str	Given an order, returns the corresponding character
<i>replication</i>	str * int	str	Replicate the string

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
strings
  strings1.ipynb
  strings1-sol.ipynb
  strings2.ipynb
  strings2-sol.ipynb
  strings3.ipynb
  strings3-sol.ipynb
  strings4.ipynb
  strings4-sol.ipynb
  strings5-chal.ipynb
  jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook strings2.ipynb
- Go on reading the exercises file, sometimes you will find paragraphs marked **EXERCISE** which will ask to write Python commands in the following cells.

¹²⁷ <https://en.softpython.org/strings/strings2-sol.html>

¹²⁸ <https://github.com/DavidLeoni/softpython-en/tree/master/strings>

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

Reading characters

A string is a sequence of characters, and often we might want to access a single character by specifying the position of the character we are interested in.

It's important to remember that the position of characters in strings start from 0. For reading a character in a certain position, we need to write the string followed by square parenthesis and specify the position inside. Examples:

```
[2]: 'park'[0]
```

```
[2]: 'p'
```

```
[3]: 'park'[1]
```

```
[3]: 'a'
```

```
[4]: #0123
```

```
'park'[2]
```

```
[4]: 'r'
```

```
[5]: #0123
```

```
'park'[3]
```

```
[5]: 'k'
```

If we try to go beyond the last character, we will get an error:

```
#0123
'park'[4]
-----
IndexError                                     Traceback (most recent call last)
<ipython-input-106-b8f1f689f0c7> in <module>
      1 #0123
      2 'park'[4]

IndexError: string index out of range
```

Before we used a string by specifying it as a literal, but we can also use variables:

```
[6]: #01234
x = 'cloud'
```

```
[7]: x[0]
```

```
[7]: 'c'
```

```
[8]: x[2]
```

[8]: 'o'

How is represented the character we've just read? If you noticed, it is between quotes like if it were a string. Let's check:

[9]: type(x[0])

[9]: str

It's really a string. To somebody this might come as a surprise, also from a philosophical standpoint: Python strings are made of... strings! Other programming languages may use a specific type for the single character, but Python uses strings to be able to better manage complex alphabets as, for example, japanese.

QUESTION: Let's suppose `x` is *any* string. If we try to execute this code:

x[0]

we will get:

1. always a character
2. always an error
3. sometimes a character, sometimes an error according to the string

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 3: we might obtain an error with the empty string (try it)

</div>

QUESTION: Let's suppose `x` is an empty string. If we try to execute this code:

x[len(x)]

we will get:

1. always a character
2. always an error
3. sometimes a character, sometimes an error according to the string at hand

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 2: since indexing starts from 0, `len` always gives us a number which is the biggest usable index plus one.

</div>

Exercise - alternate

Given two strings both of length 3, print a string which alternates characters from both strings. Your code must work with any string of this length

Example - given:

```
x="say"
y="hi!"
```

it should print:

```
shaiy!
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[10]: # write here
```

```
x="say"
y="hi!"
print(x[0] + y[0] + x[1] + y[1] + x[2] + y[2])
shaiy!
```

```
</div>
```

```
[10]: # write here
```

```
shaiy!
```

Negative indexes

In Python we can also use negative indexes, which instead to start from *the beginning* they start *from the end*:

```
[11]: #4321
"park"[-1]
```

```
[11]: 'k'
```

```
[12]: #4321
"park"[-2]
```

```
[12]: 'r'
```

```
[13]: #4321
"park"[-3]
```

```
[13]: 'a'
```

```
[14]: #4321
"park"[-4]
```

```
[14]: 'p'
```

If we go one step beyond, we get an error:

```
#4321
"park"[-5]

-----
IndexError                                     Traceback (most recent call last)
<ipython-input-126-668d8a13a324> in <module>
----> 1 "park"[-5]

IndexError: string index out of range
```

QUESTION: Suppose `x` is a NON-empty string. What do we get with the following expression?

```
x[-len(x)]
```

1. always a character
2. always an error
3. sometimes a character, sometime an error according to the string

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 1. (we supposed the string is never empty)

</div>

QUESTION: Suppose x is a some string (possibly empty), the expressions

```
x[len(x) - 1]
```

and

```
x[-len(x) - 1]
```

are equivalent ? What do they do ?

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: the expressions are not equivalent: first one gives last character only with non-empty strings, while the second always produces an error.

</div>

QUESTION: If x is a non-empty string, what does the following expression produce? Can we simplify it to a shorter one?

```
(x + x)[len(x)]
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: it's the same as $x[0]$

</div>

QUESTION: If x is a non-empty string, what does the following expression produce? An error? Something else? Can we simplify it?

```
'park'[0][0]
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: We know that $'park'[0]$ produces a character, but we also know that in Python characters extracted from strings are also strings of length 1. So, if after the expression $"park"[0]$ which produces the string $'p'$ we add another $[0]$ it's like we were writing $'p'[0]$, which returns the zeroth character found in the string in the string $'p'$, that is $'p'$ itself.

</div>

QUESTION: If x is a non-empty string, what does the following expression produce? An error? Something else? Can we simplify it?

```
(x[0])[0]
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: `x[0]` is an expression which produces the first character of the string `x`. In Python, we can place expressions among parenthesis whenever we want. So in this case the parenthesis don't produce any effect, and the expression becomes equivalent to `x[0][0]` which as we've seen before it's the same as writing `x[0]`

```
</div>
```

Substitute characters

We said strings in Python are immutable. Suppose we have a string like this:

```
[15]: #01234  
x = 'port'
```

and, for example, we want to change the character at position 2 (in this case, the `r`) into an `s`. What do we do?

We might be tempted to write like the following, but Python would punish us with an error:

```
x[2] = 's'  
  
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-113-e5847c6fa4bf> in <module>  
----> 1 x[2] = 's'  
  
TypeError: 'str' object does not support item assignment
```

The correct solution is assigning a completely new string to `x`, obtained by taking pieces from the previous one:

```
[16]: x = x[0] + x[1] + 's' + x[3]
```

```
[17]: x
```

```
[17]: 'post'
```

If seeing `x` to the right of equal sign baffles you, we can decompose the code like this and it will work the same way:

```
[18]: x = "port"  
y = x  
x = y[0] + y[1] + 's' + y[3]
```

Try it in Python Tutor:

```
[19]: x = "port"  
y = x  
x = y[0] + y[1] + 's' + y[3]  
  
jupman.pyput()  
  
[19]: <IPython.core.display.HTML object>
```

Slices

We might want to read only a subsequence which starts from a position and ends up in another one. For example, suppose we have:

```
[20]: #0123456789
x = 'mercantile'
```

and we want to extract the string 'canti', which starts at index 3 **included**. We might extract the single characters and concatenate them with + sign, but we would write a lot of code. A better option is to use the so-called **slices**¹²⁹: simply write the string followed by square parenthesis containing only start index (**included**), a colon, and finally end index (**excluded**):

```
[21]: #0123456789
x = 'mercantile'

x[3:8]    # note the : inside start and end indexes
[21]: 'canti'
```

WARNING: Extracting with slices DOES NOT modify the original string !!

Let's see an example:

```
[22]: #0123456789
x = 'mercantile'

print('           x is', x)
print('The slice x[3:8] is', x[3:8])
print('           x is', x)      # note x continues to point to old string!
                               x is mercantile
The slice x[3:8] is canti
                               x is mercantile
```

QUESTION: if x is any string of length at least 5, what does this code produce? An error? It works? Can we shorten it?

```
x[3:4]
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: If the string has length at least 5, we might have a situation like this:

```
#01234
x = 'abcde'
```

The slice x[3:4] will extract from position 3 **included** until position 4 **excluded**, so as a matter of fact it will extract only one character from position 3. So the code is equivalent to x[3]

</div>

¹²⁹ <http://greenteapress.com/thinkpython2/html/thinkpython2009.html#sec95>

Exercise - garalampog

Write some code to extract and print alam from the string "garalampog". Try guessing the correct indexes.

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[23]: x = "garalampog"

# write here

#      0123456789
print(x[3:7])
```

alam

</div>

```
[23]: x = "garalampog"

# write here
```

alam

Exercise - ifEweEfav lkSD lkWe

Write some code to extract and print kD from the string "ifE\te\nfav lkD lkWe". Be careful of spaces and special characters (before you might want to print x). Try guessing correct indexes.

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[24]: x = "ifE\te\nfav lkD lkWe"

# write here

#      0123 45 67890123456789
#x = "ifE\te\nfav lkD lkWe"

print(x[12:14])
```

kD

</div>

```
[24]: x = "ifE\te\nfav lkD lkWe"

# write here
```

kD

Slices - limits

Whenever we use slice we must be careful with index limits. Let's see how they behave:

```
[25]: #012345
"chair"[0:3] # from index 0 *included* to 3 *excluded*
```

```
[25]: 'cha'
```

```
[26]: #012345
"chair"[0:4] # from index 0 *included* to 4 *excluded*
```

```
[26]: 'chai'
```

```
[27]: #012345
"chair"[0:5] # from index 0 *included* to 5 *excluded*
```

```
[27]: 'chair'
```

```
[28]: #012345
"sedia"[0:6] # if we go beyond string length Python doesn't complain
```

```
[28]: 'sedia'
```

QUESTION: if x is any string (also empty), what does this expression do? Can it give an error? Does it return something useful?

```
x[0:len(x)]
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: It always returns a NEW copy of the whole string, because it starts from index 0 *included* and ends at index $\text{len}(x)$ *excluded*.

It also works with the empty string, as `''[0:len('')]` is equivalent to `''[0:0]` that is a substring from 0 *included* to 0 *excluded*, so we don't take any character and we do not go beyond string limits. Actually, even if we went beyond, we wouldn't upset Python (try writing `''[0:100]`

</div>

Slice - Omitting limits

If we want, it's possible to omit the starting index, in this case Python will suppose it's a 0:

```
[29]: #0123456789
"catamaran"[::3]
```

```
[29]: 'cat'
```

It's also possible to omit the ending index, in that case Python will extract until the end of the string:

```
[30]: #0123456789
"catamaran"[3::]
```

```
[30]: 'amaran'
```

By omitting both indexes we obtain the full string:

```
[31]: "catamaran"[:]
[31]: 'catamaran'
```

Exercise - ystertymyster

Write some code that given a string x prints the string composed with all the characters of x except the first one, followed by all characters of x except the last one.

- your code must work with any string

Example 1 - given:

```
x = "mystery"
```

must print:

```
ystertymyster
```

Example 2 - given:

```
x = "rope"
```

must print:

operop

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[32]: 
x = "mystery"
#x = "rope"

# write here

print(x[1:] + x[0:len(x)-1])
ystertymyster
```

</div>

```
[32]: 
x = "mystery"
#x = "rope"

# write here
```

Slice - negative limits

If we want, it's also possible to set negative limits, although it's not always intuitive:

```
[33]: #0123456
"vegetal"[3:0]    # from index 3 to positive indexes <= 3 doesn't produce anything
[33]: ''

[34]: #0123456
"vegetal"[3:1]    # from index 3 to positive indexes <= 3 doesn't produce anything
[34]: ''

[35]: #0123456
"vegetal"[3:2]    # from index 3 to positive indexes <= 3 doesn't produce anything
[35]: ''

[36]: #0123456
"vegetal"[3:3]    # from index 3 to positive indexes <= 3 doesn't produce anything
[36]: ''
```

Let's see what happens with negative indexes:

```
[37]: #0123456  positive indexes
#7654321  negative indexes
"vegetal"[3:-1]
[37]: 'eta'

[38]: #0123456  positive indexes
#7654321  negative indexes
"vegetal"[3:-2]
[38]: 'et'

[39]: #0123456  positive indexes
#7654321  negative indexes
"vegetal"[3:-3]
[39]: 'e'

[40]: #0123456  positive indexes
#7654321  negative indexes
"vegetal"[3:-4]
[40]: ''

[41]: #0123456  positive indexes
#7654321  negative indexes
"vegetal"[3:-5]
[41]: ''
```

Exercise - javarnanda

Given a string `x`, write some code to extract and print its last 3 characters joined to the first 3.

- Your code should work for any string of length equal or greater than 3

Example 1 - given:

```
x = "javarnanda"
```

it should print:

```
javnda
```

Example 2 - given:

```
x = "bang"
```

it should print:

```
banang
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[42]: x = "javarnanda"
#x = "bang"
```

```
# write here
```

```
print(x[:3] + x[-3:])
```

```
javnda
```

```
</div>
```

```
[42]: x = "javarnanda"
#x = "bang"
```

```
# write here
```

```
javnda
```

Slice - modifying

Suppose to have the string

```
[43]: #0123456789
s = "the table is placed in the center of the room"
```

and we want to change `s` assignment so it becomes associated to the string:

```
#0123456789
"the chair is placed in the center of the room"
```

Since both strings are similar, we might be tempted to only redefine the character sequence which corresponds to the word "table", which goes from index 4 included to index 9 excluded:

```
s[4:9] = "chair" # WARNING! WRONG!
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-57-0de7363c6882> in <module>
----> 1 s[4:9] = "chair" # WARNING! WRONG!
TypeError: 'str' object does not support item assignment
```

Sadly, we would receive an error, because as repeated many times strings are IMMUTABLE, so we cannot select a chunk of a particular string and try to change the original string. What we can do instead is to build a NEW string from pieces of the original string, concatenates the desired characters and associates the result to the variable of which we want to modify the assignment:

```
[44]: #0123456789
s = "the table is placed in the center of the room"
s = s[0:4] + "chair" + s[9:]
print(s)

the chair is placed in the center of the room
```

When Python finds the line

```
s = s[0:4] + "chair" + s[9:]
```

FIRST it calculates the result on the right of the =, and THEN associates the result to the variable on the left. In the expression on the right only NEW strings are generated, which once built can be assigned to variable s

Exercise - the run

Write some code such that when given the string s

```
s = 'The Gold Rush has begun.'
```

and some variables

```
what = 'Atom'
happened = 'is over'
```

substitutes the substring 'Gold' with the string in the variable what and substitutes the substring 'has begun' with the string in the variable happened.

After executing your code, the string associated to s should be

```
>>> print(s)
"The Atom Rush is over."
```

- DON'T use constant characters in your code, i.e. dots ' . ' aren't allowed !

Show solution
Hide

```
[45]: #01234567890123456789012345678
s = 'The Gold Rush has begun.'
what = 'Atom'
happened = 'is over'

# write here

s = s[0:4] + what + s[8:14] + happened + s[23:]
print(s)

The Atom Rush is over.
```

</div>

```
[45]: #01234567890123456789012345678
s = 'The Gold Rush has begun.'
what = 'Atom'
happened = 'is over'

# write here

The Atom Rush is over.
```

Inclusion operator

To check if a string is included in another one, we use the the `in` operator.

Note the result of this expression is a boolean:

```
[46]: 'the' in 'Singing in the rain'
[46]: True

[47]: 'si' in 'Singing in the rain' # in operator is case-sensitive
[47]: False

[48]: 'Si' in 'Singing in the rain'
[48]: True
```

Do not abuse `in`

WARNING: `in` is often used in a wrong / inefficient way

Always ask yourself:

1. Could the string *not* contain the substring we're looking for? Always remember to handle also this case!
2. `in` performs a search on all the string, which might be inefficient: is it really necessary, or do we already know the interval where to search?

3. if we want to know whether character is in a position we know a priori (i.e. 3), `in` is not needed, it's enough to write `my_string[3] == character`. By using `in` Python might find duplicated characters which are *before* or *after* the one we want to verify!

Exercise - contained 1

You are given two strings `x` and `y`, and a third `z`. Write some code which prints `True` if `x` and `y` are both contained in `z`.

Example 1 - given:

```
x = 'cad'
y = 'ra'
z = 'abracadabra'
```

it should print:

```
True
```

Example 2 - given:

```
x = 'zam'
y = 'ra'
z = 'abracadabra'
```

it should print:

```
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[49]:

```
x,y,z = 'cad','ra','abracadabra'    # True
#x,y,z = 'zam','ra','abracadabra'    # False

# write here

print((x in z) and (y in z))
```

```
True
```

```
</div>
```

[49]:

```
x,y,z = 'cad','ra','abracadabra'    # True
#x,y,z = 'zam','ra','abracadabra'    # False

# write here
```

Exercise - contained 2

Given three strings `x`, `y`, `z`, write some code which prints `True` if the string `x` is contained in at least one of the strings `y` or `z`, otherwise prints `False`

- your code should work with any set of strings

Example 1 - given:

```
x = "ope"
y = "honesty makes for long friendships"
z = "I hope it's clear enough"
```

it should print:

`True`

Example 2 - given:

```
x = "nope"
y = "honesty makes for long friendships"
z = "I hope it's clear enough"
```

it should print:

`False`

Example 3 - given:

```
x = "cle"
y = "honesty makes for long friendships"
z = "I hope it's clear enough"
```

it should show:

`True`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[50]:

```
x,y,z = "ope","honesty makes for long friendships","I hope it's clear enough" # True
#x,y,z = "nope","honesty makes for long friendships","I hope it's clear enough" #_
#False
#x,y,z = "cle","honesty makes for long friendships","I hope it's clear enough" # True

# write here

print((x in y) or (x in z))
```

`True`

`</div>`

[50]:

```
x,y,z = "ope","honesty makes for long friendships","I hope it's clear enough" # True
#x,y,z = "nope","honesty makes for long friendships","I hope it's clear enough" #_
#False
#x,y,z = "cle","honesty makes for long friendships","I hope it's clear enough" # True
```

(continues on next page)

(continued from previous page)

```
# write here
```

Comparisons

Python offers us the possibility to perform a *lexicographic comparison* among strings, like we would when placing names in an address book. Although sorting names is something intuitive we often do, we must be careful about special cases.

First, let's determine when two strings are equal.

Equality operators

To check whether two strings are equal, you can use te operator `==` which as result produces the boolean `True` or `False`

WARNING: `==` is written with TWO equal signs !!!

```
[51]: "dog" == "dog"  
[51]: True
```

```
[52]: "dog" == "wolf"  
[52]: False
```

Equality operator is case-sensitive:

```
[53]: "dog" == "DOG"  
[53]: False
```

To check whether two strings are NOT equal, we can use the operator `!=`, which we can expect to behave exactly as the opposite of `==`:

```
[54]: "dog" != "dog"  
[54]: False
```

```
[55]: "dog" != "wolf"  
[55]: True
```

```
[56]: "dog" != "DOG"  
[56]: True
```

As an alternative, we might use the operator `not`:

```
[57]: not "dog" == "dog"  
[57]: False
```

```
[58]: not "wolf" == "dog"
```

```
[58]: True
```

```
[59]: not "dog" == "DOG"
```

```
[59]: True
```

QUESTION: what does the following code print?

```
x = "river" == "river"  
print(x)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: When Python encounters `x = "river" == "river"` it sees an assignment, and associates the result of the expression `"river" == "river"` to the variable `x`. So FIRST it calculates the expression `"river" == "river"` which produces the boolean `True`, and THEN associates the value `True` to the variable `x`. Finally `True` is printed.

</div>

QUESTION: for each of the following expressions, try to guess whether it produces `True` or `False`

1. `'hat' != 'Hat'`

2. `'hat' == 'HAT'`

3. `'choralism'[2:5] == 'contemporary'[7:10]`

4. `'AlAbAmA'[4:] == 'aLaBaMa'`

5. `'bright'[9:20] == 'dark'[10:15]`

6. `'optical'[-1] == 'crystal'[-1]`

7. `('hat' != 'jacket') == ('trousers' != 'bow')`

8. `('stra' in 'stradivarius') == ('div' in 'digital divide')`

9. `len('note') in '5436'`

10. `str(len('note')) in '5436'`

11. `len('posters') in '5436'`

12. `str(len('posters')) in '5436'`

Exercise - statist

Write some code which prints True if a word begins with the same two characters it ends with.

- Your code should work for any word

Show solution

<div>

[60]:

```
word = 'statist'    # True
#word = 'baobab'    # False
#word = 'maxima'    # True
#word = 'karma'     # False

# write here

print(word[:2] == word[-2:len(word)])
True
```

</div>

[60]:

```
word = 'statist'    # True
#word = 'baobab'    # False
#word = 'maxima'    # True
#word = 'karma'     # False

# write here
```

Comparing characters

Characters have an inherent order we can exploit. Let's see an example:

[61]: 'a' < 'g'

[61]: True

another one:

[62]: 'm' > 'c'

[62]: True

They sound reasonable comparisons! But what about this (notice capital 'Z')?

[63]: 'a' < 'Z'

[63]: False

Maybe this doesn't look so obvious. And what if we get creative and compare with symbols such as square bracket or Unicode¹³⁰ hearts ??

¹³⁰ <https://en.softpython.org/strings/strings1-sol.html#Unicode-characters>

```
[64]: 'a' > '♥'  
[64]: False
```

To determine how to deal with this special cases, we must remember ASCII¹³¹ assignes a position number to each character, defining as a matter of fact *an ordering* between all its characters.

If we want to know the corresponding number of a character, we can use the function `ord`:

```
[65]: ord('a')  
[65]: 97
```

```
[66]: ord('b')  
[66]: 98
```

```
[67]: ord('z')  
[67]: 122
```

If we want to go the other way, given a position number we can obtain the corresponding character with `chr` function:

```
[68]: chr(97)  
[68]: 'a'
```

Uppercase characters have different positions:

```
[69]: ord('A')  
[69]: 65  
  
[70]: ord('Z')  
[70]: 90
```

EXERCISE: Using the functions above, try to find which characters are *between* capital Z and lowercase a
Show solutionHide</div>

```
[71]:  
# write here  
print(chr(91),chr(92), chr(93),chr(94), chr(95),chr(96))  
[ \ ] ^ _ `
```

</div>

```
[71]:  
# write here
```

The ordering allows us to perform *lexicographic comparisons* between single characters:

```
[72]: 'a' < 'b'
```

¹³¹ <https://en.softpython.org/strings/strings1-sol.html#ASCII-characters>

[72]: True

[73]: 'g' >= 'm'

[73]: False

EXERCISE: Write some code that:

1. prints the `ord` values of 'A', 'Z' and a given `char`
2. prints `True` if `char` is uppercase, and `False` otherwise
 - Would your code also work with accented capitalized characters such as 'Á'?
 - **NOTE:** the possible character sets are way too many, so the proper solution would be to use the method `isupper`¹³² we will see in the next tutorial.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[74]:

```
char = 'G'    # True
#char = 'g'  # False

#char = 'Á'  # True ?? Note the accent!
# write here
print('A:', ord('A'), ' Z:', ord('Z'))
print(char + ':', ord(char))
ord(char) >= ord('A') and ord(char) <= ord('Z')  # only checks simple English
  ↪alphabet cases
```

A: 65 Z: 90
G: 71

[74]: True

</div>

[74]:

```
char = 'G'    # True
#char = 'g'  # False

#char = 'Á'  # True ?? Note the accent!
# write here
```

Also, since Unicode character set *includes* ASCII, the ordering of ASCII characters can be used to safely compare them against unicode characters, so comparing characters or their `ord` should be always equivalent:

[75]: `ord('a')` # ascii

[75]: 97

[76]: `ord('♥')` # unicode

[76]: 9829

¹³² <https://en.softpython.org/strings/strings3-sol.html#isupper-and-islower-methods>

```
[77]: 'a' > '♥'
```

```
[77]: False
```

```
[78]: ord('a') > ord('♥')
```

```
[78]: False
```

Python also offers lexicographic comparisons on strings with more than one character. To understand what the expected result should be, we must distinguish among several cases, though:

- strings of equal / different length
- strings with same / mixed case

Let's begin with same length strings:

```
[79]: 'mario' > 'luigi'
```

```
[79]: True
```

```
[80]: 'mario' > 'wario'
```

```
[80]: False
```

```
[81]: 'Mario' > 'Wario'
```

```
[81]: False
```

```
[82]: 'Wario' < 'mario'      # capital case is *before* lowercase in ASCII
```

```
[82]: True
```

Comparing different lengths

Short strings which are included in longer ones come first in the ordering:

```
[83]: 'troll' < 'trolley'
```

```
[83]: True
```

If they only share a prefix with a longer string, Python compares characters after the common prefix, in this case it detects that e precedes the corresponding s:

```
[84]: 'trolley' < 'trolls'
```

```
[84]: True
```

Exercise - Character intervals

You are given a couple of strings `i1` and `i2` of two characters each.

We suppose they represent character intervals: the first character of an interval always has order number lower or equal than the second.

There are five possibilities: either the first interval ‘is contained in’, or ‘contains’, or ‘overlaps’, or ‘is before’ or ‘is after’ the second interval. Write some code which tells which containment relation we have.

Example 1 - given:

```
i1 = 'gm'
i2 = 'cp'
```

Your program should print:

```
gm is contained in cp
```

To see why, you can look at this little representation (you **don't** need to print this!):

```
c   g     m   p
abcdefghijklmnopqrstuvwxyz
```

Example 2 - given:

```
i1 = 'mr'
i2 = 'pt'
```

Your program should print:

```
mr overlaps pt
```

because `mr` is not contained nor contains nor completely precedes nor completely follows `pt` (you **don't** need to print this!):

```
      m   p   r   t
abcdefghijklmnopqrstuvwxyz
```

- if `i1` interval coincides with `i2`, it is considered as containing `i2`
- **DO NOT** use cycles nor `if`
- **HINT:** to satisfy above constraint, think about `booleans evaluation order`¹³³, for example the expression

```
'g' >= 'c' and 'm' <= 'p' and 'is contained in'
```

produces as result the string ‘`is contained in`’

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[85]:

```
i1,i2 = 'gm', 'cp'    # gm is contained in cp
#i1,i2 = 'dh', 'dh'    # gm is contained in cp  #(special case)
#i1,i2 = 'bw', 'dq'    # bw contains dq
#i1,i2 = 'ac', 'bd'    # ac overlaps bd
```

(continues on next page)

¹³³ <https://en.softpython.org/basics/basics2-bools-sol.html#Evaluation-order>

(continued from previous page)

```
#i1,i2 = 'mr', 'pt' # mr overlaps pt
#i1,i2 = 'fm', 'su' # fm is before su
#i1,i2 = 'xz', 'pq' # xz is after pq

# write here
res = (i1[0] >= i2[0] and i1[1] <= i2[1] and 'is contained in') \
    or (i1[0] < i2[0] and i1[1] > i2[1] and 'contains') \
    or (i1[0] >= i2[0] and i1[0] <= i2[1] and 'overlaps') \
    or (i1[1] >= i2[0] and i1[1] <= i2[1] and 'overlaps') \
    or (i1[1] < i2[0] and 'is before') \
    or (i1[0] > i2[1] and 'is after')

print(i1, res, i2)
gm is contained in cp
```

</div>

[85]:

```
i1,i2 = 'gm','cp' # gm is contained in cp
#i1,i2 = 'dh','dh' # gm is contained in cp # (special case)
#i1,i2 = 'bw','dq' # bw contains dq
#i1,i2 = 'ac','bd' # ac overlaps bd
#i1,i2 = 'mr','pt' # mr overlaps pt
#i1,i2 = 'fm','su' # fm is before su
#i1,i2 = 'xz','pq' # xz is after pq

# write here
```

Exercise - The Library of Encodicus

In the study room of the algorithmist Encodicus there is a bookshelf divided in 26 alphabetically ordered sections, where he scrupulously keeps his precious alchemical texts. Every section can contain at most 9 books. One day, Encodicus decides to acquire a new tome for his collection: write some code which given a string representing `bookshelf` with the counts of the books and a new `book`, finds the right position of the book and updates `bookshelf` accordingly

- assume no section contains 9 books
- assume book names are always lowercase
- **DO NOT** use cycles, if, nor string methods
- **DO NOT** manually write strings with 26 characters, or even worse create 26 variables
- **USE** `ord` to find the section position

Example - given:

```
scaffale = "|a 7|b 5|c 5|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r"
libro = "cycling in the wild"
```

after your code `bookshelf` must result updated with | c 6|:

```
>>> print(bookshelf)
|a 7|b 5|c 6|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r 2|s 4|t 6|u
→1|v 3|w 3|x 5|y 7|z 6|
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[86]:

```
book = "cycling in the wild"
#book = "algorithms of the occult"
#book = "theory of the zippo"
#book = "zoology of the software developer"

bookshelf = "|a 7|b 5|c 5|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r
→2|s 4|t 6|u 1|v 3|w 3|x 5|y 7|z 6|"

# write here

c = book[0]

i = ord(c) - ord('a')
n = int(bookshelf[(i*4)+3:(i)*4+4])

bookshelf = bookshelf[:i*4] + ' | ' + c + ' ' + str(n+1) + bookshelf[(i+1)*4:]

print(bookshelf)
|a 7|b 5|c 6|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r 2|s 4|t 6|u
→1|v 3|w 3|x 5|y 7|z 6|
```

</div>

[86]:

```
book = "cycling in the wild"
#book = "algorithms of the occult"
#book = "theory of the zippo"
#book = "zoology of the software developer"

bookshelf = "|a 7|b 5|c 5|d 8|e 2|f 0|g 4|h 8|i 7|j 1|k 6|l 0|m 5|n 0|o 3|p 7|q 2|r
→2|s 4|t 6|u 1|v 3|w 3|x 5|y 7|z 6|"

# write here
```

Replication operator

With the operator * you can replicate a string n times, for example:

[87]:

```
'beer' * 4
```

[87]:

```
'beerbeerbeerbeer'
```

Note a NEW string is created, without tarnishing the original:

[88]:

```
drink = "beer"
```

```
[89]: print(drink * 4)  
beerbeerbeerbeer
```

```
[90]: drink  
[90]: 'beer'
```

Exercise - za za za

Given a syllable and a phrase which terminates with a character n as a digit, write some code which prints a string with the syllable repeated n times, separated by spaces.

- Your code must work with any string assigned to syllable and phrase

Example - given:

```
phrase = 'the number 7'  
syllable = 'za'
```

after your code, it should print:

```
za za za za za za za
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[91]:  
phrase = 'the number 7'  
syllable = 'za'      # za za za za za za za  
#phrase = 'Give me 5'  # za za za za za  
  
# write here  
  
print((syllable + ' ') * (int(phrase[-1])))  
za za za za za za za
```

</div>

```
[91]:  
phrase = 'the number 7'  
syllable = 'za'      # za za za za za za za  
#phrase = 'Give me 5'  # za za za za za  
  
# write here
```

Continue

Go on reading notebook Strings 3 - basic methods¹³⁴

[]:

5.2.3 Strings 3 - methods

Download exercises zip

Browse files online¹³⁵

Every data type has associated particular methods for that type, let's see the simple ones associated to type string (`str`):

Method	Result	Meaning
<code>str.upper()</code>	<code>str</code>	Return the string with all characters uppercase
<code>str.lower()</code>	<code>str</code>	Return the string with all characters lowercase
<code>str.capitalize()</code>	<code>str</code>	Return the string with the first uppercase character
<code>str.startswith(str)</code>	<code>bool</code>	Check if the string begins with another one
<code>str.endswith(str)</code>	<code>bool</code>	Check whether the string ends with another one
<code>str.isalpha(str)</code>	<code>bool</code>	Check if all characters are alhpabetic
<code>str.isdigit(str)</code>	<code>bool</code>	Check if all characters are digits
<code>str.isupper()</code>	<code>bool</code>	Check if all characters are uppercase
<code>str.islower()</code>	<code>bool</code>	Check if all characters are lowercase

The others are described at the page Search methods¹³⁶

WARNING: ALL string methods ALWAYS generate a NEW string

The original string object is NEVER changed (strings are immutable).

What to do

1. Unzip `exercises zip` in a folder, you should obtain something like this:

```
strings
  strings1.ipynb
  strings1-sol.ipynb
  strings2.ipynb
  strings2-sol.ipynb
  strings3.ipynb
  strings3-sol.ipynb
  strings4.ipynb
  strings4-sol.ipynb
  strings5-chal.ipynb
jupman.py
```

¹³⁴ <https://en.softpython.org/strings/strings3-sol.html>

¹³⁵ <https://github.com/DavidLeoni/softpython-en/tree/master/strings>

¹³⁶ <https://en.softpython.org/strings/strings4-sol.html>

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `strings3.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

Example - `upper`

A method is a function of an object that takes as input the object to which it is applied and performs some calculation.

The string type `str` has predefined methods like `str.upper()` which can be applied to other string objects (i.e.: '`hello`' is a string object)

The method `str.upper()` takes the string to which it is applied, and creates a NEW string in which all the characters are in uppercase. To apply a method like `str.upper()` to the particular string object '`hello`', we must write:

```
'hello'.upper()
```

First we write the object on which apply the method ('`hello`'), then a dot ., and afterwards the method name followed by round parenthesis. The brackets can also contain further parameters according to the method.

Examples:

```
[2]: 'hello'.upper()
```

```
[2]: 'HELLO'
```

```
[3]: "I'm important".upper()
```

```
[3]: "I'M IMPORTANT"
```

WARNING: like ALL string methods, the original string object on which the method is called does NOT get modified.

Example:

```
[4]: x = "hello"  
y = x.upper()      # generates a NEW string and associates it to the variables y
```

```
[5]: x          # x variable is still associated to the old string
```

```
[5]: 'hello'
```

```
[6]: y          # y variable is associated to the new string
```

[6]: 'HELLO'

Have a look now at the same example in Python Tutor:

```
[7]: x = "hello"
y = x.upper()
print(x)
print(y)

jupman.pytut()

hello
HELLO
```

[7]: <IPython.core.display.HTML object>

Exercise - walking

Write some code which given a string x (i.e.: x='walking') prints twice the row:

```
walking WALKING walking WALKING
walking WALKING walking WALKING
```

- **DO NOT** create new variables
- your code must work with any string

```
[8]: x = 'walking'

print(x, x.upper(), x, x.upper())
print(x, x.upper(), x, x.upper())

walking WALKING walking WALKING
walking WALKING walking WALKING
```

Help: If you are not sure about a method (for example, `strip`), you can ask Python for help this way:

WARNING: when using help, DON'T put parenthesis after the method name !!

```
[9]: help("hello".strip)

Help on built-in function strip:

strip(chars=None, /) method of builtins.str instance
    Return a copy of the string with leading and trailing whitespace removed.

    If chars is given and not None, remove characters in chars instead.
```

lower method

Return the string with all lowercase characters

```
[10]: my_string = "HEllo WorLd"

another_string = my_string.lower()

print(another_string)
hello world
```

```
[11]: print(my_string)  # didn't change

Hello WorLd
```

Exercise - lowermid

Write some code that given any string x of odd length, prints a new string like x having the mid-character as lowercase.

- your code must work with any string !
- **HINT:** to calculate the position of the mid-character, use integer division with the operator $/\text{ }$

Example 1 - given:

```
x = 'ADORATION'
```

it should print:

ADORaTION

Example 2 - given:

```
x = 'LEADING'
```

it should print:

LEAdING

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[12]:      #012345678
x = 'ADORATION'
#x = 'LEADING'

# write here
k = len(x) // 2
print(x[:k] + x[k].lower() + x[k+1:])
```

ADORaTION

</div>

```
[12]:      #012345678
x = 'ADORATION'
```

(continues on next page)

(continued from previous page)

```
#x = 'LEADING'
# write here
```

capitalize method

`capitalize()` creates a NEW string having only the FIRST character as uppercase:

```
[13]: "artisan".capitalize()
```

```
[13]: 'Artisan'
```

```
[14]: "premium".capitalize()
```

```
[14]: 'Premium'
```

```
[15]: x = 'goat'
y = 'goat'.capitalize()
```

```
[16]: x      # x remains associate to the old value
```

```
[16]: 'goat'
```

```
[17]: y      # y is associated to the new string
```

```
[17]: 'Goat'
```

Exercise - Your Excellence

Write some code which given two strings `x` and `y` returns the two strings concatenated, separating them with a space and both as lowercase except the first two characters which must be uppercase

Example 1 - given:

```
x = 'yoUR'
y = 'exCellEnCE'
```

it must print:

```
Your Excellence
```

Example 2 - given:

```
x = 'hEr'
y = 'maJEsty'
```

it must print:

```
Her Majesty
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[18]:

```
x,y = 'yoUR','exCeLLENCE'
#x,y = 'hEr','maJEsty'

# write here

print(x.capitalize() + " " + y.capitalize())

Your Excellence
```

</div>

[18]:

```
x,y = 'yoUR','exCeLLENCE'
#x,y = 'hEr','maJEsty'

# write here
```

startswith method

`str.startswith` takes as parameter a string and returns `True` if the string before the dot begins with the string passed as parameter. Example:

[19]: `"the dog is barking in the road".startswith('the dog')`

[19]: `True`

[20]: `"the dog is barking in the road".startswith('is barking')`

[20]: `False`

[21]: `"the dog is barking in the road".startswith('THE DOG') # uppercase is different from lowercase`

[21]: `False`

[22]: `"THE DOG BARKS IN THE ROAD".startswith('THE DOG') # uppercase is different from lowercase`

[22]: `True`

Exercise - by Jove

Write some code which given any three strings `x`, `y` and `z`, prints `True` if both `x` and `y` start with string `z`, otherwise prints `False`

Example 1 - given:

```
x = 'by Jove'
y = 'by Zeus'
z = 'by'
```

it should print:

```
True
```

Example 2 - given:

```
x = 'by Jove'
y = 'by Zeus'
z = 'from'
```

it should print:

```
False
```

Example 3 - given:

```
x = 'from Jove'
y = 'by Zeus'
z = 'by'
```

it should print:

```
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[23]:

```
x,y,z = 'by Jove', 'by Zeus', 'by'      # True
#x,y,z = 'by Jove', 'by Zeus', 'from'    # False
#x,y,z = 'from Jove', 'by Zeus', 'by'    # False

# write here

print(x.startswith(z) and y.startswith(z))
```

```
True
```

</div>

[23]:

```
x,y,z = 'by Jove', 'by Zeus', 'by'      # True
#x,y,z = 'by Jove', 'by Zeus', 'from'    # False
#x,y,z = 'from Jove', 'by Zeus', 'by'    # False

# write here
```

endswith method

`str.endswith` takes as parameter a string and returns True if the string before the dot ends with the string passed as parameter. Example:

```
[24]: "My best wishes".endswith('st wishes')
[24]: True

[25]: "My best wishes".endswith('best')
[25]: False

[26]: "My best wishes".endswith('WISHES')      # uppercase is different from lowercase
[26]: False

[27]: "MY BEST WISHES".endswith('WISHES')      # uppercase is different from lowercase
[27]: True
```

Exercise - Snobbonis

Given couple names `husband` and `wife`, write some code which prints True if they share the surname, False otherwise.

- assume the surname is always at position 9
- your code must work for any couple `husband` and `wife`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution

</div>

```
[28]:          #0123456789          #0123456789
husband, wife = 'Antonio Snobbonis',      'Carolina Snobbonis'    # True
#husband, wife = 'Camillo De Spaparanzi', 'Matilda Degli Agi'   # False

# write here

print(wife.endswith(husband[9:]))

True
```

</div>

```
[28]:          #0123456789          #0123456789
husband, wife = 'Antonio Snobbonis',      'Carolina Snobbonis'    # True
#husband, wife = 'Camillo De Spaparanzi', 'Matilda Degli Agi'   # False

# write here
```

isalpha method

The method `isalpha` returns `True` if all characters in the string are alphabetic:

```
[29]: 'CoralReel'.isalpha()
[29]: True
```

Numbers are not considered alphabetic:

```
[30]: 'Route 666'.isalpha()
[30]: False
```

Also, blanks are *not* alphabetic:

```
[31]: 'Coral Reel'.isalpha()
[31]: False
```

... nor punctuation:

```
[32]: '!'.isalpha()
[32]: False
```

... nor weird Unicode stuff:

```
[33]: '♥'.isalpha()
[33]: False
```

```
[34]: '''.isalpha()
[34]: False
```

Exercise - Fighting the hackers

In the lower floors of Interpol, it is well known international hackers communicate using a slang called *Leet*. This fashion is also spreading in schools, where you are considered *K00l* (cool) if you know this inconvenient language. The idea is trying to substitute characters with numbers in written text ([Complete guide¹³⁷](#)).

```
1 -> i
2 -> z
3 -> e
4 -> h, a, y
etc
```

Write some code which checks `name` and `surname` given by students to detect Leet-like language.

- print `True` if at least one of the words contains numbers instead of alphabet characters, otherwise print `False`
- code must be generic, so must work with any word
- **DO NOT** use `if` command

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

¹³⁷ <https://simple.wikipedia.org/wiki/Leet>

[35]:

```
name, surname = 'K001', 'H4ck3r'      # True
#name, surname = 'Cool', 'H4ck3r'      # True
#name, surname = 'Romina', 'Rossi'      # False
#name, surname = 'Peppo', 'Sbirilli'    # False
#name, surname = 'K001', 'Sbirilli'     # True

# write here
print(not (name.isalpha() and surname.isalpha()))
```

```
True
```

```
</div>
```

[35]:

```
name, surname = 'K001', 'H4ck3r'      # True
#name, surname = 'Cool', 'H4ck3r'      # True
#name, surname = 'Romina', 'Rossi'      # False
#name, surname = 'Peppo', 'Sbirilli'    # False
#name, surname = 'K001', 'Sbirilli'     # True

# write here
```

isdigit method

isdigit method returns True if a string is only composed of digits:

[36]:

```
'391'.isdigit()
```

[36]:

```
True
```

[37]:

```
'400m'.isdigit()
```

[37]:

```
False
```

Floating point and scientific notations are not recognized:

[38]:

```
'3.14'.isdigit()
```

[38]:

```
False
```

[39]:

```
'4e29'.isdigit()
```

[39]:

```
False
```

Exercise - Selling numbers

The multinational ToxiCorp managed to acquire a wealth of private data of unaware users, and asks you to analyze it. They will then sell private information to the highest bidder on the black market. The offer looks questionable, but they pay well, so you accept.

We need to understand the data and how to organize it. You found several strings which look like phone numbers.

Every number should be composed like so:

```
+[national prefix 39][10 numbers]
```

For example, this is a valid number: +392574856985

Write some code which prints `True` if the string is a phone number, `False` otherwise

- Try the various combinations by uncommenting `phone =`
- Your code must be generic, should be valid for all numbers

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[40]:

```
phone = '+392574856985'      # True
#phone = '395851256954'      # False (missing '+')
#phone = '++7485125874'      # False (missing prefix)
#phone = '+3933342Blah'      # False (obvious :D )
#phone = "+3912"             # False (too short)
#phone = '+393481489942'      # True

# write here
number = phone[3:]  # I save the string excluding +39

# Let's make 4 bool to keep track of all conditions
has_plus = phone.startswith('+')                      # Does it start with + ?
has_national_prefix = phone[1:].startswith('39')    # Is there 39 after? We could have ↵done it also in the row above
is_10_long = len(number) == 10                      # Excluding the prefix, are the others 10 ↵characters?
has_all_digits = number.isdigit()                   # Are they all numbers?

# This is just a debug print
print("Variables check:", has_plus, has_national_prefix, is_10_long, has_all_digits)
# Final solution, combines everything
print("Is it a phone number?", has_plus and has_national_prefix and is_10_long and ↵has_all_digits)
```

Variables check: True True True True
Is it a phone number? True

</div>

[40]:

```
phone = '+392574856985'      # True
#phone = '395851256954'      # False (missing '+')
#phone = '++7485125874'      # False (missing prefix)
#phone = '+3933342Blah'      # False (obvious :D )
```

(continues on next page)

(continued from previous page)

```
#phone = "+3912"           # False (too short)
#phone = '+393481489942'   # True

# write here
```

isupper and islower methods

We can check whether a character is uppercase or lowercase with `isupper` and `islower` methods:

```
[41]: 'q'.isupper()
```

```
[41]: False
```

```
[42]: 'Q'.isupper()
```

```
[42]: True
```

```
[43]: 'b'.islower()
```

```
[43]: True
```

```
[44]: 'B'.islower()
```

```
[44]: False
```

They also work on longer strings, checking if all characters meet the criteria:

```
[45]: 'GREAT'.isupper()
```

```
[45]: True
```

```
[46]: 'NotSoGREAT'.isupper()
```

```
[46]: False
```

Note blanks and punctuation are not taken into account:

```
[47]: 'REALLY\nGREAT !'.isupper()
```

```
[47]: True
```

We could check whether a character is upper/lower case by examining its ASCII code but the best way to cover all alphabets is by using `isupper` and `islower` methods. For example, they also work with accented letters:

```
[48]: 'à'.isupper()
```

```
[48]: False
```

```
[49]: 'Á'.isupper()
```

```
[49]: True
```

Exercise - dwarves and GIANTS

In an unknown and exciting fantasy world live two populations, dwarves and GIANTS:

- dwarves love giving their offspring names containing only lowercase characters
- GIANTS don't even need to think about it, because it's written on the tablets of GROCK that GIANT names can only have uppercase characters

One day, a threat came from a far away kingdom, and so a team of fearless adventurers was gathered. The prophecy said only a mixed team of GIANTS and dwarves for a total of **4** people could defeat the evil.

1) Write some code which checks whether or not four adventurers can gather into a valid team:

- print `True` if the four names are both of dwarves and GIANTS, otherwise if they are of only one of the populations print `False`
 - your code must be generic, valid for all strings
- 2) Find some **GIANT names**¹³⁸ and **dwarves names**¹³⁹ and try to put them, making sure to translate them with the all uppercase / all lowercase capitalization, es "Jisog" is not a valid giant name, it must be translated into the gigantic writing "JISOG"

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[50] :

```
adv1, adv2, adv3, adv4 = 'gimli', 'savorlim', 'glazouc', 'hondouni'      # False
#adv1, adv2, adv3, adv4 = 'OXLOR', 'HIVAR', 'ELOR', 'SUXGROG'            # False
#adv1, adv2, adv3, adv4 = 'krakrerlig', 'GUCAM', 'SUXGROG', 'kodearen'    # True
#adv1, adv2, adv3, adv4 = 'yarnithra', 'krakrerlig', 'jandreda', 'TOVIR'   # True

# write here
a1 = adv1.isupper()
a2 = adv2.isupper()
a3 = adv3.isupper()
a4 = adv4.isupper()

print(not(a1 == a2 == a3 == a4))
False
```

</div>

[50] :

```
adv1, adv2, adv3, adv4 = 'gimli', 'savorlim', 'glazouc', 'hondouni'      # False
#adv1, adv2, adv3, adv4 = 'OXLOR', 'HIVAR', 'ELOR', 'SUXGROG'            # False
#adv1, adv2, adv3, adv4 = 'krakrerlig', 'GUCAM', 'SUXGROG', 'kodearen'    # True
#adv1, adv2, adv3, adv4 = 'yarnithra', 'krakrerlig', 'jandreda', 'TOVIR'   # True

# write here
```

¹³⁸ <https://www.fantasynamewgenerators.com/giant-names.php>

¹³⁹ https://www.fantasynamewgenerators.com/dwarf_names.php

Continue

Go on reading notebook Strings 4 - search methods¹⁴⁰

[]:

5.2.4 Strings 4 - search methods

Download exercises zip

Browse files online¹⁴¹

Strings provide methods to search and transform them into new strings, but beware: the power is nothing without control! Sometimes you will feel the need to use them, and they might even work with some small example, but often they hide traps you will regret falling into. So whenever you write code with one of these methods, **always ask yourself the questions we will stress**.

WARNING: ALL string methods ALWAYS generate a NEW string

The original string object is NEVER changed (strings are immutable).

Method	Result	Meaning
<code>str1.strip(str2)</code>	str	Remove strings from the sides
<code>str1.lstrip(str2)</code>	str	Remove strings from left side
<code>str1.rstrip(str2)</code>	str	Remove strings from right side
<code>str1.count(str2)</code>	int	Count the number of occurrences of a substring
<code>str1.find(str2)</code>	int	Return the first position of a substring starting from the left
<code>str1.rfind(str2)</code>	int	Return the first position of a substring starting from the right
<code>str1.replace(str2, str3)</code>	str	Substitute substrings

Note: the list is not exhaustive, here we report only the ones we use in the book. For the full list see Python documentation¹⁴²

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
strings
  strings1.ipynb
  strings1-sol.ipynb
  strings2.ipynb
  strings2-sol.ipynb
  strings3.ipynb
  strings3-sol.ipynb
  strings4.ipynb
  strings4-sol.ipynb
  strings5-chal.ipynb
  jupman.py
```

¹⁴⁰ <https://en.softpython.org/strings/strings4-sol.html>

¹⁴¹ <https://github.com/DavidLeoni/softpython-en/tree/master/strings>

¹⁴² <https://docs.python.org/3/library/stdtypes.html#string-methods>

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `strings3.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

strip method

Eliminates white spaces, tabs and linefeeds from the *sides* of the string. In general, this set of characters is called *blanks*.

NOTE: it does NOT removes *blanks* inside string words! It only looks on the sides.

```
[2]: x = '\t\n\n\t carpe diem \t' # we put white space, tab and line feeds at the
      ↪sides
```

```
[3]: x
```

```
[3]: '\t\n\n\t carpe diem \t'
```

```
[4]: print(x)
```

```
carpe diem
```

```
[5]: len(x) # remember that special characters like \t and \n occupy 1 character
```

```
[5]: 20
```

```
[6]: y = x.strip()
```

```
[7]: y
```

```
[7]: 'carpe diem'
```

```
[8]: print(y)
```

```
carpe diem
```

```
[9]: len(y)
```

```
[9]: 10
```

```
[10]: x # IMPORTANT: x is still associated to the old string !
```

```
[10]: '\t\n\n\t carpe diem \t '
```

Specifying character to strip

If you only want Python to remove some specific character, you can specify them in parenthesis. Let's try to specify only one:

```
[11]: 'salsa'.strip('s')      # note internal 's' is not stripped  
[11]: 'alsa'
```

If we specify two or more, Python removes all the characters it can find from the sides

Note the order in which you specify the characters does **not** matter:

```
[12]: 'caustic'.strip('aci')  
[12]: 'ust'
```

WARNING: If you specify characters, Python doesn't try anymore to remove blanks!

```
[13]: 'bouquet '.strip('b')    # it won't strip right spaces !  
[13]: 'ouquet '
```

```
[14]: '\tbouquet '.strip('b')    # ... nor strip left blanks such as tab  
[14]: '\tbouquet '
```

According to the same principle, if you specify a space ' ', then Python will **only** remove spaces and won't look for other blanks!!

```
[15]: ' careful! \t'.strip(' ')    # strips only on the left!  
[15]: 'careful! \t'
```

QUESTION: for each of the following expressions, try to guess which result it produces (or if it gives an error):

1. '\tumultuous\n'.strip()

2. ' a b c '.strip()

3. '\ta\tb\t'.strip()

4. '\tMmm'.strip()

5. 'sky diving'.strip('sky')

6. 'anacondas'.strip('sad')

7. '\nno way '.strip(' ')

8. '\nno way '.strip('\\\\n')

9. '\nno way '.strip('\\n')

10. 'salsa'.strip('as')

11. '\\t ACE '.strip('\\t')

12. ' so what? '.strip("")

13. str(-3+1).strip("+-+--")

Exercise - Biblio bank

Your dream just became true: you were hired by the Cyber-Library! Since first enrolling to the Lunar Gymnasiuz in 2365 you've been dreaming of keeping and conveying the human knowledge collected through the centuries. You will have to check the work of an AI which reads and transcribes an interesting chronicle named **White Pages 2021**.

The Pages have lists of numbers in this format:

Name Surname Prefix-Suffix

Alas, the machine is buggy and in each row inserts some *blank* characters (spaces, control characters like \t and \n, ...)

- sometimes it warms the mobile printhead, causing the reading of numerous *blank* before the test
- sometimes the AI is so impressed by the content it forgets to turn off the reading, adding some *blank* at the end

Instead, it should produce a string with an initial dash and a final dot:

- Name Surname Prefix-Suffix .

Write some code to fix the bungled AI work.

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[16] :

```
row = '      \t  \n  Mario Rossi 0323-454345 \t \t  ' # - Mario Rossi 0323-454345.
#row = '      Ernesto Spadafesso 0323-454345  \n'          # - Ernesto Spadafesso 0323-
˓→454345.
#row = '      Gianantonio Marcolina Carla Napoleone 0323-454345 \t'
#row = '\nChiara Ermellino 0323-454345  \n \n'
#row = '      \tGiada Pietraverde 0323-454345\n\t'

# write here
product = ' - ' + row.strip() + '.'
print(product)

- Mario Rossi 0323-454345.
```

</div>

[16] :

```
row = '      \t  \n  Mario Rossi 0323-454345 \t \t  ' # - Mario Rossi 0323-454345.
```

(continues on next page)

(continued from previous page)

```
#row = ' Ernesto Spadafesso 0323-454345 \n'           # - Ernesto Spadafesso 0323-
#row = ' Gianantonio Marcolina Carla Napoleone 0323-454345 \t'
#row = '\nChiara Ermellino 0323-454345 \n \n'
#row = ' \tGiada Pietraverde 0323-454345\n\t'

# write here
```

lstrip method

Eliminates white spaces, tab and line feeds from *left side* of the string.

NOTE: does NOT remove *blanks* between words of the string! Only those on left side.

```
[17]: x = '\n \t the street \t '
```

```
[18]: x
```

```
[18]: '\n \t the street \t '
```

```
[19]: len(x)
```

```
[19]: 17
```

```
[20]: y = x.lstrip()
```

```
[21]: y
```

```
[21]: 'the street \t '
```

```
[22]: len(y)
```

```
[22]: 13
```

```
[23]: x      # IMPORTANT: x is still associated to the old string !
```

```
[23]: '\n \t the street \t '
```

rstrip method

Eliminates white spaces, tab and line feeds from *left side* of the string.

NOTE: does NOT remove *blanks* between words of the string! Only those on right side.

```
[24]: x = '\n \t the lighthouse \t '
```

```
[25]: x
```

```
[25]: '\n \t the lighthouse \t '
```

```
[26]: len(x)
```

```
[26]: 21
[27]: y = x.rstrip()
[28]: y
[28]: '\n \t the lighthouse'
[29]: len(y)
[29]: 18
[30]: x      # IMPORTANT: x is still associated to the old string !
[30]: '\n \t the lighthouse \t '
```

Exercise - Bad to the bone

You have an uppercase string `s` which contains at the sides some stuff you want to remove: punctuation, a lowercase char and some blanks. Write some code to perform the removal

Example - given:

```
char = 'b'
punctuation = '!?.;,'
s = ' \t\n...bbbbbbAD TO THE BONE\n! '
```

your code should show:

```
'BAD TO THE BONE'
```

- use only `strip` (or `lstrip` and `rstrip`) methods (if necessary, you can do repeated calls)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[31]: char = 'b'
punctuation = '!?.;,'
s = ' \t\n...bbbbbbAD TO THE BONE\n! '

# write here
s.strip().strip(char + punctuation).strip()
```

```
[31]: 'BAD TO THE BONE'
```

</div>

```
[31]: char = 'b'
punctuation = '!?.;,'
s = ' \t\n...bbbbbbAD TO THE BONE\n! '

# write here
```

```
[31]: 'BAD TO THE BONE'
```

count method

The method `count` takes a substring and counts how many occurrences are there in the string before the dot.

```
[32]: "astral stars".count('a')
```

```
[32]: 3
```

```
[33]: "astral stars".count('A')      # it's case sensitive
```

```
[33]: 0
```

```
[34]: "astral stars".count('st')
```

```
[34]: 2
```

Optionally, you can pass two other parameters to indicate an index to start counting from (included) and where to end (excluded):

```
[35]: #012345678901  
"astral stars".count('a', 4)
```

```
[35]: 2
```

```
[36]: #012345678901  
"astral stars".count('a', 4, 9)
```

```
[36]: 1
```

Do not abuse count

WARNING: `count` is often used in a wrong / inefficient ways

Always ask yourself:

1. Could the string contain duplicates? Remember they will get counted!
2. Could the string contain *no* duplicate? Remember to also handle this case!
3. `count` performs a search on all the string, which could be inefficient: is it really needed, or do we already know the interval where to search?

Exercise - astro money

During 2020 lockdown, while looking at the stars above you started feeling... waves. After some thinking, you decided *THEY* wanted to communicate with you so you set up a dish antenna on your roof to receive messages from aliens. After months of apparent irrelevant noise, one day you finally receive a message you're able to translate. Aliens are *obviously* trying to tell you the winning numbers of lottery!

A message is a sequence of exactly 3 *different* character repetitions, the number of characters in each repetition is a number you will try at the lottery. You frantically start developing the translator to show these lucky numbers on the terminal.

Example - given:

```
s = '$$$$$EEEE!!'
```

it should print:

```
$ € !
4 5 2
```

- **IMPORTANT:** you can assume all sequences have ***different*** characters
- **DO NOT** use cycles nor comprehensions
- for simplicity assume each character sequence has at most 9 repetitions

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution

<div class="jupman-sol jupman-sol-code" style="display:none">

```
[37]: #01234567890      # $ € !
s = '$$$$$EEEE!!'      # 4 5 2

                                # I M Q
#s = 'IIIMMMMMQQQ'      # 3 6 3

                                # H A L
#s = 'HAL'                # 1 1 1

# write here
p1 = 0
d1 = s.count(s[p1])
p2 = p1 + d1
d2 = s.count(s[p2])
p3 = p2 + d2
d3 = s.count(s[p3])

print(s[p1], s[p2], s[p3])
print(d1, d2, d3)
```

```
$ € !
4 5 2
```

</div>

```
[37]: #01234567890      # $ € !
s = '$$$$$EEEE!!'      # 4 5 2

                                # I M Q
#s = 'IIIMMMMMQQQ'      # 3 6 3

                                # H A L
#s = 'HAL'                # 1 1 1

# write here
```

```
$ € !
4 5 2
```

find method

find returns the index of the *first* occurrence of some given substring:

```
[38]: #0123456789012345  
'bingo bongo bong'.find('ong')  
[38]: 7
```

If no occurrence is found, it returns -1:

```
[39]: #0123456789012345  
'bingo bongo bong'.find('bang')  
[39]: -1  
  
[40]: #0123456789012345  
'bingo bongo bong'.find('Bong')      # case-sensitive  
[40]: -1
```

Optionally, you can specify an index from where to start searching (included):

```
[41]: #0123456789012345  
'bingo bongo bong'.find('ong', 10)  
[41]: 13
```

And also where to end (excluded):

```
[42]: #0123456789012345  
'bingo bongo bong'.find('g', 4, 9)  
[42]: -1
```

rfind method

Like *find method*, but search starts from the right.

Do not abuse find

WARNING: find is often used in a wrong / inefficient ways

Always ask yourself:

1. Could the string contain duplicates? Remember only the *first* will be found!
2. Could the string *not* contain the search substring? Remember to also handle this case!
3. find performs a search on all the string, which could be inefficient: is it really needed, or do we already know the interval where to search?
4. If we want to know if a character is in a position we already know, find is useless: it's enough to write `my_string[3] == character`. If you used find, it could discover duplicate characters which are *before* or *after* the one we are interested in!

Exercise - The port of Monkey Island

Monkey Island has a port with 4 piers where ships coming from all the archipelago are docked. The docking point is never precise, and there could arbitrary spaces between the pier borders. The could also be duplicated ships.

- 1) Suppose each pier can only contain one ship, and we want to write some code which shows True if "The Jolly Rasta" is docked to the pier 2, or False otherwise.

Have a look at the following ports, and for each one of them try to guess whether or not the following code lines produce correct results. Try then writing some code which doesn't have the problems you will encounter.

- **DO NOT** use if instructions, loops nor comprehensions
- **DO NOT** use lists (so no split)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[43]: width = 21 # width of a pier, INCLUDED the right `/`
pier = 2

# piers      : 1           2           3           4
port =      "The Mad Monkey     |  The Jolly Rasta   |  The Sea Cucumber | LeChuck
         ↪'s Ghost Ship|"
#port =      "  The Mad Monkey    /           |  The Sea Cucumber | LeChuck
         ↪'s Ghost Ship|"
#port =      "    The Mad Monkey  /The Jolly Rasta   |  The Sea Cucumber |       ↴
         ↪   /"
#port =      "The Jolly Rasta    /           |           The Sea Cucumber/LeChuck
         ↪'s Ghost Ship|"
#port =      "                  / The Mad Monkey   |  The Jolly Rasta | LeChuck
         ↪'s Ghost Ship|"
#port =      "      The Jolly Rasta /           |  The Jolly Rasta | The
         ↪Jolly Rasta |"

print('Is Jolly Rasta docked to pier', pier, '?')
print()
print(port)

print()
print('          in:', 'The Jolly Rasta' in port)

print()
print('      find on everything:', port.find('The Jolly Rasta') != -1)

print()
print('  find since second pier:', port.find('The Jolly Rasta', width*(pier-1)) != -1)

# write here
print()
sub = port[width*(pier-1):width*pier-1]
print('          Solution:', sub.find('The Jolly Rasta') != -1)

Is Jolly Rasta docked to pier 2 ?

The Mad Monkey     |  The Jolly Rasta   |  The Sea Cucumber | LeChuck's Ghost Ship|
                           in: True
```

(continues on next page)

(continued from previous page)

```
    find on everything: True  
    find since second pier: True  
        Solution: True
```

</div>

```
[43]: width = 21 # width of a pier, INCLUDED the right `/`  
pier = 2  
  
# piers : 1 2 3 4  
port = "The Mad Monkey | The Jolly Rasta | The Sea Cucumber | LeChuck  
→'s Ghost Ship|"  
#port = " The Mad Monkey | | The Sea Cucumber | LeChuck  
→'s Ghost Ship|"  
#port = " The Mad Monkey |The Jolly Rasta | The Sea Cucumber |  
→ |"  
#port = "The Jolly Rasta | | The Sea Cucumber|LeChuck  
→'s Ghost Ship|"  
#port = " | The Mad Monkey | The Jolly Rasta |LeChuck  
→'s Ghost Ship|"  
#port = " The Jolly Rasta | | The Jolly Rasta |  
→Jolly Rasta |"  
  
print('Is Jolly Rasta docked to pier', pier, '?')  
print()  
print(port)  
  
print()  
print('           in:', 'The Jolly Rasta' in port)  
  
print()  
print('   find on everything:', port.find('The Jolly Rasta') != -1)  
  
print()  
print(' find since second pier:', port.find('The Jolly Rasta', width*(pier-1)) != -1)  
  
# write here
```

Is Jolly Rasta docked to pier 2 ?

```
The Mad Monkey | The Jolly Rasta | The Sea Cucumber | LeChuck's Ghost Ship|  
           in: True  
   find on everything: True  
find since second pier: True  
        Solution: True
```

- 2) Suppose now every pier can dock more then one ship, even with the same name. Write some code which shows True if **only one** Grog Ship is docked to the second pier, False otherwise

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[44]: width = 21 # width of a pier, INCLUDED the right `/`  
pier = 2  
  
# piers : 1 2 3 4  
port = "The Mad Monkey | The Jolly Rasta | The Sea Cucumber | LeChuck  
→'s Ghost Ship!"  
#port = "The Mad Monkey | Grog Ship Grog Ship| The Jolly Rasta | The  
→Sea Cucumber "  
#port = " The Jolly Rasta | Grog Ship | The Jolly Rasta | The  
→Jolly Rasta "  
#port = " Grog Ship | Grog Ship | LeChuck's Ghost Ship| Grog  
→Ship "  
#port = "LeChuck's Ghost Ship/  
→Jolly Rasta "  
#port = "The Jolly Rasta | Grog Ship Grog Ship| Grog Ship | The  
→Jolly Rasta "  
  
print()  
print('Is only one Grog Ship docked to pier', pier, '?')  
print()  
  
# write here  
  
sub = port[width*(pier-1):width*pier-1]  
print('Solution Grog Ship:', sub.count('Grog Ship') == 1)
```

Is only one Grog Ship docked to pier 2 ?
Solution Grog Ship: False

</div>

```
[44]: width = 21 # width of a pier, INCLUDED the right `/`  
pier = 2  
  
# piers : 1 2 3 4  
port = "The Mad Monkey | The Jolly Rasta | The Sea Cucumber | LeChuck  
→'s Ghost Ship!"  
#port = "The Mad Monkey | Grog Ship Grog Ship| The Jolly Rasta | The  
→Sea Cucumber "  
#port = " The Jolly Rasta | Grog Ship | The Jolly Rasta | The  
→Jolly Rasta "  
#port = " Grog Ship | Grog Ship | LeChuck's Ghost Ship| Grog  
→Ship "  
#port = "LeChuck's Ghost Ship/  
→Jolly Rasta "  
#port = "The Jolly Rasta | Grog Ship Grog Ship| Grog Ship | The  
→Jolly Rasta "  
  
print()  
print('Is only one Grog Ship docked to pier', pier, '?')  
print()  
  
# write here
```

(continues on next page)

(continued from previous page)

```
Is only one Grog Ship docked to pier 2 ?
```

```
Solution Grog Ship: False
```

Exercise - bananas

While exploring a remote tropical region, an ethologist discovers a population of monkeys which appear to have some concept of numbers. They collect bananas in the hundreds which are then traded with coconuts collected by another group. To communicate the quantities of up to 999 bananas, they use a series of exactly three guttural sounds. The ethologist writes down the sequencies and formulates the following theory: each sound is comprised by a sequence of the same character, repeated a number of times. The number of characters in the first sequence is the first digit (the hundreds), the number of characters in the second sequence is the second digit (the decines), while the last sequence represents units.

Write some code which puts in variable `bananas` **an integer** representing the number.

For example - given:

```
s = 'bb bbbb aaaa'
```

your code should print:

```
>>> bananas
254
>>> type(bananas)
int
```

- **IMPORTANT 1: different sequences may use the *same* character!**
- **IMPORTANT 2: you cannot assume which characters monkeys will use:** you just know each digit is represented by a repetition of the same character
- **DO NOT** use cycles nor comprehensions
- the monkeys have no concept of zero

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[45] :

```
#0123456789012
s = 'bb bbbb aaaa'      # 254
#s = 'ccc cc ccc'      # 323
#s = 'vvv rrrr ww'      # 342
#s = 'cccc h jjj'      # 413
#s = '呵呵 呵呵呵 呵呵呵' # 364 (you could get *any* weird character, also unicode ...)

# write here
p1 = s.find(' ')
bananas = len(s[:p1])*100
p2 = s.find(' ', p1+1)
bananas += len(s[p1+1:p2])*10
bananas += len(s[p2+1:])*1
```

(continues on next page)

(continued from previous page)

```

print('The bananas are',bananas)
type(bananas)

The bananas are 254
[45]: int

</div>

[45]:
    #0123456789012
s = 'bb bbbb aaaa'      # 254
#s = 'ccc cc ccc'      # 323
#s = 'vvv rrrr ww'      # 342
#s = 'cccc h jjj'       # 413
#s = '﷽ ﻭ ﻮ ﻮ ﻮ ﻮ ﻮ ﻮ' # 364 (you could get *any* weird character, also unicode ...)

# write here

```

replace method

`str.replace` takes two strings and looks in the string on which the method is called for occurrences of the first string parameter, which are substituted with the second parameter. Note it gives back a NEW string with all substitutions performed.

Example:

```

[46]: "the train runs off the tracks".replace('tra', 'ra')
[46]: 'the rain runs off the racks'

[47]: "little beetle".replace('tle', '')
[47]: 'lit bee'

[48]: "talking and joking".replace('ING', 'ed')  # it's case sensitive
[48]: 'talking and joking'

[49]: "TALKING AND JOKING".replace('ING', 'ED')  # here they are
[49]: 'TALKED AND JOKED'

```

As always with strings, `replace` DOES NOT modify the string on which it is called:

```

[50]: x = "On the bench"

[51]: y = x.replace('bench', 'bench the goat is alive')

[52]: y
[52]: 'On the bench the goat is alive'

```

```
[53]: x # IMPORTANT: x is still associated to the old string !
[53]: 'On the bench'
```

If you give an optional third argument count, only the first count occurrences will be replaced:

```
[54]: "TALKING AND JOKING AND LAUGHING".replace('ING', 'ED', 2) # replaces only first 2 occurrences
[54]: 'TALKED AND JOKED AND LAUGHING'
```

QUESTION: for each of the following expressions, try to guess which result it produces (or if it gives an error)

1. '\$feat the rich\$'.replace('£', '').replace('\$', '')
2. '\$feat the rich\$'.strip('£').strip('\$')

Do not abuse replace

WARNING: `replace` is often used in a wrong / inefficient ways

Always ask yourself:

1. Could the string contain duplicates? Remember they will *all* get substituted!
2. `replace` performs a search on the whole string, which could be inefficient: is it really needed, or do we already know the interval where the text to substitute is?

Exercise - Do not open that door

QUESTION You have a library of books, with labels like C-The godfather, R-Pride and prejudice o 'H-Do not open that door' composed by a character which identifies the type (C crime, R romance, H horror) followed by a – and the title. Given a book, you want to print the complete label, a colon and then the title, like 'Crime: The godfather'. Look at the following code fragments, and for each try writing labels among the proposed ones **or create others** which would give wrong results (if they exists).

```
book = 'C-The godfather'
book = 'R-Pride and prejudice'
book = 'H-Do not open that door'
```

1. `book.replace('C', 'Crime: ').replace('R', 'Romance: ')`
2. `book[0].replace('C', 'Crime: ') \ .replace('H', 'HORROR: ') \ .replace('R', 'Romance: ') + book[2:]`
3. `book.replace('C-', 'Crime: ').replace('R-', 'Romance: ')`
4. `book.replace('C-', 'Crime: ', 1).replace('R-', 'Romance: ', 1)`

5. book[0:2].replace('C-', 'Crime: ').replace('R-', 'Romance: ') + book[2:]

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol" jupman-sol-code" style="display:none">

[55]: # SOLUTION

```
#1
book = 'R-Clarissa'
print(book.replace('C', 'Crime: ').replace('R', 'Romance: '))

#2
book = 'H-Do not open that door'
print(book[0].replace('C', 'Crime: ').replace('H', 'HORROR: ').replace('R', 'Romance: '))
→) + book[2:])

#3
book = 'R-C-U-SooN'
print(book.replace('C-', 'Crime: ').replace('R-', 'Romance: '))

#4
book = 'C-T-H-E-R-O-B-B-E-R-Y'
print(book.replace('C-', 'Crime: ', 1).replace('R-', 'Romance: ', 1))

#5 This is quite robust, IF we assume the categories are fixed and DON'T contain
→dashes
# for example an evil category could be C- expanded to C-U-L-I-N-A-R-Y (which would
→contain R-)
#print(book[0:2].replace('C-', 'Crime: ').replace('R-', 'Romance: ').replace('H-',
→'Horror: ') + book[2:])
```

Romance: -Crime: larissa
 HORomance: Romance: ORomance: : Do not open that door
 Romance: Crime: U-SooN
 Crime: T-H-E-Romance: O-B-B-E-R-Y

</div>

[55]:

Exercise - The Kingdom of Stringards

Characters Land is ruled with the iron fist by the Dukes of Stringards. The towns managed by them are monodimensional, and can be represented as a string, hosting dukes d, lords s, vassals v and peasants p. To separate the various social circles from improper mingling, some walls |mm| have been erected.

Unfortunately, the Dukes are under siege by the tribe of the hideous Replacerons: with their short-sighted barbarian ways, they are very close to destroy the walls. To defend the town, the Stringards decide to upgrade walls, trasforming them from |mm| to |MM|.

- **DO NOT** use loops nor list comprehensions
- **DO NOT** use lists (so no split)

Stringards I: upgrading all the walls

Example - given:

```
town = 'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
```

after your code, it must result:

```
>>> town  
'ppp / MM | vvvvvv / MM | sss / MM | dd / MM | sssss / MM | vvvvvv / MM | pppppp'
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
```

```
data-jupman-show="Show solution"
```

```
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[56] :

```
town =      'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'  
# result:  'ppp / MM | vvvvvv / MM | sss / MM | dd / MM | sssss / MM | vvvvvv / MM | pppppp'  
  
# write here  
  
town = town.replace(' | mm | ', ' | MM | ')  
print(town)
```

```
ppp | MM | vvvvvv | MM | sss | MM | dd | MM | sssss | MM | vvvvvv | MM | pppppp
```

```
</div>
```

[56] :

```
town =      'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'  
# result:  'ppp / MM | vvvvvv / MM | sss / MM | dd / MM | sssss / MM | vvvvvv / MM | pppppp'  
  
# write here
```

Stringards II: Outer walls

Alas, the paesants don't work hard enough and there aren't enough coins to upgrade all the walls: upgrade **only the outer walls**

- **DO NOT** use `if`, loops nor list comprehensions
- **DO NOT** use lists (so no split)

Example - given:

```
town = 'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
```

after your code, it must result:

```
>>> town  
'ppp | MM | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | MM | pppppp'
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
```

```
data-jupman-show="Show solution"
```

```
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[57]:

```

town = 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|ssssss|mm|vvvvvv|mm|pppppp'
#result: 'ppp/MM|vvvvvv/mm|sss/mm|dd/mm|ssssss/mm|vvvvvv/MM|pppppp'
#town = '/mm|vvvvvv/mm|/mm|ddddd/mm|ssvvv/mm|pp'
#result: '/MM|vvvvvv/mm|/mm|ddddd/mm|ssvvv/MM|pp'

# write here

i = town.find('|mm|')
town = town[:i] + '|MM|' + town[i+4:]
i = town.rfind('|mm|')
town = town[:i] + '|MM|' + town[i+4:]

print(town)
ppp|MM|vvvvvv|mm|sss|mm|dd|mm|ssssss|mm|vvvvvv|MM|pppppp

```

</div>

[57]:

```

town = 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|ssssss|mm|vvvvvv|mm|pppppp'
#result: 'ppp/MM|vvvvvv/mm|sss/mm|dd/mm|ssssss/mm|vvvvvv/MM|pppppp'
#town = '/mm|vvvvvv/mm|/mm|ddddd/mm|ssvvv/mm|pp'
#result: '/MM|vvvvvv/mm|/mm|ddddd/mm|ssvvv/MM|pp'

# write here

```

Stringards III: Power to the People

An even greater threat plagues the Stringards: *democracy*.

Following the spread of this dark evil, some cities developed right and left factions, which tend to privilege only some parts of the city. If the dominant sentiment in a city is lefty, all the houses to the left of the Duke are privileged with big gold coins, otherwise with righty sentiment houses to the right get more privileged. When a house is privileged, the corresponding character is upgraded to capital.

- assume that at least a block with d is always present, and it is unique
- **DO NOT** use if, loops nor list comprehensions
- **DO NOT** use lists (so no split)

3.1) privilege only left houses

```
town = 'ppp|mm|vvvvvv|mm|sss|mm|dd|mm|ssssss|mm|vvvvvv|mm|pppppp'
```

after your code, it must result:

```
>>> town
'PPP|mm|VVVVVV|mm|SSS|mm|dd|mm|ssssss|mm|vvvvvv|mm|pppppp'
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[58]: town =      'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
# result: 'PPP | mm | VVVVVV | mm | SSS | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
#town =      '/p /ppp | /p /pp | mm | vvv | vvvv | mm | sssss | mm | ddd | mm | sssss | ss | mm | vvvvvv | mm | '
# result: '/P /PPP | /P /PP | mm | VVV | VVVV | mm | SSSSS | mm | ddd | mm | sssss | ss | mm | vvvvvv | mm | '

# write here

dpos = town.find('d')
town = town[:dpos].replace('p', 'P').replace('v', 'V').replace('s', 'S') + town[dpos:]

town
```

[58]: 'PPP | mm | VVVVVV | mm | SSS | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'

</div>

```
[58]: town =      'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
# result: 'PPP | mm | VVVVVV | mm | SSS | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
#town =      '/p /ppp | /p /pp | mm | vvv | vvvv | mm | sssss | mm | ddd | mm | sssss | ss | mm | vvvvvv | mm | '
# result: '/P /PPP | /P /PP | mm | VVV | VVVV | mm | SSSSS | mm | ddd | mm | sssss | ss | mm | vvvvvv | mm | '

# write here
```

3.2) privilege only right houses

Example - given:

```
town = 'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
```

after your code, it must result:

```
>>> town
'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | SSSSS | mm | VVVVVV | mm | PPPPPP'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[59]: town =      'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
#result: 'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | SSSSS | mm | VVVVVV | mm | PPPPPP'
#town =      '/p /ppp | /p /pp | mm | vvv | vvvv | mm | sssss | mm | ddd | mm | sssss | ss | mm | vvvvvv | p /pp | mm | '
#result: '/p /ppp | /p /pp | mm | vvv | vvvv | mm | sssss | mm | ddd | mm | SSSSS | SS | mm | VVVVVV | P /PP | mm | '

# write here
dpos = town.rfind('d')
town = town[:dpos] + town[dpos:].replace('p', 'P').replace('v', 'V').replace('s', 'S')

town
```

[59]: 'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | SSSSS | mm | VVVVVV | mm | PPPPPP'

</div>

[59]:

```

town =      'ppp | mm | vvvvvv | mm | sss | mm | dd | mm | sssss | mm | vvvvvv | mm | pppppp'
#result: 'ppp / mm / vvvvvv / mm / sss / mm / dd / mm / SSSSS / mm / VVVVVV / mm / PPPPPP'
#town =      '/p / ppp / /p / pp / mm / vvv / vvvv / mm / sssss / mm / ddd / mm / sssss / ss / mm / vvvvvv / p / pp / mm / '
#result: '/p / ppp / /p / pp / mm / vvv / vvvv / mm / sssss / mm / ddd / mm / SSSS / SS / mm / VVVVVV / P / PP / mm / '

# write here

```

Stringards IV: Power struggle

Over time, the Dukes family has expanded and alas ruthless feuds occurred. According to the number of town people to the left/right of the dukes, a corresponding number of royal members to the left/right receives support for playing their power games. A member of the dukes palace who receives support becomes uppercase. Each character 'p', 'v' or 's' contributes support (but not the walls). The royal members who are not reached by support are slaughtered by their siblings, and substituted with a Latin Cross Unicode¹⁴³ †

- assume at least a block of d is always present, and it is unique
- assume that for each left/right house, there is *at least* a left/right duke

Example - given:

```
town = "ppp | mm | vv | mm | v | s | mm | dddddddddd | mm | ss | mm | vvvvv | mm | pppp";
```

After your code, it must print:

```

Members of the royal family:24
    left:7
    right:11

After the deadly struggle, the new town is

ppp | mm | vv | mm | v | s | mm | DDDDDDDDDDDDDDDDDDDDDDDDDDDDD | mm | ss | mm | vvvvv | mm | pppp

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[60]:

```

town =      'ppp | mm | vv | mm | v | s | mm | dddddddddd | mm | ss | mm | vvvvv | mm | pppp'
#result: 'ppp / mm / vv / mm / v / s / mm / DDDDDDDDDDDDDDDDDDDDDDDDDDD / mm / ss / mm / vvvvv / mm / pppp'   tot:
↪24 sx:7 dx:11
#town =      'ppp / mm / ppp / mm / vv / mm / ss / mm / dddddd | mm / ss / mm / mm / s / v / mm / p / p / '
#result: 'ppp / mm / ppp / mm / vv / mm / ss / mm / DDDDDDDDDDDDDDDDDDDDDDDDD / mm / ss / mm / mm / s / v / mm / p / p / ' tot:
↪20 sx:10 dx:6

# write here

d = town.count('d')

print('Members of the royal family:', d)

dpos_sx = town.find('d')

```

(continues on next page)

¹⁴³ <https://www.compart.com/en/unicode/U+271D>

(continued from previous page)

```
c_sx = town.count('p', 0, dpos_sx) + town.count('v', 0, dpos_sx) + town.count('s', 0, ↵dpos_sx)
print('left:', c_sx)

dpos_dx = town.rfind('d')
c_dx = town.count('p', dpos_dx) + town.count('v', dpos_dx) + town.count('s', dpos_dx)
print('right:', c_dx)

town = town[:dpos_sx] + 'D'*c_sx + 'D'*(d-c_sx-c_dx) + 'D'*c_dx + town[dpos_dx+1:]

print()
print('After the deadly struggle, the new town is:')
print()
print(town)

Members of the royal family: 24
    left: 7
    right: 11

After the deadly struggle, the new town is:

ppp|mm|vv|mm|v|s|mm|DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD|mm|ss|mm|vvvvv|mm|pppp
```

</div>

[60]:

```
town = 'ppp|mm|vv|mm|v|s|mm|ddddd|mm|ss|mm|vvvvv|mm|pppp'
#result: 'ppp|mm|vv|mm|v|s|mm|DDDDDDDDDDDDDDDDDDDDDDDDDD|mm|ss|mm|vvvvv|mm|pppp' tot:
→24 sx:7 dx:11
#town = 'ppp|mm|ppp|mm|vv|mm|ss|mm|ddddd|mm|ss|mm|mm|s|v|mm|p|p| '
#result: 'ppp|mm|ppp|mm|vv|mm|ss|mm|DDDDDDDDDDDDDDDDDD|mm|ss|mm|mm|s|v|mm|p|p|' tot:
→20 sx:10 dx:6

# write here
```

Other exercises

QUESTION: For each following expression, try to find the result

1. `'gUrP'.lower() == 'GuRp'.lower()`
2. `'NaNo'.lower() != 'nAnO'.upper()`
3. `'o' + 'ortaggio'.replace('o','\t \n ').strip() + 'o'`
4. `'DaDo'.replace('D','b') in 'barbados'`

Continue

Go on reading notebook Strings 5 - first challenges¹⁴⁴

5.2.5 Strings 5 - First challenges

Download exercises zip

Browse file online¹⁴⁵

We now propose some exercises without solution, do you accept the challenge?

Challenge - a strange zoo

⊕ You are given a phrase and you know there are a couple strange char at the boundaries of the phrase. Write some code to fix the phrase so it doesn't has the extremities delimited by the first occurrences of char.

- **DO NOT** use loops

For example - given:

```
phrase = "There is za strange zoo nearby, with many animalsz you wouldn't believe."
```

after your code, it must result:

```
>>> phrase
'a strange zoo nearby, with many animals'
```

[1]:

```
char, phrase = "z", "There is za strange zoo nearby, with many animalsz you wouldn't believe." # 'a strange zoo nearby, with many animals'
#char, phrase = "Z", "Zthere is a Zorg in the ZooZ outside the neighborhood" # 'there is a Zorg in the Zoo'

# write here
```

Challenge - nuclear fusion

⊕ Given a phrase of words separated by spaces and a word, write some code to produce a string where all occurrences beginning with that word are substituted with sub

- phrase never begins with the word to substitute
- **DO NOT** use loops

[2]:

```
phrase = "it's clear nuclear fusion is the future - clearly, there is a lot of interest around it"
```

(continues on next page)

¹⁴⁴ <https://en.softpython.org/strings/strings5-chal.html>

¹⁴⁵ <https://github.com/DavidLeoni/softpython-en/strings>

(continued from previous page)

```
word, sub = "clear", "unclear" # "it's unclear nuclear fusion is the future -  
#unclearly, there is a lot of interest around it"  
  
#word, sub = "is", "can be"      # "it's clear nuclear fusion can be the future -  
#clearly, there can be a lot of interest around it"  
  
# write here
```

Challenge - gold

⊕⊕ You are given a string s which begins with a sequence of the same repeated character, then it has a treasure, and then continues with another sequence of another repeated character. Both initial and ending sequences have **unknown length**.

- assume the string has **at least three** characters
- assume the characters of starting sequence are **always different** from end sequence
- **DO NOT** use loops
- **DO NOT** write constant characters in your code (so no €, \$, ..)

[3]:

```
s = "*****gold---"    # gold  
#s = "////////gems!!!!!!!"  # gems  
#s = "-----€_____"      # €  
#s = "p$q"  
  
# write here
```

[]:

5.3 Lists

5.3.1 Lists 1 - Introduction

Download exercises zip

Browse files online¹⁴⁶

A Python list is a **mutable** sequence of heterogeneous elements, in which we can put the objects we want. The order in which we put them is preserved.

¹⁴⁶ <https://github.com/DavidLeoni/softpython-en/tree/master/lists>

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
lists
lists1.ipynb
lists1-sol.ipynb
lists2.ipynb
lists2-sol.ipynb
lists3.ipynb
lists3-sol.ipynb
lists4.ipynb
lists4-sol.ipynb
lists5-chal.ipynb
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook lists1.ipynb
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

Creating lists

We can create a list by specifying the elements it contains between square brackets, separating them with a comma.

For example, in this list we insert the numbers 7, 4 e 9:

```
[2]: [7, 4, 9]
```

```
[2]: [7, 4, 9]
```

Like all Python objects, we can associate them to a variable, in this case we create a new one we call my_list:

```
[3]: my_list = [7, 4, 9]
```

```
[4]: my_list
```

```
[4]: [7, 4, 9]
```

Let's see what happens in memory, and compare strings representation with lists representation:

```
[5]: # WARNING: before using the function jupman.pytut() which follows,
#           it is necessary to first execute this cell with Shift+Enter
#           it's sufficient to execute it only once
```

(continues on next page)

(continued from previous page)

```
import jupman
```

```
[6]: my_string = "prova"  
  
my_list = [7, 4, 9]  
  
jupman.pytut()  
  
[6]: <IPython.core.display.HTML object>
```

We suddenly note a relevant difference. The string remained in the azure region where associations among variables and values usually stay. From variable `my_list` we see instead an arrow departing to a new yellow memory region, which is created as soon the execution reaches the row where the list is defined.

Later we will analyze more in detail the consequences of this.

In a list the same elements may appear many times:

```
[7]: numbers = [1, 2, 3, 1, 3]  
  
[8]: numbers  
[8]: [1, 2, 3, 1, 3]
```

We can put any element, for example strings:

```
[9]: fruits = ["apple", "pear", "peach", "strawberry", "cherry"]  
  
[10]: fruits  
[10]: ['apple', 'pear', 'peach', 'strawberry', 'cherry']
```

We can also mix the object types contained in a list, for example we can have integers and strings:

```
[11]: mix = ["table", 4, "chair", 8, 5, 1, "chair"]
```

In Python Tutor it will be shown like this:

```
[12]: mix = ["table", 5, 4, "chair", 8, "chair"]  
  
jupman.pytut()  
  
[12]: <IPython.core.display.HTML object>
```

For convenience we can also write the list on many rows (the spaces in this case do not count, only remember to terminate rows with commas ,)

```
[13]: mix = ["table",  
           5,  
           4,  
           "chair",  
           8,  
           "chair"]
```

EXERCISE: try writing the list above WITHOUT putting a comma after the 5, which error appears?

```
[14]: # write here
```

Empty list

There are two ways to create an empty list.

- 1) with square brackets:

```
[15]: my_empty_list = []
```

```
[16]: my_empty_list
```

```
[16]: []
```

- 2) Or with `list()`:

```
[17]: another_empty_list = list()
```

```
[18]: another_empty_list
```

```
[18]: []
```

WARNING: When you create an empty list (independently from the used notation), a NEW region in memory is allocated to place the list.

Let's see what this means with Python Tutor:

```
[19]: a = []
b = []

jupman.pytut()
```

```
[19]: <IPython.core.display.HTML object>
```

Note two arrows appeared, which point to **different** memory regions. The same would have happened by initializing the lists with some elements:

```
[20]: la = [8, 6, 7]
lb = [9, 5, 6, 4]

jupman.pytut()
```

```
[20]: <IPython.core.display.HTML object>
```

We would have two lists in different memory regions also by placing identical elements inside the lists:

```
[21]: la = [8, 6, 7]
lb = [8, 6, 7]

jupman.pytut()
```

```
[21]: <IPython.core.display.HTML object>
```

Things get complicated when we start using assignment operations:

```
[22]: la = [8, 6, 7]
```

```
[23]: lb = [9, 5, 6, 4]
```

```
[24]: lb = la
```

By writing `lb = la`, we told Python to ‘forget’ the previous assignment of `lb` to `[9, 5, 6, 4]`, and instead to associate `lb` to the same value associated to `la`, that is `[8, 6, 7]`. Thus, in memory we will see an arrow departing from `lb` and arriving into `[8, 6, 7]`, and the memory region where the list `[9, 5, 6, 4]` was placed will be removed (won’t be associated to any variable anymore). Let’s see what happens with Python Tutor:

```
[25]: la = [8, 6, 7]
lb = [9, 5, 6, 4]
lb = la

jupman.pytut()
```

```
[25]: <IPython.core.display.HTML object>
```

Exercise - list swaps

Try swapping the lists associated to variables `la` and `lb` by using only assignments and **without creating new lists**. If you want, you can overwrite a third variable `lc`. Verify what happens with Python Tutor.

- your code must work for any value of `la`, `lb` and `lc`

Example - given:

```
la = [9, 6, 1]
lb = [2, 3, 4, 3, 5]
lc = None
```

After your code, it must result:

```
>>> print(la)
[2, 3, 4, 3, 5]
>>> print(lb)
[9, 6, 1]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[26]: la = [9, 6, 1]
lb = [2, 3, 4, 3, 5]
lc = None

# write here

lc = la
la = lb
lb = lc

print(la)
print(lb)
```

```
[2, 3, 4, 3, 5]
[9, 6, 1]
```

</div>

[26]:

```
la = [9, 6, 1]
lb = [2, 3, 4, 3, 5]
lc = None

# write here
```

Tables

A list can also contain other lists:

[27]:

```
table = [['a', 'b', 'c'], ['d', 'e', 'f']]
```

Typically, whenever we have structures like this, it's convenient to dispense them on many rows (it's not mandatory but improves clarity):

[28]:

```
table = [
    ['a', 'b', 'c'],           # start external big list
    ['d', 'e', 'f']           # internal list 1
]                                # internal list 2
                                # end external big list
```

[29]:

```
table
```

[29]:

```
[['a', 'b', 'c'], ['d', 'e', 'f']]
```

Let's see how it's shown in Python Tutor:

[30]:

```
table = [
    ['a', 'b', 'c'],
    ['d', 'e', 'f']
]

jupman.pytut()
```

[30]:

```
<IPython.core.display.HTML object>
```

As we previously said, in a list we can put the elements we want, so we can mix lists with different dimensions, strings, numbers and so on:

[31]:

```
so_much = [
    ['hello', 3, 'world'],
    'a string',
    [9, 5, 6, 7, 3, 4],
    8,
]
```

[32]:

```
print(so_much)
```

```
[['hello', 3, 'world'], 'a string', [9, 5, 6, 7, 3, 4], 8]
```

Let's see how it appears in Python Tutor:

```
[33]: so_much = [
    ['hello', 3, 'world'],
    'a string',
    [9, 5, 6, 7, 3, 4],
    8,
]

jupman.pytut()
```

[33]: <IPython.core.display.HTML object>

Question - list creation

Have a look at these two pieces of code. For each case, try thinking how they might be represented in memory and then verify with Python Tutor.

- could there be a difference?
- how many memory cells will be allocated in total?
- how many arrows will you see?

```
# first case
lb = [
    [8, 6, 7],
    [8, 6, 7],
    [8, 6, 7],
    [8, 6, 7],
]
```

```
# second case
la = [8, 6, 7]
lb = [
    la,
    la,
    la,
    la
]
```

```
[34]: # first case
lb = [
    [8, 6, 7],
    [8, 6, 7],
    [8, 6, 7],
    [8, 6, 7],
]
jupman.pytut()
```

[34]: <IPython.core.display.HTML object>

```
[35]: # second case
```

(continues on next page)

(continued from previous page)

```

la = [8, 6, 7]
lb = [
    la,
    la,
    la,
    la
]
jupman.pytut()
[35]: <IPython.core.display.HTML object>

```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: In the first case, we have a ‘big list’ associated to the variable `lb` which contains 4 sublists each of 3 elements. Each sublist is created as new, so in total in memory we end up with 4 cells of the big list `lb` + (4 sublists * 3 cells each) = 16 cells

In the second case we have instead always the ‘big list’ associated to the variable `lb` of 4 cells, but inside it contains some pointers to the same identical list `la`. So the total number of occupied cells is 4 cells of big list `lb` + (1 sublist * 3 cells) = 7 cells

</div>

Exercise - domino

In your neighborhood a super domino match is being held: since the first prize is a card to get 10 pies made my mythical Grandmother Severina you decide to put serious effort.

You start thinking about how to train and decide to start matching the tiles in the correct way:

```

tile1 = [1, 3]
tile3 = [1, 5]
tile2 = [3, 9]
tile5 = [9, 7]
tile4 = [8, 2]

```

Given these tiles, generate a list which will contain two lists: in the first one insert a possible sequence of chained tiles; in the second one put the tiles which were left excluded from the first one sequence.

Example:

```
[ [ [1, 3], [3, 9], [9, 7] ], [ [1, 5], [8, 2] ] ]
```

- DO NOT write numbers
- USE only lists of variables

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```

tile1 = [1, 3]
tile2 = [3, 2]
tile3 = [1, 5]
tile4 = [2, 4]
tile5 = [3, 3]

```

(continues on next page)

(continued from previous page)

```
tile6 = [5, 4]
tile7 = [1, 2]

# write here
sequence = [tile1, tile5, tile2, tile4]
remained = [tile3, tile6, tile7]
print([sequence, remained])

[[[1, 3], [3, 3], [3, 2], [2, 4]], [[1, 5], [5, 4], [1, 2]]]
```

</div>

[36]:

```
tile1 = [1, 3]
tile2 = [3, 2]
tile3 = [1, 5]
tile4 = [2, 4]
tile5 = [3, 3]
tile6 = [5, 4]
tile7 = [1, 2]

# write here
```

Exercise - create lists 2

Insert some values in the lists la, lb such that

```
print([[la,la],[lb,la]])
```

prints

```
[[[8, 4], [8, 4]], [[4, 8, 4], [8, 4]]]
```

- **Insert only NUMBERS**
- Observe in Python Tutor how arrows are represented

```
[37]: la = [] # insert numbers
lb = [] # insert numbers

print([[la,la],[lb,la]])

[[[], []], [[], []]]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[38]: # SOLUTION

```
la = [8, 4]
lb = [4, 8, 4]

print([[la,la],[lb,la]])
```

```
[[[8, 4], [8, 4]], [[4, 8, 4], [8, 4]]]
```

</div>

[38]:

Exercise - create lists 3

Insert some values as elements of the lists `la`, `lb` e `lc` such that

```
print([[lb,lb,[lc,la]],lc])
```

prints

```
[[[8, [7, 7]], [8, [7, 7]], [[8, 7], [8, 5]]], [8, 7]]
```

- **insert only NUMBERS or NEW LISTS OF NUMBERS**
- Observe in Python Tutor are arrows are represented

[39]:

```
la = [] # insert elements (numbers or lists of numbers)
lb = [] # insert elements (numbers or lists of numbers)
lc = [] # insert elements (numbers or lists of numbers)

print([[lb,lb,[lc,la]],lc])
[[[], [], [[], []]], []]
```

[Show solution](#)<div class="jupman-sol" data-jupman-code" style="display:none">

[40]: # SOLUTION

```
la = [8,5]
lb = [8,[7,7]]
lc = [8,7]

print([[lb,lb,[lc,la]],lc])
[[[8, [7, 7]], [8, [7, 7]], [[8, 7], [8, 5]]], [8, 7]]]
```

</div>

[40]:

Exercise - create lists 4

Insert some values in the lists `la`, `lb` such that

```
print([[la,lc,la], lb])
```

prints

```
[[[3, 2], [[3, 2], [8, [3, 2]]], [3, 2]], [8, [3, 2]]]
```

- **insert only NUMBERS or VARIABLES** `la,lb` or `lc`
- Observe in Python Tutor how arrows are represented

```
[41]: la = [] # insert numbers or variables la, lb, lc  
lb = [] # insert numbers or variables la, lb, lc  
lc = [] # insert numbers or variables la, lb, lc  
  
print([[la,lc,la], lb])  
[[[], [], []], []]
```

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol" jupman-sol-code" style="display:none">

```
[42]: # SOLUTION  
  
la = [3,2]  
lb = [8,la]  
lc = [la,lb]  
  
print([[la,lc,la], lb])  
[[[3, 2], [[3, 2], [8, [3, 2]]], [3, 2]], [8, [3, 2]]]
```

</div>

```
[42]:
```

Convert sequences into lists

`list` may also be used to convert any sequence into a NEW list. A sequence type we've already seen are strings, so we can check what happens when we use `list` like if it were a function, by passing a string as parameter:

```
[43]: list("train")  
[43]: ['t', 'r', 'a', 'i', 'n']
```

We obtained a list in which each element is made of a character from the original string.

What happens if we call instead `list` on another list?

```
[44]: list( [7,9,5,6] )  
[44]: [7, 9, 5, 6]
```

Apparently, nothing particular, we obtained a list with the same start elements. But is it really the same list? Let's have a better look with Python Tutor:

```
[45]: la = [7, 9, 5, 6]
lb = list( la )
jupman.pytut()
[45]: <IPython.core.display.HTML object>
```

We note a NEW memory region was created with the same elements of `la`.

Exercise - gulp

Given a string with mixed uppercase and lowercase characters, write some code which creates a list containing as first element a list with characters from the string lowercased and as second element a list containing all the uppercased characters

- your code must work with any string
- if you don't remember the string methods, look here¹⁴⁷

Example - given:

```
s = 'GuLp'
```

your code must print:

```
[['g', 'u', 'l', 'p'], ['G', 'U', 'L', 'P']]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[46]: s = 'GuLp'

# write here
print([list(s.lower()), list(s.upper())])
[['g', 'u', 'l', 'p'], ['G', 'U', 'L', 'P']]
```

</div>

```
[46]: s = 'GuLp'

# write here

[['g', 'u', 'l', 'p'], ['G', 'U', 'L', 'P']]
```

QUESTION: This code:

- produces an error or assigns something to `x` ?
- After its execution, how many lists remain in memory?
- Can we shorten it?

```
s = "marathon"
x = list(list(list(list(s))))
```

¹⁴⁷ <https://en.softpython.org/strings/strings3-sol.html>

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show answer"
  data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: The code assigns the list ['m', 'a', 'r', 'a', 't', 'h', 'o', 'n'] to variable x. The first time list(s) generates a NEW list ['m', 'a', 'r', 'a', 't', 'h', 'o', 'n']. Successive calls to list take as input the just generated list and keep creating NEW lists with the same identical content. Since no produced list except the last one is assigned to a variable, the intermediate ones are eliminated at the end of execution. We can thus safely shorten the code by writing:

```
s = "marathon"
x = list(s)
```

```
</div>
```

QUESTION: This code:

- produces an error or assigns something to x ?
- After its execution, how many lists remain in memory?

```
s = "chain"
a = list(s)
b = list(a)
c = b
x = list(c)
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show answer"
  data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: Only 3 lists remain in memory, each containing 6 cells. This time the lists persist in memory because they are associated to variables a, b and c. We have 3 and not 4 lists because in instruction c = b the c variable is associated to the same identical memory region associated as variable b

```
</div>
```

Exercise - garaga

Given

```
sa = "ga"
sb = "ra"
la = ['ga']
lb = list(la)
```

- Assign to lc a list built in such a way so that once printed produces:

```
>>> print(lc)
```

```
[[['g', 'a', 'r', 'a'], ['ga'], ['ga'], ['r', 'a', 'g', 'a']]]
```

- in Python Tutor, ALL the arrows must point to a different memory region

```
[47]: sa = "ga"
sb = "ra"
la = ['ga']
lb = list(la)
```

(continues on next page)

(continued from previous page)

```
# insert come code in the list
lc = []

print(lc)
jupman.pytut()

[]

[47]: <IPython.core.display.HTML object>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
    data-jupman-show="Show solution"
    data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[48]: # SOLUTION

```
sa = "ga"
sb = "ra"
la = ['ga']
lb = list(la)
lc = [list(sa + sb), list(la), list(lb), list(sb + sa) ]

print(lc)
jupman.pytut()
[['g', 'a', 'r', 'a'], ['ga'], ['ga'], ['r', 'a', 'g', 'a']]
```

[48]: <IPython.core.display.HTML object>

```
</div>
```

[48]:

```
[['g', 'a', 'r', 'a'], ['ga'], ['ga'], ['r', 'a', 'g', 'a']]
```

[48]: <IPython.core.display.HTML object>

Continue

Go on reading notebook Lists 2 - operators¹⁴⁸

[]:

5.3.2 Lists 2 - operators

Download exercises zip

Browse online files¹⁴⁹

There are several operators to manipulate lists. The following ones behave like the ones we've seen in strings:

¹⁴⁸ <https://en.softpython.org/lists/lists2-sol.html>

¹⁴⁹ <https://github.com/DavidLeoni/softpython-en/tree/master/lists>

Operator	Syntax	Result	Meaning
<i>length</i>	<code>len(lst)</code>	<code>int</code>	Return the list length
<i>index</i>	<code>list[int]</code>	<code>obj</code>	Reads/writes an element at the specified index
<i>slice</i>	<code>list[int:int]</code>	<code>list</code>	Extracts a sublist - return a NEW list
<i>membership</i>	<code>obj in list</code>	<code>bool</code>	Checks if an object is contained in the list
<i>concatenation</i>	<code>list + list</code>	<code>list</code>	Concatenates two lists - return a NEW list
<i>maximum</i>	<code>max(lst)</code>	<code>int</code>	Given a list of numbers, return the greatest one
<i>minimum</i>	<code>min(lst)</code>	<code>int</code>	Given a list of numbers, returns the smallest one
<i>sum</i>	<code>sum(lst)</code>	<code>int</code>	Given a list of numbers, sums all of them
<i>replication</i>	<code>list * int</code>	<code>list</code>	Replicates the list - return a NEW list
<i>equality</i>	<code>==, !=</code>	<code>bool</code>	Checks whether lists are equal or different

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
lists
lists1.ipynb
lists1-sol.ipynb
lists2.ipynb
lists2-sol.ipynb
lists3.ipynb
lists3-sol.ipynb
lists4.ipynb
lists4-sol.ipynb
lists5-chal.ipynb
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `lists2.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

Length of a list

A list is a sequence, and like any sequence you can use the function `len` to obtain the length:

```
[2]: a = [7, 5, 8]
```

```
[3]: len(a)
```

```
[3]: 3
```

```
[4]: b = [8, 3, 6, 4, 7]
```

```
[5]: len(b)
```

```
[5]: 5
```

If a list contains other lists, they count as single elements:

```
[6]: mixed = [
    [4, 5, 1],
    [8, 6],
    [7, 6, 0, 8],
]
```

```
[7]: len(mixed)
```

```
[7]: 3
```

WARNING: YOU CAN'T use `len` as a method

```
[3,4,2].len() # WRONG
```

EXERCISE: Try writing `[3, 4, 2].len()` here, which error appears?

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[8]: # write here
```

```
# [3,4,2].len()
```

</div>

```
[8]: # write here
```

EXERCISE: Try writing `[3, 4, 2].len` WITHOUT the round parenthesis at the end, which error appears?

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[9]: # write here
```

```
# [3,4,2].len
```

</div>

```
[9]: # write here
```

QUESTION: If `x` is some list, by writing:

```
len(len(x))
```

what do we get?

1. the length of the list
2. an error
3. something else

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show answer"
  data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 2: `len` wants a *sequence* as argument and gives back a *number*, so the internal call to `len(x)` produces a number which is given to the external `len` and at that point Python will complain it received a number instead of a sequence. Verify which error appears by writing `len(len(x))` down here.

```
</div>
```

```
[10]: # write code here
```

QUESTION: Look at this expression, without executing it. What does it produce?

```
[len([]), len([len(['a', 'b'])])]
```

1. an error (which one?)
2. a number (which one?)
3. a list (which one?)

Try writing the result by hand, and then compare it with the one obtained by executing the code in a cell.

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show answer"
  data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 3: the list `[0, 1]`

```
</div>
```

QUESTION: Look at this expression, without executing it. What does it produce?

```
len([[], [], [], [[], []], [[], []]])
```

1. an error (which one?)
2. a number (which one?)
3. a list (which one?)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show answer"
  data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 2. produces the number 4

```
</div>
```

QUESTION: What does the following expression produce?

```
[ [((len('ababb')))], len(["argg", ('b'), ("c")]), len([len("bc")]) ]
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: [[5], 3, 1]

</div>

Reading an element

Like for strings, we can access an element a list element by putting the index of the position we want to access among square brackets:

```
[11]: # 0 1 2 3
la = [70, 60, 90, 50]
```

As for any sequence, the positions start from 0:

```
[12]: la[0]
```

```
[12]: 70
```

```
[13]: la[1]
```

```
[13]: 60
```

```
[14]: la[2]
```

```
[14]: 90
```

```
[15]: la[3]
```

```
[15]: 50
```

Like for any string, if we exaggerate with the index we get an error:

```
la[4]
-----
IndexError                                 Traceback (most recent call last)
<ipython-input-134-09bfed834fa2> in <module>
----> 1 la[4]

IndexError: list index out of range
```

As in strings, we can obtain last element by using a negative index:

```
[16]: # 0 1 2 3
la = [70, 60, 90, 50]
```

```
[17]: la[-1]
```

```
[17]: 50
```

```
[18]: la[-2]
```

```
[18]: 90
```

```
[19]: la[-3]
```

```
[19]: 60
```

```
[20]: la[-4]
```

```
[20]: 70
```

If we go beyond the list length, we get an error:

```
la[-5]
```

```
-----  
IndexError Traceback (most recent call last)  
<ipython-input-169-f77280923dce> in <module>  
----> 1 la[-5]
```

```
IndexError: list index out of range
```

QUESTION: if `x` is some list, by writing:

```
x[0]
```

what do we get?

1. the first element of the list
2. always an error
3. sometimes an element, sometimes an error according to the list

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 3: if the list is empty Python will not find the element and will give us an error. Which one? Try writing in the cell down here `[] [0]` and see what happens.

```
</div>
```

```
[21]: # write code here
```

QUESTION: if `x` is some list, by writing:

```
x[len(x)]
```

what do we get?

1. an element of the list
2. always an error
3. sometimes an element, sometimes an error according to the list

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 2. always an error: `len(x)` will always be a number equal to the last available index + 1

```
</div>
```

Exercise - Gutenberg apprentice

Such honor! So young and you have been hired as master Gutenberg apprentice! Your job is to compose the pages with the characters made with iron blocks, so your collaborators can then send everything to the printing press.

You have a `chars` list in which the original blocks are saved. Can you print the writing Gutenberg?

- **DO NOT** write characters nor additional strings (so no 'g' nor 'G'!)
- every character **MAY** be reused more than once

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[22]:

```
chars = ['b', 'e', 'g', 'n', 'r', 't', 'u']    # Gutenberg
#chars = ['a', 'm', 's', 'p', 'o', 'a', 't']    # Stampamos

l = chars      # Let's create a new handy variable

# write here
# SOLUTION 1: not optimal
print(l[2].upper(), l[6], l[5], l[1], l[3], l[0], l[1], l[4], l[2])
# SOLUTION 2
print(f"{l[2].upper()}{l[6]}{l[5]}{l[1]}{l[3]}{l[0]}{l[1]}{l[4]}{l[2]}")
# SOLUTION 3
print("%s%s%s%s%s%s%s%s" % (l[2].upper(), l[6], l[5], l[1], l[3], l[0], l[1], l[4], l[2]))
```

G u t e n b e r g
Gutenberg
Gutenberg

```
</div>
```

[22]:

```
chars = ['b', 'e', 'g', 'n', 'r', 't', 'u']    # Gutenberg
#chars = ['a', 'm', 's', 'p', 'o', 'a', 't']    # Stampamos

l = chars      # Let's create a new handy variable

# write here
```

Writing an element

Since all the lists are MUTABLE, given a list object we can change the content of any cell inside.

For example, suppose you want to change the cell at index 2 of the list `la`, from 6 to 5:

[23]:

#0	1	2	3
la = [7, 9, 6, 8]			

We might write like this:

```
[24]: la[2] = 5
```

```
[25]: la
```

```
[25]: [7, 9, 5, 8]
```

Let's see what's happening with Python Tutor:

```
[26]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
#           (it's sufficient to execute it only once)
```

```
import jupman
```

```
[27]: #      0   1   2   3
la = [7, 9, 6, 8]
la[2] = 5
```

```
jupman.pytut()
```

```
[27]: <IPython.core.display.HTML object>
```

As you see, no new memory regions are created, it just overwrites an existing cell.

Exercise - a jammed parking lot

You are the administrator of the condominium “The Pythonic Joy”. Every apartment has one or two parking spaces assigned, and each one is numbered from 1 to 11.

What follows is the current parking lot, and as you can see there are three spaces not assigned, because the flats 3, 4 and 7 have no tenants anymore:

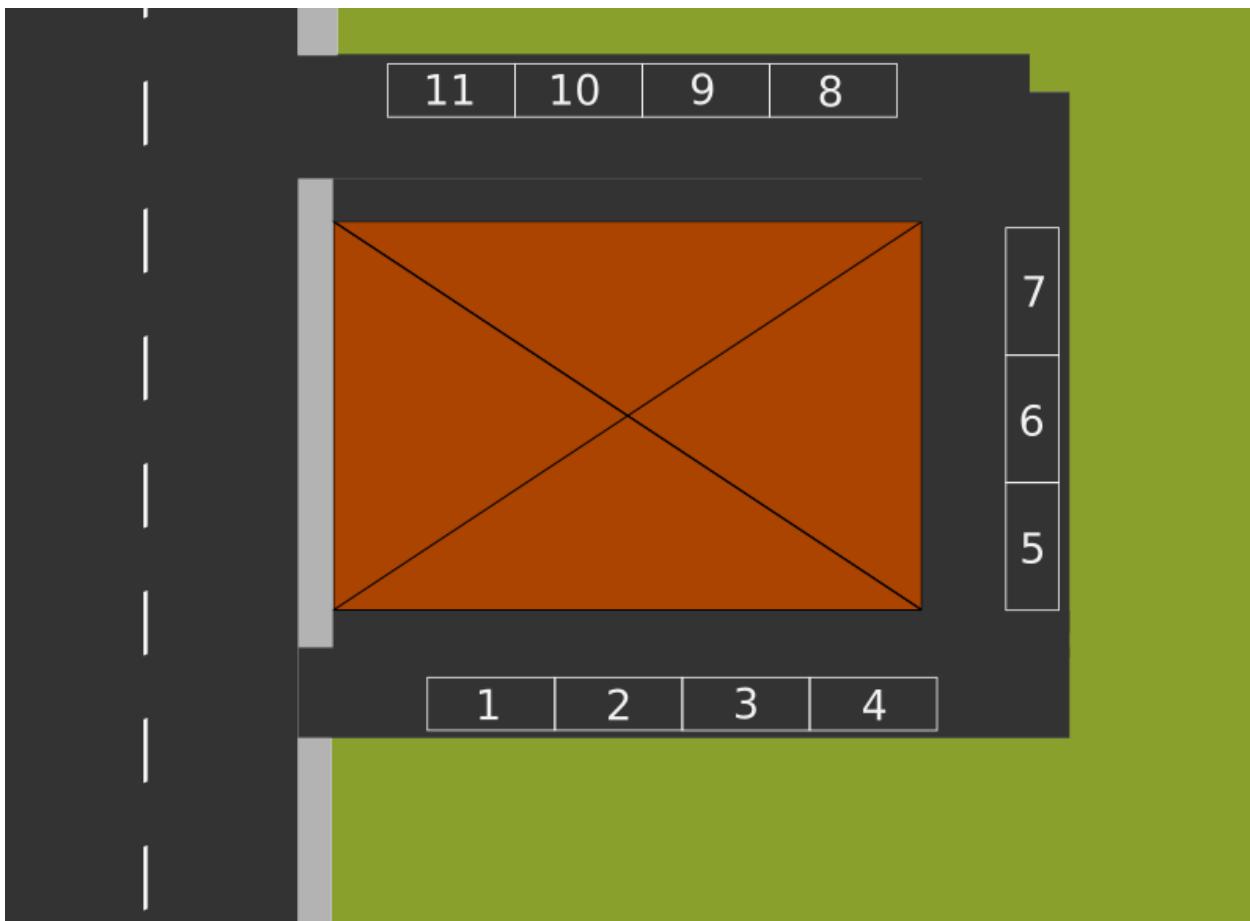
```
[28]: parking_lot = ["Carlo", "Apt.3", "Ernesto", "Apt.4", "Apt.7", "Pam", "Giovanna",
                   ↪"Camilla", "Giorgia", "Jessica", "Jim"]
```

To keep the order you decide to compact the assignments and leave the empty spaces at the far end (could be handy for parking the movers!)

Write some code to MODIFY `parking_lot` so to have:

```
>>> print(parking_lot)
['Carlo', 'Jessica', 'Ernesto', 'Jim', 'Giorgia', 'Pam', 'Giovanna', 'Camilla', 'App.7
 ↪', 'App.3', 'App.4']
```

- **DO NOT** create new lists (no `[a,b, ...]` nor `list(a,b,...)`)
- **DO NOT** write tenants nor apartment names (so no `'Jessica'` nor `'Apt.3'`)
- `parking_lot` may have variable length
- assume the unassigned places are always 3 and in fixed position



Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[29]:

```

parking_lot = ["Carlo", "Apt.3", "Ernesto", "Apt.4", "Apt.7", "Pam", "Giovanna",
               ↵"Camilla", "Giorgia", "Jessica", "Jim"]
#result:      ['Carlo', 'Jessica', 'Ernesto', 'Jim', 'Giorgia', 'Pam', 'Giovanna',
               ↵'Camilla', 'Apt.7', 'Apt.3', 'Apt.4']
#parking_lot = ["Cristian", "Apt.3", "Edgar", "Apt.4", "Apt.7", "Pamela", "Giusy",
               ↵"Cristina", "John"]
#result:      ['Cristian', 'Cristina', 'Edgar', 'John', 'Giusy', 'Pamela', 'Apt.7',
               ↵'Apt.3', 'Apt.4']

# write here

parking_lot[1],parking_lot[-2] = parking_lot[-2],parking_lot[1]
parking_lot[3],parking_lot[-1] = parking_lot[-1],parking_lot[3]
parking_lot[4],parking_lot[-3] = parking_lot[-3],parking_lot[4]

print(parking_lot)
['Carlo', 'Jessica', 'Ernesto', 'Jim', 'Giorgia', 'Pam', 'Giovanna', 'Camilla', 'Apt.7
               ↵', 'Apt.3', 'Apt.4']
```

</div>

[29]:

```
parking_lot = ["Carlo", "Apt.3", "Ernesto", "Apt.4", "Apt.7", "Pam", "Giovanna",
    ↪"Camilla", "Giorgia", "Jessica", "Jim"]
#result:      ['Carlo', 'Jessica', 'Ernesto', 'Jim', 'Giorgia', 'Pam', 'Giovanna',
    ↪'Camilla', 'Apt.7', 'Apt.3', 'Apt.4']
#parking_lot = ["Cristian", "Apt.3", "Edgar", "Apt.4", "Apt.7", "Pamela", "Giusy",
    ↪"Cristina", "John"]
#result:      ['Cristian', 'Cristina', 'Edgar', 'John', 'Giusy', 'Pamela', 'Apt.7',
    ↪'Apt.3', 'Apt.4']

# write here
```

Mutating shared lists

WARNING: READ VERY WELL !!!

90% OF PROGRAMMING ERRORS ARE CAUSED BY MISUNDERSTANDING THIS TOPIC !!!

What happens when we associate the same identical mutable object to two variables, like for example a list, and then we mutate the object using one of the two variables?

Let's look at an example - first, we associate the list [7, 9, 6] to variable la:

[30]: la = [7, 9, 6]

Now we define a new variable lb, and we associate the *same value* that was already associated to variable la. Note: we are NOT creating new lists !

[31]: lb = la

[32]: print(la) # la is always the same

[7, 9, 6]

[33]: print(lb) # lb is the *same* list associated to la

[7, 9, 6]

We can now try modifying a cell of lb, putting 5 in the cell at index 0:

[34]: lb[0] = 5

If we try printing the variables la and lb, Python will look at the values associated to each variable. Since the value is the same identical list (which is in the same identical memory region), in both cases you will see the change we just did !

[35]: print(la)

[5, 9, 6]

[36]: print(lb)

[5, 9, 6]

Let's see in detail what happens with Python Tutor:

```
[37]: la = [7, 9, 6]
lb = la
lb[0] = 5
print('la is', la)
print('lb is', lb)

jupman.pytut()

la is [5, 9, 6]
lb is [5, 9, 6]

[37]: <IPython.core.display.HTML object>
```

Let's see the difference when we explicitly create a list equal to `la`.

In this case we will have two distinct memory regions and `la` will NOT be modified:

```
[38]: la = [7, 9, 6]
lb = [7, 9, 6]
lb[0] = 5
print('la is', la)
print('lb is', lb)

jupman.pytut()

la is [7, 9, 6]
lb is [5, 9, 6]

[38]: <IPython.core.display.HTML object>
```

QUESTION: After executing this code, what will be printed? How many lists will be present in memory?

Try drawing **ON PAPER** what is supposed to happen in memory, and then compare with Python Tutor!

```
la = [8, 7, 7]
lb = [9, 6, 7, 5]
lc = lb
la = lb
print('la is', la)
print('lb is', lb)
print('lc is', lc)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: it will print:

```
la is [9, 6, 7, 5]
lb is [9, 6, 7, 5]
lc is [9, 6, 7, 5]
```

because

```
la = [8, 7, 7]
lb = [9, 6, 7, 5]

lc = lb    # variable lc is associated to the same identical list of lb
```

(continues on next page)

(continued from previous page)

```
la = lb    # variable la is associated to the same identical list of lb  
          # the list previously associated to la is lost
```

```
</div>
```

```
[39]: la = [8, 7, 7]  
lb = [9, 6, 7, 5]  
lc = lb  
la = lb  
#print('la is', la)  
#print('lb is', lb)  
#print('lc is', lc)  
jupman.pytut()
```

```
[39]: <IPython.core.display.HTML object>
```

QUESTION: Look at the following code. After its execution, by printing `la`, `lb` and `lc` what will we get?

Try drawing **ON PAPER** what is happening in memory, then compare the result with Python Tutor!

```
la = [7, 8, 5]  
lb = [6, 7]  
lc = lb  
lb = la  
lc[0] = 9  
print('la is', la)  
print('lb is', lb)  
print('lc is', lc)
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show answer"  
data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: The print will produce

```
la is [7, 8, 5]  
lb is [7, 8, 5]  
lc is [9, 7]
```

because :

```
la = [7, 8, 5]  
lb = [6, 7]  
# the variable lc is assigned to the same list of lb [6, 7]  
lc = lb  
# the variable lb is associated to the same list of la [7, 8, 5].  
# This doesn't change the assignment of lc, which remains associated to [6, 7] !  
lb = la  
# Modifies the first element of the list associated to lc which from [6, 7] becomes [9,  
# 7]  
lc[0] = 9  
print('la is', la)  
print('lb is', lb)  
print('lc is', lc)
```

```
</div>
```

```
[40]: la = [7,8,5]
lb = [6,7]
lc = lb
lb = la
lc[0] = 9
#print('la is', la)
#print('lb is', lb)
#print('lc is', lc)

jupman.pytut()

[40]: <IPython.core.display.HTML object>
```

List of strings

We said we can put any object into a list, for example some strings:

```
[41]: vegetables = ['tomatoes', 'onions', 'carrots', 'cabbage']
```

Let's try extracting a vegetable by writing this expression:

```
[42]: vegetables[2]
[42]: 'carrots'
```

Now, the preceding expression produces the result 'carrots', which we know is a string. This suggests we can use the expression exactly like if it were a string.

Suppose we want to obtain the first character of the string 'carrots', if we directly have the string we can write like this:

```
[43]: 'carrots'[0]
[43]: 'c'
```

But if the string is inside the previous list, we could directly do like this:

```
[44]: vegetables[2][0]
[44]: 'c'
```

Exercise - province codes

Given a list with exactly 4 province codes in lowercase, write some code which creates a NEW list containing the same codes in uppercase characters.

- your code must work with any list of 4 provinces
- hint: if you don't remember the right method, have a look here¹⁵⁰

Example 1 - given:

```
provinces = ['tn', 'mi', 'to', 'ro']
```

your code must print:

¹⁵⁰ <https://en.softpython.org/strings/strings3-sol.html>

```
[ 'TN', 'MI', 'TO', 'RO' ]
```

Example 2 - given:

```
provinces = ['pa', 'ge', 've', 'aq']
```

Your code must print:

```
[ 'PA', 'GE', 'VE', 'AQ' ]
```



Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[45]:

```
provinces = ['tn', 'mi', 'to', 'ro']
#provinces = ['pa', 'ge', 've', 'aq']

# write here

print([provinces[0].upper(), provinces[1].upper(), provinces[2].upper(), provinces[3].
       upper()])

['TN', 'MI', 'TO', 'RO']
```

</div>

[45]:

```
provinces = ['tn', 'mi', 'to', 'ro']
#provinces = ['pa', 'ge', 've', 'aq']

# write here
```

Exercise - games

Given a list `games` of exactly 3 strings, write some code which MODIFIES the list so it contains only the first characters of each string.

- Your code must work with any list of exactly 3 strings

Example - given:

```
games = ["Monopoly", "RISK", "Bingo"]
```

After executing the code, it must result:

```
>>> print(games)
["M", "R", "B"]
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[46]:

```
games = ["Monopoly", "RISK", "Bingo"] # ['M', 'R', 'T']
#games = ["Frustration", "Game of the Goose", "Scrabble"] # ['F', 'G', 'S']

# write here

games = [ games[0][0], games[1][0], games[2][0] ]
print(games)

['M', 'R', 'B']
```

</div>

[46]:

```
games = ["Monopoly", "RISK", "Bingo"] # ['M', 'R', 'T']
#games = ["Frustration", "Game of the Goose", "Scrabble"] # ['F', 'G', 'S']

# write here
```

Slices

We can extract sequences from lists by using *slices*. A slice is produced by placing square brackets after the list with inside the starting index (INCLUDED), followed by a colon :, followed by the end index (EXCLUDED). It works exactly as with strings: in that case the slice produces a new string, in this case it produces a NEW list. Let's see an example:

[47]:

```
#0 1 2 3 4 5 6 7 8 9
la = [40, 30, 90, 80, 60, 10, 40, 20, 50, 60]
```

[48]:

```
la[3:7]
```

[48]:

```
[80, 60, 10, 40]
```

We extracted a NEW list [80, 60, 10, 40] from the list la starting from index 3 INCLUDED until index 7 EXCLUDED. We can see the original list is preserved:

[49]:

```
la
```

[49]:

```
[40, 30, 90, 80, 60, 10, 40, 20, 50, 60]
```

Let's verify what happens with Python Tutor, by assigning the new list to a variable lb:

[50]:

```
# 0 1 2 3 4 5 6 7 8 9
la = [40, 30, 90, 80, 60, 10, 40, 20, 50, 60]
lb = la[3:7]

jupman.pytut()
```

[50]:

```
<IPython.core.display.HTML object>
```

You will notice a NEW memory region, associated to variable lb.

Slice - limits

When we operate with slices we must be careful about indeces limits. Let's see how they behave:

```
[51]: #0 1 2 3 4  
[50,90,70,80,60][0:3] # from index 0 *included* to 3 *excluded*
```

```
[51]: [50, 90, 70]
```

```
[52]: #0 1 2 3 4  
[50,90,70,80,60][0:4] # from index 0 *included* a 4 *excluded*
```

```
[52]: [50, 90, 70, 80]
```

```
[53]: #0 1 2 3 4  
[50,90,70,80,60][0:5] # from index 0 *included* to 5 *excluded*
```

```
[53]: [50, 90, 70, 80, 60]
```

```
[54]: #0 1 2 3 4  
[50,90,70,80,60][0:6] # if we go beyond the list length Python does not complain
```

```
[54]: [50, 90, 70, 80, 60]
```

```
[55]: #0 1 2 3 4  
[50,90,70,80,60][8:12] # Python doesn't complain even if we start from non-existing  
→indeces
```

```
[55]: []
```

QUESTION: This expression:

```
[] [3:8]
```

1. produces a result (which one?)
2. produces an error (which one?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: given an empty list, we are trying yo create a sublist which goes from index 3 INCLUDED to index 8 EXCLUDED. As we've seen before, if we start after the limit and also if we go beyond the limit Python does not complain, and when elements are not found we are simply served with an empty list.

</div>

QUESTION: if `x` is some list (may also empty), what does this expression do? Can it give an error? Does it return something useful?

```
x[0:len(x)]
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: Always return a NEW copy of the entire list, because it starts from index 0 INCLUDED and ends at index `len(x)` EXCLUDED.

It also works with the empty list, because `[] [0:len([])]` is equivalent to `[] [0:0]` that is sublist from 0 included to 0 excluded, so we are not taking any character and are not going beyod list limits. In fact, as we've seen before, even if we went beyond Python wouldn't complain.

```
</div>
```

Exercise - The ‘treccia mochena’

As you well know, a wonderful pastry is made in the Mocheni valley in Trentino: the famous ‘treccia mochena’.

At a quick glance, it may look like a braid loaf, between 30 and 60 cm long with inside a mix of ingredients along a marvellous and secret cream.

With your friends Camilla and Giorgio, you bought a treccia divided in a certain number of portions stuffed with walnuts, blubberries and red currants.

```
treccia = ['w', 'w', 'w', 'w', 'w', 'b', 'b', 'b', 'b', 'b', 'c', 'c', 'c', 'c']
```

```
walnuts,blubberries,currants = 5,6,4
```

You like the blubberries, Giorgio likes walnuts and Camilla the red currants.

Write some code to place into variables `mine`, `giorgio` and `camilla` some lists obtained from `treccia`, and PRINT the result:

```
Mine: ['b', 'b', 'b', 'b', 'b', 'b']
Giorgio: ['w', 'w', 'w', 'w', 'w']
Camilla: ['c', 'c', 'c', 'c']
```

- suppose `treccia` has always only 3 ingredients
- **DO NOT** write constant numbers (except 0)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[56]:

```
#                               Walnuts          Blubberries
˓→      Currants
walnuts,blubberries,currants,treccia = 5,6,4,['w', 'w', 'w', 'w', 'w', 'b', 'b', 'b',
˓→'b', 'b', 'b', 'c', 'c', 'c']
#walnuts,blubberries,currants,treccia = 2,4,3,['W', 'W', 'B', 'B', 'B', 'C', 'C',
˓→'C']

# write here
mine = treccia[walnuts : walnuts+blubberries]
giorgio = treccia[0 : walnuts]
camilla = treccia[walnuts+blubberries : walnuts+blubberries+currants]
print("  Mine: %s \nGiorgio: %s \nCamilla: %s" % (mine, giorgio, camilla))

Mine: ['b', 'b', 'b', 'b', 'b', 'b']
Giorgio: ['w', 'w', 'w', 'w', 'w']
Camilla: ['c', 'c', 'c', 'c']
```

```
</div>
```

[56]:

```
#                               Walnuts          Blubberries
˓→      Currants
walnuts,blubberries,currants,treccia = 5,6,4,['w', 'w', 'w', 'w', 'w', 'b', 'b', 'b',
˓→'b', 'b', 'b', 'c', 'c', 'c']
```

(continues on next page)

(continued from previous page)

```
#walnuts,bluberries,currants,treccia = 2,4,3,['W', 'W', 'B', 'B', 'B', 'B', 'C', 'C',
↪'C']

# write here
```

Slices - omitting limits

If we will, it's possible to omit start index, in which case Python will suppose it's 0:

```
[57]: #0 1 2 3 4 5 6 7 8 9
[90,60,80,70,60,90,60,50,70][:3]

[57]: [90, 60, 80]
```

It's also possible to omit the end index, in this case Python will extract elements until the list end:

```
[58]: #0 1 2 3 4 5 6 7 8 9
[90,60,80,70,60,90,60,50,70][3:]

[58]: [70, 60, 90, 60, 50, 70]
```

By omitting both indexes we obtain the full list:

```
[59]: #0 1 2 3 4 5 6 7 8 9
[90,60,80,70,60,90,60,50,70][:]

[59]: [90, 60, 80, 70, 60, 90, 60, 50, 70]
```

QUESTION: What is this code going to print? Will `la` get modified or not?

```
la = [7,8,9]
lb = la[:]
lb[0] = 6
print('la =',la)
print('lb =',lb)
```

Show answer<div class="jupman-sol" jupman-sol-question" style="display:none">

ANSWER: `lb = la[:]` creates a NEW list containing all the elements which are in `la`. When we write `lb[0] = 6` we are only modifying the memory region associated to `lb`. If you observe it in Python Tutor, you will see that `la` and `lb` are pointing to different memory regions:

</div>

```
[60]: la = [7,8,9]
lb = la[:]
lb[0] = 6
#print('la =',la)
#print('lb =',lb)

jupman.pytut()

[60]: <IPython.core.display.HTML object>
```

QUESTION: For each of the following expressions, try guessing which value it produces, or if it gives an error.

1. [9 , 7 , 8 , 6] [1 : 1]

2. [9, 7, 8, 6] [1:2]

3. [9, 7, 8, 6] [2:3] [0]

4. [] []

5. [] [:]

6. [3] [:]

7. [:] []

Exercise - An out of tune guitar

In the attic you found an old guitar which was around when you were a child. Now that you have a degree in Sound Engineering you try playing it while a sensor measures the notes it produces.

The sensor is a microphone, and each one tenth of second it records the main note it detected.

You discover this phenomena: when played, some guitar strings have an oscillating behaviour, but then they synthonize on a precise note until the end:

'D' 'A' 'F' 'E' 'B' 'G' 'B' 'F' 'F' 'F' 'F' 'F' 'F' 'F' 'F' 'F'

Other strings do the opposite:

'C', 'C', 'C', 'C', 'C', 'D', 'G', 'F', 'B', 'B', 'C', 'B', 'B', '...'

A list `cutoffs` records for each string the instants in which the weird behaviour starts or ends. The instants are represented as a sequence of beats '*' , for example the first string starts an anomalous behaviour after 5 beats: '*****', while the seconds ends the anomalous behaviour after 7 beats: '*****' .

Write some code to cut the sensor output and obtain only the sequences of continuous and correct notes. In the end, it should PRINT:

```
[['C', 'C', 'C', 'C', 'C'],
 ['F', 'F', 'F', 'F', 'F', 'F', 'F', '...'],
 ['D', 'D', 'D', 'D', 'D', 'D', 'D', '...'],
 ['G', 'G', 'G', 'G']]
```

- **DO NOT** write constant numbers of instants in the code, instead, use the variable `cutOffs`.

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[61] i

```
cutoffs = [ '*****',  
           '*****',  
           '***',  
           '***' ]
```

(continues on next page)

(continued from previous page)

```
string1 = ['C', 'C', 'C', 'C', 'C', 'D', 'G', 'F', 'B', 'B', 'C', 'B', ...]
string2 = ['D', 'A', 'F', 'E', 'B', 'G', 'B', 'F', 'F', 'F', 'F', 'F', 'F', ...
    ↵']
string3 = ['B', 'E', 'G', 'D', 'D', 'D', 'D', 'D', 'D', 'D', ...]
string4 = ['G', 'G', 'G', 'G', 'D', 'D', 'A', 'A', 'B', 'F', 'B', ...]

# write here
string1_clean = string1[:len(cutoffs[0])]
string2_clean = string2[len(cutoffs[1]):]
string3_clean = string3[len(cutoffs[2]):]
string4_clean = string4[:len(cutoffs[3])]

print(string1_clean)
print(string2_clean)
print(string3_clean)
print(string4_clean)

['C', 'C', 'C', 'C', 'C']
['F', 'F', 'F', 'F', 'F', 'F', 'F', '...']
['D', 'D', 'D', 'D', 'D', 'D', '...', ...]
['G', 'G', 'G', 'G']
```

</div>

[61]:

```
cutoffs = ['*****',
           '*****',
           '***',
           '****',]

string1 = ['C', 'C', 'C', 'C', 'C', 'D', 'G', 'F', 'B', 'B', 'C', 'B', ...]
string2 = ['D', 'A', 'F', 'E', 'B', 'G', 'B', 'F', 'F', 'F', 'F', 'F', 'F', ...
    ↵']
string3 = ['B', 'E', 'G', 'D', 'D', 'D', 'D', 'D', 'D', 'D', ...]
string4 = ['G', 'G', 'G', 'G', 'D', 'D', 'A', 'A', 'B', 'F', 'B', ...]

# write here
```

Slices - negative limits

It's also possible to set inverse and negative limits, although it's not always intuitive:

```
[62]: #0 1 2 3 4 5 6
[70,40,10,50,60,10,90][3:0]    # from index 3 to positive indexes <= 3 produces nothing
[62]: []

[63]: #0 1 2 3 4 5 6
[70,40,10,50,60,10,90][3:1]    # from index 3 to positive indexes <= 3 produces nothing
[63]: []
```

```
[64]: # 0 1 2 3 4 5 6
[70,40,10,50,60,10,90][3:2]    # from index 3 to positive indexes <= 3 produces nothing
[64]: []
```

```
[65]: # 0 1 2 3 4 5 6
[70,40,10,50,60,10,90][3:3]    # from index 3 to positive indexes <= 3 produces nothing
[65]: []
```

Let's see what happens with negative indexes:

```
[66]: # 0 1 2 3 4 5 6
#-7 -6 -5 -4 -3 -2 -1
[70,40,10,50,60,10,90][3:-1]
[66]: [50, 60, 10]
```

```
[67]: # 0 1 2 3 4 5 6
#-7 -6 -5 -4 -3 -2 -1
[70,40,10,50,60,10,90][3:-2]
[67]: [50, 60]
```

```
[68]: # 0 1 2 3 4 5 6
#-7 -6 -5 -4 -3 -2 -1
[70,40,10,50,60,10,90][3:-3]
[68]: [50]
```

```
[69]: # 0 1 2 3 4 5 6
#-7 -6 -5 -4 -3 -2 -1
[70,40,10,50,60,10,90][3:-4]
[69]: []
```

```
[70]: # 0 1 2 3 4 5 6
#-7 -6 -5 -4 -3 -2 -1
[70,40,10,50,60,10,90][3:-5]
[70]: []
```

It's also possible to start from a negative index and arrive to a positive one. As long as the first index marks a position which precedes the second index, something gets returned:

```
[71]: # 0 1 2 3 4 5 6
#-7 -6 -5 -4 -3 -2 -1
[70,40,10,50,60,10,90][-7:3]
[71]: [70, 40, 10]
```

```
[72]: # 0 1 2 3 4 5 6
#-7 -6 -5 -4 -3 -2 -1
[70,40,10,50,60,10,90][-6:3]
[72]: [40, 10]
```

```
[73]: # 0 1 2 3 4 5 6  
#-7 -6 -5 -4 -3 -2 -1  
[70,40,10,50,60,10,90][-5:3]
```

```
[73]: [10]
```

```
[74]: # 0 1 2 3 4 5 6  
#-7 -6 -5 -4 -3 -2 -1  
[70,40,10,50,60,10,90][-4:3]
```

```
[74]: []
```

```
[75]: # 0 1 2 3 4 5 6  
#-7 -6 -5 -4 -3 -2 -1  
[70,40,10,50,60,10,90][-3:3]
```

```
[75]: []
```

```
[76]: # 0 1 2 3 4 5 6  
#-7 -6 -5 -4 -3 -2 -1  
[70,40,10,50,60,10,90][-2:3]
```

```
[76]: []
```

QUESTION: For each of the following expressions, try guessing which value is produced, or if it gives an error

1.

2.

3.

4.

5.

6.

7.

8.

Exercise - The NonsenShop

To keep your expenses under control you want to write a software which tracks everything you buy, and reduce pointless expenses.

You just need to take a picture of the receipt with the smartphone, and an OCR (Optical Character Recognition) module will automatically read the text.

- **all receipts** have the same schema: shop name, date, “Bought items”, 1 row per item, total, thanks.

- assume all the receipts are always ordered by price

- 1) Write some code to create a NEW list with only the items rows, for example:

```
[ 'Green varnish for salad      1,12€',
  'Anti-wind lead confetti    4,99€',
  'Comb for pythons           12,00€',
  'Cigarette lighter for diving 23,00€',
  'Transparent home shoes       35,56€']
```

2) Print the most expensive item like this:

```
Most expensive item was: Transparent home shoes
cost: 35,56€
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[77]:

```
receipt = ["NonsenseShop",
           "July 21, 2021 14:54",
           "Items:",
           "Green varnish for salad      1,12€",
           "Anti-wind lead confetti    4,99€",
           "Comb for pythons           12,00€",
           "Cigarette lighter for diving 23,00€",
           "Transparent home shoes       35,56€",
           "Total                      56,66€",
           "Thanks for buying our nonsense!"]

#receipt = ["Eleganz",
#           "January 3 2020 12:53",
#           "Items:",
#           "Wedge heels with aquarium and fishes      342,00€",
#           "'Elvis' suit                         20.000,00€",
#           "Total 20.000,342€",
#           "And now unleash the party animal in you!"]

# write here
items = receipt[3:-2]
last = receipt[-3]

from pprint import pprint
pprint(items)
print()
print("Most expensive item was:", last[:-10])
print("                           cost:", last[-10:].lstrip())

['Green varnish for salad      1,12€',
 'Anti-wind lead confetti    4,99€',
 'Comb for pythons           12,00€',
 'Cigarette lighter for diving 23,00€',
 'Transparent home shoes       35,56€']

Most expensive item was: Transparent home shoes
cost: 35,56€
```

</div>

[77]:

```
receipt = ["NonsenseShop",
```

(continues on next page)

(continued from previous page)

```
"July 21, 2021 14:54",
"Items:",
"Green varnish for salad      1,12€",
"Anti-wind lead confetti    4,99€",
"Comb for pythons            12,00€",
"Cigarette lighter for diving 23,00€",
"Transparent home shoes       35,56€",
"Total                      56,66€",
"Thanks for buying our nonsense!"]

#recepit = ["Eleganz",
#           "January 3 2020 12:53",
#           "Items:",
#           "'Wedge heels with aquarium and fishes'      342,00€",
#           "'Elvis' suit                                20.000,00€",
#           "Total 20.000,342€",
#           "And now unleash the party animal in you!"]

# write here
```

Slice - step

It's also possible to specify a third parameter called 'step' to tell Python how many cells to skip at each read. For example, here we start from index 3 and arrive to index 9 excluded, **skipping by 2**:

```
[78]: # 0 1 2 3 4 5 6 7 8 9
[ 0,10,20,30,40,50,60,70,80,90][3:9:2]
[78]: [30, 50, 70]
```

All the sequence, skipping by 3:

```
[79]: # 0 1 2 3 4 5 6 7 8 9
[ 0,10,20,30,40,50,60,70,80,90][0:10:3]
[79]: [0, 30, 60, 90]
```

We can also omit the limits to obtain the equivalent expression:

```
[80]: # 0 1 2 3 4 5 6 7 8 9
[ 0,10,20,30,40,50,60,70,80,90][::3]
[80]: [0, 30, 60, 90]
```

Slices - modifying

Suppose we have the list

```
[81]: # 0 1 2 3 4 5 6 7
la = [30, 40, 80, 10, 70, 60, 40, 20]
```

and we want to change `la` cells from index 3 INCLUDED to index 6 EXCLUDED in such a way they contain the numbers taken from list `[91, 92, 93]`. We can do it with this special notation which allows us to write a slice *to the left* of operator `=`:

```
[82]: la[3:6] = [91, 92, 93]
```

```
[83]: la
```

```
[83]: [30, 40, 80, 91, 92, 93, 40, 20]
```

In this slightly more complex example we verify in Python Tutor that the original memory region gets actually modified:

```
[84]: # 0 1 2 3 4 5 6 7
la = [30, 40, 80, 10, 70, 60, 40, 20]
lb = la
lb[3:6] = [91, 92, 93]

jupman.pytut()
```

```
[84]: <IPython.core.display.HTML object>
```

QUESTION: Look at the following code - what does it produce?

```
la = [9, 6, 5, 8, 2]
la[1:4] = [4, 7, 0]
print(la)
```

1. modify `la` (how?)
2. an error (which one?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 1 - MODIFIES `la` like this:

```
# 0 1 2 3 4
[ 9, 4, 7, 0, 2 ]
```

so from index 1 INCLUDED to index 4 EXCLUDED

</div>

QUESTION: Look at the following code. What does it produce?

```
la = [7, 6, 8, 4, 2, 4, 2, 3, 1]
i = 3
lb = la[0:i]
la[i:2*i] = lb
print(la)
```

1. modifies `la` (how?)

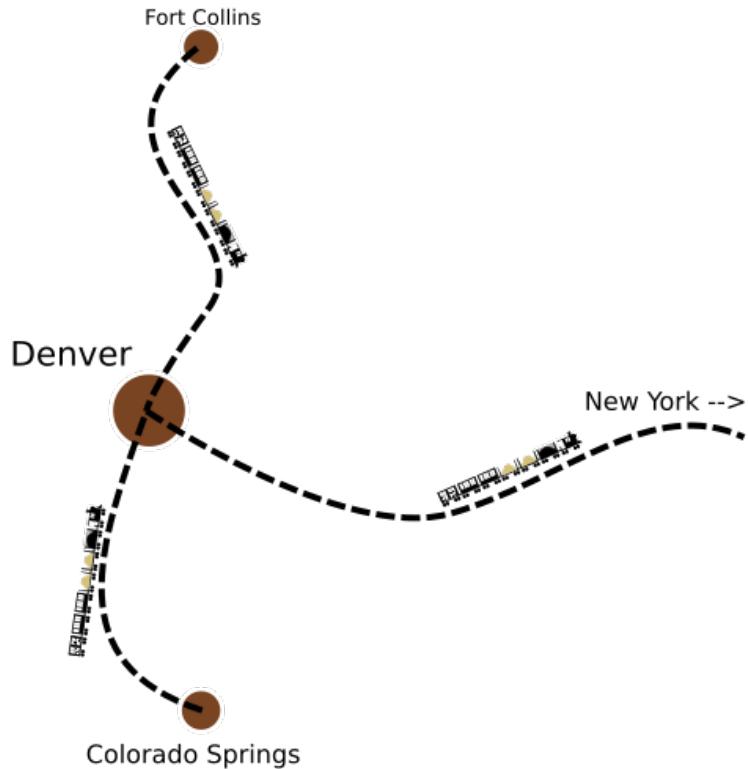
2. an error (which one?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 1 - modifies `la` by copying first `i` cells into successive ones.

</div>

Exercise - The railway outlaws



United States - May 13th, 1857

The colonization of the West is a hard job but somebody gotta do it. Mountains are stuffed with rare minerals and cute animals to be transformed into precious fur coats for noblewomans of Europe. As always, with great wealth come great outlaws.

You are the station master of Denver and you must manage the trains. There are three main lines, Colorado Springs, Fort Collins e New York:

- from Colorado Springs always arrive exactly 1 wagon of coal, 3 of minerals and 3 of passengers **alternated**.
- from Fort Collins always arrive exactly 2 wagons of coal, 2 of passengers and 2 of cattle

When trains reach Denver, their content is transferred into the empty wagons of the New York train like this:

- to prevent robberies, all the precious wagons are to be positioned **nearby the locomotive**
- the cattle is to be placed **always behind passengers**, because as much as hygiene in the west is lacking, it's still easier to pretend humans bathe more than cattle

Write some code which MODIFIES the ORIGINAL memory region of new_york list by copying the strings from colorado and fort

- **MINIMIZE the number of assignments!** (the best solution with just slices has only has three assignments!)
- **DO NOT** write constant strings in your code (so no "cowboy", "gold", ...)

Example - given:

```
[85]: colorado = ["CS locomotive", "coal", "cowboy", "gold", "miners", "gold", "cowboy",
    ↪ "silver"]
fort = ["FC locomotive", "coal", "coal", "gentlmen", "ladies", "cows", "horses"]
new_york = ["NY locomotive", "coal", "", "", "", "", "", "", ""]
    ↪ "", "", "", ""]
```

after your code it must result:

```
>>> print(new_york)
['NY locomotive', 'coal', 'gold', 'gold', 'silver', 'cowboy', 'miners', 'cowboy',
    ↪ 'gentlmen', 'ladies', 'cows', 'horses']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[86]: # 0          1          2          3          4          5          6          7
    ↪ 8 9 10
colorado = ["CS locomotive", "coal", "cowboy", "gold", "miners", "gold", "cowboy",
    ↪ "silver"]
fort = ["FC locomotive", "coal", "coal", "gentlmen", "ladies", "cows", "horses"]
new_york = ["NY locomotive", "coal", "", "", "", "", "", "", ""]
    ↪ "", "", "", ""]

# write here
new_york[2:5] = colorado[3::2]
new_york[5:] = colorado[2::2]
new_york[8:] = fort[3:]
print(new_york)

['NY locomotive', 'coal', 'gold', 'gold', 'silver', 'cowboy', 'miners', 'cowboy',
    ↪ 'gentlmen', 'ladies', 'cows', 'horses']
```

</div>

```
[86]: # 0          1          2          3          4          5          6          7
    ↪ 8 9 10
colorado = ["CS locomotive", "coal", "cowboy", "gold", "miners", "gold", "cowboy",
    ↪ "silver"]
fort = ["FC locomotive", "coal", "coal", "gentlmen", "ladies", "cows", "horses"]
new_york = ["NY locomotive", "coal", "", "", "", "", "", "", ""]
    ↪ "", "", "", ""

# write here
```

(continues on next page)

(continued from previous page)

List of lists

NOTE: We will talk much more in detail of lists of lists in the tutorial [Matrices - list of lists¹⁵¹](#), this is just a brief introduction.

The consideration we've seen so far about string lists are also valid for a list of lists:

```
[88]: couples = [
    [67, 95],      # external list
    [60, 59],      #   internal list at index 0
    [86, 75],      #           index 1
    [96, 90],      #           index 2
    [88, 87],      #           index 3
    ]              #           index 4
```

If we want to extract the number 90, we must first extract the sublist from index 3:

```
[89]: couples[3]  # NOTE: the expression result is a list
[89]: [96, 90]
```

and so in the extracted sublist (which has only two elements) we can recover the number at index 0:

```
[90]: couples[3][0]
[90]: 96
```

and at index 1:

```
[91]: couples[3][1]
[91]: 90
```

Exercise - couples

1. Write some code to extract and print the numbers 86, 67 and 87
2. Given a row with index i and a column j , print the number at row i and column j multiplied by the number at successive row and same column

After your code, you should see printed

```
1) 86 67 87
2) i = 3 j = 1 result = 7830
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

¹⁵¹ <https://en.softpython.org/matrices-lists/matrices-lists1-sol.html>

[92]:

```

couples = [          # external list
            [67,95],    # internal list at index 0
            [60,59],    # internal list at index 1
            [86,75],    # internal list at index 2
            [96,90],    # internal list at index 3
            [88,87],    # internal list at index 4
        ]

i = 3
j = 1

# write here

print("1)", couples[2][0], couples[0][0], couples[4][1])
print()
print("2)", " i =", i, " j =", j, " result =", couples[i][j]*couples[i+1][j])
1) 86 67 87
2) i = 3 j = 1 result = 7830

```

</div>

[92]:

```

couples = [          # external list
            [67,95],    # internal list at index 0
            [60,59],    # internal list at index 1
            [86,75],    # internal list at index 2
            [96,90],    # internal list at index 3
            [88,87],    # internal list at index 4
        ]

i = 3
j = 1

# write here

```

Exercise - Glory to Gladiators!

The gladiators fight for the glory of the battle and the entertainment of the Emperor and the people! Sadly, not all gladiators manage to fight the same number of battles..

For each fight, each gladiator receives a reward in sesterces (in case he doesn't survive, it will be offered to his patron...)

At the end of the games, the Emperor throws the prize in sesterces to his favourite gladiator. The Emperor has bad aim and his weak arms don't allow him to throw everything at once, so he always ends up throwing half of the money to the chosen gladiator and half to the next gladiator.

- NOTE: the Emperor **never** chooses the last of the list

Given a `prize` and `gladiators` list of sublists of arbitrary length, and a gladiator at index `i`, write some code which MODIFIES the sublists of `gladiators` at row `i` and following one so to increase the last element of both lists of half prize.

- Your code must work with any `prize`, `gladiators` and `i`

Example - given:



67	95			
60	23	23	13	59
86	75			
96	90	92		
88	87			

and given:

```
prize, i = 40, 1
```

after your code, by writing (we use pprint so printing happens on many lines) it must result:

```
>>> from pprint import pprint
>>> pprint(gladiatori, width=30)
[[67, 95],
 [60, 23, 23, 13, 79],
 [86, 95],
 [96, 90, 92],
 [88, 87]]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[93]:

```
prize, i = 40, 1                      # sesterces, gladiator to award
#prize, i = 10, 3

gladiators = [
    [67, 95],                         # external list
    [60, 23, 23, 13, 79],             # internal list at index 0
    [86, 95],                         # internal list at index 1
    [96, 90, 92],                     # internal list at index 2
    [88, 87],                          # internal list at index 3
]
```

(continues on next page)

(continued from previous page)

```
# write here
gladiators[i][-1] += prize // 2
gladiators[i+1][-1] += prize // 2

from pprint import pprint
pprint(gladiators, width=30)

[[67, 95],
 [60, 23, 23, 13, 79],
 [86, 95],
 [96, 90, 92],
 [88, 87]]
```

</div>

[93]:

```
prize, i = 40, 1                      # sesterces, gladiator to award
#prize, i = 10, 3

gladiators = [
    [67, 95],                         # external list
    [60, 23, 23, 13, 59],             # internal list at index 0
    [86, 75],                          # internal list at index 1
    [96, 90, 92],                     # internal list at index 2
    [88, 87],                          # internal list at index 3
]
# write here
```

Membership

To verify whether an object is contained in a list, we can use the `in` operator.

Note the result of this expression is a boolean:

[94]: 9 in [6, 8, 9, 7]

[94]: True

[95]: 5 in [6, 8, 9, 7]

[95]: False

[96]: "apple" in ["watermelon", "apple", "banana"]

[96]: True

[97]: "carrot" in ["watermelon", "apple", "banana"]

[97]: False

Do not abuse in

WARNING: in is often used in a wrong / inefficient way

Always ask yourself:

1. Could the list *not* contain the substring we're looking for? Always remember to also handle his case!
2. in performs a search on all the list, which might be inefficient: is it really necessary, or we already know the interval where to search?
3. If we wanted to know whether element is in a position we know a priori (i.e. 3), in is not needed, it's enough to write my_list[3] == element. By using in Python might find duplicated elements which are *before* or *after* the one we want to verify!

QUESTION: What's the result of this expression? True or False?

```
True in [ 5 in [6,7,5],  
          2 in [8,1]  
      ]
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: Gives back True because

```
[ 5 in [6,7,5],  
  2 in [8,1]  
]
```

represents a list of two elements. Each element is an expression with in which gets evaluated. In the first case, 5 in [6,7,5] results True. So the final list becomes [True, False] and by writing True in [True, False] we obtain True

</div>

not in

We can write the check of **non** belonging in two ways:

Way 1:

```
[98]: "carrot" not in ["watermelon", "banana", "apple"]  
[98]: True
```

```
[99]: "watermelon" not in ["watermelon", "banana", "apple"]  
[99]: False
```

Way 2:

```
[100]: not "carrot" in ["watermelon", "banana", "apple"]  
[100]: True
```

```
[101]: not "watermelon" in ["watermelon", "banana", "apple"]
[101]: False
```

QUESTION: Given any element x and list y , what does the following expression produce?

```
x in y and not x in y
```

1. False
2. True
3. False or True according to the values of x and y
4. an error

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 1. Gives back `False`, because internally Python brackets the expression like this:

```
(x in y) and (not x in y)
```

and one element cannot be both contained in the list and not contained in the same list

</div>

QUESTION: For each of the following expressions, try to guess the result

1. `3 in [3]`
2. `[4, 5] in [1, 2, 3, 4, 5]`
3. `[4, 5] in [[1, 2, 3], [4, 5]]`
4. `[4, 5] in [[1, 2, 3, 4], [5, 6]]`
5. `'n' in ['alien'[-1]]`
6. `'rts' in 'karts'[1:4]`
7. `[] in [[[[]]]`
8. `[] in [[]]`
9. `[] in ["[]"]`

QUESTION: For each of the following expressions, independently from the value of x and y , tell whether it always results `True`:

1. `x in x`
2. `x in [x]`
3. `x not in []`

4. `x in [[x]]`

5. `x in [[x][0]]`

6. `(x and y) in [x,y]`

7. `x in [x,y] and y in [x,y]`

Exercise - vegetables

Given the list `vegetables` of exactly 5 strings and the list of strings `fruits`, MODIFY the variable `vegetables` so that in each cell there is True if the vegetable is a fruit or False otherwise.

- your code must work with any list of 5 strings `vegetables` and any list `fruits`

Example - given:

```
vegetables = ["carrot",
              "cabbage",
              "apple",
              "aubergine",
              "watermelon"]

fruits = ["watermelon", "banana", "apple", ]
```

after execution your code must print:

```
>>> print(vegetables)
[False, False, True, False, True]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[102]:

```
vegetables = ["carrot",
              "cabbage",
              "apple",
              "aubergine",
              "watermelon"]

fruits = ["watermelon", "banana", "apple", ]

# write here

vegetables = [vegetables[0] in fruits,
              vegetables[1] in fruits,
              vegetables[2] in fruits,
              vegetables[3] in fruits,
              vegetables[4] in fruits,
            ]

print(vegetables)
```

```
[False, False, True, False, True]
```

</div>

```
[102]: vegetables = ["carrot",
                   "cabbage",
                   "apple",
                   "aubergine",
                   "watermelon"]

fruits = ["watermelon", "banana", "apple",]

# write here
```

List concatenation with +

Given two lists `la` and `lb`, we can concatenate them with the operator `+` which produces a NEW list:

```
[103]: la = [70, 60, 80]
lb = [90, 50]

la + lb
```

```
[103]: [70, 60, 80, 90, 50]
```

Note the operator `+` produces a NEW list, so `la` and `lb` remained unchanged:

```
[104]: print(la)
[70, 60, 80]
```

```
[105]: print(lb)
[90, 50]
```

Let's check with Python Tutor:

```
[106]: la = [70, 60, 80]
lb = [90, 50]
lc = la + lb

print(la)
print(lb)
print(lc)

jupman.pytut()

[70, 60, 80]
[90, 50]
[70, 60, 80, 90, 50]
```

```
[106]: <IPython.core.display.HTML object>
```

Exercise - concatenation

Write some code which given lists `la` and `lb`, puts into list `lc` the last two elements of `la` and the first two of `lb`

Example - given:

```
la = [18, 26, 30, 45, 55]
lb = [16, 26, 37, 45]
```

after your code it must print:

```
>>> print(la)
[18, 26, 30, 45, 55]
>>> print(lb)
[16, 26, 37, 45]
>>> print(lc)
[45, 55, 16, 26]
```

[Show solution](#)[Hide](#)>

[107]:

```
la = [18, 26, 30, 45, 55]
lb = [16, 26, 37, 45]

# write here
lc = la[-2:] + lb[:2]
print(la)
print(lb)
print(lc)
```

```
[18, 26, 30, 45, 55]
[16, 26, 37, 45]
[45, 55, 16, 26]
```

</div>

[107]:

```
la = [18, 26, 30, 45, 55]
lb = [16, 26, 37, 45]

# write here
```

QUESTION: For each of the following expressions, try guessing the result

1.

2.

3.

4.

5.

6. `[[]] + []`
7. `[[]] + [[]]`
8. `([6] + [8]) [0]`
9. `([6] + [8]) [1]`
10. `([6] + [8]) [2 :]`
11. `len([4, 2, 5]) + len([3, 1, 2])`
12. `len([4, 2, 5] + [3, 1, 2])`
13. `[5, 4, 3] + "3, 1"`
14. `[5, 4, 3] + "[3, 1]"`
15. `"[5, 4, 3]" + "[3, 1]"`
16. `["4", "1", "7"] + ["3", "1"]`
17. `list('coca') + ['c', 'o', 'l', 'a']`

min and max

A list is a sequence of elements, and as such we can pass it to functions `min` or `max` for finding respectively the minimum or the maximum element of the list.

```
[108]: min([4, 5, 3, 7, 8, 6])
```

```
[108]: 3
```

```
[109]: max([4, 5, 3, 7, 8, 6])
```

```
[109]: 8
```

V COMMANDMENT¹⁵² : You shall never use `min` and `max` as variable names

If you do, you will lose the functions!

Note it's also possible to directly pass to `min` and `max` the elements to compare without including them in a list:

```
[110]: min(4, 5, 3, 7, 8, 6)
```

```
[110]: 3
```

```
[111]: max(4, 5, 3, 7, 8, 6)
```

¹⁵² <https://en.softpython.org/commandments.html#V-COMMANDMENT>

```
[111]: 8
```

But if we pass only one, without including it in a list, we will get an error:

```
min(4)
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-156-bb3db472b52e> in <module>
----> 1 min(4)

TypeError: 'int' object is not iterable
```

The error tells us that when we pass only an argument, Python expects a sequence like a list:

```
[112]: min([4])
[112]: 4
```

To min and max we can also pass strings, and we will get the character which is alphabetically lesser or greater:

```
[113]: min("orchestra")
[113]: 'a'

[114]: max("orchestra")
[114]: 't'
```

If we pass a list of strings, we will obtain the lesser or greater string in lexicographical order (i.e. the phonebook order)

```
[115]: min(['the', 'sailor', 'walks', 'around', 'the', 'docks'])
[115]: 'around'

[116]: max(['the', 'sailor', 'walks', 'around', 'the', 'docks'])
[116]: 'walks'
```

QUESTION: For each of the following expressions, try guessing the result (or if it gives an error)

1. `max(7)`

2. `max([7])`

3. `max([5, 4, 6, 2])`

4. `max([min([7, 3])])`

5. `max([])`

6. `max(2, 9, 3)`

7. `max([3, 2, 5] + [9, 2, 3])`

8. `max(max([3, 2, 5], max([9, 2, 3])))`

9. `max(min(3, 6), min(8, 2))`
10. `min(max(3, 6), max(8, 2))`
11. `max(['a', 'b', 'd', 'c'])`
12. `max(['boat', 'dice', 'aloha', 'circle'])`
13. `min(['void', '', 'null', 'nada'])`
14. `max(['hammer'[-1], 'socket'[-1], 'wrench'[-1]])`
15. `min(['hammer'[-1], 'socket'[-1], 'wrench'[-1]])`

sum

With `sum` we can sum all the elements in a list:

```
[117]: sum([1, 2, 3])
```

```
6
```

```
[118]: sum([1.0, 2.0, 0.14])
```

```
3.14
```

WARNING: DO NOT use `sum` as variable name!

If you do, you will lose the function with **very bad consequences**¹⁵³!

QUESTION: For each of the following expressions, try guessing the result (or if it gives an error):

1. `sum[3, 1, 2]`
2. `sum(1, 2, 3)`
3. `la = [1, 2, 3]
sum(la) > max(la)`
4. `la = [1, 2, 3]
sum(la) > max(la)*len(la)`
5. `la = [4, 2, 6, 4, 7]
lb = [max(la), min(la), max(la)]
print(max(lb) != max(la))`

¹⁵³ <https://en.softpython.org/commandments.html#V-COMMANDMENT>

Exercise - balance

Given a list of n numbers `balance` with n even, write some code which prints `True` if the sum of all first $n/2$ numbers is equal to the sum of all successive ones.

- your code must work for *any* number list

Example 1 - given:

```
balance = [4, 3, 7, 1, 5, 8]
```

after your code, it must print:

```
True
```

Example 2 - given:

```
balance = [4, 3, 3, 1, 9, 8]
```

after your code, it must print:

```
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[119]:

```
balance = [4, 3, 7, 1, 5, 8]
#balance = [4, 3, 3, 1, 9, 8]

# write here
n = len(balance)
sum(balance[:n//2]) == sum(balance[n//2:])
```

[119]:

```
True
```

```
</div>
```

[119]:

```
balance = [4, 3, 7, 1, 5, 8]
#balance = [4, 3, 3, 1, 9, 8]

# write here
```

List replication

To replicate the elements of a list, it's possible to use the operator `*` which produces a NEW list:

[120]:

```
[7, 6, 8] * 2
```

[120]:

```
[7, 6, 8, 7, 6, 8]
```

[121]:

```
[7, 6, 8] * 3
```

[121]:

```
[7, 6, 8, 7, 6, 8, 7, 6, 8]
```

Note a NEW list is produced, and the original one is not modified:

```
[122]: la = [7, 6, 8]
[123]: lb = [7, 6, 8] * 3
[124]: la    # original
[124]: [7, 6, 8]
[125]: lb    # expression result
[125]: [7, 6, 8, 7, 6, 8, 7, 6, 8]
```

We can multiply a list of strings:

```
[126]: la = ["a", "world", "of", "words"]
[127]: lb = la * 2
[128]: print(la)
['a', 'world', 'of', 'words']
[129]: print(lb)
['a', 'world', 'of', 'words', 'a', 'world', 'of', 'words']
```

As long as we multiply lists which contain immutable elements like numbers or strings, no particular problems arise:

```
[130]: la = ["a", "world", "of", "words"]
lb = la * 2
jupman.pytut()
[130]: <IPython.core.display.HTML object>
```

The matter becomes much more sophisticated when we multiply lists which contain mutable objects like other lists. Let's see an example:

```
[131]: la = [5, 6]
lb = [7, 8, 9]
lc = [la, lb] * 2
[132]: print(la)
[5, 6]
[133]: print(lb)
[7, 8, 9]
[134]: print(lc)
[[5, 6], [7, 8, 9], [5, 6], [7, 8, 9]]
```

By printing it, we see that the lists `la` and `lb` are represented inside `lc` - but how, exactly? `print` calls may trick you about the effective state of memory - to investigate further it's convenient to use Python Tutor:

```
[135]: la = [5, 6]
lb = [7, 8, 9]
lc = [la, lb] * 2

jupman.pytut()

[135]: <IPython.core.display.HTML object>
```

Arggh ! A jungle of arrows will appear ! This happens because when we write `[la, lb]` we create a list with two *references* to other lists `[5, 6]` and `[7, 8, 9]`, and the operator `*` when duplicating it just copies *references*.

For now we stop here, we will see the implications details later in the tutorial [matrices - lists of lists](#)¹⁵⁴

Equality

We can check whether two lists are equal with equality operator `==`, which given two lists returns `True` if they contain equal elements or `False` otherwise:

```
[136]: [4, 3, 6] == [4, 3, 6]
```

```
[136]: True
```

```
[137]: [4, 3, 6] == [4, 3]
```

```
[137]: False
```

```
[138]: [4, 3, 6] == [4, 3, 6, 'ciao']
```

```
[138]: False
```

```
[139]: [4, 3, 6] == [2, 2, 8]
```

```
[139]: False
```

We can check equality of lists with heterogenous elements:

```
[140]: ['apples', 3, ['cherries', 2], 6] == ['apples', 3, ['cherries', 2], 6]
```

```
[140]: True
```

```
[141]: ['bananas', 3, ['cherries', 2], 6] == ['apples', 3, ['cherries', 2], 6]
```

```
[141]: False
```

To check for inequality, we can use the operatpr `!=`:

```
[142]: [2, 2, 8] != [2, 2, 8]
```

```
[142]: False
```

```
[143]: [4, 6, 0] != [2, 2, 8]
```

```
[143]: True
```

```
[144]: [4, 6, 0] != [4, 6, 0, 2]
```

¹⁵⁴ <https://en.softpython.org/matrices-lists/matrices-lists1-sol.html>

[144] : True

QUESTION: For each of the following expressions, guess whether it is `True`, `False` or it produces an error:

1. `[2, 3, 1] != [2, 3, 1]`

2. `[4, 8, 12] == [2*2, 4*2, 6*2]`

3. `[7, 8][:] == [7, 9-1]`

4. `[7][0] == [[7]][0]`

5. `[9] == [9][0]`

6. `[max(7, 9)] == [max([7]), max([9])]`

7. `['a', 'b', 'c'] == ['A', 'B', 'C']`

8. `['a', 'b'] != ['a', 'b', 'c']`

9. `["ciao"] != ["CIAO".lower()]`

10. `[True in [True]] != [False]`

11. `[][:] == []`

12. `[[]] == [] + []`

13. `[[], []] == [] + []`

14. `[[[[]]] == [[[[]+[]]]]`

Continue

You can find more exercise in the notebook [Lists 3 - basic methods](#)¹⁵⁵

[] :

5.3.3 Lists 3 - Basic methods

Download exercises zip

Browse files online¹⁵⁶

Lists are objects of type `list` and have several methods for performing operations on them, let's see the basic ones:

¹⁵⁵ <https://en.softpython.org/lists/lists3-sol.html>

¹⁵⁶ <https://github.com/DavidLeoni/softpython-en/tree/master/lists>

Method	Returns	Description
<code>list.append(obj)</code>	None	Adds a new element at the end of the list
<code>list1.extend(list2)</code>	None	Adds many elements at the end of the list
<code>list.insert(int,obj)</code>	None	Adds a new element into some given position
<code>list.pop()</code>	obj	Removes and return the element at last position
<code>list.pop(int)</code>	obj	Given an index, removes and return the element at that position
<code>list.reverse()</code>	None	Inverts the order of elements
<code>list.sort()</code>	None	Sorts the elements <i>in-place</i>
<code>"sep".join(seq)</code>	string	produces a string concatenating all the elements in seq separated by "sep"
<code>list.copy()</code>	list	Copia superficialmente la list

The others are described at the page [Search methods](#)¹⁵⁷

WARNING 1: LIST METHODS *MODIFY* THE LIST ON WHICH ARE CALLED !

Whenever you call a method of a list (the object to the left of the dot .), you MODIFY the list itself (differently from string methods which always generate a new string without changing the original)

WARNING 2: LIST METHODS RETURN NOTHING!

They almost always return the object None (differently from strings which always return a new string)

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
lists
lists1.ipynb
lists1-sol.ipynb
lists2.ipynb
lists2-sol.ipynb
lists3.ipynb
lists3-sol.ipynb
lists4.ipynb
lists4-sol.ipynb
lists5-chal.ipynb
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `lists3.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter

¹⁵⁷ <https://en.softpython.org/lists/lists4-sol.html>

- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

append method

We can MODIFY a list adding a single element at a time with the method `append`.

Suppose to start from an empty list:

[2]: `la = []`

If we want to add as element the number 50, we can write like this:

[3]: `la.append(50)`

Note the list we initialilly created got MODIFIED:

[4]: `la`

[4]: `[50]`

WARNING: `la.append(50)` returned NOTHING !!!!

Observe carefully the output of cell with instruction `la.append(50)`, you will notice there is absolutely nothing. This happens because the purpose of `append` is to MODIFY the list on which it is called, NOT generating new lists.

We append another number *at the end* of the list:

[5]: `la.append(90)`

[6]: `la`

[6]: `[50, 90]`

[7]: `la.append(70)`

[8]: `la`

[8]: `[50, 90, 70]`

Let's see what happened in Python Tutor:

```
[9]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
#           (it's sufficient to execute it only once)
```

```
import jupman
```

```
[10]: la = []
       la.append(50)
       la.append(90)
       la.append(70)

       jupman.pytut()
```

```
[10]: <IPython.core.display.HTML object>
```

Note there is only one yellow memory region associated to variable `la` which gets expanded as you click on Next.

We said `append` method returns nothing, let's try to add some detail. In the methods table, there is present a column named *Returns*. If you check it, for almost all methods included `append` there is indicated it returns `None`.

`None` is the most boring object in Python, because it literally means nothing. What can you do with nothing? Very few things, so few that whenever Jupyter finds as result the `None` object it doesn't even print it. Try directly inserting `None` in a cell, you will see it won't be reported in cell output:

```
[11]: None
```

A way to force the print is by using the command `print`:

```
[12]: print(None)
```

```
None
```

EXERCISE: What is the type of the object `None`? Discover it by using the function `type`

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
```

```
data-jupman-show="Show solution"
```

```
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[13]:
```

```
# write here
```

```
type(None)
```

```
[13]: NoneType
```

```
</div>
```

```
[13]:
```

```
# write here
```

Let's try repeating what happens with `append`. If you call the method `append` on a list, `append` silently MODIFIES the list, and RETURNS the object `None` as call result. Notice that Jupyter considers this object as non-interesting, so it doesn't even print it.

Let's try to get explicit about this mysterious `None`. If it's true that `append` produces it as call result, it means we can associate this result to some variable. Let's try to associate it to variable `x`:

```
[14]: la = []
x = la.append(70)
```

Now, if everything went as we wrote, `append` should have modified the list:

```
[15]: la
```

```
[15]: [70]
```

and there should be associated `None` to variable `x`. So, if we ask Jupyter to show the value associated to `x` and that value is `None`, nothing will appear:

```
[16]: x
```

Note there is no output in the cell, apparently we are really in presence of a `None`. Let's force the `print`:

[17]: `print(x)`

None

Here it is! Probably you will be a little confused by all of this, so let's check again what happens in Python Tutor:

[18]: `la = []
x = la.append(70)
print("la is", la)
print("x is", x)`

jupman.pytut()

la is [70]
x is None

[18]: <IPython.core.display.HTML object>

What's the final gist?

REUSING THE RESULT OF LIST METHODS CALLS IS ALMOST ALWAYS AN ERROR !

Since calling list methods returns `None`, which is a 'useless' object, trying to reuse it will almost surely produce an error

EXERCISE: Build a list by adding one element at a time with the method `append`. Add the elements `77, "test", [60, 93]` with three calls to `append`, and finally print the list.

After your code, you should see `[77, 'test', [60, 93]]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[19]:

```
la = []

# write here
la.append(77)
la.append("test")
la.append([60, 93])

print(la)
```

[77, 'test', [60, 93]]

</div>

[19]:

```
la = []

# write here
```

QUESTION: The following code:

```
la = []
la.append(80, 70, 90)
```

1. produces an error (which one?)
2. modifies the list (how?)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 1: append accepts only one argument, by passing more than one will produce an error, to see which try to execute the code in a cell.

```
</div>
```

QUESTION: The following code:

```
la = []
la.append(80).append(90)
```

1. produces an error
2. appends to la the numbers 80 and 90

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 1: produces an error, because we said the call to `la.append(80)` MODIFIES the list `la` on which it is called and return the value `None`. If on `None` we try calling `.append(90)`, since `None` is not a list we will get an error message. Make sure of this using Python Tutor.

```
</div>
```

QUESTION: let's briefly go back to strings. Look at the following code (if you don't remember what string methods do see here¹⁵⁸)

```
sa = '    trento    '
sb = sa.strip().capitalize()
print(sb)
```

1. produces an error (which one?)
2. changes `sa` (how?)
3. prints something (what?)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 3: prints `Trento`. Differently from lists, the strings are *immutable* sequences: this means that when you call a method of strings you are sure it will RETURN a NEW string. So the first call to `sa.strip()` RETURNS the string without spaces at beginning and end of '`trento`', and on this string the method `capitalize()` is called to make the first character uppercase.

If this is not clear to you, try to executing the following code in Python Tutor. It is equivalent to the one in the example but it explicitly shows the passage by assigning the result of calling `sa.strip()` to the extra variable `x`

```
sa = '    trento    '
x = sa.strip()
sb = x.capitalize()
print(sb)
```

```
</div>
```

QUESTION: Have a look at this code. Will it print something at the end? Or will it produce an error?

¹⁵⁸ <https://en.softpython.org/strings/strings3-sol.html#Methods>

```

la = []
lb = []
la.append(lb)

lb.append(90)
lb.append(70)

print(la)

```

 Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: It will print [[90, 70]], because we put lb inside la.

Even if with first append we added lb as first element of la, afterwards it is perfectly legit keeping on modifying lb by calling lb.append(90).

Try executing the code in Python Tutor, and see the arrows.

</div>

Exercise - augmenting a list 1

Given the list la of *fixed dimension 7*, write some code to augment the empty list lb so to *only* contain the elements of la with even index (0, 2, 4, ...).

- Your code should work with *any* list la of fixed dimension 7

```

#   0 1 2 3 4 5 6
la=[8,4,3,5,7,3,5]
lb=[]

```

After your code, you should obtain:

```

>>> print(lb)
[8,3,7,5]

```

 Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[20]:

```

#   0 1 2 3 4 5 6
la=[8,4,3,5,7,3,5]
lb=[]

# write here
lb.append(la[0])
lb.append(la[2])
lb.append(la[4])
lb.append(la[6])
print(lb)

```

[8, 3, 7, 5]

</div>

[20]:

```
#      0 1 2 3 4 5 6
la=[8,4,3,5,7,3,5]
lb=[]

# write here
```

extend method

We've seen that with `append` we can augment a list *one element at a time*.

What if we wanted to add many elements in a single shot, maybe taken from another list?

We should use the method `extend`, which MODIFIES the list on which it is called by adding all the elements it finds in the input sequence.

[21]: la = [70, 30, 50]

[22]: lb = [40, 90, 30, 80]

[23]: la.extend(lb)

[24]: la

[24]: [70, 30, 50, 40, 90, 30, 80]

[25]: lb

[25]: [40, 90, 30, 80]

In the example above, `extend` is called on the variable `la`, and we passed `lb` as parameter

WARNING: `la` is MODIFIED, but the sequence we passed in round parenthesis is not (`lb` in the example)

QUESTION: the execution of method `extend` returns something? What do you see in the output of cell `la.extend(lb)`?

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: `extend`, as all list methods, doesn't return anything. To be more explicit, it returns the object `None`, which is not even printed by Jupyter.

</div>

Let's verify what happened with Python Tutor:

[26]: la = [70, 30, 50]
lb = [40, 90, 30, 80]
la.extend(lb)

jupman.pytut()

[26]: <IPython.core.display.HTML object>

QUESTION: Look inside this code. Which will be the values associated to variables `la`, `lb` and `x` after its execution?

```
la = [30, 70, 50]
lb = [80, 40]
x = la.extend(lb)

print('la is ', la)
print('lb is ', lb)
print('x is ', x)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: It will print this:

```
la is [30, 70, 50, 80, 40]
lb is [80, 40]
x is None
```

`la` was MODIFIED by adding all the elements of `lb`.

The call to `extend`, like all list methods, returned the object `None` which was associated to variable `x`. Try to understand well what happened by using Python Tutor.

</div>

Extending with sequences

We said that `extend` can take any generic sequence in the round parenthesis, not only lists. This means we can also try to pass a string. For example:

[27]:

```
la = [70, 60, 80]

s = "hello"

la.extend(s)
```

[28]:

[28]:

```
la
[70, 60, 80, 'h', 'e', 'l', 'l', 'o']
```

Since the string is a character sequence, `extend` took each of these elements and added them to `la`

QUESTION: was the value associated to variable `s` modified?

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: absolutely impossible, because a) `extend` only modifies the list on which it is called and b) strings are immutable anyway.

</div>

QUESTION: The following code:

```
la = [60, 50]
la.extend(70, 90, 80)
```

1. produces un error (which one?)
2. modifies la (how?)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 1: produces an error, because we have to pass a SINGLE parameter to `extend`, which must be a *sequence*. Here instead we are passing many parameters. An alternative might be to build a list like this:

```
la = [60, 50]
la.extend([70, 90, 80])
```

```
</div>
```

QUESTION: If this code is executed, what happens?

```
sa = "hello"
sb = "world"
sa.extend(sb)
```

1. sa is modified (how?)
2. we get an error (which one?)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 2: we obtain an error, because `extend` is an exclusive method of lists. It only belongs to lists because MODIFIES the object on which it is called - since strings are immutable objects, it wouldn't make sense to change them.

```
</div>
```

QUESTION: If this code is executed, what happens?

```
la = [1, 2, 3]
lb = [4, 5]
lc = [6, 7, 8]

la.extend(lb).extend(lc)
```

1. la becomes [1, 2, 3, 4, 5, 6, 7, 8]
2. an error (which one?)
3. la becomes [1, 2, 3, 4, 5] and an error (which one?)

QUESTION: 3: la becomes [1, 2, 3, 4, 5] and right after we get an error, because the call to `la.extend(lb)` MODIFIES la to [1, 2, 3, 4, 5] and RETURN the value None. At that point, Python tries to call the method `extend` on the object None, but since it is not a list, we get an error (**to convince yourself, verify everything with Python Tutor !!!**)

```
-----
AttributeError                                                 Traceback (most recent call last)
<ipython-input-45-0a08a154ada4> in <module>
      3 lc = [6, 7, 8]
      4
----> 5 la.extend(lb).extend(lc)

AttributeError: 'NoneType' object has no attribute 'extend'
```

Exercise: augmenting a list 2

Given two *lists* `la` and `lb` and an element `x`, write some code to MODIFY `la` so that `la` contains at the end the element `x` followed by all other elements of `lb`

- **NOTE 1:** your code should work with any `la` and `lb`
- **NOTE 2:** `id` is a Python function which associates to each memory region a unique identifier. If you try printing `id(la)` before modifying `la` and `id(la)` afterwards, you should obtain *exactly* the same id. If you obtain a different one, it means you generated an entirely new list. In that case, verify how it's working with Python Tutor.

```
la = [5, 9, 2, 4]
lb = [7, 1, 3]
x = 8
```

You should obtain:

```
>>> print(la)
[5, 9, 2, 4, 8, 7, 1, 3]
>>> print(lb)
[7, 1, 3]
>>> print(x)
8
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[29]:
```

```
la = [5, 9, 2, 4]
lb = [7, 1, 3]
x = 8

# write here
la.append(x)
la.extend(lb)
print(la)
print(lb)
print(x)
```

```
[5, 9, 2, 4, 8, 7, 1, 3]
[7, 1, 3]
8
```

</div>

```
[29]:
```

```
la = [5, 9, 2, 4]
lb = [7, 1, 3]
x = 8

# write here
```

Exercise - zslice

Write some code which given two lists `la` (of at least 3 elements) and `lb`, MODIFIES `lb` in such a way to add 3 elements of `la` followed by the last 3 of `la`.

- your code must work with any list
- use extends and slices

```
la = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o']
lb = ['z']
```

You should obtain:

```
>>> print(la)
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o']
>>> print(lb)
['z', 'a', 'b', 'c', 'm', 'n', 'o']
```

[Show solution](#)[Hide](#)</div>

[30]:

```
la = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o']
lb = ['z']

# write here

lb.extend(la[:3]) # a slice generates a list
lb.extend(la[-3:])

print(la)
print(lb)
```

```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o']
['z', 'a', 'b', 'c', 'm', 'n', 'o']
```

</div>

[30]:

```
la = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'l', 'm', 'n', 'o']
lb = ['z']

# write here
```

Exercise - Zebarerun

Write some code which given a list of three strings `words` and an empty list `la`, fills `la` with all the first 3 characters of every string in `words`.

- your code must work with any list of 3 strings
- use slices

Example given:

```
words = ["Zebras", "are", "running"]
la = []
```

Your code must show:

```
>>> print(t)
['Z', 'e', 'b', 'a', 'r', 'e', 'r', 'u', 'n']
```

[Show solution](#)

>

[31]:

```
words = ["Zebras", "are", "running"]

la = []

# write here
la.extend(words[0][:3])
la.extend(words[1][:3])
la.extend(words[2][:3])
print(la)

['Z', 'e', 'b', 'a', 'r', 'e', 'r', 'u', 'n']
```

</div>

[31]:

```
words = ["Zebras", "are", "running"]

la = []

# write here
```

insert method

`insert` MODIFIES the list by inserting an element at a specific index - all elements starting from that index will be shifted of one position to the right.

[32]:

```
#0 1 2 3
la = [6, 7, 8, 9]
```

[33]:

```
la.insert(2, 55) # insert the number 55 at index 2
```

```
[34]: la  
[34]: [6, 7, 55, 8, 9]
```

```
[35]: la.insert(0,77) # insert the number 77 at index 0
```

```
[36]: la  
[36]: [77, 6, 7, 55, 8, 9]
```

We can insert after the end:

```
[37]: la.insert(6,88) # insert the number 88 at index 6
```

```
[38]: la  
[38]: [77, 6, 7, 55, 8, 9, 88]
```

Note that if we go beyond the end, the element is placed right after the end and no empty cells are created:

```
[39]: la.insert(1000,99) # insert number 99 at index 7
```

```
[40]: la  
[40]: [77, 6, 7, 55, 8, 9, 88, 99]
```

QUESTION: Given any list x , what does this code produce? Can we rewrite it in some other way?

```
x.insert(len(x), 66)
```

1. produces a new list (which one?)
2. modifies x (how?)
3. an error

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 2 - the code MODIFIES the list x by adding the element 66 at the end. The code is then equivalent to code

```
x.append(66)
```

```
</div>
```

QUESTION: What does the following code produce?

```
la = [3, 4, 5, 6]  
la.insert(0, [1, 2])  
print(la)
```

1. prints $[1, 2, 3, 4, 5, 6]$
2. an error (which one?)
3. something else (what?)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 3 - the code inserts in `la` the list `[1, 2]` as zero-th element. The print will then show `[[1, 2], 3, 4, 5, 6]`

</div>

QUESTION: What does the following code produce?

```
la = [4, 5, 6]
la.insert(0, 1, 2, 3)
print(la)
```

1. prints `[1, 2, 3, 4, 5, 6]`
2. an error (which one?)
3. something else (what?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 2 - an error, because we can only pass 2 parameters to `insert`, the insertion index and the single object to insert.

</div>

QUESTION: What does the following code produce?

```
la = [4, 5, 6]
lb = la.insert(0, 3)
lc = lb.insert(0, 2)
ld = lc.insert(0, 1)
print(ld)
```

1. prints `[1, 2, 3, 4, 5, 6]`
2. an error (which one?)
3. something else (what?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 2 - an error: like almost all list methods, `insert` returns `None`, so by writing `lb = la.insert(0, 3)` we are associating `None` to `lb`, so when Python in the next line encounters `lc = lb.insert(0, 2)` and tries to execute `None.insert(0, 2)` it will complain because `None` not being a list doesn't have the `insert` method.

</div>

Exercise - insertando

Given the list

```
la = [7, 6, 8, 5, 6]
```

write some code which MODIFIES the list by using only calls to `insert`. After your code, `la` should appear like this:

```
>>> print(la)
[7, 70, 90, 6, 8, 80, 5, 6, 50]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[41]:

```
la = [7, 6, 8, 5, 6]

# write here

la.insert(3, 80)
la.insert(1, 90)
la.insert(1, 70)
la.insert(len(la), 50)

print(la)
```

```
[7, 70, 90, 6, 8, 80, 5, 6, 50]
```

```
</div>
```

[41]:

```
la = [7, 6, 8, 5, 6]

# write here
```

WARNING: calling insert is much slower than append !!

A call to `insert` rewrites all the cells after the insertion point, while `append` instead adds only one cell. Given the computer is fast, very often we don't realize the difference, but whenever possible try writing code using `append` instead of `insert`, especially if you have to write programs which operate on big amounts of data.

Exercise - insappend

This code takes as input an empty list `la` and a list of numbers `lb`. Try to understand what it does, and rewrite it using some `append`.

[42]:

```
la = []
lb = [7, 6, 9, 8]
la.insert(0, lb[0]*2)
la.insert(0, lb[1]*2)
la.insert(0, lb[2]*2)
la.insert(0, lb[3]*2)
print(la)
```

```
[16, 18, 12, 14]
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[43]:

```
la = []
lb = [7, 6, 9, 8]

# write here
la.append(lb[-1]*2)
la.append(lb[-2]*2)
```

(continues on next page)

(continued from previous page)

```
la.append(lb[-3]*2)
la.append(lb[-4]*2)
print(la)

[16, 18, 12, 14]
```

</div>

```
[43]: la = []
lb = [7, 6, 9, 8]

# write here
```

pop method

pop method does two things: when called without arguments MODIFIES the list by removing the last element, and also RETURNS the removed element:

```
[44]: basket = ['melon', 'strawberry', 'apple']
```

```
[45]: basket.pop()
```

```
[45]: 'apple'
```

```
[46]: basket
```

```
[46]: ['melon', 'strawberry']
```

```
[47]: basket.pop()
```

```
[47]: 'strawberry'
```

```
[48]: basket
```

```
[48]: ['melon']
```

Since the last element is *returned* by pop, we can also assign it to a variable:

```
[49]: fruit = basket.pop()
```

Note we don't see no result printed because the returned element was assigned to the variable fruit:

```
[50]: fruit
```

```
[50]: 'melon'
```

We also notice that basket was MODIFIED indeed:

```
[51]: basket
```

```
[51]: []
```

If you further call pop on an empty list you will get an error:

```
basket.pop()  
-----  
IndexError Traceback (most recent call last)  
<ipython-input-67-086f38c9fbco> in <module>()  
----> 1 basket.pop()  
  
IndexError: pop from empty list
```

Optionally, to remove an element from a specific position we can pass `pop` an index from 0 INCLUDED to the length of the list EXCLUDED:

```
[52]: #          0          1          2          3  
       tools = ['hammer', 'screwdriver', 'plier', 'hammer']  
  
[53]: tools.pop(2)  
[53]: 'plier'  
  
[54]: tools  
[54]: ['hammer', 'screwdriver', 'hammer']
```

QUESTION: Have a look at following code snippets, and for each of them try to guess the result it produces (or if it gives an error):

1.

```
la = ['a']  
print(la.pop())  
print(la.pop())
```

2.

```
la = [4, 3, 2, 1]  
print(la.pop(4))  
print(la)
```

3.

```
la = [1, 2, 3, 4]  
print(la.pop(3))  
print(la)
```

4.

```
la = [1, 2, 3, 4]  
print(la.pop(-1))  
print(la)
```

5.

```
s = 'raw'  
print(s.pop())  
print(s)
```

6.

```
la = ['so', 'raw']  
print(la.pop())  
print(la)
```

7.

```
la = ['a', ['a']]  
print(la.pop())  
print(la)
```

Exercise - popcorn

Given the list `corn` of exactly 4 characters, write some code which transfers in reverse order all the characters from `corn` to another list `box` which is initially empty.

- **DO NOT** use methods like `reverse` or functions like `reversed`
- Your code must work with *any* list `corn` of 4 elements

Example - given:

```
corn = ['G', 'u', 'r', 'u']
box = []
```

after your code, it must result:

```
>>> print(corn)
[]
>>> print(box)
['u', 'r', 'u', 'G']
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[55]:

```
corn = ['G', 'u', 'r', 'u']
box = []

# write here

box.append(corn.pop())
box.append(corn.pop())
box.append(corn.pop())
box.append(corn.pop())
print(box)

['u', 'r', 'u', 'G']
```

</div>

[55]:

```
corn = ['G', 'u', 'r', 'u']
box = []

# write here
```

Exercise - zonzo

Given a list `la` containing some characters, and a list `lb` containing exactly two positions *in ascending order*, write some code which eliminates from `la` the characters at positions specified in `lb`.

- **WARNING:** by calling `pop` the first time you will MODIFY `la`, so the index from the second element to eliminate will need to be properly adjusted !
- **DO NOT** create new lists, so no rows beginning with `la =`
- Your code must work with *any* `la` and *any* `lb` of two elements

Example - given:

```
#      0   1   2   3   4
la = ['z', 'o', 'n', 'z', 'o']
lb = [2, 4]
```

at position 2 in `la` we find the `n` and at 4th the `o`, so after your code it must result:

```
>>> print(la)
['z', 'o', 'z']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[56]:

```
#      0   1   2   3   4
la = ['z', 'o', 'n', 'z', 'o']
lb = [2, 4]

# write here
la.pop(lb[0])
la.pop(lb[1]-1)
print(la)
```

```
['z', 'o', 'z']
```

```
</div>
```

[56]:

```
#      0   1   2   3   4
la = ['z', 'o', 'n', 'z', 'o']
lb = [2, 4]

# write here
```

reverse method

`reverse` method MODIFIES the list on which it is called by inverting the order of elements.

Let's see an example:

```
[57]: la = [7, 6, 8, 4]
```

```
[58]: la.reverse()
```

```
[59]: la
```

```
[59]: [4, 8, 6, 7]
```

WARNING: reverse RETURNS NOTHING!

To be precise, it returns None

```
[60]: lb = [7, 6, 8, 4]
```

```
[61]: x = lb.reverse()
```

```
[62]: print(x)
```

```
None
```

```
[63]: print(lb)
```

```
[4, 8, 6, 7]
```

QUESTION: Which effect does the following code produce?

```
s = "transatlantic"
s.reverse()
print(s)
```

1. an error (which one?)
2. prints the string in reverse

 Show answer <div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: `.reverse()` is a method ONLY present in LISTS, so by using it on strings we will get an error. And we have to expect it, as `reverse` MODIFIES the object on which it is called and since strings are *immutable* no string method can possibly modify the string on which it is called.

</div>

QUESTION: If `x` is some list, which effect does the following produce?

```
x.reverse().reverse()
```

1. changes the list (how?)
2. it doesn't change the list
3. generates an error (which one?)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 3 - generates an error, because `reverse()` returns `None` which is not a list so it doesn't have `reverse()` method.

```
</div>
```

Exercise - good manners

Write some code which given two lists `la` and `lb` MODIFY `la` adding all the elements of `lb` and then reversing the whole list.

- your code must work with any `la` and `lb`
- **DO NOT** modify `lb`

Example - given:

```
la = ['g', 'o', 'o', 'd']
lb = ['m', 'a', 'n', 'n', 'e', 'r', 's']
```

After your code, it must print:

```
>>> print('la=', la)
la= ['s', 'r', 'e', 'n', 'a', 'm', 'd', 'o', 'o', 'g']
>>> print('lb=', lb)
lb= ['m', 'a', 'n', 'n', 'e', 'r', 's']
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[64]:

```
la = ['g', 'o', 'o', 'd']
lb = ['m', 'a', 'n', 'n', 'e', 'r', 's']

# write here
la.extend(lb)
la.reverse()
print('la=', la)
print('lb=', lb)

la= ['s', 'r', 'e', 'n', 'a', 'm', 'd', 'o', 'o', 'g']
lb= ['m', 'a', 'n', 'n', 'e', 'r', 's']
```

```
</div>
```

[64]:

```
la = ['g', 'o', 'o', 'd']
lb = ['m', 'a', 'n', 'n', 'e', 'r', 's']

# write here
```

Exercise - precious things

Given two lists `la` and `lb` write some code which PRINTS a list with the elements of `la` and `lb` in reverse order.

- **DO NOT** modify `la` and **DO NOT** modify `lb`
- your code must work with any list `la` and `lb`

Example - given:

```
la = ['p', 'r', 'e', 'c', 'i', 'o', 'u', 's']
lb = ['t', 'h', 'i', 'n', 'g', 's']
```

After your code, it must print:

```
['s', 'g', 'n', 'i', 'h', 't', 's', 'u', 'o', 'i', 'c', 'e', 'r', 'p']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[65]: la = ['p', 'r', 'e', 'c', 'i', 'o', 'u', 's']
       lb = ['t', 'h', 'i', 'n', 'g', 's']

       # write here
       lc = la + lb  # the + creates a NEW list
       lc.reverse()
       print(lc)

['s', 'g', 'n', 'i', 'h', 't', 's', 'u', 'o', 'i', 'c', 'e', 'r', 'p']
```

</div>

```
[65]: la = ['p', 'r', 'e', 'c', 'i', 'o', 'u', 's']
       lb = ['t', 'h', 'i', 'n', 'g', 's']

       # write here
```

Exercise - powers

The following code uses some `insert` which as we already said it is not very efficient. Try to understand what it does, and rewrite it using only `append` and `reverse`

- your code must work for any value of `x`

```
[66]: x = 2
       la = [x]
       la.insert(0,la[0]**2)
       la.insert(0,la[0]**2)
       la.insert(0,la[0]**2)
       la.insert(0,la[0]**2)
       print(la)

[32, 16, 8, 4, 2]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[67]:

```
x = 2
la = [x]

# write here
la.append(la[-1]*2)
la.append(la[-1]*2)
la.append(la[-1]*2)
la.append(la[-1]*2)
la.reverse()
print(la)
```

```
[32, 16, 8, 4, 2]
```

</div>

[67]:

```
x = 2
la = [x]

# write here
```

sort method

If a list contains homogenous elements, it is possible to sort it rapidly with the `sort` method, which MODIFIES the list on which it is called (also called sorting *in-place*):

[68]:

```
la = [8, 6, 7, 9]
```

[69]:

```
la.sort() # NOTE: sort returns nothing !!!
```

[70]:

```
la
```

[70]:

```
[6, 7, 8, 9]
```

Strings are also sortable¹⁵⁹

[71]:

```
lb = ['Boccaccio', 'Alighieri', 'Manzoni', 'Leopardi']
```

[72]:

```
lb.sort()
```

[73]:

```
lb
```

[73]:

```
['Alighieri', 'Boccaccio', 'Leopardi', 'Manzoni']
```

A list with non-comparable elements it's not sortable, and Python will complain:

[74]:

```
lc = [3, 4, 'cabbage', 7, 'potatoes']
```

¹⁵⁹ <https://en.softpython.org/strings/strings2-sol.html#Comparing-characters>

```
>>> lc.sort()

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-288-0cabfae30939> in <module>
----> 1 lc.sort()

TypeError: '<' not supported between instances of 'str' and 'int'
```

Optionally, for reverse order you can pass the parameter `reverse=True`:

```
[75]: la = [4, 2, 5, 3]
la.sort(reverse=True)
```

```
[76]: la
```

```
[76]: [5, 4, 3, 2]
```

Custom sorting

If you have custom needs like for example a lists of strings in the format '`name surname`' that you want to sort according to the surname, you should use optional parameter `key` with `lambda` functions, see [Python docs](#)¹⁶⁰

Exercise - numlist

Given the list `la = [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]`

1. finds the min, max and the median value (HINT: sort it and extract the right values)
2. create a list only with elements at even indexes (i.e. `[10, 72, 11, ..]`, note that “..” means the list is still not complete!) and ricalculates the values of min, max and median
3. redo the same with the elements at odd indexes (i.e. `[60, 118,..]`)

You should obtain:

```
original: [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]
even: [10, 72, 11, 56, 120]
odd: [60, 118, 71, 89, 175]

sorted: [10, 11, 56, 60, 71, 72, 89, 118, 120, 175]
sorted even: [10, 11, 56, 72, 120]
sorted odd: [60, 71, 89, 118, 175]

original: Min: 10 Max. 175 Median: 72
even: Min: 10 Max. 120 Median: 56
odd: Min: 60 Max. 175 Median: 89
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"< data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[77]:
```

(continues on next page)

¹⁶⁰ <https://docs.python.org/3/howto/sorting.html#key-functions>

(continued from previous page)

```
la = [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]

# write here

even = la[0::2]      # we take only elements at even indeces
odd = la[1::2]       # we take only elements at odd indeces

print("original:    " , la)
print("even:        " , even)
print("odd:         " , odd)

la.sort()
even.sort()
odd.sort()

print()
print("sorted:      " , la)
print("sorted even: " , even)
print("sorted odd:  " , odd)
print()
print("original:    Min:", la[0], " Max.", la[-1], " Median: ", la[len(la) // 2])
print("even:        Min:", even[0], " Max.", even[-1], " Median: ", even[len(even) // 
    ↵ 2])
print("odd:         Min:", odd[0], " Max.", odd[-1], " Median: ", odd[len(odd) // 2])

original:    [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]
even:        [10, 72, 11, 56, 120]
odd:         [60, 118, 71, 89, 175]

sorted:      [10, 11, 56, 60, 71, 72, 89, 118, 120, 175]
sorted even: [10, 11, 56, 72, 120]
sorted odd:  [60, 71, 89, 118, 175]

original:    Min: 10  Max. 175  Median: 72
even:        Min: 10  Max. 120  Median: 56
odd:         Min: 60  Max. 175  Median: 89
```

</div>

[77]:

```
la = [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]

# write here
```

join - build strings from lists

Given a string to use as separator, and a sequence like for example a list `la` which only contains strings, it's possible to concatenate them into a (new) string with `join` method:

```
[78]: la = ["When", "the", "sun", "raises"]

'SEPARATOR'.join(la)

[78]: 'WhenSEPARATORtheSEPARATORSunSEPARATORraises'
```

As separator we can put any character, like a space:

```
[79]: ' '.join(la)

[79]: 'When the sun raises'
```

Note the original list is not modified:

```
[80]: la

[80]: ['When', 'the', 'sun', 'raises']
```

QUESTION: What does this code produce?

```
' '.join(['a', 'b', 'c']).upper()
```

1. an error (which one?)
2. a string (which one?)
3. a list (which one?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 2: it produces the string 'ABC': first it takes all characters from the list `['a', 'b', 'c']` and it joins them with empty space '' separator to form 'abc', then this string is set all uppercase with `upper()`.

</div>

QUESTION: What does this code produce?

```
'a'.join('KRT') + 'E'
```

1. a string (which one?)
2. an error (which one?)
3. a list (which one?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 1: produces the string 'KaRaTE' - we said that `join` takes as input a sequence, so we are not bounded to pass lists but we can directly pass any string, which is a character sequence. `join` will then interval each character in the string with the separator we provide before the dot.

</div>

QUESTION: What does this code produce?

```
'\\''.join('mmmm')
```

1. an error (which one?)
2. a string (which one?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 2: `\\` is an escape sequence which represents the single character apex 'm', so we will obtain m'm'm'm

</div>

QUESTION: Given any string s and a list of strings la of at least two characters, the following code will always give us the same result - which one? (think about it, and if you don't know how to answer try putting random values for s and la)

```
len(s) <= len(s.join(la))
```

1. an errore (which one?)
2. a stringa (which one?)
3. something else (what?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 3: the code will always produce the boolean True because s.join(la) produces a string containing all the strings in la alternated with the string s. So the length of this string will always be greater or equal to the length of s: by comparing the two lengths with <= operator, we will always obtain the boolean True.

Example:

```
s = "ab"  
la = ['uiief', 'cb', 'sd']  
len(s) <= len(s.join(la))
```

</div>

Exercise - barzoletta

Given the string

```
sa = 'barzoletta'
```

write some code which creates a NEW string sb by changing the original string in such a way it results:

```
>>> print(sb)  
'barzelletta'
```

- **USE** the method `insert` and cell reassignment
- **NOTE:** you cannot use them an a string, because it is IMMUTABLE - you will then first convert the string to a list

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[81]:

```
sa = 'barzoletta'

# write here

la = list(sa)
la[4] = 'e'
la.insert(5, 'l')
sb = ''.join(la)
print(sb)
```

barzelletta

</div>

[81]:

```
sa = 'barzoletta'

# write here
```

Exercise - dub dab dib dob

Write some code which given a list of strings `la`, associates to variable `s` a string with the concatenated strings, separating them with a comma and a space.

Example - given:

```
la = ['dub', 'dab', 'dib', 'dob']
```

After your code, you should obtain this:

```
>>> print(s)
dub, dab, dib, dob
>>> len(s)
18
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[82]:

```
la = ['dub', 'dab', 'dib', 'dob']

# write here

s = ', '.join(la)

print(s)
len(s)
```

dub, dab, dib, dob

[82]:

18

</div>

[82]:

```
la = ['dub', 'dab', 'dib', 'dob']

# write here
```

Exercise - ghirgori

Given a list of strings `la` and a list with three separators `seps`, write some code which prints the elements of `la` separated by first separator, followed by the second separator, followed by the elements of `la` separated by the third separator.

- your code must work with any list `la` and `seps`

Example - given:

```
la = ['ghi', 'ri', 'go', 'ri']
seps = [',', '_', '+']
```

After your code, it must print:

```
ghi,ri,go,ri_ghi+ri+go+ri
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[83]:

```
la = ['ghi', 'ri', 'go', 'ri']
seps = [',', '_', '+']

# write here

print(seps[0].join(la) + seps[1] + seps[2].join(la))
```

```
ghi,ri,go,ri_ghi+ri+go+ri
```

```
</div>
```

[83]:

```
la = ['ghi', 'ri', 'go', 'ri']
seps = [',', '_', '+']

# write here
```

Exercise - welldone

Given the list:

```
la = ["walnut", "eggplant", "lemon", "lime", "date", "onion", "nectarine", "endive"]:
```

1. Create another list (call it new) containing the first character of every element of la
2. Add a space to new at position 4 and attach an exclamation mark (' ! ') at the end
3. Print the list
4. Print the list content by joining all elements with an empty space

You should get:

```
['w', 'e', 'l', 'l', ' ', 'd', 'o', 'n', 'e', '!']
```

well done!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[84]:

```
la = ["walnut", "eggplant", "lemon", "lime", "date", "onion", "nectarine", "endive"]

# write here

new = []
new.append(la[0][0])
new.append(la[1][0])
new.append(la[2][0])
new.append(la[3][0])
new.append(la[4][0])
new.append(la[5][0])
new.append(la[6][0])
new.append(la[7][0])

new.insert(4, " ")
new.append("!")

print(new)
print("\n", "".join(new))

['w', 'e', 'l', 'l', ' ', 'd', 'o', 'n', 'e', '!']

well done!
```

</div>

[84]:

```
la = ["walnut", "eggplant", "lemon", "lime", "date", "onion", "nectarine", "endive"]

# write here
```

copy method

If we want to copy a mutable data structure, we can use the `.copy()` method, which performs a so-called *shallow copy*. Let's see what it means.

Let's start with a simple case, for example a list of immutable objects like strings we can visualize in Python Tutor:

```
[85]: satellites = ["Hubble", "Sputnik 1"]

copia = satellites.copy()

jupman.pytut()

[85]: <IPython.core.display.HTML object>
```

We clearly see how a completely new memory region was created. If we later try to modify the copy, we will see the original is not changed:

```
[86]: satellites = ["Sputnik 1", "Hubble"]

my_copy = satellites.copy()

my_copy.append("James Webb")

jupman.pytut()

[86]: <IPython.core.display.HTML object>
```

copy is shallow

So far, we didn't notice any particular problem. But what happens if we try `copy()` on a list which contains other lists, in other words, other mutable elements, and then try mutating one of the two?

```
[87]: biglistA = [ ['Pay', 'attention'],
                 ['to', 'where'],
                 ['the', 'arrows', 'point to']
             ]

biglistB = biglistA.copy()

biglistA[2][0] = 'CAREFUL!' # we write into the original...

print(biglistA)
print(biglistB)

jupman.pytut()

[[['Pay', 'attention'], ['to', 'where'], ['CAREFUL!', 'arrows', 'point to']],
[['Pay', 'attention'], ['to', 'where'], ['CAREFUL!', 'arrows', 'point to']]]

[87]: <IPython.core.display.HTML object>
```

Note we have two big lists containing cells that point to shared sublists: as a matter of fact, by writing into a subcell of `biglistA` we also write into `biglistB`!

In other words, `.copy()` performs only a *shallow copy*, for a proper deep copy we will need to find some other way!

deepcopy function

To avoid sharing problems we can use the so-called deep copy, available in the function `deepcopy` from module `copy`

WARNING: `deepcopy` IS NOT a list method!

Let's try again the example with `copy.deepcopy`: we will now get completely distinct data structures:

```
[88]: # first we import `copy`, which is a PYTHON MODULE
import copy

biglistA = [ ['Pay', 'attention'],
            ['to', 'where'],
            ['the', 'arrows', 'point to']
        ]

# then we call its function deepcopy, passing the parameter biglistA:

biglistB = copy.deepcopy(biglistA)
biglistA[2][0] = 'CAREFUL!' # we write into the original...

print(biglistA)
print(biglistB)

jupman.pytut()

[['Pay', 'attention'], ['to', 'where'], ['CAREFUL!', 'arrows', 'point to']]
[['Pay', 'attention'], ['to', 'where'], ['the', 'arrows', 'point to']]

[88]: <IPython.core.display.HTML object>
```

Continue

You can find more exercises in the worksheet [Lists 4 - Search methods](#)¹⁶¹

5.3.4 Lists 4 - Search methods

Download exercises zip

Browse files online¹⁶²

Lists offer several different methods to perform searches and transformations inside them, but beware: the power is nothing without control! Sometimes you might feel the need to use them, but very often they hide traps you will later regret. So whenever you write code with one of these methods, **always ask yourself the questions we will stress**.

Method	Returns	Description
<code>str1.split(str2)</code>	list	Produces a list with all the words in str1 separated from str2
<code>list.count(obj)</code>	int	Counts the occurrences of an element
<code>list.index(obj)</code>	int	Searches for the first occurrence of an element and returns its position
<code>list.remove(obj)</code>	None	Removes the first occurrence of an element

¹⁶¹ <https://en.softpython.org/lists/lists4-sol.html>

¹⁶² <https://github.com/DavidLeoni/softpython-en/tree/master/lists>

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
lists
lists1.ipynb
lists1-sol.ipynb
lists2.ipynb
lists2-sol.ipynb
lists3.ipynb
lists3-sol.ipynb
lists4.ipynb
lists4-sol.ipynb
lists5-chal.ipynb
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook lists4.ipynb
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

split method - from strings to lists

The split method of strings must be called on a string and a separator must be passed as parameter, which can be a single character or a substring. The result is a list of strings without the separator.

```
[2]: "Finally the pirates shared the treasure".split("the")
[2]: ['Finally ', ' pirates shared ', ' treasure']
```

In practice this method is the opposite of lists method `join`¹⁶³ we've already seen, with the important difference this method must be called on strings and not lists.

By calling `split` without arguments generic *blanks* are used as separators (space, \n, tab \t, etc)

```
[3]: s = "Finally the\npirates\tshared      the treasure"
print(s)
Finally the
pirates shared      the treasure
```

```
[4]: s.split()
```

¹⁶³ <https://en.softpython.org/lists/lists3-sol.html#join---build-strings-from-lists>

```
[4]: ['Finally', 'the', 'pirates', 'shared', 'the', 'treasure']
```

It's also possible to limit the number of elements to split by specifying the parameter `maxsplit`:

```
[5]: s.split(maxsplit=2)
```

```
[5]: ['Finally', 'the', 'pirates\tshared      the treasure']
```

WARNING: What happens if the string does *not* contain the separator? Remember to also consider this case!

```
[6]: "I talk and overtalk and I never ever take a break".split(',')
```

```
[6]: ['I talk and overtalk and I never ever take a break']
```

QUESTION: Look at this code. Will it print something? Or will it produce an error?

1. `"revolving\tdoor".split()`

2. `"take great\t\ncare".split()`

3. `"do not\tforget\nabout\tme".split('\t')`

4. `"non ti scordar\ndi\tme".split(' ')`

5. `"The Guardian of the Abyss stared at us".split('abyss')[1]`

6. `"".split('abyss')[0]`

7. `"abyss_0000_abyss".split('abyss')[0]`

Exercise - trash dance

You've been hired to dance in the last video of the notorious band *Melodic Trash*. You can't miss this golden opportunity. Excited, you start reading the score, but you find a lot of errors - of course the band doesn't need to know about writing scores to get tv time. There are strange symbols, and the last bar is too long (after the sixth bar) and needs to be put one row at a time. Write some code which fixes the score in a list `dance`.

- **DO NOT** write string constants from the input in your code (so no "Ra Ta Pam" ...)

Example - given:

```
music = "Zam Dam\tZa Bum Bum\tZam\tBam To Tum\tRa Ta Pam\tBar Ra\tRammaGumma Unza\n\t\tTACAUACA \n BOOMBOOM!"
```

after your code it must result:

```
>>> print(dance)
['Zam Dam',
 'Za Bum Bum',
 'Zam',
 'Bam To Tum',
```

(continues on next page)

(continued from previous page)

```
'Ra Ta Pam',
'Bar Ra',
'RammaGumma',
'Unza',
'TACAUACA',
'BOOMBOOM! ']
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[7]:

```
music = "Zam Dam\tZa Bum Bum\tZam\tBam To Tum\tRa Ta Pam\tBar Ra\tRammaGumma Unza\n\t\nTACAUACA \n BOOMBOOM!"  
# write here  
  
dance = music.split('\t',maxsplit=6)  
dance = dance[:-1] + dance[-1].split()  
dance
```

[7]:

```
['Zam Dam',
'Za Bum Bum',
'Zam',
'Bam To Tum',
'Ra Ta Pam',
'Bar Ra',
'RammaGumma',
'Unza',
'TACAUACA',
'BOOMBOOM! ']
```

</div>

[7]:

```
music = "Zam Dam\tZa Bum Bum\tZam\tBam To Tum\tRa Ta Pam\tBar Ra\tRammaGumma Unza\n\t\nTACAUACA \n BOOMBOOM!"  
# write here
```

Exercise - Trash in tour

The *Melodic Trash* band strikes again! In a new tour they present the summer hits. The records company only provides the sales numbers in anglosaxon format, so before communicating them to Italian media we need a conversion.

Write some code which given the `hits` and a position in the hit parade, (from 1 to 4), prints the sales number.

- **NOTE:** commas must be substituted with dots

Example - given:

```
hits = """6,230,650 - I love you like the moldy tomatoes in the fridge
2,000,123 - The pain of living filthy rich
```

(continues on next page)

(continued from previous page)

```
100,000 - Groupies are never enough
837 - Do you remember the trashcans in the summer..."""

position = 1    # the tomatoes
#position = 4  # the trashcans
```

Prints:

```
Number 1 in hit parade "I love you like the moldy tomatoes in the fridge" sold 6.230.
↪650 copies
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[8]:

```
hits = """6,230,650 - I love you like the moldy tomatoes in the fridge
2,000,123 - The pain of living filthy rich
100,000 - Groupies are never enough
837 - Do you remember the trashcans in the summer..."""

position = 1    # the tomatoes
#position = 4  # the trashcans

# write here

lst = hits.split('\n')
ext = lst[position-1].split(' - ')

print("Number", position, "in hit parade", "''' + ext[1] + '''",
      'sold', '.'.join(ext[0].split(',')), 'copies')

Number 1 in hit parade "I love you like the moldy tomatoes in the fridge" sold 6.230.
↪650 copies
```

</div>

[8]:

```
hits = """6,230,650 - I love you like the moldy tomatoes in the fridge
2,000,123 - The pain of living filthy rich
100,000 - Groupies are never enough
837 - Do you remember the trashcans in the summer..."""

position = 1    # the tomatoes
#position = 4  # the trashcans

# write here
```

Exercise - manylines

Given the following string of text:

```
"""This is a string  
of text on  
several lines which tells nothing."""
```

1. print it
2. prints how many lines, words and characters it contains
3. sort the words in alphabetical order and print the first and last ones in lexicographical order

You should obtain:

```
This is a string  
of text on  
several lines which tells nothing.  
  
Lines: 3    words: 12    chars: 62  
  
['T', 'h', 'i', 's', ' ', 'i', 's', ' ', 'a', ' ', 's', 't', 'r', 'i', 'n', 'g', '\n',  
 ↪ 'o', 'f', ' ', 't', 'e', 'x', 't', ' ', 'o', 'n', '\n', 's', 'e', 'v', 'e', 'r', 'a',  
 ↪ 'l', ' ', 'l', 'i', 'n', 'e', 's', ' ', 'w', 'h', 'i', 'c', 'h', ' ', 't', 'e',  
 ↪ 'l', 'l', 's', ' ', 'n', 'o', 't', 'h', 'i', 'n', 'g', '.']  
62  
  
First word: This  
Last word : which  
['This', 'a', 'is', 'lines', 'nothing.', 'of', 'on', 'several', 'string', 'tells',  
 ↪ 'text', 'which']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[9]:

```
s = """This is a string  
of text on  
several lines which tells nothing."""  
  
# write here  
  
# 1) print  
print(s)  
print("")  
  
# 2) prints the lines, words and characters  
lines = s.split('\n')  
  
# NOTE: words are separated by a space or a newline  
  
words = lines[0].split(' ') + lines[1].split(' ') + lines[2].split(' ')  
num_chars = len(s)  
print("Lines:", len(lines), " words:", len(words), " chars:", num_chars)  
  
# alternative method for number of characters
```

(continues on next page)

(continued from previous page)

```

print("")
characters = list(s)
num_chars2 = len(characters)
print(characters)
print(num_chars2)

# 3. alphabetically order the words and prints the first and last one in
# lexicographical order

words.sort() # NOTE: it returns NOTHING !!!!
print("")
print("First word:", words[0])
print("Last word :", words[-1])
print(words)

This is a string
of text on
several lines which tells nothing.

Lines: 3    words: 12    chars: 62

['T', 'h', 'i', 's', ' ', ' ', 'i', 's', ' ', ' ', 'a', ' ', ' ', 's', 't', 'r', 'i', 'n', 'g', '\n',
 'o', 'f', ' ', 't', 'e', 'x', 't', ' ', 'o', 'n', '\n', 's', 'e', 'v', 'e', 'r', 'a', 'l', ' ', 'l', 'i', 'n', 'e',
 ' ', 'l', 'i', 'n', 'e', 's', ' ', 'w', 'h', 'i', 'c', 'h', ' ', 't', 'e', ' ', 'l', 'i', 'n', 'e',
 ' ', 'l', 'i', 'n', 'e', 's', ' ', 'n', 'o', 't', 'h', 'i', 'n', 'g', '.']

62

First word: This
Last word : which
['This', 'a', 'is', 'lines', 'nothing.', 'of', 'on', 'several', 'string', 'tells',
 'text', 'which']

```

</div>

[9]:

```

s = """This is a string
of text on
several lines which tells nothing."""

# write here

```

Exercise - takechars

⊕ Given a phrase which contains **exactly** 3 words and has **always** as a central word a number n , write some code which PRINTS the first n characters of the third word.

Example - given:

```
phrase = "Take 4 letters"
```

your code must print:

```
lett
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[10]:
```

```
phrase = "Take 4 letters"      # lett
#phrase= "Getting 5 caratters"  # carat
#phrase= "Take 10 characters"   # characters

# write here
words = phrase.split()
n = int(words[1])
print(words[2][:n])

lett
```

```
</div>
```

```
[10]:
```

```
phrase = "Take 4 letters"      # lett
#phrase= "Getting 5 caratters"  # carat
#phrase= "Take 10 characters"   # characters

# write here
```

count method

We can find the number of occurrences of a certain element in a list by using the method `count`

```
[11]: la = ['a', 'n', 'a', 'c', 'o', 'n', 'd', 'a']
```

```
[12]: la.count('n')
```

```
[12]: 2
```

```
[13]: la.count('a')
```

```
[13]: 3
```

```
[14]: la.count('d')
```

```
[14]: 1
```

Do not abuse count

WARNING: count is often used in a wrong / inefficient ways

Always ask yourself:

1. Could the list contain duplicates? Remember they will get counted!
2. Could the list contain *no* duplicate? Remember to also handle this case!
3. count performs a search on all the list, which could be inefficient: is it really needed, or do we already know the interval where to search?

QUESTION: Look at the following code fragments, and for each of them try guessing the result (or if it produces an error)

1. `['A', 'aa', 'a', 'aaAah', "a", "aaaa"[1], " a "].count("a")`
2. `["the", "punishment", "of", "the", "fools"].count('Fools') == 1`
3. `lst = ['oasis', 'date', 'oasis', 'coconut', 'date', 'coconut']
print(lst.count('date') == 1)`
4. `lst = ['oasis', 'date', 'oasis', 'coconut', 'date', 'coconut']
print(lst[4] == 'date')`
5. `['2', 2, "2", 2, float("2"), 2.0, 4/2, "1+1", int('3')-float('1')].count(2)`
6. `[[]].count([])`
7. `[[[], [], []]].count([])`

Exercise - country life

Given a list `country`, write some code which prints `True` if the first half contains a number of elements `e11` equal to the number of elements `e12` in the second half.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[15]:

```
e11,e12 = 'shovels', 'hoes'          # True
#e11,e12 = 'shovels', 'shovels'       # False
#e11,e12 = 'wheelbarrows', 'plows'    # True
#e11,e12 = 'shovels', 'wheelbarrows'  # False

country = ['plows', 'wheelbarrows', 'shovels',      'wheelbarrows', 'shovels', 'hoes',
           ↪'wheelbarrows',
           'hoes', 'plows',      'wheelbarrows', 'plows',      'shovels', 'plows',
           ↪'hoes']

# write here
```

(continues on next page)

(continued from previous page)

```
mid = len(country)//2
country[:mid].count(e11) == country[mid:].count(e12)

[15]: True

</div>

[15]: e11,e12 = 'shovels', 'hoes'          # True
#e11,e12 = 'shovels', 'shovels'        # False
#e11,e12 = 'wheelbarrows', 'plows'     # True
#e11,e12 = 'shovels', 'wheelbarrows'   # False

country = ['plows','wheelbarrows', 'shovels',      'wheelbarrows', 'shovels','hoes',
           ↪'wheelbarrows',
           'hoes', 'plows',           'wheelbarrows', 'plows',           'shovels','plows',
           ↪'hoes']

# write here
```

index method

The `index` method allows us to find the index of the FIRST occurrence of an element.

```
[16]: #      0   1   2   3   4   5
la = ['p','a','e','s','e']
```

```
[17]: la.index('p')
```

```
[17]: 0
```

```
[18]: la.index('a')
```

```
[18]: 1
```

```
[19]: la.index('e') # we find the FIRST occurrence
```

```
[19]: 2
```

If the element we're looking for is not present, we will get an error:

```
>>> la.index('z')
-----
ValueError                                Traceback (most recent call last)
<ipython-input-303-32d9c064ebe0> in <module>
----> 1 la.index('z')

ValueError: 'z' is not in list
```

Optionally, you can specify an index to start from (**included**):

```
[20]: # 0   1   2   3   4   5   6   7   8   9   10
['a','c','c','a','p','a','r','r','a','r','e'].index('a',6)
```

[20]: 8

And also where to end (**excluded**):

```
# 0   1   2   3   4   5   6   7   8   9   10
['a','c','c','a','p','a','r','r','a','r','e'].index('a',6,8)

-----
ValueError                                     Traceback (most recent call last)
<ipython-input-17-7f344c26b62e> in <module>
      1 # 0   1   2   3   4   5   6   7   8   9   10
----> 2 ['a','c','c','a','p','a','r','r','a','r','e'].index('a',6,8)

ValueError: 'a' is not in list
```

Do not abuse index

WARNING: `index` is often used in a wrong / inefficient ways

Always ask yourself:

1. Could the list contain duplicates? Remember only the *first* will be found!
2. Could the list *not* contain the searched element? Remember to also handle this case!
3. `index` performs a search on all the list, which could be inefficient: is it really needed, or do we already know the interval where to search?
4. If we want to know if an element is in a position we already know, `index` is useless, it's enough to write `my_list[3] == element`. If you used `index`, it could discover duplicate characters which are *before* or *after* the one we are interested in!

QUESTION: Look at the following code fragments, and for each one try guessing the result it produces (or if it gives error).

1. `['arc','boat','hollow','dune'].index('hollow') == ['arc','boat','hollow','dune'].index('hollow',1)`
2. `['azure','blue','sky blue','smurfs'][-1:].index('sky blue')`
3. `road = ['asphalt','bitumen','cement','gravel']
print('mortar' in road or road.index('mortar'))`
4. `road = ['asphalt','bitumen','cement','gravel']
print('mortar' in road and road.index('mortar'))`
5. `road = ['asphalt','bitumen','mortar','gravel']
print('mortar' in road and road.index('mortar'))`
6. `la = [0,5,10]
la.reverse()
print(la.index(5) > la.index(10))`

Exercise - Spatoč

In the past you met the Slavic painter Spatoč when he was still dirt poor. He gifted you with 2 or 3 paintings (you don't remember) of dubious artistic value that you hid in the attic, but now watching TV you just noticed that Spatoč has gained international fame. You run to the attic to retrieve the paintings, which are lost among junk. Every painting is contained in a [] box, but you don't know in which rack it is. Write some code which prints where they are.

- racks are **numbered from 1**. If the third painting was not found, print 0.
- **DO NOT** use loops nor `if`
- **HINT:** printing first two is easy - to print the last one have a look at [Booleans - evaluation order¹⁶⁴](#)

Example 1 - given:

```
[21]:      # 1      2      3      4      5
attic = [3,      'VV',      ['painting'], '---',      ['painting'],
         # 6      7      8      9      10
         5.23, ['shovel'], ['ski'],      ["painting"], ['lamp']]
```

prints:

```
rack of first painting : 3
rack of second painting: 5
rack of third painting : 9
```

Example 2 - given:

```
[22]:      # 1      2      3      4      5      6      7
attic = [['painting'], '---', ['ski'], ['painting'], ['statue'], ['shovel'], ['boots']]
```

prints

```
rack of first painting : 1
rack of second painting: 4
rack of third painting : 0
```

[Show solution](#)</div>

```
[23]:      # 1 2      3      4      5      6      7      8      9      -
      ↵ 10
attic = [3, 'VV', ['painting'], '---', ['painting'], 5.23, ['shovel'], ['ski'], ['painting'], -
      ↵ ['lamp']]
# 3,5,9
      # 1      2      3      4      5      6      7
#attic = [['painting'], '---', ['ski'], ['painting'], ['statue'], ['shovel'], ['boots']]
# 1,4,0

# write here

i1 = attic.index(['painting'])
print("rack of first painting :", i1+1)
i2 = attic.index(['painting'], i1+1)
print("rack of second painting:", i2+1)
```

(continues on next page)

¹⁶⁴ <https://en.softpython.org/basics/basics2-bools-sol.html#Evaluation-order>

(continued from previous page)

```
i3 = int(['painting'] in attic[i2+1:]) and (attic.index(['painting'], i2+1) + 1)
print("rack of third painting :", i3)
```

```
rack of first painting : 3
rack of second painting: 5
rack of third painting : 9
```

</div>

[23]:

```
# 1 2      3          4      5          6      7          8      9
↔ 10
attic = [3, '...', ['painting'], '---', ['painting'], 5.23, ['shovel'], ['ski'], ['painting'],
↔ ['lamp']]
# 3,5,9
# 1           2       3       4           5       6       7
#attic = [['painting'], '--', ['ski'], ['painting'], ['statue'], ['shovel'], ['boots']]
# 1,4,0

# write here
```

remove method

remove takes an object as parameter, searches for the FIRST cell containing that object and eliminates it:

```
[24]: # 0 1 2 3 4 5
la = [6, 7, 9, 5, 9, 8] # the 9 is in the first cell with index 2 and 4
```

```
[25]: la.remove(9) # searches first cell containing 9
```

```
[26]: la
```

```
[26]: [6, 7, 5, 9, 8]
```

As you can see, the cell which was at index 2 and that contained the FIRST occurrence of 9 has been eliminated. The cell containing the SECOND occurrence of 9 is still there.

If you try removing an object which is not present, you will receive an error:

```
la.remove(666)

-----
ValueError                                Traceback (most recent call last)
<ipython-input-121-5d04a71f9d33> in <module>
----> 1 la.remove(666)

ValueError: list.remove(x): x not in list
```

Do not abuse remove

WARNING: `remove` is often used in a wrong / inefficient ways

Always ask yourself:

1. Could the list contain duplicates? Remember only the *first* will be removed!
2. Could the list *not* contain the searched element? Remember to also handle this case!
3. `remove` performs a search on all the list, which could be inefficient: is it really needed, or do we already know the position *i* where the element to be removed is? In such case it's much better using `.pop(i)`

QUESTION: Look at the following code fragments, and for each try guessing the result (or if it produces an error).

1.

```
la = ['a', 'b', 'c', 'b']
la.remove('b')
print(la)
```

2.

```
la = ['a', 'b', 'c', 'b']
x = la.remove('b')
print(x)
print(la)
```

3.

```
la = ['a', 'd', 'c', 'd']
la.remove('b')
print(la)
```

4.

```
la = ['a', 'bb', 'c', 'bbb']
la.remove('b')
print(la)
```

5.

```
la = ['a', 'b', 'c', 'b']
la.remove('B')
print(la)
```

6.

```
la = ['a', 9, '99', 9, 'c', str(9), '999']
la.remove("9")
print(la)
```

7.

```
la = ["don't", "trick", "me"]
la.remove("don't").remove("trick").remove("me")
print(la)
```

8.

```
la = ["don't", "trick", "me"]
la.remove("don't")
la.remove("trick")
la.remove("me")
print(la)
```

9.

```
la = [4,5,7,10]
11 in la or la.remove(11)
print(la)
```

```
10. la = [4, 5, 7, 10]
    11 in la and la.remove(11)
    print(la)
```

```
11. la = [4, 5, 7, 10]
    5 in la and la.remove(5)
    print(la)
```

```
12. la = [9, [9], [[9]], [[[9]]] ]
    la.remove([9])
    print(la)
```

```
13. la = [9, [9], [[9]], [[[9]]] ]
    la.remove([[9]])
    print(la)
```

Exercise - nob

Write some code which removes from list `la` all the numbers contained in the 3 elements list `lb`.

- your code must work with any list `la` and `lb` of three elements
- you can assume that list `la` contains exactly TWO occurrences of all the elements of `lb` (plus also other numbers)

Example - given:

```
lb = [8, 7, 4]
la = [7, 8, 11, 8, 7, 4, 5, 4]
```

after your code it must result:

```
>>> print(la)
[11, 5]
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[27]:

```
lb = [8, 7, 4]
la = [7, 8, 11, 8, 7, 4, 5, 4]

# write here

la.remove(lb[0])
la.remove(lb[0])
la.remove(lb[1])
la.remove(lb[1])
la.remove(lb[2])
la.remove(lb[2])
print(la)
```

```
[11, 5]
```

</div>

[27]:

```
lb = [8, 7, 4]
la = [7, 8, 11, 8, 7, 4, 5, 4]

# write here
```

Continue

Go on with [first challenges](#)¹⁶⁵

[]:

5.3.5 Lists 5 - First challenges

Download exercises zip

Browse files online¹⁶⁶

We now propose some exercises without solution, do you accept the challenge?

Challenge - super DUPER sorted

- 1) Given a string `s` as a series of *exactly* 3 words separated by various *blanks*, write some code which sorts each word separately and puts in `s` a string with all the words joined and sorted.

Example - given:

```
s = 'super \t \n    DUPER \n    sorted'
```

after your code, it should result:

```
>>> s
'eprsuDEPRUdeorst'
```

- **DO NOT** use if statements nor cycles
- **DO NOT** write string constants in your code (so no '`\t`' ...)

[1]:

```
s = 'super \t \n    DUPER \n    sorted' # 'eprsuDEPRUdeorst'
#s = 'cba BCAD dcab' # 'abcABCDabcd'

# write here
```

¹⁶⁵ <https://en.softpython.org/lists/lists5-chal.html>

¹⁶⁶ <https://github.com/DavidLeoni/softpython-en/lists>

Challenge - What a nasty problem

Suppose you have a list of strings `to_mod` with an asterisk `'*'`, which you want to MODIFY by inserting another list `to_ins` at asterisk position.

Example - given:

```
to_ins = ['I', 'mean', 'truly', 'darn']
to_mod = ['What', 'a', 'nasty', '*', 'nasty', 'problem']
```

After your code, it must result:

```
>>> to_mod
['What', 'a', 'nasty', 'I', 'mean', 'truly', 'darn', 'nasty', 'problem']
```

- **DO NOT** change the assignment of `to_mod`, so no `to_mod =` statement is allowed !!! If you are thinking about converting everything to a string and back to a list, even if not optimal it could still be a solution provided you don't use a `to_mod =` statement
- **DO NOT** use loops nor `if` statements
- **HINT:** think about methods that MODIFY a list, for example you could either:
 - reset the list with the `clear()` method and re-extend it with what you need (of course you will need to save prior resetting ...)
 - (harder) use `slices` ressignment¹⁶⁷

[2]:

```
to_ins = ['I', 'mean', 'truly', 'darn']
to_mod = ['What', 'a', 'nasty', '*', 'nasty', 'problem']
#to_ins, to_mod = ['looks', 'like', 'a'], ['This', '*', 'punishment']

# write here
```

Challenge - Toys in the Attic

There are **Toys in the Attic**¹⁶⁸! Let's take them back!

Unfortunately, they are mixed with other stuff so we need to reorganize the mess. The attic is very tiny and dark, so you put your stuff in a long row. At the beginning of the `attic` there is a little note you put with the last inventory you did. It reports the number of items of a particular category that you will find after the note. After all those objects, you will find another note with the number of objects for another category of objects which follows and so on.

We can represent the `attic` as a list of mixed object types, numbers and strings. The list contains **exactly** three categories, in this order: toys, painting, and sports. You are given three separate empty lists `toys`, `painting`, `sports` and your goal is to separate the objects into these 3 different lists

After your code, you should obtain:

```
toys:     ['doll', 'lego', 'minicar']
painting: ['frame', 'brushes']
sports:   ['bike', 'pump', 'racket', 'ball']
```

¹⁶⁷ <https://en.softpython.org/lists/lists2-sol.html#Slices---modifying>

¹⁶⁸ <https://www.youtube.com/watch?v=Q9NAerwlYWw>

- **DO NOT** replace the lists, so no `toys = statements` ! You can only MODIFY them.
- **DO NOT** use loops nor `if` statements

[4]:

```
# 0   1   2   3   4   5   6   7   8   9   10  11
attic=[3,'doll','lego','minicar',2,'frame','brushes', 4,'bike','pump','racket','ball']

# 0   1   2   3   4   5   6   7
#attic = [2,'cards','monopoly',1,'colors',2,'snowboard','ski']

# these are given, you have to somehow MODIFY these lists
toys = []
painting = []
sports = []

# write here
```

[]:

5.4 Tuples

5.4.1 Tuples

[Download exercise zip](#)

Browse files online¹⁶⁹

A tuple in Python is an *immutable* sequence of heterogenous elements which allows duplicates, so we can put inside the objects we want, of different types, and with repetitions.

What to do

1. Unzip `exercises.zip` in a folder, you should obtain something like this:

```
tuples
    tuples1.ipynb
    tuples1-sol.ipynb
    tuples2-chal.ipynb
    jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `tuples.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

¹⁶⁹ <https://github.com/DavidLeoni/softpython-en/tree/master/tuples>

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

Creating tuples

Tuples are created with round parenthesis () and by separating the elements with commas ,

Some example:

```
[2]: numbers = (6, 7, 5, 7, 7, 9)
```

```
[3]: print(numbers)
(6, 7, 5, 7, 7, 9)
```

Tuples of one element: You can create a tuple of a single element by adding a comma after the element:

```
[4]: little_tup = (4,) # notice the comma !!!
```

Let's verify the type is the expected one:

```
[5]: type(little_tup)
[5]: tuple
```

To see the difference, we write down here (4) without comma and we verify the type of the obtained object:

```
[6]: fake = (4)
[7]: type(fake)
[7]: int
```

We see that fake is an int, because 4 has been evaluated as an expression inside round brackets so the result is the content inside the parenthesis.

Empty tuple

We can also create an empty tuple:

```
[8]: empty = ()
[9]: print(empty)
()
[10]: type(empty)
[10]: tuple
```

Tuples without brackets

When we assign values to some variable, (and *only* when we assign values to variables) it is possible to use a notation like the following, in which on the left of = we put names of variables and on the right we place a sequence of values:

```
[11]: a,b,c = 1, 2, 3
```

```
[12]: a
```

```
[12]: 1
```

```
[13]: b
```

```
[13]: 2
```

```
[14]: c
```

```
[14]: 3
```

If we ask ourselves what that 1, 2, 3 is, we can try putting on the left a single variable:

```
[15]: # WARNING: BETTER AVOID THIS!
x = 1,2,3
```

```
[16]: type(x)
```

```
[16]: tuple
```

We see that Python considered that 1, 2, 3 as a tuple. Typically, you would never write assignments with less variables than values to put, but if it happens, probably you will find yourself with some undesired tuple !

QUESTION: Have a look at the following code snippets, and for each try guessing which result it produces (or if it gives an error)

```
1. z,w = 5,6
   print(type(z))
   print(type(w))
```

```
2. a,b = 5,6
   a,b = b,a
   print('a=',a)
   print('b=',b)
```

```
3. z = 5,
   print(type(z))
```

```
4. z = ,
   print(type(z))
```

Heterogenous elements

In a tuple we can put elements of different types, like numbers and strings:

```
[17]: stuff = (4, "paper", 5, 2, "scissors", 7)
```

```
[18]: stuff
```

```
[18]: (4, 'paper', 5, 2, 'scissors', 7)
```

```
[19]: type(stuff)
```

```
[19]: tuple
```

We can also insert other tuples:

```
[20]: salad = ( ("lettuce", 3), ("tomatoes", 9), ("carrots", 4) )
```

```
[21]: salad
```

```
[21]: (('lettuce', 3), ('tomatoes', 9), ('carrots', 4))
```

```
[22]: type(salad)
```

```
[22]: tuple
```

And also lists:

```
[23]: mix = ( ["when", "it", "rains"], ["I", "program"], [7,3,9] )
```

WARNING: avoid mutable objects inside tuples!

Inserting *mutable* objects like lists inside tuples may cause problems in some situations like when you later want to use the tuple as element of a set or a key in a dictionary (we will see the details in the respective tutorials)

Let's see how the previous examples are represented in Python Tutor:

```
[24]: # WARNING: before using the function jupman.pytut() which follows,
#           it is necessary to first execute this cell with Shift+Enter (once is
#           enough)

import jupman
```

```
[25]: stuff = (4, "paper", 5, 2, "scissors", 7)
salad = ( ("lettuce", 3), ("tomatoes", 9), ("carrots", 4) )
mix = ( ["when", "it", "rains"], ["I", "program"], [7,3,9] )

jupman.pytut()
```

```
[25]: <IPython.core.display.HTML object>
```

Creating tuples from sequences

You can create a tuple from any sequence, like for example a list:

```
[26]: tuple( [8,2,5] )
```

```
[26]: (8, 2, 5)
```

Or a string (which is a character sequence):

```
[27]: tuple("abc")
```

```
[27]: ('a', 'b', 'c')
```

Creating sequences from tuples

Since the tuple is a sequence, it is also possible to generate lists from tuples:

```
[28]: list( (3,4,2,3) )
```

```
[28]: [3, 4, 2, 3]
```

QUESTION: Does it make sense creating a tuple from another tuple like this? Can we rewrite the code in a more concise way?

```
[29]: x = (4,2,5)
y = tuple(x)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: since a tuple is IMMUTABLE, once we create in memory the object (4, 2, 5) we are sure nobody will modify it, so it's not necessary to create a new tuple and we can directly write:

```
x = (4,2,5)
y = x
```

</div>

QUESTION: Have a look at the following expressions, and for each try to guess which result produces (or if it gives an error):

1.

2.

3.

4.

5.

6.

7. `((),)`
8. `tuple([('a'), ('b'), ('c')))`
9. `tuple(tuple(('z', 'u', 'm')))`
10. `str(('a', 'b', 'c'))`
11. `"".join(('a', 'b', 'c'))`

Operators

The following operators work on tuples and behave exactly as in lists:

Operator	Syntax	Result	Meaning
<i>length</i>	<code>len(tuple)</code>	<code>int</code>	Return the length of a tuple
<i>indexing</i>	<code>tuple[int]</code>	<code>object</code>	Reads an element at specified index
<i>slice</i>	<code>tuple[int1:int2]</code>	<code>tuple</code>	Extracts a sub-tuple - return a NEW tuple
<i>concatenation</i>	<code>tuple1 + tuple2</code>	<code>tuple</code>	Concatenates two tuples - return a NEW tuple
<i>membership</i>	<code>obj in tuple</code>	<code>bool</code>	Checks whether an element is present in a tuple
<i>replication</i>	<code>tuple * int</code>	<code>tuple</code>	Replicates the tuple - return a NEW tuple
<i>equality</i>	<code>==, !=</code>	<code>bool</code>	Checks if two tuples are equal or different

len

`len` function returns the tuple length:

```
[30]: len( (4,2,3) )
```

```
[30]: 3
```

```
[31]: len( (7,) )
```

```
[31]: 1
```

```
[32]: len( () )
```

```
[32]: 0
```

QUESTION: Have a look at following expressions, and for each try to guess the result (or if it gives an error)

1. `len(3,2,4)`
2. `len((3,2,4))`
3. `len(('a',))`
4. `len(('a',))`

5. `len((((),()),(),))`
6. `len(len((1,2,3,4)))`
7. `len([(('d','a','c','d'),((('ab'))),[('a','b','c')])])`

[]:

Reading an element

Like in strings and lists we can read an element at a certain position:

```
[33]: #      0  1  2  3
      tup = (10,11,12,13)
```

```
[34]: tup[0]
```

```
[34]: 10
```

```
[35]: tup[1]
```

```
[35]: 11
```

```
[36]: tup[2]
```

```
[36]: 12
```

```
[37]: tup[3]
```

```
[37]: 13
```

We can also use negative indexes:

```
[38]: tup[-1]
```

```
[38]: 13
```

QUESTION: Have a look at the following expressions and for each of them try to guess the result or if it produces an error:

1. `(1,2,3)[0]`
2. `(1,2,3)[3]`
3. `(1,2,3)0`
4. `('a,')[0]`
5. `('a',)[0]`
6. `(1,2,3)[-0]`

7.

8.

9.

10.

11.

[]:

Exercise - animals

Given the string `animals = "Siamese cat,dog,canary,piglet,rabbit,hamster"`

1. convert it to a list
2. create a tuple of tuples where each tuple has two elements: the animal name and the name length, i.e. ((“dog”,3), (“hamster”,7))
3. print the tuple

You should obtain:

```
Siamese cat,dog,canary,piglet,rabbit,hamster
 (('Siamese cat', 11), ('dog', 3), ('canary', 6), ('piglet', 6), ('rabbit', 6), (
 ↪'hamster', 7))
```

- you can assume `animals` always contains exactly 6 animals

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[39]:

```
animals = "Siamese cat,dog,canary,piglet,rabbit,hamster"

# write here
my_list = animals.split(',')

#print(animals)
print()

animals_tuple = ( (my_list[0], len(my_list[0])), 
                  (my_list[1], len(my_list[1])), 
                  (my_list[2], len(my_list[2])), 
                  (my_list[3], len(my_list[3])), 
                  (my_list[4], len(my_list[4])), 
                  (my_list[5], len(my_list[5])))

print(animals_tuple)

 (('Siamese cat', 11), ('dog', 3), ('canary', 6), ('piglet', 6), ('rabbit', 6), (
 ↪'hamster', 7))
```

```
</div>
```

[39]:

```
animals = "Siamese cat,dog,canary,piglet,rabbit,hamster"  
# write here
```

Slices

As with strings and lists, by using *slices* we can also extract subsequences from a tuple, that is, on the right of the tuple we can write square brackets with inside a start index INCLUDED, a colon : and an end index EXCLUDED:

[40]: `tup = (10,11,12,13,14,15,16,17,18,19)`

[41]: `tup[2:6] # from index 2 INCLUDED to 6 EXCLUDED`

[41]: `(12, 13, 14, 15)`

It is possible to alternate the gathering of elements by adding the number of elements to skip as a third numerical parameter in the square brackets, for example:

[42]: `tup = (10,11,12,13,14,15,16,17)`

[43]: `tup[0:8:5]`

[43]: `(10, 15)`

[44]: `tup[0:8:2]`

[44]: `(10, 12, 14, 16)`

[45]: `tup[1:8:1]`

[45]: `(11, 12, 13, 14, 15, 16, 17)`

WARNING: remeber that slices produce a NEW tuple !

QUESTION: Have a look at the following code snippets, and for each try to guess which result it produces (or if it gives an error)

1. `(7, 6, 8, 9, 5) (1:3)`

2. `(7, 6, 8, 9, 5) [1:3]`

3. `(10,11,12,13,14,15,16) [3:100]`

4. `(10,11,12,13,14,15,16) [-3:5]`

5. `(1, 0, 1, 0, 1, 0) [::2]`

6. `(1, 2, 3) [::1]`

7. `(1, 0, 1, 0, 1, 0) [1::2]`

8. `tuple("postcards") [0::2]`

9. `(4, 5, 6, 3, 4, 7) [0:::2]`

Concatenation

It is possible to concatenate two tuples by using the operator `+`, which creates a NEW tuple:

[46]: `t = (1, 2, 3) + (4, 5, 6, 7, 8)`

[47]: `t`

[47]: `(1, 2, 3, 4, 5, 6, 7, 8)`

[48]: `type(t)`

[48]: `tuple`

Let's verify that original tuples are not modified:

[49]: `x = (1, 2, 3)
y = (4, 5, 6, 7, 8)`

[50]: `t = x + y`

[51]: `t`

[51]: `(1, 2, 3, 4, 5, 6, 7, 8)`

[52]: `x`

[52]: `(1, 2, 3)`

[53]: `y`

[53]: `(4, 5, 6, 7, 8)`

Let's see how they are represented in Python Tutor:

[54]: `# FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE HERE THIS CELL with Shift+Enter
(it's sufficient to execute it only once, it's also at the beginning of this
→notebook)
import jupman`

[55]: `x = (1, 2, 3)
y = (4, 5, 6, 7, 8)
t = x + y
print(t)
print(x)`

(continues on next page)

(continued from previous page)

```
print(y)

jupman.pytut()

(1, 2, 3, 4, 5, 6, 7, 8)
(1, 2, 3)
(4, 5, 6, 7, 8)

[55]: <IPython.core.display.HTML object>
```

QUESTION: Have a look at the following code snippets, and for each try guessing which result it produces (or if it gives an error)

1. `(2, 3, 4) + tuple([5, 6, 7])`

2. `"crazy"+('r', 'o', 'c', 'k', 'e', 't')`

3. `() + ()`

4. `type((()) + ())`

5. `len((()) + ())`

6. `() + []`

7. `[] + ()`

Membership

As in all sequences, if we want to verify whether an element is contained in a tuple we can use the operator `in` which returns a boolean value:

```
[56]: 'e' in ('h', 'e', 'l', 'm', 'e', 't')
```

```
[56]: True
```

```
[57]: 'z' in ('h', 'e', 'l', 'm', 'e', 't')
```

```
[57]: False
```

not in

To check whether something is **not** belonging to a tuple, we can use two forms:

not in - form 1:

```
[58]: "carrot" not in ("watermelon", "banana", "apple")
```

```
[58]: True
```

```
[59]: "watermelon" not in ("watermelon", "banana", "apple")
```

[59]: False

not in - form 2

[60]: `not "carrot" in ("watermelon", "banana", "apple")`

[60]: True

[61]: `not "watermelon" in ("watermelon", "banana", "apple")`

[61]: False

QUESTION: Have a look at the following code snippets, and for each try to guess which result it produces (or if it gives an error)

1. `3 in (1.0, 2.0, 3.0)`

2. `3.0 in (1, 2, 3)`

3. `3 not in (3)`

4. `3 not in (3,)`

5. `6 not in ()`

6. `0 in (0) [0]`

7. `[] in ()`

8. `() in []`

9. `not [] in ()`

10. `() in ()`

11. `() in (())`

12. `() in ((),)`

13. `'ciao' in ('c', 'i', 'a', 'o')`

Replication

To replicate the elements in a tuple, it is possible to use the operator * which produces a NEW tuple:

[62]: `(7, 8, 5) * 3`

[62]: `(7, 8, 5, 7, 8, 5, 7, 8, 5)`

[63]: `(7, 8, 5) * 1`

[63]: (7, 8, 5)

[64]: (7, 8, 5) * 0

[64]: ()

QUESTION: What is the following code going to print?

```
x = (5, 6, 7)
y = x * 3
print('x=', x)
print('y=', y)
```

ANSWER: It will print:

```
x = (5, 6, 7)
y = (5, 6, 7, 5, 6, 7, 5, 6, 7)
```

because the multiplication generates a NEW tuple which is associated to y. The tuple associated to x remains unchanged.

QUESTION: Have a look at the following expressions, and for each try to guess which result it produces (or if it gives an error)

1. (5, 6, 7) * (3.0)

2. (5, 6, 7) * (3, 0)

3. (5, 6, 7) * (3)

4. (5, 6, 7) * 3

5. (4, 2, 3) * int(3.0)

6. (1, 2) * [3][0]

7. (1, 2) * (3, 4)[-1]

8. [(9, 8)] * 4

9. (1+2, 3+4) * 5

10. (1+2,) * 4

11. (1+2) * 4

12. (1, 2, 3) * 0

13. (7) * 0

14. (7,) * 0

[]:

Exercise - welcome

Given a tuple `x` containing exactly 3 integers, and a tuple `y` containing exactly 3 tuples of characters, write some code to create a tuple `z` containing each tuple of `y` replicated by the corresponding integer in `x`.

Example - given:

```
x = (2, 4, 3)
y = (('w', 'e', 'l', 'c'), ('o',), ('m', 'e'))
```

after your code it should print:

```
>>> print(z)
('w', 'e', 'l', 'c', 'w', 'e', 'l', 'c', 'o', 'o', 'o', 'o', 'm', 'e', 'm', 'e', 'm',
 ↵'e')
```

 Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[65]:

```
x = (2, 4, 3)
y = (('w', 'e', 'l', 'c'), ('o',), ('m', 'e'))

# write here
z = y[0]*x[0] + y[1]*x[1] + y[2]*x[2]

print(z)
```

('w', 'e', 'l', 'c', 'w', 'e', 'l', 'c', 'o', 'o', 'o', 'o', 'm', 'e', 'm', 'e', 'm',
 ↵'e')

</div>

[65]:

```
x = (2, 4, 3)
y = (('w', 'e', 'l', 'c'), ('o',), ('m', 'e'))

# write here
```

Write an element

Tuples are *immutable*, so trying to i.e. write an assignment for placing the number 12 into the cell at index 3 provokes an error:

```
#      0 1 2 3 4
tup = (5, 8, 7, 9, 11)
tup[3] = 666

-----
TypeError                                Traceback (most recent call last)
                                         (continues on next page)
```

(continued from previous page)

```
<ipython-input-118-83949b0c81e2> in <module>
    1 tup = (5,8,7,9,11)
----> 2 tup[3] = 666
TypeError: 'tuple' object does not support item assignment
```

What we can do is to create a NEW tuple by composing it from sequences takes from the original one:

```
[66]: #      0  1  2  3  4  5  6
       tup = (17,54,34,87,26,95,34)

[67]: tup = tup[0:3] + (12,) + tup[4:]

[68]: tup
[68]: (17, 54, 34, 12, 26, 95, 34)
```

WARNING: append, extend, insert, sort DO NOT WORK WITH TUPLES !

All the methods you used to modify lists will *not* work with tuples.

Exercise - badmod

Try writing down here `(1, 2, 3).append(4)` and see which error appears:

```
[69]: # write here
```

Exercise - abde

Given a tuple `x`, save in a variable `y` another tuple containing:

- at the beginning, the same elements of `x` *except* the last one
- at the end, the elements '`d`' and '`e`' .
- Your code should work with *any* tuple `x`

Example - given:

```
x = ('a', 'b', 'c')
```

after your code, you should see printed:

```
x = ('a', 'b', 'c')
y = ('a', 'b', 'd', 'e')
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[70]:

```
x = ('a', 'b', 'c')

# write here
y = x[:-1] + ('d', 'e')

print('x=', x)
print('y=', y)

x= ('a', 'b', 'c')
y= ('a', 'b', 'd', 'e')
```

</div>

[70]:

```
x = ('a', 'b', 'c')

# write here
```

Exercise - charismatic

Given a tuple t having alternating uppercase / lowercase characters, write some code which modifies the assignment of t so that t becomes equal to a tuple having all characters lowercase as first ones and all uppercase characters as last ones.

Example - given:

```
t = ('C', 'h', 'A', 'r', 'I', 's', 'M', 'a', 'T', 'i', 'C')
```

after your code it must result:

```
>>> print(t)
('C', 'A', 'I', 'M', 'T', 'C', 'h', 'r', 's', 'a', 'i')
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[71]:

```
t = ('C', 'h', 'A', 'r', 'I', 's', 'M', 'a', 'T', 'i', 'C')

# write here
t = t[::-2] + t[1::2]
print(t)

('C', 'A', 'I', 'M', 'T', 'C', 'h', 'r', 's', 'a', 'i')
```

</div>

[71]:

```
t = ('C', 'h', 'A', 'r', 'I', 's', 'M', 'a', 'T', 'i', 'C')

# write here
```

Exercise - sorting

Given a tuple `x` of unordered numbers, write some code which changes the assignment of `x` so that `x` results assigned to a sorted tuple

- your code must work for *any* tuple `x`
- **HINT:** as we've already written, tuples DO NOT have `sort` method (because it would mutate them), but lists have it ...

Example - given:

```
x = (3, 4, 2, 5, 5, 5, 2, 3)
```

after your code it must result:

```
>>> print(x)
(2, 2, 3, 3, 4, 5, 5, 5)
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[72]:

```
x = (3, 4, 2, 5, 5, 5, 2, 3)

# write here
y = list(x)
y.sort()
x = tuple(y)
print(x)

(2, 2, 3, 3, 4, 5, 5, 5)
```

</div>

[72]:

```
x = (3, 4, 2, 5, 5, 5, 2, 3)

# write here
```

Methods

Tuples are objects of type `tuple` and have methods which allows to operate on them:

Method	Returns	Description
<code>tuple.index(obj)</code>	int	Searches for the first occurrence of an element and returns its position
<code>tuple.count(obj)</code>	int	Count the occurrences of an element

index method

index method allows to find the index of the FIRST occurrence of an element.

```
[73]: tup = ('b', 'a', 'r', 'a', 't', 't', 'o')
```

```
[74]: tup.index('b')
```

```
[74]: 0
```

```
[75]: tup.index('a')
```

```
[75]: 1
```

```
[76]: tup.index('t')
```

```
[76]: 4
```

If the element we're looking for is not present, we will get an error:

```
>>> tup.index('z')
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-318-96cf33478b69> in <module>
----> 1 tup.index('z')
```

```
ValueError: tuple.index(x): x not in tuple
```

Optionally, you can specify an index to start searching from (**included**):

```
[77]: # 0   1   2   3   4   5   6   7   8
      ('b', 'a', 'r', 'a', 't', 't', 'a', 'r', 'e').index('r', 3)
```

```
[77]: 7
```

and also where to end (**excluded**):

```
# 0   1   2   3   4   5   6   7   8
('b', 'a', 'r', 'a', 't', 't', 'a', 'r', 'e').index('r', 3, 7)
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-12-e91a1f6569d7> in <module>
      1 # 0   1   2   3   4   5   6   7   8
----> 2 ('b', 'a', 'r', 'a', 't', 't', 'a', 'r', 'e').index('r', 3, 7)
```

```
ValueError: tuple.index(x): x not in tuple
```

Do not abuse index

WARNING: `index` is often used in a wrong / inefficient ways

Always ask yourself:

1. Could the tuple contain duplicates? Remember only the *first* will be found!
2. Could the tuple *not* contain the searched element? Remember to also handle this case!
3. `index` performs a search on all the tuple, which could be inefficient: is it really needed, or do we already know the interval where to search?
4. If we want to know if an element is in a position we already know (i.e. 3), `index` is useless, it's enough to write `my_tuple[3] == element`. If you used `index`, it could discover duplicate characters which are *before* or *after* the one we are interested in!

QUESTION: Have a look at the following expressions, and for each try to guess which result (or if it gives an error)

1. `(3, 4, 2).index(4)`
2. `(3, 4, ---1).index(-1)`
3. `(2.2, .2, 2,).index(2)`
4. `(3, 4, 2).index(len([3, 8, 2, 9]))`
5. `(6, 6, 6).index(666)`
6. `(4, 2, 3).index(3).index(3)`
7. `tuple("GUG").index("g")`
8. `(tuple("ci") + ("a", "o")).index('a')`
9. `((()).index(()))`
10. `(((),).index(()))`

Exercise - The chinese boxes

Write some code which searches the word "Chinese" in each of 3 tuples nested into each other, printing the actual position relative to the tuple which contains the occurrence.

- the tuples always start with 4 strings

Example - given:

```
tup = ('Open', 'The', 'Chinese', 'Boxes', ('Boxes', 'Open', 'The', 'Chinese', ('Chinese',
    ↪ 'Open', 'The', 'Boxes')))
```

after your code, it must print:

```
('Open', 'The', 'Chinese', 'Boxes') contains Chinese at position 2
('Boxes', 'Open', 'The', 'Chinese') contains Chinese at position 3
('Chinese', 'Open', 'The', 'Boxes') contains Chinese at position 0
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[78]:

```
word = 'Chinese'
tup = ('Open', 'The', 'Chinese', 'Boxes', ('Boxes', 'Open', 'The', 'Chinese', ('Chinese',
˓→'Open', 'The', 'Boxes')))
#           2                   3                   0

#word = 'c'
#tup = ('a', 'b', 'c', 'd', ('e', 'f', 'g', 'h', ('a', 'b', 'd', 'c')))
#       2           1           3

# write here

t1 = tup[:4]
t2 = tup[4][:4]
t3 = tup[4][4]
i1 = t1.index(word)
i2 = t2.index(word)
i3 = t3.index(word)

print(t1,"contains", word, "at position", i1)
print(t2,"contains", word, "at position", i2)
print(t3,"contains", word, "at position", i3)

('Open', 'The', 'Chinese', 'Boxes') contains Chinese at position 2
('Boxes', 'Open', 'The', 'Chinese') contains Chinese at position 3
('Chinese', 'Open', 'The', 'Boxes') contains Chinese at position 0
```

</div>

[78]:

```
word = 'Chinese'
tup = ('Open', 'The', 'Chinese', 'Boxes', ('Boxes', 'Open', 'The', 'Chinese', ('Chinese',
˓→'Open', 'The', 'Boxes')))
#           2                   3                   0

#word = 'c'
#tup = ('a', 'b', 'c', 'd', ('e', 'f', 'g', 'h', ('a', 'b', 'd', 'c')))
#       2           1           3

# write here
```

count method

We can obtain the number of occurrences of a certain element in a list by using the method `count`:

```
[79]: t = ('a', 'c', 'a', 'd', 'e', 'm', 'i', 'a')
```

```
[80]: t.count('a')
```

```
[80]: 3
```

```
[81]: t.count('d')
```

```
[81]: 1
```

If an element is not present 0 is returned:

```
[82]: t.count('z')
```

```
[82]: 0
```

Do not abuse count

WARNING: `count` is often used in a wrong / inefficient ways

Always ask yourself:

1. Could the tuple contain duplicates? Remember they will *all* be counted!
2. Could the tuple *not* contain the element to count? Remember to also handle this case!
3. `count` performs a search on all the tuple, which could be inefficient: is it really needed, or do we already know the interval where to search?

QUESTION: Have a look at the following expressions, and for each try to guess which result (or if it gives an error)

1. `('p', 'o', 'r', 't', 'e', 'n', 't', 'o', 's', 'o').count('o')`

2. `('p', 'o', 'r', 't', 'e', 'n', 't', 'o', 's', 'o').count(('o'))`

3. `('p', 'o', 'r', 't', 'e', 'n', 't', 'o', 's', 'o').count(('o',))`

4. `(1, 0, 0, 0).count(0)`

5. `(1, 0, 0, 0).count((0))`

6. `(1, 0, 0, 0).count((0,))`

7. `(1, 0, (0,), (0,)).count((0,))`

8. `(1, 0, (0,0), ((0,0), (0,0))).count((0,0))`

Exercise - fruits

Given the string `s = "apple|pear|apple|cherry|pear|apple|pear|pear|cherry|pear|strawberry"`

Insert the elements separated by " | " (pipe character) in a list.

1. How many elements must the list have?
2. Knowing the list created at previous point has only four distinct elements (es "apple", "pear", "cherry", and "strawberry"), create another list where each element is a tuple containing the name of the fruit and its multiplicity (that is, the number of times it appears in the original list).

Example - given:

```
counts = [("apple", 3), ("pear", 5), ...]
```

Here you can write code which works given a specific constant, so you don't need cycles.

3. Print the content of each tuple in a separate line (i.e.: first line; "apple" is present 3 times)

You should obtain:

```
[('apple', 3), ('pear', 5), ('cherry', 2), ('strawberry', 1)]  
  
apple is present 3 times  
pear is present 5 times  
cherry is present 2 times  
strawberry is present 1 times
```

Show solution<div class="jupman-sol" jupman-sol-code" style="display:none">

```
[83]: s = "apple|pear|apple|cherry|pear|apple|pear|pear|cherry|pear|strawberry"
```

```
# write here

words = s.split("|")
#print(words)

tapples = ("apple", words.count("apple"))
tpears = ("pear", words.count("pear"))
tcherries = ("cherry", words.count("cherry"))
tstrawberries = ("strawberry", words.count("strawberry"))
counts =[tapples, tpears, tcherries, tstrawberries]

print(counts)
print()
print(tapples[0], "is present", tapples[1], "times")
print(tpears[0], "is present", tpears[1], "times")
print(tcherries[0], "is present", tcherries[1], "times")
print(tstrawberries[0], "is present", tstrawberries[1], "times")

[('apple', 3), ('pear', 5), ('cherry', 2), ('strawberry', 1)]  
  
apple is present 3 times  
pear is present 5 times  
cherry is present 2 times  
strawberry is present 1 times
```

```
</div>

[83]: s = "apple|pear|apple|cherry|pear|apple|pear|pear|cherry|pear|strawberry"

# write here

[('apple', 3), ('pear', 5), ('cherry', 2), ('strawberry', 1)]

apple is present 3 times
pear is present 5 times
cherry is present 2 times
strawberry is present 1 times
```

Continue

Go on with the [first challenges](#)¹⁷⁰

5.4.2 Tuples 2 - First challenges

Download exercises zip

Browse file online¹⁷¹

We now propose some exercises without solution, do you accept the challenge?

Challenge - The Temple Of Rounded Doom

You are exploring an uncharted tropical region, and among the vegetation you discover the entrance of a temple devoted to the ancient God Tuplaranda. Cautiously, you enter. You see a massive door: on a side lies a long series of tablets, some rounded and some squared. They contain tokens with mystical symbols. In order to open the door, you must build a single long round tablet in front of the door, with all the tokens in the same order you find them. Write some code to produce such rounded tablet.

Example - given:

```
[1]: t = ('wara', 'zuna', ('nabu', 'zebi'), [('vi','la')], ('gur',), ('gar'), 'zat', ['ben
      ↪', 'elz','ub'])
```

The tokens are 'wara', 'zuna', 'vila', 'nabu' etc

Your code must produce:

```
('wara', 'zuna', 'nabu', 'zebi', 'vi', 'la', 'gur', 'gar', 'z', 'a', 't', 'ben', 'elz
      ↪', 'ub')
```

IMPORTANT: DO NOT upset Tuplaranda! Pay attention to the warning signs on the door:

- **DO NOT** write strings (so don't manually write string constants like 'zuna' in your code)
- **DO NOT** type more than **8** opening square brackets [

¹⁷⁰ <https://en.softpython.org/tuples/tuples2-chal.html>

¹⁷¹ <https://github.com/DavidLeoni/softpython-en/tuples>

WARNING 1: there are traps - be very careful about commas !

WARNING 2: 'zat' must become 'z', 'a', 't'

[2]:

```
t = ('wara', 'zuna', ('nabu', 'zebi'), [('vi','la')], ('gur',), ('gar'), 'zat', ['ben
˓→', 'elz','ub'])

# write here
```

[]:

5.5 Sets

5.5.1 Sets

[Download exercises zip](#)

Browse online files¹⁷²

A set is a *mutable unordered* collection of *immutable distinct* elements (that is, without duplicates). The Python datatype to represent sets is called `set`.

What to do

1. Unzip `exercises.zip` in a folder, you should obtain something like this:

```
sets
sets1.ipynb
sets1-sol.ipynb
sets2-chal.ipynb
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `sets.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

¹⁷² <https://github.com/DavidLeoni/softpython-en/tree/master/sets>

Creating a set

We can create a set using curly brackets, and separating the elements with commas ,

Let's try a set of characters:

```
[2]: s = {'b', 'a', 'd', 'c'}
```

```
[3]: type(s)
```

```
[3]: set
```

WARNING: SETS ARE *NOT* ORDERED !!!

DO NOT BELIEVE IN WHAT YOU SEE !!

Let's try printing the set:

```
[4]: print(s)
{'a', 'd', 'b', 'c'}
```

The output shows the order in which the print was made is different from the order in which we built the set. Also, according to the Python version you're using, on your computer it might be even different!

This is because order in sets is NOT guaranteed: the only thing that matters is whether or not an element belongs to a set.

As a further demonstration, we may ask Jupyter to show the content of the set, by writing only the variable `s` WITHOUT `print`:

```
[5]: s
[5]: {'a', 'b', 'c', 'd'}
```

Now it appears in alphabetical order! It happens like so because Jupyter show variables by implicitly using the `pprint`¹⁷³ (*pretty print*), which ONLY for sets gives us the courtesy to order the result before printing it. We can thank Jupyter, but let's not allow it to confuse us!

Elements index: since sets have no order, asking Python to extract an element at a given position would make no sense. Thus, differently from strings, lists and tuples, with sets it's NOT possible to extract an element from an index:

```
s[0]
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-352-c9c96910e542> in <module>
----> 1 s[0]

TypeError: 'set' object is not subscriptable
```

We said that a set has only *distinct* elements, that is without duplicates - what happens if we try to place some duplicate anyway?

```
[6]: s = {6, 7, 5, 9, 5, 5, 7}
```

¹⁷³ <https://docs.python.org/3/library/pprint.html>

```
[7]: s
[7]: {5, 6, 7, 9}
```

We note that Python silently removed the duplicates.

Converting sequences to sets

As for lists and strings, we can create a `set` from another sequence:

```
[8]: set('acacia') # from string
[8]: {'a', 'c', 'i'}
```



```
[9]: set([1,2,3,1,2,1,2,1,3,1]) # from list
[9]: {1, 2, 3}
```



```
[10]: set((4,6,1,5,1,4,1,5,4,5)) # from tuple
[10]: {1, 4, 5, 6}
```

Again, we notice in the generated set there are no duplicates

REMEMBER: Sets are useful to remove duplicates from a sequence

Mutable elements and hashes

Let's see again the definition from the beginning:

A set is a *mutable unordered* collection of *immutable distinct* elements

So far we only created the set using *immutable* elements like numbers and strings.

What happens if we place some mutable elements, like lists?

```
>>> s = {[1,2,3], [4,5]}

-----
TypeError                                     Traceback (most recent call last)
<ipython-input-40-a6c538692ccb> in <module>
----> 1 s = {[1,2,3], [4,5]}

TypeError: unhashable type: 'list'
```

We obtain `TypeError: unhashable type: 'list'`, which literally means Python didn't manage to calculate the *hash* of the list. What could this particular dish ever be?

What is the hash? The *hash* of an object is a number that Python can associate to it, for example you can see the hash of an object by using the function with the same name:

```
[11]: hash("This is a nice day") # string
[11]: 1137365577994337037
```

```
[12]: hash( 1111122222333333444444555555555 )    # number  
[12]: 651300278308214397
```

Imagine the *hash* is some kind of label with these properties:

- it is too short to completely describe the object to which it is associated (that is: given a hash label, you *cannot* reconstruct the object it represents)
- it is enough long to identify *almost uniquely* the object...
- ... even if in the world there *might* be different objects which have associated exactly the same label

What's the relation with our sets? The *hash* has various applications, but typically Python uses it to quickly find an object in collections which are based on hashes, like sets and dictionaries. How much fast? Very fast: even with homogenous sets, we always obtain an answer in a constant very short time! In other words, the answer speed *does not* depend on the set dimension (except for pathological cases we don't review here).

This velocity is permitted by the fact that given some object to search, Python is able to rapidly calculate its *hash* label: then, with the label in the hand, so to speak, it can manage to quickly find in the memory store whether there are objects which have the same label. If they are found, they will almost surely be very few, so Python will only need to compare them with the searched one.

***Immutable* objects always have the same hash label** from when they are created until the end of the program. Instead, the *mutable* ones behave differently: each time we change an object, the *hash* also changes. Imagine a market where employees place food by looking at labels and separating accordingly for example the coffee in the shelves for the breakfast and bleach in the shelves for detergents. If you are a customer and you want some coffee, you look at signs and directly go toward the shelves for breakfast stuff. Image what could happen if an evil sorcerer could transform the objects already placed into other objects, like for example the coffee into bleach (let's assume that at the moment of the transmutation the *hash* label also changes). Much confusion would certainly follow, and, if we aren't cautious, also a great stomachache or worse.

So to offer you the advantage of a fast search while avoiding disastrous situations, Python imposes to place inside sets only objects with a stable *hash*, that is *immutable* objects.

QUESTION: Can we insert a tuple inside a set? Try to verify your intuition with a code example.

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
```

Show answer

```
></a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: Yes, tuples are *immutable*, so they have a corresponding *hash* which remains stable for all the program duration, for example this is a tuple set: { (1, 2), (3, 4, 5) }

Note we can consider a tuple as really immutable only if it contains elements which are also immutable.

```
</div>
```

Empty set

WARNING: If you write {} you will obtain a dictionary, NOT a set !!!

To create an empty set we must call the function `set()`:

```
[13]: s = set()
```

```
[14]: s
```

[14]: `set()`

EXERCISE: try writing `{ }` in the cell below and look at the object type obtained with `type`

[15]: `# write here`

QUESTION: Can we try inserting a set inside another set? Have a careful look at the set definition, then verify your suppositions by writing some code to create a set which has another set inside.

WARNING: To perform the check, DO NOT use the `set` function, only use creation with curly brackets

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show answer"
  data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: A set is *mutable*, so we *cannot* insert it as an element of another set (its *hash* label could vary over time). By writing `{ {1, 2, 3} }` you will get an error.

`</div>`

QUESTION: If we write something like this, what do we get? (careful!)

```
set(set(['a', 'b']))
```

1. a set with a and b inside
2. a set containing another set which contains a and b as elements
3. an error (which one?)

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show answer"
  data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: 1:

- inside we have the expression `set(['a', 'b'])` which generates the set `{'a', 'b'}`
- outside we have the expression `set(set(['a', 'b']))` which is given the set just created, so we can rewrite it as `set({'a', 'b'})`
- Since `set` when used as a function expects a sequence, and a set *is* a sequence, the external `set` takes all the elements it finds inside the sequence `{'a', 'b'}` we passed, and generates a new set with '`a`' and '`b`' inside.

`</div>`

QUESTION: Have a look at following expressions, and for each of them try to guess which result it produces (or if it gives an error):

1. `{'oh', 'la', 'la'}`

2. `set([3, 4, 2, 3, 2, 2, 2, -1])`

3. `{(1, 2), (2, 3)}`

4. `set('aba')`

5. `str({'a'})`

6. `{1, 2, 3}`

7. `set(1, 2, 3)`

8. `set({1, 2, 3})`

9. `set([1, 2, 3])`

10. `set((1, 2, 3))`

11. `set("abc")`

12. `set("1232")`

13. `set([{1, 2, 3, 2}])`

14. `set([[1, 2, 3, 2]])`

15. `set([(1, 2, 3, 2)])`

16. `set(["abcb"])`

17. `set(["1232"])`

18. `set((1, 2, 3, 2))`

19. `set([(), ()])`

20. `set([])`

21. `set(list(set()))`

Exercise - dedup

Write some brief code to create a list `lb` which contains all the elements of the list `la` without duplicates and alphabetically sorted.

- DO NOT change original list `la`
- DO NOT use cycles
- your code should work for any `la`

```
la = ['c', 'a', 'b', 'c', 'd', 'b', 'e']
```

After your code, you should obtain:

```
>>> print(la)
['c', 'a', 'b', 'c', 'd', 'b', 'e']
>>> print(lb)
['a', 'b', 'c', 'd', 'e']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[16]: la = ['c', 'a', 'b', 'c', 'd', 'b', 'e']

# write here

lb = list(set(la))
lb.sort()
#lb = list(sorted(set(la))) # alternative, NOTE sorted generates a NEW sequence

print("la =",la)
print("lb =",lb)

la = ['c', 'a', 'b', 'c', 'd', 'b', 'e']
lb = ['a', 'b', 'c', 'd', 'e']
```

</div>

```
[16]: la = ['c', 'a', 'b', 'c', 'd', 'b', 'e']

# write here

la = ['c', 'a', 'b', 'c', 'd', 'b', 'e']
lb = ['a', 'b', 'c', 'd', 'e']
```

Frozenset

INFO: this topic is optional for the purposes of the book

In Python also exists *immutable* sets which are called `frozenset`. Here we just remind that since frozensets are *immutable* they do have associated a `hash` label and thus they can be inserted as elements of other sets. For other info we refer to the [official documentation](#)¹⁷⁴.

Operators

Operator	Syntax	Result	Description
<i>length</i>	<code>len(set)</code>	<code>int</code>	the number of elements in the set
<i>membership</i>	<code>el in set</code>	<code>bool</code>	verifies whether an element is contained in the set
<i>union</i>	<code>set1 set2</code>	<code>set</code>	union, creates a NEW set
<i>intersection</i>	<code>set1 & set2</code>	<code>set</code>	intersection, creates a NEW set
<i>difference</i>	<code>set1 - set2</code>	<code>set</code>	difference, creates a NEW set
<i>symmetric difference</i>	<code>set1 ^ set2</code>	<code>set</code>	symmetric difference, creates a NEW set
<i>equality</i>	<code>==, !=</code>	<code>bool</code>	checks whether two sets are equal or different

¹⁷⁴ <https://docs.python.org/3/library/stdtypes.html#frozenset>

len

```
[17]: len( {'a','b','c'} )
```

```
[17]: 3
```

```
[18]: len( set() )
```

```
[18]: 0
```

Exercise - distincts

Given a string `word`, write some code that:

- prints the distinct characters present in `word` as alphabetically ordered (without the square brackets!), together with their number
- prints the number of duplicate characters found in total

Example 1 - given:

```
word = "ababbbbcddd"
```

after your code it must print:

```
word      : ababbbbcddd
4 distincts : a,b,c,d
6 duplicates
```

Example 2 - given:

```
word = "cccccaaabbba"
```

after your code it must print:

```
word      : ccccaaaabbba
3 distinct : a,b,c
9 duplicates
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[19]: # write here
word = "ababbbbcddd"
#word = "cccccaaabbba"
s = set(word)
print("word      :", word)
la = list(s)
la.sort()
print(len(s), 'distincts :', ", ".join(la))
#print(len(s), 'distincts :', list(sorted(s)))  # ALTERNATIVE WITH SORTED
print(len(word) - len(s), 'duplicates')

word      : ababbbbcddd
4 distincts : a,b,c,d
6 duplicates
```

</div>

```
[19]: # write here
```

```
word      : ababbbbcd
4 distincts : a,b,c,d
6 duplicates
```

Membership

As for any sequence, when we want to check whether an element is contained in a set we can use the `in` operator which returns a boolean value:

```
[20]: 'a' in {'m', 'e', 'n', 't', 'a'}
```

```
[20]: True
```

```
[21]: 'z' in {'m', 'e', 'n', 't', 'a'}
```

```
[21]: False
```

in IS VERY FAST WHEN USED WITH SETS

The speed of `in` operator DOES NOT depend on the set dimension

This is a substantial difference with respect to other sequences we've already seen: if you try searching for an element with `in` through strings, lists or tuples, and the searched element is toward the end (or isn't there at all), Python will have to look through the whole sequence.

What can we search?

The price to pay for having a fast search is that we can only search *immutable* elements. Before searching, Python tries to calculate the `hash` label of the object to search: if it succeeds, it goes on searching, otherwise complains and raises an exception. For example, if we try searching a mutable element like a list, Python will refuse:

```
["lattuce", "rucola"] in { ("cabbage", "potatoes"),
                           ("lattuce", "rucola"),
                           ("radishes", "courgette") }

-----
TypeError                                     Traceback (most recent call last)
/tmp/ipykernel_23977/3154816283.py in <module>
----> 1 ["lattuce", "rucola"] in { ("cabbage", "potatoes"),
                                   ("lattuce", "rucola"),
                                   ("radishes", "courgette") }

TypeError: unhashable type: 'list'
```

QUESTION: What result do you expect from this code?

```
("rucola", "radishes") in { ("cabbage", "potatoes"),
                           ("lattuce", "rucola"),
                           ("radishes", "courgette") }
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show answer"
  data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: the set is created correctly because it only contains tuples as elements, which are immutable. Afterwards, by using the `in` operator we ask Python to verify whether the tuple ("rucola", "radishes") is an element of the set. We are looking for a tuple which is an immutable sequence, so Python succeeds in calculating the `hash` label and the search ends without errors. Since the tuple does not belong to the set, the expression produces the value `False`.

```
</div>
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `2*10 in {10, 20, 30, 40}`

2. `'four' in {'f', 'o', 'u', 'r'}`

3. `'aa' in set('aa')`

4. `'a' in set(['a', 'a'])`

5. `[3 in {3, 4}, 6 in {3, 4}]`

6. `4 in set([1, 2, 3]*4)`

7. `2 in {len('3.4'.split('.'))}`

8. `{'c', 'd'} in {('a', 'b'), ('c', 'd'), ('e', 'f')}`

9. `{'c', 'd'} in [{('a', 'b'), ('d', 'c')}, {'e', 'f'}]`

not in

To check whether something is **not** belonging to a sequence, we can use two forms:

not in - form 1:

```
[22]: "carrot" not in {"watermelon", "banana", "apple"}
```

```
[22]: True
```

```
[23]: "watermelon" not in {"watermelon", "banana", "apple"}
```

```
[23]: False
```

not in - forma 2

```
[24]: not "carrot" in {"watermelon", "banana", "apple"}
```

```
[24]: True
```

```
[25]: not "watermelon" in {"watermelon", "banana", "apple"}
```

```
[25]: False
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `4 not in {1,2,3}`
2. `'3' not in {1,2,3}`
3. `not 'a' in {'b', 'c'}`
4. `not {} in set([])`
5. `{not 'a' in {'a'}}`
6. `4 not in set((4,))`
7. `() not in set([(0)])`

QUESTION: the following expressions are similar. What do they have in common? What is the difference with the last one (beyond the fact it is a set)?

1. `'e' in 'abcde'`
2. `'abcde'.find('e') >= 0`
3. `'abcde'.count('e') > 0`
4. `'e' in ['a', 'b', 'c', 'd', 'e']`
5. `['a', 'b', 'c', 'd', 'e'].count('e') > 0`
6. `'e' in ('a', 'b', 'c', 'd', 'e')`
7. `('a', 'b', 'c', 'd', 'e').count('e') > 0`
8. `'e' in {'a', 'b', 'c', 'd', 'e'}`

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: All the expressions reported above return a boolean which is `True` if the element '`e`' is present in the sequence.

All the operations of search and/counting (`in`, `find`, `index`, `count`) on strings, lists and tuples take a search time which in the worst case like here can be equal to the sequence dimension ('`e`' is at the end).

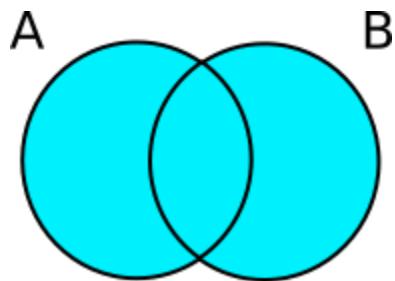
On the other hand, since sets (expression 8.) are based on *hashes*, they allow an immediate search, independently from the set dimension or the elements position (so creating the set with `e` at the end makes no difference).

To make performant searches it's preferable to use hash based collections, like sets or dictionaries !

</div>

Union

The union operator `|` (called *pipe*) produces a NEW set containing all the elements from both the first and second set.



```
[26]: {'a', 'b', 'c'} | {'b', 'c', 'd', 'e'}
```

```
[26]: {'a', 'b', 'c', 'd', 'e'}
```

Note there aren't duplicated elements

EXERCISE: What if we use the `+`? Try writing in a cell `{'a', 'b'} + {'c', 'd', 'e'}`. What happens?

```
[27]: # write here
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if they give an error):

1. `{'a', 'd', 'b'} | {'a', 'b', 'c'}`

2. `{'a'} | {'a'}`

3. `{'a' | 'b'}`

4. `{1 | 2 | 3}`

5. `{'a' | 'b' | 'a'}`

6. `{}{'a'} | {'b'} | {'a'}`

7. `[1, 2, 3] | [3, 4]`

8. `(1, 2, 3) | (3, 4)`

9. `"abc" | "cd"`

10. `{'a'} | set(['a', 'b'])`

11. `set(".".join('pacca'))`

12. `'{a}' | '{b}' | '{a}'`

13. `set((1, 2, 3)) | set([len([4, 5])])`

14. { () } | { () }

15. { ' | ' } | { ' | ' }

QUESTION: Given two sets x and y , the expression

```
len(x | y) <= len(x) + len(y)
```

produces:

1. an error (which one?)
2. always True
3. always False
4. sometimes True sometimes False according to values of x and y

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 2: the number of elements from the union will always be lesser or equal to the sum of the number of elements of each single set we are going to merge, so from the \leq comparison we will always get True.

</div>

Exercise - everythingbut 1

Write some code which creates a set $s4$ which contains all the elements of $s1$ and $s2$ but does not contain the elements of $s3$.

- Your code should work with *any* set $s1$, $s2$, $s3$

Example - given:

```
s1 = set(['a', 'b', 'c', 'd', 'e'])
s2 = set(['b', 'c', 'f', 'g'])
s3 = set(['b', 'f'])
```

After your code you should obtain:

```
>>> print(s4)
{'d', 'a', 'c', 'g', 'e'}
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[28]: s1 = set(['a', 'b', 'c', 'd', 'e'])
s2 = set(['b', 'c', 'f', 'g'])
s3 = set(['b', 'f'])

# write here
s4 = (s1 | s2) - s3
#print(s4)
```

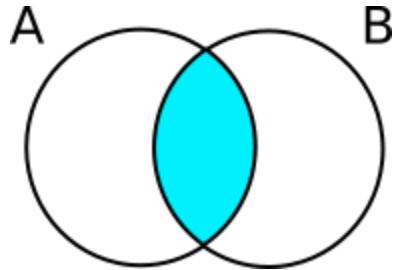
</div>

```
[28]: s1 = set(['a', 'b', 'c', 'd', 'e'])
s2 = set(['b', 'c', 'f', 'g'])
s3 = set(['b', 'f'])

# write here
```

Intersection

The intersection operator & produces a NEW set which contains all the common elements of the first and second set.



```
[29]: {'a', 'b', 'c'} & {'b', 'c', 'd', 'e'}
[29]: {'b', 'c'}
```

QUESTION: Look at the following expressions, and for each try guessing wthe result (or if it gives an error):

1. `{0}&{0,1}`

2. `{0,1}&{0}`

3. `set("capra") & set("campa")`

4. `set("cba") & set("dcba")`

5. `{len([1,2,3]),4} & {len([5,6,7])}`

6. `{1,2}&{1,2}`

7. `{0,1}&{ }`

8. `{0,1}&set()`

9. `'cc' in (set('pacca') & set('zucca'))`

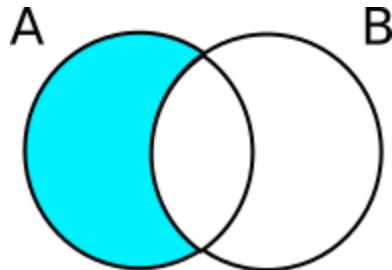
10. `set([1,2,3,4,5][::2]) & set([1,2,3,4,5][2::2])`

11. `{(),}&{()}`

12. `{()}&{()}`

Difference

The difference operator – produces a NEW set containing all the elements of the first set except the ones from the second:



```
[30]: {'a', 'b', 'c', 'd'} - {'b', 'c', 'e', 'f', 'g'}
```

```
[30]: {'a', 'd'}
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `{3, 4, 2} - 2`

2. `{1, 2, 3} - {3, 4}`

3. `'{"a"} - {"a"}`

4. `{1, 2, 3} -- {3, 4}`

5. `{1, 2, 3} - (-{3, 4})`

6. `set("chiodo") - set("chiave")`

7. `set("prova") - set("prova".capitalize())`

8. `set("BarbA") - set("BARBA".lower())`

9. `'c' in (set('parco') - set('cassa'))`

10. `set([(1, 2), (3, 4), (5, 6)]) - set([(2, 3), (4, 5)])`

11. `set([(1, 2), (3, 4), (5, 6)]) - set([(3, 4), (5, 6)])`

12. `{1, 2, 3} - set()`

13. `set() - {1, 2, 3}`

QUESTION: Given two sets `x` and `y`, what does the following code produce? An error? Is it simplifiable?

```
(x & y) | (x-y)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

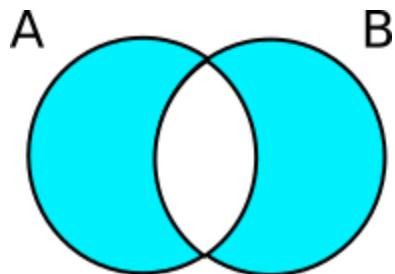
ANSWER: We are merging the common elements between `x` and `y`, with the elements present in `x` but not in `y`. Thus, we are taking all the elements of `x`, so the expression can be greatly simplified by just writing:

```
x
```

```
</div>
```

Symmetric difference

The symmetric difference of two sets is their union except their intersection, that is all elements except the common ones:



In Python you can directly express it with the `^` operator:

```
[31]: {'a', 'b', 'c'} ^ {'b', 'c', 'd', 'e'}  
[31]: {'a', 'd', 'e'}
```

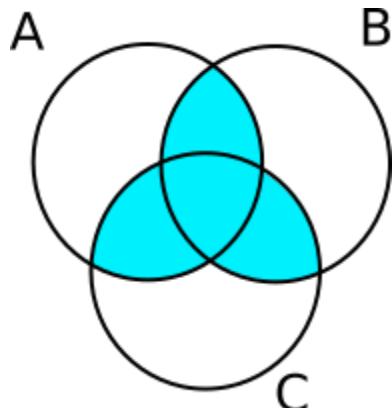
Let's check the result corresponds to the definition:

```
[32]: s1 = {'a', 'b', 'c'}  
s2 = {'b', 'c', 'd', 'e'}  
  
(s1 | s2) - (s1 & s2)  
[32]: {'a', 'd', 'e'}
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `{'p', 'e', 'p', 'p', 'o'} ^ {'p', 'a', 'p', 'p', 'e'}`
2. `{'ab', 'cd'} ^ {'ba', 'dc'}`
3. `set('broadino') ^ set('bordo')`
4. `set((1, 2, 5, 3, 2, 3, 1)) ^ set((1, 4, 3, 2))`

QUESTION: given 3 sets A, B, C, what's the expression to obtain the azure part?



Show answer

$(A \cap B) \cup (A \cap C) \cup (B \cap C)$

ANSWER:

```
(A & B) | (A & C) | (B & C)
```

</div>

QUESTION: If we use the following values in the previous exercise, what would the set which denotes the purple part contain?

```
A = {'a', 'ab', 'ac', 'abc'}
B = {'b', 'ab', 'bc', 'abc'}
C = {'c', 'ac', 'bc', 'abc'}
```

Once you guessed the result, try executing the formula you obtained in the previous exercise with the provided values and compare the results with the solution.

Show answer

$\{abc, ac, bc, ab\}$

ANSWER: If the formula is correct you should obtain:

```
{'abc', 'ac', 'bc', 'ab'}
```

</div>

Equality

We can check whether two sets are equal by using the equality operator `==`, which given two sets return `True` if they contain the same elements or `False` otherwise:

```
[33]: {4, 3, 6} == {4, 3, 6}
```

```
[33]: True
```

```
[34]: {4, 3, 6} == {4, 3}
```

```
[34]: False
```

```
[35]: {4, 3, 6} == {4, 3, 6, 'hello'}
```

[35]: False

Careful about removal of duplicates !

[36]: {2,8} == {2,2,8}

[36]: True

To verify the inequality, we can use the != operator:

[37]: {2,5} != {2,5}

[37]: False

[38]: {4,6,0} != {2,8}

[38]: True

[39]: {4,6,0} != {4,6,0,2}

[39]: True

Beware of duplicates and order!

[40]: {0,1} != {1,0,0,0,0,0,0,0}

[40]: False

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. {2 == 2, 3 == 3}

2. {1,2,3,2,1} == {1,1,2,2,3,3}

3. {'aa'} == {'a'}

4. set('aa') == {'a'}

5. [{1,2,3}] == {[1,2,3]}

6. set({1,2,3}) == {1,2,3}

7. set((1,2,3)) == {(1,2,3)}

8. {'aa'} != {'a', 'aa'}

9. {set()} != set()

10. set('scarpa') == set('capras')

11. set('papa') != set('pappa')

12. set('pappa') != set('reale')

13. `{(), ()} == {(())}`

14. `{(), ()} != {((()), (())}`

15. `[set()] == [set(), set()]`

16. `(set('gosh') | set('posh')) == (set('shopping') - set('in'))`

Methods like operators

There are methods which behave like the operators `|`, `&`, `-`, `^` by creating a **NEW** set.

NOTE: differently from operators, these methods accept as parameter *any* sequence, not just sets:

Method	Result	Description	Related operator
<code>set.union(seq)</code>	set	union, creates a NEW set	<code> </code>
<code>set.intersection(seq)</code>	set	intersection, creates a NEW set	<code>&</code>
<code>set.difference(seq)</code>	set	difference, creates a NEW set	<code>-</code>
<code>set.symmetric_difference(seq)</code>	set	symmetric difference, creates a NEW set	<code>^</code>

Methods which **MODIFY** the first set on which they are called (and return `None!`):

Method	Result	Description
<code>setA.update(setB)</code>	<code>None</code>	union, MODIFIES <code>setA</code>
<code>setA.intersection_update(setB)</code>	<code>None</code>	intersection, MODIFIES <code>setA</code>
<code>setA.difference_update(setB)</code>	<code>None</code>	difference, MODIFIES <code>setA</code>
<code>setA.symmetric_difference_update(setB)</code>	<code>None</code>	symmetric difference, MODIFIES <code>setA</code>

union

We'll only have a look at `union/update`, all other methods behave similarly

With `union`, given a set and a generic sequence (so not necessarily a set) we can create a NEW set:

[41]: `sa = {'g', 'a', 'r', 'a'}`

[42]: `la = ['a', 'g', 'r', 'a', 'r', 'i', 'o']`

[43]: `sb = sa.union(la)`

[44]: `sb`

[44]: `{'a', 'g', 'i', 'o', 'r'}`

EXERCISE: with `union` we can use any sequence, but that's not the case with operators. Try writing `{1, 2, 3} | [2, 3, 4]` and see what happens.

```
[45]: # write here
```

We can verify union creates a new set with Python Tutor:

```
[46]: sa = {'g', 'a', 'r', 'a'}
la = ['a', 'g', 'r', 'a', 'r', 'i', 'o']
sb = sa.union(la)

jupman.pytut()

[46]: <IPython.core.display.HTML object>
```

update

If we want to MODIFY the first set instead, we can use the methods ending with update:

```
[47]: sa = {'g', 'a', 'r', 'a'}

[48]: la = ['a', 'g', 'r', 'a', 'r', 'i', 'o']

[49]: sa.update(la)

[50]: print(sa)
{'a', 'r', 'i', 'g', 'o'}
```

QUESTION: what did the call to update return?

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: since Jupyter didn't show anything, it means the call to update method implicitly returned the None object.

</div>

Let's look what happened with Python Tutor - we also added a x = to put in evidence what was returned by calling .update:

```
[51]: sa = {'g', 'a', 'r', 'a'}
la = ['a', 'g', 'r', 'a', 'r', 'i', 'o']
x = sa.update(la)
print(sa)
print(x)

jupman.pytut()

{'a', 'r', 'i', 'g', 'o'}
None

[51]: <IPython.core.display.HTML object>
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `set('case').intersection('sebo') == 'se'`

```
2. set('naso').difference('caso')
```

```
3. s = {1,2,3}
s.intersection_update([2,3,4])
print(s)
```

```
4. s = {1,2,3}
s = s & [2,3,4]
```

```
5. s = set('cartone')
s = s.intersection('parto')
print(s)
```

```
6. sa = set("mastice")
sb = sa.difference("mastro").difference("collo")
print(sa)
print(sb)
```

```
7. sa = set("mastice")
sb = sa.difference_update("mastro").difference_update("collo")
print(sa)
print(sb)
```

Exercise - everythingbut 2

Given sets `s1`, `s2` e `s3`, write some code which MODIFIES `s1` so that it also contains the elements of `s2` but not the elements of `s3`:

- Your code should work with *any* set `s1`, `s2`, `s3`
- **DO NOT** create new sets

Example - given:

```
s1 = set(['a', 'b', 'c', 'd', 'e'])
s2 = set(['b', 'c', 'f', 'g'])
s3 = set(['b', 'f'])
```

After your code you should obtain:

```
>>> print(s1)
{'a', 'g', 'e', 'd', 'c'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[52]: s1 = set(['a', 'b', 'c', 'd', 'e'])
s2 = set(['b', 'c', 'f', 'g'])
s3 = set(['b', 'f'])

# write here
s1.update(s2)
s1.difference_update(s3)
print(s1)
```

```
{'a', 'd', 'c', 'g', 'e'}  
</div>  
[52]: s1 = set(['a','b','c','d','e'])  
s2 = set(['b','c','f','g'])  
s3 = set(['b','f'])  
  
# write here  
  
{'a', 'd', 'c', 'g', 'e'}
```

Other methods

Method	Result	Description
<code>set.add(el)</code>	None	adds the specified element - if already present does nothing
<code>set.remove(el)</code>	None	removes the specified element - if not present raises an error
<code>set.discard(el)</code>	None	removes the specified element - if not present does nothing
<code>set.pop()</code>	obj	removes an arbitrary element from the set and returns it
<code>set.clear()</code>	None	removes all the elements
<code>setA.issubset(setB)</code>	bool	checks whether setA is a subset of setB
<code>setA.issuperset(setB)</code>	bool	checks whether setA contains all the elements of setB
<code>setA.isdisjoint(setB)</code>	bool	checks whether setA has no element in common with setB

add method

Given a set, we can add an element with the method `.add`:

```
[53]: s = {3, 7, 4}  
  
[54]: s.add(5)  
  
[55]: s  
[55]: {3, 4, 5, 7}
```

If we add the same element twice, nothing happens:

```
[56]: s.add(5)  
  
[57]: s  
[57]: {3, 4, 5, 7}
```

QUESTION: If we write this code, which result do we get?

```
s = {'a', 'b'}  
s.add({'c', 'd', 'e'})  
print(s)
```

1. prints {'a', 'b', 'c', 'd', 'e'}

2. prints {{ 'a', 'b', 'c', 'd', 'e' }}
3. prints { 'a', 'b', {'c', 'd', 'e' }}
4. an error (which one?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 4 - produces `TypeError: unhashable type: 'set'`: we are trying to insert a set as element of another set, but sets are *mutable* so their *hash* label (which allows Python to find them quickly) might vary over time.

</div>

QUESTION: Look at the following code, which result does it produce?

```
x = {'a', 'b'}
y = set(x)
x.add('c')
print('x=', x)
print('y=', y)
```

1. an error (which one?)
2. x and y will be the same (how?)
3. x and y will be different (how?)

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 3. It will print:

```
x= {'c', 'a', 'b'}
y= {'a', 'b'}
```

because `y=set(x)` creates a NEW set by copying all the elements in the input sequence `x`.

Let's verify with Python Tutor:

</div>

```
[58]: x = {'a', 'b'}
y = set(x)
x.add('c')

jupman.pytut()
```

[58]: <IPython.core.display.HTML object>

remove method

The `remove` method takes the specified element out of the set. If it doesn't exist, it produces an error:

[59]: s = {'a', 'b', 'c'}

[60]: s.remove('b')

[61]: s

```
[61]: { 'a', 'c'}
```

```
[62]: s.remove('c')
```

```
[63]: s
```

```
[63]: {'a'}
```

```
s.remove('z')
```

```
-----  
KeyError Traceback (most recent call last)  
<ipython-input-266-a9e7a977e50c> in <module>  
----> 1 s.remove('z')  
  
KeyError: 'z'
```

Exercise - bababiba

Given a string `word` of exactly 4 syllabs of two characters each, create a set `s` which contains tuples with 2 characters each. Each tuple must represent a syllab taken from `word`.

- to add elements to the set, only use `add`
- your code must work for any `word` of 4 bisyllabs

Example 1 - given:

```
word = "bababiba"
```

after your code, it must result:

```
>>> print(s)
{('b', 'a'), ('b', 'i')}
```

Example 2 - given

```
word = "rubareru"
```

after your code, it must result:

```
>>> print(s)
{('r', 'u'), ('b', 'a'), ('r', 'e')}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[64]: word = "bababiba"
#word = "rubareru"

# write here

s = set()
s.add(tuple(word[:2]))
s.add(tuple(word[2:4]))
```

(continues on next page)

(continued from previous page)

```
s.add(tuple(word[4:6]))
s.add(tuple(word[6:8]))
print(s)

{('b', 'a'), ('b', 'i')}
```

</div>

```
[64]: word = "bababiba"
#word = "rubareru"

# write here
```

```
{('b', 'a'), ('b', 'i')}
```

discard method

The `discard` method removes the specified element from the set. If it doesn't exists, it does nothing (we may also say it *silently* discards the element):

```
[65]: s = {'a', 'b', 'c'}
```

```
[66]: s.discard('a')
```

```
[67]: s
```

```
[67]: {'b', 'c'}
```

```
[68]: s.discard('c')
```

```
[69]: s
```

```
[69]: {'b'}
```

```
[70]: s.discard('z')
```

```
[71]: s
```

```
[71]: {'b'}
```

Exercise - trash

⊕⊕ A waste processing plant receives a load of trash, which we represent as a set of strings:

```
trash = {'alkenes', 'vegetables', 'mercury', 'paper'}
```

To remove the contaminant elements which *might* be present (NOTE: they're not always present), the plant has exactly 3 filters (as list of strings) which will apply in series to the trash:

```
filters = ['cadmium', 'mercury', 'alkenes']
```

In order to check whether filters have effectively removed the contaminant(s), for each applied filter we want to see the state of the processed trash.

At the end, we also want to print all and *only* the contaminants which were actually removed (put them together in the variable `separated`)

- **DO NOT** use `if` commands
- **DO NOT** use cycles (the number of filters is fixed to 3, so you can just copy and paste code)
- Your code must work for *any* list `filters` of 3 elements and *any* set `trash`

Example - given:

```
filters = ['cadmium', 'mercury', 'alkenes']
trash = {'alkenes', 'vegetables', 'mercury', 'paper'}
```

After your code, it must show:

```
Initial trash: {'mercury', 'alkenes', 'vegetables', 'paper'}
Applying filter for cadmium : {'mercury', 'alkenes', 'vegetables', 'paper'}
Applying filter for mercury : {'alkenes', 'vegetables', 'paper'}
Applying filter for alkenes : {'vegetables', 'paper'}

Separated contaminants: {'mercury', 'alkenes'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[72]:

```
filters = ['cadmium', 'mercury', 'alkenes']
trash = {'alkenes', 'vegetables', 'mercury', 'paper'}

separated = trash.intersection(filters) # creates a NEW set

# write here
s = "Applying filter for"
print("Initial trash:", trash)
trash.discard(filters[0])
print(s,filters[0],":", trash)
trash.discard(filters[1])
print(s,filters[1],":", trash)
trash.discard(filters[2])
print(s,filters[2],":", trash)
print("")
```

```
print("Separated contaminants:", separated)
```

```
Initial trash: {'mercury', 'alkenes', 'paper', 'vegetables'}
Applying filter for cadmium : {'mercury', 'alkenes', 'paper', 'vegetables'}
Applying filter for mercury : {'alkenes', 'paper', 'vegetables'}
Applying filter for alkenes : {'paper', 'vegetables'}
```

```
Separated contaminants: {'mercury', 'alkenes'}
```

```
</div>
```

```
[72]: filters = ['cadmium', 'mercury', 'alkenes']
trash = {'alkenes', 'vegetables', 'mercury', 'paper'}

separated = trash.intersection(filters) # creates a NEW set

# write here
```

issubset method

To check whether all elements in a set `sa` are contained in another set `sb` we can write `sa.issubset(sb)`. Examples:

```
[73]: {2,4}.issubset({1,2,3,4})
```

```
[73]: True
```

```
[74]: {3,5}.issubset({1,2,3,4})
```

```
[74]: False
```

WARNING: the empty set is always considered a subset of any other set

```
[75]: set().issubset({3,4,2,5})
```

```
[75]: True
```

issuperset method

To verify whether a set `sa` contains all the elements of another set `sb` we can write `sa.issuperset(sb)`. Examples:

```
[76]: {1,2,3,4,5}.issuperset({1,3,5})
```

```
[76]: True
```

```
[77]: {1,2,3,4,5}.issuperset({2,4})
```

```
[77]: True
```

```
[78]: {1,2,3,4,5}.issuperset({1,3,5,7,9})
```

```
[78]: False
```

WARNING: the empty set is always considered a subset of any other set

```
[79]: {1,2,3,4,5}.issuperset({})
```

```
[79]: True
```

isdisjoint method

A set is disjoint from another one if it doesn't have any element in common, we can check for disjointness by using the method `isdisjoint`:

```
[80]: {1, 3, 5}.isdisjoint({2, 4})
```

```
[80]: True
```

```
[81]: {1, 3, 5}.isdisjoint({2, 3, 4})
```

```
[81]: False
```

QUESTION: Given a set `x`, what does the following expression produce?

```
x.isdisjoint(x)
```

1. an error (which one?)
2. always True
3. always False
4. True or False according to the value of `x`

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: 4, True or False according to the value of `x`.

Probably you thought the expression always returns `False`: after all, how could a set ever be disjoint from itself? In fact the expression almost always returns `False` *except* for the particular case of the empty set:

```
x = set()  
x.isdisjoint(x)
```

in which it returns `True`.

MORAL OF THE STORY: ALWAYS CHECK FOR THE EMPTY SET !

For this and many other methods the empty set often causes behaviours which aren't always intuitive, so we invite you to always check case by case.

</div>

Exercise - matrioska

⊕⊕ Given a list `sets` of exactly 4 sets, we define it a *matrioska* if each set contains all the elements of the previous set (plus eventually others). Write some code which PRINTS `True` if the sequence is a matrioska, otherwise PRINTS `False`.

- **DO NOT** use `if`
- your code must work for *any* sequence of exactly 4 sets
- **HINT:** you can create a list of 3 booleans which verify whether a set is contained in the next one ...

Example 1 - given:

```
sets = [{ 'a', 'b' },
        { 'a', 'b', 'c' },
        { 'a', 'b', 'c', 'd', 'e' },
        { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i' }]
```

after your code, it must print:

```
Is the sequence a matrioska? True
```

Example 2 - given:

```
sets = [{ 'a', 'b' },
        { 'a', 'b', 'c' },
        { 'a', 'e', 'd' },
        { 'a', 'b', 'd', 'e' }]
```

after your code, it must print:

```
Is the sequence a matrioska? False
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[82]:

```
sets = [{ 'a', 'b' },
        { 'a', 'b', 'c' },
        { 'a', 'b', 'c', 'd', 'e' },
        { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i' }]

#sets = [{ 'a', 'b' },
#         { 'a', 'b', 'c' },
#         { 'a', 'e', 'd' },
#         { 'a', 'b', 'd', 'e' }]

# write here

checks = [ sets[0].issubset(sets[1]),
            sets[1].issubset(sets[2]),
            sets[2].issubset(sets[3]) ]

print("Is the sequence a matrioska?", checks.count(True) == 3)
```

```
Is the sequence a matrioska? True
```

</div>

[82]:

```
sets = [{ 'a', 'b' },
        { 'a', 'b', 'c' },
        { 'a', 'b', 'c', 'd', 'e' },
        { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i' }]

#sets = [{ 'a', 'b' },
#         { 'a', 'b', 'c' },
```

(continues on next page)

(continued from previous page)

```
#         {'a', 'e', 'd'},
#         {'a', 'b', 'd', 'e'}]

# write here
```

Continue

Go on with [first challenges](#)¹⁷⁵

5.5.2 Sets 2 - First challenges

Download exercises zip

[Browse file online](#)¹⁷⁶

We now propose some exercises without solution, do you accept the challenge?

[4] :

[] :

5.6 Dictionaries

5.6.1 Dictionaries 1 - Introduction

Download exercises zip

[Browse files online](#)¹⁷⁷

Dictionaries are mutable containers which allow us to rapidly associate elements called *keys* to some *values*

- *Keys* are immutable, don't have order and there cannot be duplicates
- *Values* can be duplicated

Given a key, we can find the corresponding value very fast.

¹⁷⁵ <https://en.softpython.org/sets/sets2-chal.html>

¹⁷⁶ <https://github.com/DavidLeoni/softpython-en/sets>

¹⁷⁷ <https://github.com/DavidLeoni/softpython-en/tree/master/dictionaries>

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
sets
dictionaries1.ipynb
dictionaries1-sol.ipynb
dictionaries2.ipynb
dictionaries2-sol.ipynb
dictionaries3.ipynb
dictionaries3-sol.ipynb
dictionaries4.ipynb
dictionaries4-sol.ipynb
dictionaries5-chal.ipynb
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `dictionaries1.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

Creating a dictionary

In everyday life, when thinking about a dictionary we typically refer to a book which given an item (for example 'chair'), allows us to **rapidly** find the related description (i.e. a piece of furniture to sit on).

In Python we have a data structure called `dict` which provides an easy way to represent dictionaries.

Following the previous example, we might create a `dict` with different items like this:

```
[2]: {'chair': 'a piece of furniture to sit on',
      'cupboard': 'a cabinet for storage',
      'lamp': 'a device to provide illumination'}
```



```
[2]: {'chair': 'a piece of furniture to sit on',
      'cupboard': 'a cabinet for storage',
      'lamp': 'a device to provide illumination'}
```

Let's be clear about the naming:

Dictionaries are mutable containers which allow us to rapidly associate elements called **keys** to some **values**.

The definition says we have *keys* (in the example 'chair', 'cupboard', etc), while the descriptions from the example ('a piece of furniture to sit on') in Python are going to be called *values*.

When we create a dictionary, we first write a curly bracket {, then we follow it with a series of key : value couples, each followed by a comma , (except the last one, in which the comma is optional). At the end we close with a curly bracket }

Placing spaces or newlines inside **is optional**. So we can also write like this:

```
[3]: {'chair' : 'a piece of furniture to sit on',
       'cupboard' : 'a cabinet for storage',
       'lamp' : 'a device to provide illumination'}
```

```
[3]: {'chair': 'a piece of furniture to sit on',
      'cupboard': 'a cabinet for storage',
      'lamp': 'a device to provide illumination'}
```

Or also everything on a row:

```
[4]: {'chair':'a piece of furniture to sit on','cupboard':'a cabinet for storage','lamp':
      ↪'a device to provide illumination'}
```

```
[4]: {'chair': 'a piece of furniture to sit on',
      'cupboard': 'a cabinet for storage',
      'lamp': 'a device to provide illumination'}
```

Note if we use short words Python will probably print the dictionary in single a row anyway:

```
[5]: {'barca': 'remo',
      'auto': 'ruota',
      'aereo': 'ala'}
```

```
[5]: {'barca': 'remo', 'auto': 'ruota', 'aereo': 'ala'}
```

Putting a comma after the last couple does not give errors:

```
[6]: {
      'ship': 'paddle',
      'car': 'wheel',
      'airplane': 'wing', # note 'extra' comma
    }
```

```
[6]: {'ship': 'paddle', 'car': 'wheel', 'airplane': 'wing'}
```

Let's see how a dictionary is represented in Python Tutor - to ease the job, we will assign the variable `furniture` to it

```
[7]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
#           (it's sufficient to execute it only once)

import jupman
```

```
[8]: furniture = {
      'chair' : 'a piece of furniture to sit on',
      'cupboard': 'a cabinet for storage',
      'lamp' : 'a device to provide illumination'
}
print(furniture)

jupman.pytut()
```

```
{'chair': 'a piece of furniture to sit on', 'cupboard': 'a cabinet for storage', 'lamp  
→': 'a device to provide illumination'}
```

[8]: <IPython.core.display.HTML object>

We note that once executed, an arrow appears pointing from `furniture` to an orange/yellow memory region. The keys have orange background, while the corresponding values have yellow background. Looking at arrows and colors, we can guess that whenever we're assigning variables, dictionaries behave like other data structures, like lists and sets.

QUESTION: Look at the following code, and try guessing what happens during execution - at the end, how will memory be organized? What will be printed? Where will arrows go?

[9]:

```
da = {  
    'chair' : 'a piece of furniture to sit on',  
    'cupboard' : 'a cabinet for storage',  
    'lamp' : 'a device to provide illumination'  
}  
  
db = {  
    'ship': 'paddle',  
    'car': 'wheel',  
    'airplane': 'wing'  
}  
dc = db  
db = da  
da = dc  
dc = db  
#print(da)  
#print(db)  
#print(dc)  
  
jupman.pytut()
```

[9]: <IPython.core.display.HTML object>

The keys

Let's try to better understand which keys we can use by looking again at the definition:

Dictionaries are mutable containers which allow us to rapidly associate elements called *keys* to some *values*

- **Keys are immutable, don't have order and there cannot be duplicates**
- Values can be duplicated

QUESTION: have a careful look at the words in bold - can you tell a data structure we've already seen which has these features?

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: The keys of dictionaries for many aspects behave like elements of a set.

Have you read the tutorial on sets?¹⁷⁸?

Before going on, make sure to understand well the section on mutable elements and hashes¹⁷⁹

</div>

Keys are immutable

QUESTION: The definition does not force us to use strings as keys, other types are also allowed. But can we use all the types we want?

For each of the following examples, try to tell whether the dictionary can be created or we will get an error (which one?). Also check how they are represented in Python Tutor.

1. integers

```
{  
    4 : 'cats',  
    3 : 'dogs'  
}
```

2. float

```
{  
    4.0 : 'cats',  
    3.0 : 'dogs'  
}
```

3. strings

```
{  
    'a' : 'cats',  
    'b' : 'dogs'  
}
```

4. lists

```
{  
    [1,2] : 'zam',  
    [3,4] : 'zum'  
}
```

5. tuples

```
{  
    (1,2) : 'zam',  
    (4,3) : 'zum'  
}
```

6. sets

```
{  
    {1,2} : 'zam',  
    {3,4} : 'zum'  
}
```

7. other dictionaries (check the first part of the definition !)

¹⁷⁸ <https://en.softpython.org/sets/sets-sol.html>

¹⁷⁹ <https://eb.softpython.org/sets/sets-sol.html#Mutable-elements-and-hashes>

```
{
    {'a':'x', 'b':'y'} : 'zam',
    {'c':'w', 'd':'z'} : 'zum'
}
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: integers, float, strings and tuples are IMMUTABLE and so we can use them as keys (see definition). Instead, lists, sets (and other dictionaries) are MUTABLE, so we cannot use them as keys. If we try using a MUTABLE element such as a list like if it were the key of a dictionary, Python will complain, telling us the object is not *hashable* (exactly as it would complain if we tried to insert it in a set)

```
>>> { [1,2]:'zam',
      [3,4]:'zum' }

-----
TypeError                                     Traceback (most recent call last)
<ipython-input-12-c3c2d6cc97b8> in <module>
      1 { [1,2]:'zam',
----> 2     [3,4]:'zum' }

TypeError: unhashable type: 'list'
```

</div>

Keys don't have order

In a real-life dictionary, items are always ordered according to some criteria, typically in alphabetical order.

With Python we need to consider this important difference:

- The keys are immutable, **don't have order** and there cannot be duplicates

When we say that a collection ‘does not have order’, it means that the order of elements we see when we insert or print them does not matter to determine whether a collection is equal to another one. In dictionaries, it means that if we specify couples in a different order, we obtain dictionaries that Python considers as equal.

For example, the following dictionaries can all be considered as equal:

```
[10]: {
    'ships' : 'port',
    'airplanes': 'airport',
    'trains': 'station'
}

[10]: {'ships': 'port', 'airplanes': 'airport', 'trains': 'station'}

[11]: {
    'airplanes': 'airport',
    'ships' : 'port',
    'trains': 'station'
}

[11]: {'airplanes': 'airport', 'ships': 'port', 'trains': 'station'}
```

```
[12]: {
    'trains': 'station',
    'ships' : 'port',
    'airplanes': 'airport'
}

[12]: {'trains': 'station', 'ships': 'port', 'airplanes': 'airport'}
```

Printing a dictionary: you may have noticed that Jupyter always prints the keys in alphabetical order. This is just a courtesy for us, but do not be fooled by it! If we try a native print we will obtain a different result!

```
[13]: print({
    'ships' : 'port',
    'airplanes': 'airport',
    'trains': 'station'
})

{'ships': 'port', 'airplanes': 'airport', 'trains': 'station'}
```

Key duplicates

- Keys are immutable, don't have order and **there cannot be duplicates**

We might ask ourselves how Python manages duplicates in keys. Let's try to create a duplicated couple on purpose:

```
[14]: {
    'chair' : 'a piece of furniture to sit on',
    'chair' : 'a piece of furniture to sit on',
    'lamp'   : 'a device to provide illumination'
}

[14]: {'chair': 'a piece of furniture to sit on',
       'lamp': 'a device to provide illumination'}
```

We notice Python didn't complain and silently discarded the duplicate.

What if we try inserting a couple with the same key but different value?

```
[15]: {
    'chair' : 'a piece of furniture to sit on',
    'chair' : 'a type of seat',
    'lamp'   : 'a device to provide illumination'
}

[15]: {'chair': 'a type of seat', 'lamp': 'a device to provide illumination'}
```

Notice Python kept only the last couple.

The values

Let's see once again the definition:

Dictionaries are mutable containers which allow us to rapidly associate elements called keys to some values

- Keys are immutable, don't have order and there cannot be duplicates
- **Values can be duplicated**

Seems like values have less constraints than keys.

QUESTION: For each of the following examples, try to tell whether we can create the dictionary or we will get an error (which one?). Check how they are represented in Python Tutor.

1. integers

```
{
    'a':3,
    'b':4
}
```

2. duplicated integers

```
{
    'a':3,
    'b':3
}
```

3. float

```
{
    'a':3.0,
    'b':4.0
}
```

4. strings

```
{
    'a' : 'ice',
    'b' : 'fire'
}
```

5. lists

```
{
    'a' : ['t', 'w'],
    'b' : ['x'],
    'c' : ['y', 'z', 'k']
}
```

6. duplicated lists

```
{
    'a' : ['x', 'y', 'z'],
    'b' : ['x', 'y', 'z']
}
```

7. lists containing duplicates

```
{  
    'a' : [ 'x', 'y', 'y' ],  
    'b' : [ 'z', 'y', 'z' ]  
}
```

8. tuples

```
{  
    'a': (6, 9, 7),  
    'b': (8, 1, 7, 4)  
}
```

9. sets

```
{  
    'a' : { 6, 5, 6 },  
    'b' : { 2, 4, 1, 5 }  
}
```

10. dictionaries

```
{  
    'a' : {  
        'x': 3,  
        'y': 9  
    },  
    'b' : {  
        'x': 3,  
        'y': 9,  
        'z': 10  
    },  
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show answer"
data-jupman-hide="Hide">Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: We can freely put whatever we please as values, Python will not complain. In particular, notice how different keys can have the same value.

</div>

Empty dictionary

We can create an empty dictionary by writing {}:

WARNING: THIS IS NOT THE EMPTY SET¹⁸⁰ !!

[16]: {}

[16]: {}

[17]: type({})

¹⁸⁰ <https://en.softpython.org/sets/sets-sol.html#Empty-set>

[17]: dict

A dictionary is a collection, and as we've already seen (with lists, tuples and sets), we can create an empty collection by typing its type, in this case `dict`, followed by round brackets:

[18]: `dict()`

[18]: `{}`

Let's see how it's represented in Python Tutor:

[19]: `diz = dict()`

`jupman.pytut()`

[19]: <IPython.core.display.HTML object>

Keys and heterogenous values

So far we've always used keys all of the same type and values all of the same type, but this is not mandatory. (the only required thing is for key types to be immutable):

[20]: {

```
"a": 3,
"b": ["a", "list"],
7 : ("this", "is", "a", "tuple")
}
```

[20]: `{'a': 3, 'b': ['a', 'list'], 7: ('this', 'is', 'a', 'tuple')}`

NOTE: Although mixing types is possible, it's not advisable!

Throwing different types inside a dictionary often brings misfortune, as it increases probability of incurring into bugs.

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `{'a': 'b',
 'c': 'd'}`

2. `{'a b': 'c',
 'c d': 'e f'}`

3. `{'a' = 'c',
 'b' = 'd'}`

4. `{'a': 'b':
 'c': 'd'}`

5. `{
 "1": [2, 3],
 "2, 3": 1,
}`

6. `type({'a:b', 'c:d'})`

7. `{'a': 'b',
'c': 'd'}`

8. `{'a:b',
'c:d'}`

9. `{5, 2:
4, 5}`

10. `{1:2,
1:3}`

11. `{2:1,
3:1}`

12. `{'a': 'b',
'c': 'd', }`

13. `type({'a', 'b',
'c', 'd'})`

14. `{'a': 'b',
'c': 'd',
'e', 'f'}`

15. `{(): 2}`

16. `{(1, 2): [3, 4]}`

17. `{[1, 2]: (3, 4)}`

18. `{'[1, 2]': (3, 4)}`

19. `{ {1, 2}: (3, 4) }`

20. `{len({1, 2}): (3, 4)}`

21. `{5:{'a': 'b'}}`

22. `{"a":{1:2}}`

23. `{"a":{[1]:2}}`

24. `{"a":{1:[2]}}`

25. `{["a":{1:[2]}]}`

```
26. set([{2:4}])
```

Exercise - barone

Given a list of **exactly** 6 characters, build a dictionary `diz` as follows:

Example 1 - given:

```
lst = ['b', 'a', 'r', 'o', 'n', 'e']
```

after your code it must result (NOTE: the key order DOESN'T matter!)

```
>>> diz
{'b': ['a', 'r', 'o', 'n', 'e'],
 ('b', 'a', 'r', 'o', 'n', 'e'): {'a': 'b', 'b': 'e', 'n': 'o', 'o': 'r', 'r': 'n', 'e': 'e'},
 ('b', 'a', 'b', 'a'): ['r', 'o', 'r', 'o', 'n', 'e', 'n', 'e'],
 'b/a/r/o/n/e': {'b': 'a', 'r': 'o', 'n': 'e'}}
```

Example 2 - given:

```
lst = ['p', 'r', 'i', 'o', 'r', 'e']
```

it must result:

```
>>> diz
{'p': ['r', 'i', 'o', 'r', 'e'],
 ('p', 'r', 'i', 'o', 'r', 'e'): {'e': 'p', 'i': 'r', 'o': 'i', 'p': 'r', 'r': 'e', 'r': 'o'},
 ('p', 'r', 'p', 'r'): ['i', 'o', 'i', 'o', 'r', 'e', 'r', 'e'],
 'p/r/i/o/r/e': {'p': 'r', 'i': 'o', 'r': 'e'}}}
```

- USE only `lst`
- **IMPORTANT: DO NOT write string constants** (so no "barone", "b")

[Show solution](#)<div class="jupman-sol" data-jupman-code" style="display:none">

[21]:

```
lst = ['b', 'a', 'r', 'o', 'n', 'e']
lst = ['p', 'r', 'i', 'o', 'r', 'e']

# write here

{lst[0]: lst[1:],
 tuple(lst) : set(lst),
 tuple(lst[:2]) * 2 : lst[2:4]*2 + lst[4:]*2,
 '/'.join(lst) : {lst[0]:lst[1],
                   lst[2]:lst[3],
                   lst[4]:lst[5]}

[21]: {'p': ['r', 'i', 'o', 'r', 'e'],
 ('p', 'r', 'i', 'o', 'r', 'e'): {'e': 'p', 'i': 'r', 'o': 'i', 'p': 'r', 'r': 'e'},
 ('p', 'r', 'p', 'r'): ['i', 'o', 'i', 'o', 'r', 'e', 'r', 'e'],
 'p/r/i/o/r/e': {'p': 'r', 'i': 'o', 'r': 'e'}}}
```

</div>

```
[21]:  
lst = ['b', 'a', 'r', 'o', 'n', 'e']  
lst = ['p', 'r', 'i', 'o', 'r', 'e']  
  
# write here
```

Dictionary from a sequence of couples

We can obtain a dictionary by specifying a sequence of key/value couples as parameter of the function `dict`. For example we could pass a list of tuples:

```
[22]: dict( [  
    ('flour', 500),  
    ('eggs', 2),  
    ('sugar', 200),  
])  
  
[22]: {'flour': 500, 'eggs': 2, 'sugar': 200}
```

We can also use other sequences, the important bit is that subsequences must all have two elements. For example, here is a tuple of lists:

```
[23]: dict( (  
    ['flour', 500],  
    ['eggs', 2],  
    ['sugar', 200],  
))  
  
[23]: {'flour': 500, 'eggs': 2, 'sugar': 200}
```

If a subsequence has a number of elements different from two, we obtain this error:

```
>>> dict( (  
    ['flour', 500],  
    ['rotten', 'eggs', 3],  
    ['sugar', 200],  
))  
  
-----  
ValueError Traceback (most recent call last)  
<ipython-input-88-563d301b4aef> in <module>  
      2     ['flour', 500],  
      3     ['rotten', 'eggs', 3],  
      4     ['sugar', 200],  
      5   ) )  
  
ValueError: dictionary update sequence element #1 has length 3; 2 is required
```

QUESTION: Compare the following expressions. Do they do the same thing? If so, which one would you prefer?

```
dict( {  
    ('a', 5),  
    ('b', 8),  
    ('c', 3)  
} )
```

```
dict( (
    { 'a', 5 },
    { 'b', 8 },
    { 'c', 3 }
)
)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: The expressions do NOT produce the same result, and we must definitely prefer the first one.

On our pc, we obtained this:

WARNING: on your computer you may get different results!

```
# first
>>> dict( {
    ('a', 5),
    ('b', 8),
    ('c', 3) } )

{'b': 8, 'a': 5, 'c': 3}
```

```
# second
>>> dict( (
    { 'a', 5 },
    { 'b', 8 },
    { 'c', 3} ) )

{'a': 5, 8: 'b', 3: 'c'}
```

In the first case we started with a set of tuples: since it is a set, the elements inside it are memorized in an order we *cannot* predict. When Python checks the tuples inside, for each of them obtains a key/value couple. Now, from the dictionary definition we know dictionary keys are also memorized without a precise order. Thus, inserting keys in an order or another doesn't matter, the only important thing is keeping the key/value distinction. In the dictionary print we see the same couples we specified, only in different order: the proper couples have been created because tuples *are* ordered indeed.

In the second case we started instead from a tuple of sets, so Python visited the elements of the tuple in the same order as the one we see: alas, by specifying the couples like sets the order in which Python read the elements becomes unpredictable. On our computer, with the first set we've been lucky and Python first read 'a' and then 5, with the following sets it read instead first the number and then the character! On your computer you might see a completely different result!

</div>

QUESTION: Look at the following expressions, and for each try guessing which result it produces (or if it gives an error):

1. `dict('abcd')`

2. `dict([('ab', 'cd')])`

3. `dict([('a1', 'c2')])`

```
4. dict([])  
5. dict()  
6. dict(' ',)    # nasty
```

Exercise - galattico veramente

Given some variables use the constructor from sequences of couples to obtain the variable `diz`

- **DO NOT** use string constants in the code, nor particular numbers (so no 'Ga' nor 759). Using indexes is allowed.

Example 1 - given:

```
s = 'Ga'  
t = ('LA', 'tt')  
l1 = ['Ic', 'Co', 'Ve']  
l2 = ['Ra', 'Me', 'Nt']  
l3 = [[[ 'EEE', '...', ]]]  
n = 43.759
```

After your code, it must result (NOTE: the order of keys DOESN'T matter!)

```
>>> diz  
{'G': 'a',  
'LA': 'tt',  
'I': 'c',  
'C': 'o',  
'V': 'e',  
'R': 'a',  
'M': 'e',  
'N': 't',  
'EEE': '...',  
'43': '759'}
```

Example 2 - given:

```
s = 'Sp'  
t = ('Az', 'ia')  
l1 = ['Le', 'Si', 'De']  
l2 = ['Ra', 'Le', 'In']  
l3 = [[[ 'CREDIBBILE', '!!!!!!' ]]]  
n = 8744.92835
```

must result in:

```
>>> diz  
{'S': 'i',  
'Az': 'ia',  
'L': 'e',  
'D': 'e',  
'R': 'a',  
'I': 'n',  
'CREDIBBILE': '!!!!!!',  
'8744': '92835'}
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[24]:

```
s = 'Ga'
t = ('LA', 'tt')
l1 = ['Ic', 'Co', 'Ve']
l2 = ['Ra', 'Me', 'Nt']
l3 = [[[ 'EEE', '...']]]
n = 43.759

#s = 'Sp'
#t = ('Az', 'ia')
#l1 = ['Le', 'Si', 'De']
#l2 = ['Ra', 'Le', 'In']
#l3 = [[[ 'CREDIBBILE', '!!!!!!']]]
#n = 8744.92835

# write here

diz = dict([s, t] + l1 + l2 + l3[0] + [str(n).split('.')])
diz
```

[24]:

```
{'G': 'a',
'LA': 'tt',
'I': 'c',
'C': 'o',
'Ve': 'e',
'R': 'a',
'M': 'e',
'N': 't',
'EEE': '...',
'43': '759'}
```

</div>

[24]:

```
s = 'Ga'
t = ('LA', 'tt')
l1 = ['Ic', 'Co', 'Ve']
l2 = ['Ra', 'Me', 'Nt']
l3 = [[[ 'EEE', '...']]]
n = 43.759

#s = 'Sp'
#t = ('Az', 'ia')
#l1 = ['Le', 'Si', 'De']
#l2 = ['Ra', 'Le', 'In']
#l3 = [[[ 'CREDIBBILE', '!!!!!!']]]
#n = 8744.92835

# write here
```

Dictionary from keyword arguments

As further creation method, we can specify keys as they were parameters with a name:

```
[25]: dict(a=5,b=6)  
[25]: {'a': 5, 'b': 6}
```

WARNING: keys will be subject to the same restrictive rules of function parameter names!

For example, by using curly brackets this dictionary is perfectly lecit:

```
[26]: {'a b' : 2,  
       'c d' : 6}  
[26]: {'a b': 2, 'c d': 6}
```

But if we try creating it using `a b` as argument of `dict`, we will incur into problems:

```
>>> dict(a b=2, c d=6)  
  
File "<ipython-input-97-444f8661585a>", line 1  
    dict(a b=2, c d=6)  
          ^  
SyntaxError: invalid syntax
```

Strings will also give trouble:

```
>>> dict('a b'=2,'c d'=6)  
  
File "<ipython-input-98-45aafbb56e81>", line 1  
    dict('a b'=2,'c d'=6)  
          ^  
SyntaxError: keyword can't be an expression
```

And be careful about tricks like using variables, we won't obtain the desired result:

```
[27]: ka = 'a b'  
kc = 'c d'  
  
dict(ka=2,kc=6)  
[27]: {'ka': 2, 'kc': 6}
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `dict(3=5,2=8)`

2. `dict('costs'=9,'benefits'=15)`

3. `dict(_costs=9,_benefits=15)`

4. `dict(33trentini=5)`

5. `dict(trentini33=5)`
6. `dict(trentini_33=5)`
7. `dict(trentini-33=5)`
8. `dict(costs=1=2,benefits=3=3)`
9. `dict(costs=1==2,benefits=3==3)`
10. `v1 = 6
v2 = 8
dict(k1=v1,k2=v2)`

Copying a dictionary

There are two ways to copy a dictionary, you can either do a *shallow* copy or a *deep* copy.

Shallow copy

It is possible to create a shallow copy by passing another dictionary to function `dict`:

```
[28]: da = {'x':3,  
          'y':5,  
          'z':1}
```

```
[29]: db = dict(da)
```

```
[30]: print(da)  
{'x': 3, 'y': 5, 'z': 1}
```

```
[31]: print(db)  
{'x': 3, 'y': 5, 'z': 1}
```

In Python Tutor we will see two different memory regions:

```
[32]: da = {'x':3,  
          'y':5,  
          'z':1}  
db = dict(da)  
  
jupman.pytut()  
  
[32]: <IPython.core.display.HTML object>
```

QUESTION: can we also write like this? With respect to the previous example, will we obtain different results?

```
[33]: da = {'x':3,  
          'y':5,  
          'z':1}
```

(continues on next page)

(continued from previous page)

```
[33]: db = dict(dict(da))

jupman.pytut()

<IPython.core.display.HTML object>
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
    data-jupman-show="Show answer"
    data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">
```

ANSWER: The code produces the same results of previous example, although it is not efficient (a temporary dictionary will be created by the internal `dict` and then it will be immediately discarded)

```
</div>
```

Mutable values: In the example we used integer values, which are *immutable*. If we tried *mutable* values like lists, what would happen?

```
[34]: da = {'x': ['a', 'b', 'c'],
           'y': ['d'],
           'z': ['e', 'f']}
db = dict(da)

jupman.pytut()

<IPython.core.display.HTML object>
```

If you try executing Python Tutor, you will see an explosion of arrows which go from the new dictionary `db` to the values of `da` (which are lists). No panic! We are going to give a better explanation in the next notebook, for now just note that **with the shallow copy of mutable values the new dictionary will have memory regions in common with the original dictionary.**

Deep copy

When there are mutable shared memory regions like in the case above, it's easy to do mistakes and introduce subtle bugs you might notice much later in the development cycle.

In order to have completely separated memory regions, we can use *deep copy*.

First we must tell Python we intend to use functions from the module `copy`, and then we will be allowed to call its `deepcopy` function:

```
[35]: from copy import deepcopy

da = {'x': ['a', 'b', 'c'],
       'y': ['d'],
       'z': ['e', 'f']}
db = deepcopy(da)

jupman.pytut()

<IPython.core.display.HTML object>
```

If you execute the code in Python Tutor, you will notice that by following the arrow from `db` we will end up in a totally new orange/yellow memory region, which shares nothing with the memory region pointed by `da`.

QUESTION: Have a look at the following code - after its execution, will you see arrows going from `db` to elements of `da`?

```
[36]: da = { 'x': {1, 2, 3},
            'y': {4, 5}}
db = dict(da)
jupman.pytut()
```

```
[36]: <IPython.core.display.HTML object>
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: Yes, because the values from da are sets which are mutable.

</div>

Continue

Go on reading Dictionaries 2¹⁸¹

[]:

5.6.2 Dictionaries 2 - operators

[Download exercise zip](#)

Browse online files¹⁸²

There are several operators to manipulate dictionaries:

Operator	Syntax	Return	Description
<i>length</i>	len(dict)	int	Retorn the number of keys
<i>reading</i>	dict[key]	obj	Return the value associated to the key
<i>writing</i>	dict[key] = value		Adds or modify the value associated to the key
<i>deletion</i>	del dict[key]		Removes the key/value couple
<i>membership</i>	obj in dict	bool	Return True if the key obj is present in dict
<i>equality</i>	==, !=	bool	Checks whether two dictionaries are equal or different

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
dictionaries
dictionaries1.ipynb
dictionaries1-sol.ipynb
dictionaries2.ipynb
dictionaries2-sol.ipynb
dictionaries3.ipynb
dictionaries3-sol.ipynb
dictionaries4.ipynb
dictionaries4-sol.ipynb
dictionaries5-chal.ipynb
jupman.py
```

¹⁸¹ <https://en.softpython.org/dictionaries/dictionaries2-sol.html>

¹⁸² <https://github.com/DavidLeoni/softpython-en/tree/master/dictionaries>

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `dictionaries2.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

len

We can obtain the number of key/value associations in a dictionary by using the function `len`:

```
[2]: len({'a':5,  
         'b':9,  
         'c':7  
})
```

```
[2]: 3
```

```
[3]: len({3:8,  
         1:3  
})
```

```
[3]: 2
```

```
[4]: len({})
```

```
[4]: 0
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `len(dict())`

2. `len({'a':{}})`

3. `len({(1,2):{3}, (4,5):{6}, (7,8):{9}})`

4. `len({1:2, 1:2, 2:4, 2:4, 3:6, 3:6})`

5. `len({1:2, ':3', ':4, })`

6. `len(len({3:4, 5:6}))`

Reading a value

At the end of dictionaries definition, it is reported:

Given a key, we can find the corresponding value very fast

How can we specify the key to search? It's sufficient to use square brackets [], a bit like we already did for lists:

```
[5]: furniture = {
    'chair' : 'a piece of furniture to sit on',
    'cupboard' : 'a cabinet for storage',
    'lamp' : 'a device to provide illumination'
}
```

```
[6]: furniture['chair']
[6]: 'a piece of furniture to sit on'
```

```
[7]: furniture['lamp']
[7]: 'a device to provide illumination'
```

WARNING: What we put in square parenthesis **must** be a key present in the dictionary

If we put keys which are not present, we will get an error:

```
>>> furniture['armchair']

-----
KeyError                                                 Traceback (most recent call last)
<ipython-input-19-ee891f51417b> in <module>
----> 1 furniture['armchair']

KeyError: 'armchair'
```

Fast disorder

Whenever we give a key to Python, how fast is it in getting the corresponding value? Very fast, so much so the speed *does not depend on the dictionary dimension*. Whether it is small or huge, given a key it will always find the associated value in about the same time.

When we hold a dictionary in real life, we typically have an item to search for and we turn pages until we get what we're looking for: the fact items are sorted allows us to rapidly find the item.

We might expect the same also in Python, but if we look at the definition we find a notable difference:

Dictionaries are mutable containers which allow us to rapidly associate elements called keys to some values

Keys are immutable, **don't have order** and there cannot be duplicates Values can be duplicated

If keys are *not* ordered, how can Python get the values so fast? The speed stems from the way Python memorizes keys, which is based on *hashes*, similarly for what happens with sets¹⁸³. The downside is we can only *immutable* objects as keys.

¹⁸³ <https://en.softpython.org/sets/sets-sol.html#Mutable-elements-and-hashes>

QUESTION: If we wanted to print the value 'a device to provide illumination' we see at the bottom of the dictionary, without knowing it corresponds to lamp, would it make sense to write something like this?

```
furniture = {'chair':'a piece of furniture to sit on',
              'cupboard':'a cabinet for storage',
              'lamp': 'a device to provide illumination'
}

print( furniture[2] )
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: Absolutely NOT. The couples key/value in the dictionary *are not* ordered, so it makes no sense to get a value at a given position.

</div>

QUESTION: Look at the following expressions, and for each try guessing which result it produces (or if it gives an error):

```
kabbalah = {
    1 : 'Progress',
    3 : 'Love',
    5 : 'Creation'
}
```

- kabbalah[0]
- kabbalah[1]
- kabbalah[2]
- kabbalah[3]
- kabbalah[4]
- kabbalah[5]
- kabbalah[-1]

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: In the dictionary we have keys which are integer numbers: so we can use numbers among square brackets, which we will call *keys*, but not *positions*.

The unique expressions which will produce results are those for which the number specified among the square brackets is effectively present among the keys:

```
>>> kabbalah[1]
'Progress'
>>> kabbalah[3]
'Love'
>>> kabbalah[5]
'Creation'
```

All others will give `KeyError`, like:

```
>>> kabbalah[2]
-----
KeyError                                     Traceback (most recent call last)
<ipython-input-29-de66b9721e9b> in <module>
      5 }
      6
----> 7 kabbalah[2]

KeyError: 2
```

</div>

QUESTION: Look at the following code fragments, and for each try guessing which result it produces (or if it gives an error):

1. `{'a':4, 'b':5}('a')`2. `{1:2, 2:3, 3:4}[2]`3. `{'a':1, 'b':2}['c']`4. `{'a':1, 'b':2}[a]`5. `{'a':1, 'b':2}[1]`6. `{'a':1, 'b':2, 'c':3}['c']`7. `{'a':1, 'b':2, 'c':3}[len(['a', 'b', 'c'])]`8. `{(3,4):(1,2)}[(1,2)]`9. `{(1,2):(3,4)}[(1,2)]`10. `{[1,2]:[3,4]}[[1,2]]`11. `{'a','b','c'}['a']`12. `{'a:b','c:d'}['c']`13. `{'a':4, 'b':5}{'a'}`14. `d1 = {'a':'b'}
d2 = {'b':'c'}
print(d1[d2['c']])`15. `d1 = {'a':'b'}
d2 = {'b':'c'}
print(d2[d1['a']])`16. `{[]}`

17. { [] : 3 } [[]]

18. { 1 : 7 } ['1']

19. { '1' : 7 } " [] "

20. { '1' : 7 } [""]

21. { "" : 7 } ['1']

22. { '1' : () } ['1']

23. { () : 7 } [()]

24. { (()) : 7 } [()]

25. { (()) : 7 } [((),)]

Exercise - z7

⊕ Given a dictionary d1 with keys 'b' and 'c' and integer values, create a dictionary d2 containing the key 'z' and associate to it the sum of values of keys from d1

- your code must work for *any* d1 with keys 'b' and 'c'

Example - given:

```
d1 = { 'a':6, 'b':2, 'c':5}
```

After your code, it must result:

```
>>> print(d2)
{ 'z': 7}
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[8]:

```
d1 = { 'a':6, 'b':2, 'c':5}

# write here

d2 = { 'z' : d1['b'] + d1['c'] }

print(d2)
```

```
{ 'z': 7}
```

```
</div>
```

[8]:

```
d1 = { 'a':6, 'b':2, 'c':5}
```

(continues on next page)

(continued from previous page)

```
# write here
```

Writing in the dictionary

Can we write in a dictionary?

Dictionaries are mutable containers which allow us to rapidly associate elements called keys to some values

The definition talks about mutability, so we are allowed to modify dictionaries after creation.

Dictionaries are collections of key/value couples, and among the possible modifications we find:

1. adding a key/value couple
2. associate an existing key to a different value
3. remove a key/value couple

Writing - adding key/value

Suppose we created our dictionary furniture:

[9]:

```
furniture = {
    'chair' : 'a piece of furniture to sit on',
    'cupboard' : 'a cabinet for storage',
    'lamp' : 'a device to provide illumination'
}
```

and afterwards we want to add a definition for 'armchair'. We can reuse the variable furniture followed by square brackets with inside the key we want to add ['armchair'] and after the brackets we will put an equality sign =

[10]: furniture['armchair'] = 'a chair with armrests'

Note Jupyter didn't show results, because the previous operation is an assignment *command* (only *expressions* generate results).

But something did actually happen in memory, we can check it by furniture:

[11]: furniture

```
{'chair': 'a piece of furniture to sit on',
 'cupboard': 'a cabinet for storage',
 'lamp': 'a device to provide illumination',
 'armchair': 'a chair with armrests'}
```

Note the dictionary associated to the variable furniture was **MODIFIED** with the addition of 'armchair'.

When we add a key/value couple, we can use heterogenous types:

```
[12]: trashcan = {
    'bla' : 3,
    4     : 'boh',
(7, 9) : ['gar', 'bage']
}
```

```
[13]: trashcan[5.0] = 'a float'
```

```
[14]: trashcan
```

```
[14]: {'bla': 3, 4: 'boh', (7, 9): ['gar', 'bage'], 5.0: 'a float'}
```

We are subject to the same constraints on keys we have during the creation, so we can only use *immutable* keys. If we try inserting a *mutable* type, for example a list, we will get an error:

```
>>> trashcan[ ['some', 'list'] ] = 8
-----
TypeError                                     Traceback (most recent call last)
<ipython-input-51-195ac9c21bcd> in <module>
----> 1 trashcan[ ['some', 'list'] ] = 8
TypeError: unhashable type: 'list'
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if gives an error):

```
1. d = {1:'a'}
d[2] = 'a'
print(d)
```

```
2. d = {}
print(len(d))
d['a'] = 'b'
print(len(d))
```

```
3. d1 = {'a':3, 'b':4}
diz2 = diz1
diz1['a'] = 5
print(diz1)
print(diz2)
```

```
4. diz1 = {'a':3, 'b':4}
diz2 = dict(diz1)
diz1['a'] = 5
print(diz1)
print(diz2)
```

```
5. la = ['a','c']
diz = {'a':3,
       'b':4,
       'c':5}
diz['d'] = diz[la[0]] + diz[la[1]]
print(diz)
```

```
6. diz = {}
diz[()] = ''
diz[('a',)] = 'A'
diz[('a', 'b')] = 'AB'
print(diz)
```

```
7. la = [5, 8, 6, 9]
diz = {}
diz[la[0]] = la[2]
diz[la[2]] = la[0]
print(diz)
```

```
8. diz = {}
diz[(4, 5, 6)[2]] = 'c'
diz[(4, 5, 6)[1]] = 'b'
diz[(4, 5, 6)[0]] = 'a'
print(diz)
```

```
9. diz1 = {
    'a' : 'x',
    'b' : 'x',
    'c' : 'y',
    'd' : 'y',
}

diz2 = {}
diz2[diz1['a']] = 'a'
diz2[diz1['b']] = 'b'
diz2[diz1['c']] = 'c'
diz2[diz1['d']] = 'd'
print(diz2)
```

Writing - reassocciate a key

Let's suppose to change the definition of a lamp:

```
[15]: furniture = {'chair':'a piece of furniture to sit on',
                  'cupboard':'a cabinet for storage',
                  'lamp': 'a device to provide illumination'
}
```

```
[16]: furniture['lamp'] = 'a device to provide visible light from electric current'
```

```
[17]: furniture
{'chair': 'a piece of furniture to sit on',
 'cupboard': 'a cabinet for storage',
 'lamp': 'a device to provide visible light from electric current'}
```

Exercise - workshop

⊕ MODIFY the dictionary `workshop`:

1. set the '`bolts`' key value equal to the value of the '`pincers`' key
2. increment the value of `wheels` key of 1
 - your code must work with any number associated to the keys
 - **DO NOT** create new dictionaries, so no lines beginning with `workshop = {`

Example - given:

```
workshop = {'wheels':3,
            'bolts':2,
            'pincers':5}
```

after your code, you should obtain:

```
>>> print(workshop)
{'bolts': 5, 'wheels': 4, 'pincers': 5}
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: workshop = {'wheels' : 3,
                  'bolts' : 2,
                  'pincers': 5}

# write here

workshop['wheels'] = workshop['wheels'] + 1
workshop['bolts'] = workshop['pincers']
#print(workshop)
```

</div>

```
[18]: workshop = {'wheels' : 3,
                  'bolts' : 2,
                  'pincers': 5}

# write here
```

QUESTION: Look at the following code fragments expressions, and for each try guessing the result it produces (or if it gives an error):

```
1. diz = {'a':'b'}
diz['a'] = 'a'
print(diz)
```

```
2. diz = {'1':'2'}
diz[1] = diz[1] + 5    # nasty
print(diz)
```

```
3. diz = {1:2}
diz[1] = diz[1] + 5
print(diz)
```

```
4. d1 = {1:2}
d2 = {2:3}
d1[1] = d2[d1[1]]
print(d1)
```

Writing - deleting

To remove a key/value couple the special command `del` is provided. Let's take a dictionary:

```
[19]: kitchen = {
    'pots' : 3,
    'pans' : 7,
    'forks' : 20
}
```

If we want to eliminate the couple `pans : 7`, we will write `del` followed by the name of the dictionary and the key to eliminate among square brackets:

```
[20]: del kitchen['pans']
```

```
[21]: kitchen
```

```
[21]: {'pots': 3, 'forks': 20}
```

Trying to delete a non-existent key will produce an error:

```
>>> del cucina['crankshaft']

-----
KeyError                                     Traceback (most recent call last)
<ipython-input-34-c0d541348698> in <module>
----> 1 del cucina['crankshaft']

KeyError: 'crankshaft'
```

QUESTION: Look at the following code fragments, and for each try guessing which result it produces (or if it gives an error):

```
1. diz = {'a':'b'}
del diz['b']
print(diz)
```

```
2. diz = {'a':'b', 'c':'d'}
del diz['a']
print(diz)
```

```
3. diz = {'a':'b', 'c':'d'}
del diz['a']
del diz['a']
print(diz)
```

```
4. diz = {'a':'b'}
new_diz = del diz['a']
print(diz)
print(new_diz)
```

```
5. diz1 = {'a':'b', 'c':'d'}
diz2 = diz1
del diz1['a']
print(diz1)
print(diz2)
```

```
6. diz1 = {'a':'b', 'c':'d'}
diz2 = dict(diz1)
del diz1['a']
print(diz1)
print(diz2)
```

```
7. diz = {'a':'b'}
del diz['c']
print(diz)
```

```
8. diz = {'a':'b'}
diz.del('a')
print(diz)
```

```
9. diz = {'a':'b'}
diz['a'] = None
print(diz)
```

Exercise - desktop

Given a dictionary desktop:

```
desktop = {
    'paper' : 5,
    'pencils':2,
    'pens'   :3
}
```

write some code which MODIFIES it so that after executing your code, the dictionary appears like this:

```
>>> print(desktop)
{'pencil sharpeners': 1, 'paper': 5, 'pencils': 2, 'papers': 4}
```

- **DO NOT** write lines which begin with desktop =

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[22]:

```
desktop = {
    'paper' : 5,
    'pencils':2,
    'pens'   :3
```

(continues on next page)

(continued from previous page)

```

}

# write here
desktop['papers'] = 4
del desktop['pens']
desktop['pencil sharpeners'] = 1
print(desktop)

{'paper': 5, 'pencils': 2, 'papers': 4, 'pencil sharpeners': 1}

```

</div>

[22]:

```

desktop = {
    'paper' : 5,
    'pencils': 2,
    'pens'   : 3
}

# write here

```

Exercise - garden

You have a dictionary `garden` which associates the names of present objects and their quantity. You are given:

- a list `to_remove` containing the names of exactly two objects to eliminate
- a dictionary `to_add` containing exactly two names of flowers associated to their quantity to add

MODIFY the dictionary `garden` according to the quantities given in `to_remove` (**deleting the keys**) and `to_add` (**increasing the corresponding values**)

- assume that `garden` always contains the objects given in `to_remove` and `to_add`
- assume that `to_add` always and only contains `tulips` and `roses`

Example - given:

```

to_remove = ['weeds', 'litter']
to_add = { 'tulips': 4,
           'roses' : 2
}

garden = { 'sunflowers': 3,
           'tulips'     : 7,
           'weeds'      : 10,
           'roses'      : 5,
           'litter'     : 6,
}

```

after your code, it must result:

```

>>> print(garden)
{'roses': 7, 'tulips': 11, 'sunflowers': 3}

```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[23]:

```
to_remove = ['weeds', 'litter']
to_add = { 'tulips': 4,
           'roses' : 2
         }

garden = { 'sunflowers': 3,
           'tulips' : 7,
           'weeds' : 10,
           'roses' : 5,
           'litter' : 6,
         }

# write here

del garden[to_remove[0]]
del garden[to_remove[1]]
garden['roses'] = garden['roses'] + to_add['roses']
garden['tulips'] = garden['tulips'] + to_add['tulips']
print(garden)

{'sunflowers': 3, 'tulips': 11, 'roses': 7}
```

</div>

[23]:

```
to_remove = ['weeds', 'litter']
to_add = { 'tulips': 4,
           'roses' : 2
         }

garden = { 'sunflowers': 3,
           'tulips' : 7,
           'weeds' : 10,
           'roses' : 5,
           'litter' : 6,
         }

# write here
```

Exercise - translations

Given two dictionaries `en_it` and `it_es` of English-Italian and Italian-Spanish translations, write some code which MODIFIES a third dictionary `en_es` by placing translations from English to Spanish

- assume that `en_it` always and only contains translations of `hello` and `road`
- assume that `it_es` always and only contains translations of `ciao` and `strada`
- in the solution, **ONLY** use the constants '`hello`' and '`road`', you will take the others you need from the dictionaries
- **DO NOT** create a new dictionary - so no lines beginning with `en_es = {`

Example - given:

```
en_it = {
    'hello' : 'ciao',
    'road' : 'strada'
}

it_es = {
    'ciao' : 'hola',
    'strada' : 'carretera'
}
en_es = {}
```

after your code, it must print:

```
>>> print(en_es)
{'hello': 'hola', 'road': 'carretera'}
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[24]:

```
en_it = {
    'hello' : 'ciao',
    'road' : 'strada'
}

it_es = {
    'ciao' : 'hola',
    'strada' : 'carretera'
}

en_es = {}

# write here
en_es['hello'] = it_es[en_it['hello']]
en_es['road'] = it_es[en_it['road']]
print(en_es)
```

</div>

[24]:

```
en_it = {
    'hello' : 'ciao',
    'road' : 'strada'
}

it_es = {
    'ciao' : 'hola',
    'strada' : 'carretera'
}

en_es = {}

# write here
```

Membership with `in`

We can check whether a *key* is present in a dictionary by using the operator `in`:

```
[25]: 'a' in {'a':5,'b':7}
```

```
[25]: True
```

```
[26]: 'b' in {'a':5,'b':7}
```

```
[26]: True
```

```
[27]: 'z' in {'a':5,'b':7}
```

```
[27]: False
```

WARNING: `in` searches among the *keys*, not in *values*!

```
[28]: 5 in {'a':5,'b':7}
```

```
[28]: False
```

As always when dealing with keys, we *cannot* search for a mutable object, like for example lists:

```
>>> [3,5] in {'a':'c','b':'d'}  
-----  
TypeError Traceback (most recent call last)  
<ipython-input-41-3e3e336117aa> in <module>  
----> 1 [3,5] in {'a':'c','b':'d'}  
  
TypeError: unhashable type: 'list'
```

`not in`

It is possible to check for *non* belonging with the `not in` operator:

```
[29]: 'z' not in {'a':5,'b':7}
```

```
[29]: True
```

```
[30]: 'a' not in {'a':5,'b':7}
```

```
[30]: False
```

Equivalently, we can use this other form:

```
[31]: not 'z' in {'a':5,'b':7}
```

```
[31]: True
```

```
[32]: not 'a' in {'a':5,'b':7}
```

```
[32]: False
```

QUESTION: Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. ('a') in {'a':5}
2. ('a', 'b') in {('a', 'b'):5}
3. ('a', 'b',) in {('a', 'b'):5}
4. ['a', 'b'] in {('a', 'b'):5}
5. {3: 'q' in {'q':5}}
6. {'q' not in {'q':0} : 'q' in {'q':0}}
7. {'a' in 'b'}
8. {'a' not in {'b':'a'}})
9. len({'a':6, 'b':4}) in {1:2}
10. 'ab' in {('a', 'b'): 'ab'}
11. None in {}
12. None in {'None':3}
13. None in {None:3}
14. not None in {0:None}

Exercise - The Helmsman

The restaurant “The Helmsman” serves a menu with exactly 3 courses each coupled with a side dish. The courses and the side dishes are **numbered from 1 to 12**. There are many international clients who don’t speak well the local language, so they often simply point a course number. They never point a side dish. Once the `order` is received, the waiter with a tablet verifies whether the course is ready with the correct side dish. Write some code which given an index of a `course` shows `True` if this is in the `kitchen` coupled with the course, `False` otherwise.

- **DO NOT** use `if`
- **DO NOT** use loops nor list comprehensions
- **HINT:** if you don’t know how to do it, look at [Booleans - Evaluation order](#)¹⁸⁴

Example 1 - given:

```
# 1          2          3          4          5          6
menu = ['herring', 'butter', 'orata', 'salad',   'salmon',   'potatoes',
# 7          8          9          10         11         12
      'tuna',    'beans',    'salmon',  'lemon',  'herring', 'salad']
```

(continues on next page)

¹⁸⁴ <https://en.softpython.org/basics/basics2-bools-sol.html#Evaluation-order>

(continued from previous page)

```
kitchen = {'orata':'salad',
           'salmon':'potatoes',
           'herring':'salad',
           'tuna':'beans'}
```

order = 1

The program will show False, because there is no association "herring" : "butter" in kitchen

Example 2 - given:

```
order = 3
```

the program will show True because there is the association "orata" : "salad" in cambusa

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[33]:

```
order = 1      # False
#order = 3    # True
#order = 5    # True
#order = 7    # True
#order = 9    # False
#order = 11   # True

      # 1          2          3          4          5          6
menu = ['herring', 'butter', 'orata', 'salad', 'salmon', 'potatoes',
        # 7          8          9          10         11         12
        'tuna',     'beans',    'salmon',   'lemon',   'herring', 'salad']
```

```
kitchen = {'orata':'salad',
           'salmon':'potatoes',
           'herring':'salad',
           'tuna':'beans'}
```

write here

```
menu[order-1] in kitchen and kitchen[menu[order-1]] == menu[order]
```

[33]:

False

</div>

[33]:

```
order = 1      # False
#order = 3    # True
#order = 5    # True
#order = 7    # True
#order = 9    # False
#order = 11   # True

      # 1          2          3          4          5          6
menu = ['herring', 'butter', 'orata', 'salad', 'salmon', 'potatoes',
        # 7          8          9          10         11         12
        'tuna',     'beans',    'salmon',   'lemon',   'herring', 'salad']
```

(continues on next page)

(continued from previous page)

```
kitchen = {'orata':'salad',
           'salmon':'potatoes',
           'herring':'salad',
           'tuna':'beans'}

# write here
```

Dictionaries of sequences

So far we almost always associated a single value to keys. What if wanted to associate more? For example, suppose we are in a library and we want to associate users with the books they borrowed. We could represent everything as a dictionary where a list of borrowed books is associated to each customer:

```
[34]: loans = {'Marco': ['Les Misérables', 'Ulysses'],
             'Gloria': ['War and Peace'],
             'Rita': ['The Shining', 'Dracula', '1984']}
```

Let's see how it gets represented in Python Tutor:

```
[35]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
#           (it's sufficient to execute it only once)

import jupman
```

```
[36]: loans = {'Marco': ['Les Misérables', 'Ulysses'],
             'Gloria': ['War and Peace'],
             'Rita': ['The Shining', 'Dracula', '1984']}
jupman.pyput()
```

```
[36]: <IPython.core.display.HTML object>
```

If we try writing the expression:

```
[37]: loans['Rita']
[37]: ['The Shining', 'Dracula', '1984']
```

Python shows the corresponding list: for all intents and purposes Python considers `loans['Rita']` as if it were a list, and we can use it as such. For example, if we wanted to access the 1-indexed book of the list, we would write `[1]` after the expression:

```
[38]: loans['Rita'][1]
[38]: 'Dracula'
```

Equivalently, we might also save a pointer to the list by assigning the expression to a variable:

```
[39]: ritas_list = loans['Rita']
[40]: ritas_list
```

```
[40]: ['The Shining', 'Dracula', '1984']
```

```
[41]: ritas_list[1]
```

```
[41]: 'Dracula'
```

Let's see everything in Python Tutor:

```
[42]: loans = {'Marco': ['Les Misérables', 'Ulysses'],
              'Gloria': ['War and Peace'],
              'Rita': ['The Shining', 'Dracula', '1984']}
ritas_list = loans['Rita']
print(ritas_list[1])

jupman.pytut()

Dracula
```

```
[42]: <IPython.core.display.HTML object>
```

If you execute the code in Python Tutor, you will notice that as soon as we assign `ritas_list`, the corresponding list appears to 'detach' from the dictionary. This is only a graphical effect caused by Python Tutor, but from the point of view of the dictionary nothing changed. The intention is to show the list now is *reachable* both from the dictionary and from the new variable `ritas_list`.

Exercise - loans

Write some code to extract and print:

1. The first book borrowed by Gloria ('War and Peace') and the last one borrowed by Rita ('1984')
2. The number of books borrowed by Rita
3. True if everybody among Marco, Gloria and Rita borrowed at least a book, `False` otherwise

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution

><div class="jupman-sol jupman-sol-code" style="display:none">

```
[43]: loans = {'Marco': ['Les Misérables', 'Ulysses'],
              'Gloria': ['War and Peace'],
              'Rita': ['The Shining', 'Dracula', '1984']}

# write here
print("1. The first book borrowed by Gloria is", loans['Gloria'][0])
print("  The last book borrowed by Rita is", loans['Rita'][-1])
print("2. Rita borrowed", len(loans['Rita']), "book(s)")
res = len(loans['Marco']) > 0 and len(loans['Gloria']) > 0 and len(loans['Rita']) > 0
print("3. Have everybody borrowed at least a book?", res)
```

1. The first book borrowed by Gloria is War and Peace
The last book borrowed by Rita is 1984
2. Rita borrowed 3 book(s)
3. Have everybody borrowed at least a book? True

</div>

```
[43]: loans = {'Marco': ['Les Misérables', 'Ulysses'],
              'Gloria': ['War and Peace'],
```

(continues on next page)

(continued from previous page)

```
'Rita': ['The Shining', 'Dracula', '1984']}}

# write here
```

1. The first book borrowed by Gloria is War and Peace
The last book borrowed by Rita is 1984
2. Rita borrowed 3 book(s)
3. Have everybody borrowed at least a book? True

Exercise - Shark Bay

The West India Company asked you to explore the tropical seas, which are known for the dangerous species which live in their waters. You are provided with a dmap which associates places to species found therein:

```
dmap = {
    "Shark Bay" : ["sharks"],
    "Estuary of Bad Luck" : ["crocodiles", "piraña"],
    "Shipwreck Trench" : ["killer whales", "tiger fishes"],
}
```

You are also given vague directions about how to update the dmap, using these variables:

```
place = "Shipwreck Trench"
dangers = ["morays", "blue spotted octopus"]
travel = "Sunken Sails Offshore"
exploration = ["barracudas", "jellyfishes"]
```

Try writing some code which uses the variables above (or data from the map itself) MODIFIES dmap so to obtain:

```
>>> dmap
{'Shark Bay' : ['sharks'],
 'Estuary of Bad Luck' : ['crocodiles', 'piraña', 'jellyfishes'],
 'Shipwreck Trench' : ['killer whales', 'tiger fishes'],
 'Jellyfishes Offshore': ['barracudas', 'jellyfishes', 'crocodiles', 'piraña']}
```

- **IMPORTANT: DO NOT use constant strings in your code** (so no "Shipwreck Trench" ...). Numerical constants are instead allowed.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[44]:

```
place = "Estuary of Bad Luck"
dangers = ["morays", "blue spotted octopus"]
travel = "Sunken Sails Offshore"
exploration = ["barracudas", "jellyfishes"]

dmap = {
    "Shark Bay" : ["sharks"],
    "Estuary of Bad Luck": ["crocodiles", "piraña"],
    "Shipwreck Trench" : ["killer whales", "tiger fishes"],
}
```

(continues on next page)

(continued from previous page)

```
# write here

dmap[travel] = dangers
dmap[exploration[1].capitalize() + travel[-9:]] = exploration + dmap[place]
dmap[place].append(exploration[1])
del dmap[travel]

dmap

[44]: {'Shark Bay': ['sharks'],
 'Estuary of Bad Luck': ['crocodiles', 'piraña', 'jellyfishes'],
 'Shipwreck Trench': ['killer whales', 'tiger fishes'],
 'Jellyfishes Offshore': ['barracudas', 'jellyfishes', 'crocodiles', 'piraña']}
```

</div>

```
[44]: place = "Estuary of Bad Luck"
dangers = ["morays", "blue spotted octopus"]
travel = "Sunken Sails Offshore"
exploration = ["barracudas", "jellyfishes"]

dmap = {
    "Shark Bay" : ["sharks"],
    "Estuary of Bad Luck": ["crocodiles", "piraña"],
    "Shipwreck Trench" : ["killer whales", "tiger fishes"],
}
# write here
```

Exercise - The Storm Sea

The West India Company asks you now to produce a new map starting from `dmap1` and `dmap2`. The new map must contain **all** the items from `dmap1`, expanded with the items from `place1` and `place2`.

- assume the items `place1` and `place2` are always present in `dmap1` and `dmap2`.
- IMPORTANT:** the execution of your code must **not** change `dmap1` nor `dmap2`

Example - given:

```
dmap1 = {
    "Shark Bay" : ["sharks"],
    "Estuary of Bad Luck" : ["crocodiles", "piraña"],
    "Storm Sea" : ["barracudas", "morays"]
}

dmap2 = {
    "Estuary of Bad Luck" : ["morays", "shark fishes"],
    "Storm Sea" : ["giant octupses"],
    "Shipwreck Trench" : ["killer whales"],
    "Lake of the Hopeless" : ["water vortexes"]
}

place1, place2 = "Estuary of Bad Luck", "Storm Sea"
```

After your code, it must result:

```
>>> new
{'Estuary of Bad Luck': ['crocodiles', 'piraña', 'morays', 'shark fishes'],
 'Shark Bay': ['sharks'],
 'Storm Sea': ['barracudas', 'morays', 'giant octupses']}
>>> dmap1 # not changed
{'Estuary of Bad Luck': ['crocodiles', 'piraña'],
 'Shark Bay': ['sharks'],
 'Storm Sea': ['barracudas', 'morays']}
>>> dmap2 # not changed
{'Estuary of Bad Luck': ['morays', 'shark fishes'],
 'Lake of the Hopeless': ['water vortexes'],
 'Shipwreck Trench': ['killer whales'],
 'Storm Sea': ['giant octupses']}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[45]:

```
dmap1 = {
    "Shark Bay" : ["sharks"],
    "Estuary of Bad Luck" : ["crocodiles", "piraña"],
    "Storm Sea" : ["barracudas", "morays"]
}

dmap2 = {
    "Estuary of Bad Luck" : ["morays", "shark fishes"],
    "Storm Sea" : ["giant octupses"],
    "Shipwreck Trench" : ["killer whales"],
    "Lake of the Hopeless" : ["water vortexes"]
}

place1, place2 = "Estuary of Bad Luck", "Storm Sea"

# write here

import copy

new = copy.deepcopy(dmap1)
new[place1].extend(dmap2[place1])
new[place2].extend(dmap2[place2])

from pprint import pprint
print("new:")
pprint(new)
print("dmap1:")
pprint(dmap1)
print("dmap2:")
pprint(dmap2)

new:
{'Estuary of Bad Luck': ['crocodiles', 'piraña', 'morays', 'shark fishes'],
 'Shark Bay': ['sharks'],
 'Storm Sea': ['barracudas', 'morays', 'giant octupses']}
dmap1:
{'Estuary of Bad Luck': ['crocodiles', 'piraña'],
 'Shark Bay': ['sharks'],
```

(continues on next page)

(continued from previous page)

```
'Storm Sea': ['barracudas', 'morays']}
dmap2:
{'Estuary of Bad Luck': ['morays', 'shark fishes'],
 'Lake of the Hopeless': ['water vortexes'],
 'Shipwreck Trench': ['killer whales'],
 'Storm Sea': ['giant octupses']}
```

```
</div>
```

```
[45]: dmap1 = {
    "Shark Bay" : ["sharks"],
    "Estuary of Bad Luck" : ["crocodiles", "piraña"],
    "Storm Sea" : ["barracudas", "morays"]
}

dmap2 = {
    "Estuary of Bad Luck" : ["morays", "shark fishes"],
    "Storm Sea" : ["giant octupses"],
    "Shipwreck Trench" : ["killer whales"],
    "Lake of the Hopeless" : ["water vortexes"]
}

place1, place2 = "Estuary of Bad Luck", "Storm Sea"

# write here
```

Equality

We can verify whether two dictionaries are equal with `==` operator, which given two dictionaries return `True` if they contain key/value couples or `False` otherwise:

```
[46]: {'a':3, 'b':4} == {'a':3, 'b':4}
```

```
[46]: True
```

```
[47]: {'a':3, 'b':4} == {'c':3, 'b':4}
```

```
[47]: False
```

```
[48]: {'a':3, 'b':4} == {'a':3, 'b':999}
```

```
[48]: False
```

We can verify equality of dictionaries with a different number of elements:

```
[49]: {'a':3, 'b':4} == {'a':3}
```

```
[49]: False
```

```
[50]: {'a':3, 'b':4} == {'a':3, 'b':3, 'c':5}
```

```
[50]: False
```

... and with heterogenous elements:

```
[51]: {'a':3, 'b':4} == {2:('q','p'), 'b':[99,77]}
[51]: False
```

Equality and order

From the definition:

- Keys are immutable, **don't have order** and there cannot be duplicates

Since order has no importance, dictionaries created by inserting the same key/value couples in a different order will be considered equal.

For example, let's try direct creation:

```
[52]: {'a':5, 'b':7} == {'b':7, 'a':5}
[52]: True
```

What about incremental update?

```
[53]: diz1 = {}
diz1['a'] = 5
diz1['b'] = 7

diz2 = {}
diz2['b'] = 7
diz2['a'] = 5

print(diz1 == diz2)
True
```

QUESTION: Look at the following code fragments, and for each try guessing which result it produces (or if it gives an error):

1. `{1:2} == {2:1}`
2. `{1:2,3:4} == {3:4,1:2}`
3. `{'a'.upper():3} == {'a':3}`
4. `{'A'.lower():3} == {'a':3}`
5. `{'a': {1:2}} == {3:4}`
6.

```
diz1 = {}
diz1[2] = 5
diz1[3] = 7

diz2 = {}
diz2[3] = 7
diz2[2] = 5
print(diz1 == diz2)
```

```
7. diz1 = {'a':3, 'b':8}
diz2 = diz1
diz1['a'] = 7
print(diz1 == diz2)
```

```
8. diz1 = {}
diz1['a']=3
diz2 = diz1
diz2['a']=4
print(diz1 == diz2)
```

```
9. diz1 = {'a':3, 'b':4, 'c':5}
diz2 = {'a':3, 'c':5}
del diz1['a']
print(diz1 == diz2)
```

```
10. diz1 = {}
diz2 = {'a':3}
diz1['a'] = 3
diz1['b'] = 5
diz2['b'] = 5
print(diz1 == diz2)
```

Equality and copies

When duplicating containers which hold mutable objects, if we do not pay attention we might get surprises. Let's go back on the topic of shallow and deep copies of dictionaries, this time trying to verify the effective equality in Python.

WARNING: Have you read Dictionaries 1 - Copying a dictionary¹⁸⁵ ?

If not, do it now!

QUESTION: Let's see a simple example, with a 'manual' copy. If you execute the following code in Python Tutor, what will it print? How many memory regions will you see?

```
d1 = {'a':3,
      'b':8}
d2 = {'a':d1['a'],
      'b':d1['b']}
d1['a'] = 6

print('equal?', d1 == d2)
print('d1=', d1)
print('d2=', d2)
```

NOTE: all values (3 and 8) are **immutable**.

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: In this case we manually created a dictionary `d2` using *immutable* values taken from `d1`. So in Python Tutor we will see two distinct memory regions and a successive modification to `d1` will not alter `d2`:

¹⁸⁵ <https://en.softpython.org/dictionaries/dictionaries1-sol.html#Copying-a-dictionary>

</div>

```
[54]: d1 = {'a':3,
           'b':8}
d2 = {'a':d1['a'],
       'b':d1['b'] }
d1['a'] = 6

print('equal?', d1 == d2)
print('d1=', d1)
print('d2=', d2)

jupman.pytut()

equal? False
d1= {'a': 6, 'b': 8}
d2= {'a': 3, 'b': 8}

[54]: <IPython.core.display.HTML object>
```

QUESTION: If you execute the following code in Python Tutor, what will it print?

1. Which type of copy did we do? Shallow? Deep? (or both ...?)
2. How many memory regions will you see?

```
d1 = {'a':3,
      'b':8}
d2 = dict(d1)
d1['a'] = 7

print('equal?', d1 == d2)
print('d1=', d1)
print('d2=', d2)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: when used as a function, `dict` executes a *shallow* copy, that is, copies the structure of the dictionary without duplicating the mutable values. In this specific case, all values we have are immutable integers, so the copy can also be considered a complete duplication. When we assign the value `7` to the key '`a`' in `d1` we are modifying the original data structure , leaving the copy we just made `d2` unaltered, so `d1 == d2` will be `False`.

Let's verify it in Python Tutor:

</div>

```
[55]: d1 = {'a':3,
           'b':8}
d2 = dict(d1)
d1['a'] = 7

print('equal?', d1 == d2)
print('d1=', d1)
print('d2=', d2)

jupman.pytut()

equal? False
d1= {'a': 7, 'b': 8}
d2= {'a': 3, 'b': 8}
```

```
[55]: <IPython.core.display.HTML object>
```

QUESTION: If you execute the following code in Python Tutor, what will it print?

1. Which type of copy did we do? Shallow? Deep? (or both ...?)
2. How many memory regions will you see?

NOTE: the values are lists, thus they are **mutable**

```
d1 = {'a': [1, 2],  
      'b': [4, 5, 6]}  
d2 = dict(d1)  
d1['a'].append(3)  
  
print('equal?', d1 == d2)  
print('d1=', d1)  
print('d2=', d2)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: We used `dict` like a function, so we did a *shallow copy*. In this case we have lists as values, which are *mutable* objects. This means the shallow copy only copied references to the lists, but *not* the lists themselves. For this reason you will see arrows going from the copy of the dictionary `d2` to memory regions of the original lists. This means that if you try to modify a list after the copy occurred (for example with the method `.append(3)`), as a matter of fact you will also modify the list reachable from the copied dictionary `d2`. Let's check this out in Python Tutor:

</div>

```
[56]: d1 = {'a': [1, 2],  
           'b': [4, 5, 6]}  
d2 = dict(d1)  
d1['a'].append(3)  
  
print('equal?', d1 == d2)  
print('d1=', d1)  
print('d2=', d2)  
  
jupman.pytut()  
  
equal? True  
d1= {'a': [1, 2, 3], 'b': [4, 5, 6]}  
d2= {'a': [1, 2, 3], 'b': [4, 5, 6]}
```

```
[56]: <IPython.core.display.HTML object>
```

QUESTION: If you execute the following code in Python Tutor, what will it print?

1. Which type of copy did we do? Shallow? Deep? (or both ...?)
2. How many memory regions will you see?

NOTE: the values are lists, so they are **mutable**

```
import copy  
d1 = {'a': [1, 2],  
      'b': [4, 5, 6]}  
d2 = copy.deepcopy(d1)  
d1['a'].append(3)
```

(continues on next page)

(continued from previous page)

```
print('equal?', d1 == d2)
print('d1=', d1)
print('d2=', d2)
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: We used `copy.deepcopy`, making an in-depth copy. In this case we have mutable lists as values. The deep copy duplicated all the objects it was able to reach, lists included. So in this case we will obtain two completely distinct memory regions. After the copy, if we modify a list reachable from the original `d1`, we will be sure that we cannot tarnish objects reachable from `d2`. Let's check it in Python Tutor:

</div>

```
[57]: import copy
d1 = {'a': [1, 2],
      'b': [4, 5, 6]}
d2 = copy.deepcopy(d1)
d1['a'].append(3)

print('equal?', d1 == d2)
print('d1=', d1)
print('d2=', d2)

jupman.pytut()

equal? False
d1= {'a': [1, 2, 3], 'b': [4, 5, 6]}
d2= {'a': [1, 2], 'b': [4, 5, 6]}
```

[57]: <IPython.core.display.HTML object>

QUESTION: Look at the following code fragments, and for each try guessing which result it produces (or if it gives an error):

1.

```
diz1 = {'a':[4,5],
        'b':[6,7]}
diz2 = dict(diz1)
diz2['a'] = diz1['b']
diz2['b'][0] = 9
print(diz1 == diz2)
print(diz1)
print(diz2)
```

2.

```
da = {'a':['x','y','z']}
db = dict(da)
db['a'] = ['w','t']
dc = dict(db)
print(da)
print(db)
print(dc)
```

3.

```
import copy

la = ['x','y','z']
diz1 = {'a':la,
```

(continues on next page)

(continued from previous page)

```
'b':la }
diz2 = copy.deepcopy(diz1)
diz2['a'][0] = 'w'
print('uguali?', diz1 == diz2)
print('diz1=', diz1)
print('diz2=', diz2)
```

Exercise - Zoom Doom

Write some code which given a string s (i.e. 'ZOOM'), creates a dictionary zd and assigns to keys 'a', 'b' and 'c' the *same identical list* containing the string characters as elements (i.e. ['Z', 'O', 'O', 'M']).

- in Python Tutor you should see 3 arrows which go from keys to *the same identical memory region*
- by modifying the list associated to each key, you should see the modification also in the lists associated to other keys
- your code must work for *any* string s

Example - given:

```
s = 'ZOOM'
```

After your code, it should result:

```
>>> print(zd)
{'a': ['Z', 'O', 'O', 'M'],
 'b': ['Z', 'O', 'O', 'M'],
 'c': ['Z', 'O', 'O', 'M'],
}
>>> zd['a'][0] = 'D'
>>> print(zd)
{'a': ['D', 'O', 'O', 'M'],
 'b': ['D', 'O', 'O', 'M'],
 'c': ['D', 'O', 'O', 'M'],
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"< data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[58]:

```
s = 'ZOOM'

# write here

zoom = list(s)

zd = {'a':zoom,
      'b':zoom,
      'c':zoom}
print(zd)
zd['a'][0] = 'D'
print(zd)

#jupman.pytut()
```

```
{'a': ['Z', 'O', 'O', 'M'], 'b': ['Z', 'O', 'O', 'M'], 'c': ['Z', 'O', 'O', 'M']}
{'a': ['D', 'O', 'O', 'M'], 'b': ['D', 'O', 'O', 'M'], 'c': ['D', 'O', 'O', 'M']}
```

</div>

[58]:

```
s = 'ZOOM'

# write here
```

Continue

Go on reading Dictionaries 3 - methods¹⁸⁶

[]:

5.6.3 Dictionaries 3 - Methods

Download exercise zip

Browse online files¹⁸⁷

In this notebook we will see the main methods to extract data and manipulate dictionaries.

Methods:

Method	Return	Description
<code>dict.keys()</code>	<code>dict_keys</code>	Return a <i>view</i> of keys which are present in the dictionary
<code>dict.values()</code>	<code>dict_values</code>	Return a <i>view</i> of values which are present in the dictionary
<code>dict.items()</code>	<code>dict_items</code>	Return a <i>view</i> of (key/value) couples present in the dictionary
<code>dict1.update(dict2)</code>	None	MODIFY the dictionary <code>dict1</code> with the key / value couples found in <code>dict2</code>

What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
sets
dictionaries1.ipynb
dictionaries1-sol.ipynb
dictionaries2.ipynb
dictionaries2-sol.ipynb
dictionaries3.ipynb
dictionaries3-sol.ipynb
dictionaries4.ipynb
dictionaries4-sol.ipynb
```

(continues on next page)

¹⁸⁶ <https://en.softpython.org/dictionaries/dictionaries3-sol.html>

¹⁸⁷ <https://github.com/DavidLeoni/softpython-en/tree/master/dictionaries>

(continued from previous page)

```
dictionaries5-chal.ipynb  
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `dictionaries3.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press **Control + Enter**
- to execute Python code inside a Jupyter cell AND select next cell, press **Shift + Enter**
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press **Alt + Enter**
- If the notebooks look stuck, try to select **Kernel -> Restart**

keys method

By calling the method `.keys()` we can obtain the dictionary keys:

```
[2]: vegetables = {'carrots' : 5,  
                  'tomatoes' : 8,  
                  'cabbage' : 3}
```

```
[3]: vegetables.keys()  
[3]: dict_keys(['carrots', 'tomatoes', 'cabbage'])
```

WARNING: THE RETURNED SEQUENCE IS OF TYPE `dict_keys`

`dict_keys` might look like a list but it is well different!

In particular, the returned sequence `dict_keys` is a **view** on the original dictionary. In computer science, when we talk about *views* we typically intend collections which contain a part of the objects contained in another collection, *and if the original collection gets modified, so is the view at the same time*.

Let's see what this means. First let's assign the sequence of keys to a variable:

```
[4]: ks = vegetables.keys()
```

Then we modify the original dictionary, adding an association:

```
[5]: vegetables['potatoes'] = 8
```

If we now print `ks`, we should see the change:

```
[6]: ks  
[6]: dict_keys(['carrots', 'tomatoes', 'cabbage', 'potatoes'])
```

Sequence returned by `.keys()` can change over time!

When reusing the sequence from `.keys()`, ask yourself if the dictionary could have changed in the meanwhile

If we want a stable version as a sort of static ‘picture’ of dictionary keys at a given moment in time, we must explicitly convert them to another sequence, like for example a list:

```
[7]: as_list = list(vegetables.keys())
[8]: as_list
[8]: ['carrots', 'tomatoes', 'cabbage', 'potatoes']
[9]: vegetables['cocumber'] = 9
[10]: as_list      # no cocumbers
[10]: ['carrots', 'tomatoes', 'cabbage', 'potatoes']
```

Let’s see again the example in Python Tutor:

```
[11]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
#           (it's sufficient to execute it only once)

import jupman

[12]: vegetables = {'carrots' : 5,
                  'tomatoes' : 8,
                  'cabbage' : 3}
keys = vegetables.keys()
vegetables['potatoes'] = 8
as_list = list(vegetables.keys())
vegetables['cocumbers'] = 9
#print(as_list)

jupman.pytut()

[12]: <IPython.core.display.HTML object>
```

WARNING: WE CAN'T USE INDEXES WITH `dict_keys`

If we try, we will obtain an error:

```
>>> vegetables = {'carrots' : 5,
                  'tomatoes' : 8,
                  'cabbage' : 3}
>>> ks = vegetables.keys()
>>> ks[0]

-----
TypeError                                     Traceback (most recent call last)
<ipython-input-90-c888bf602918> in <module>()
----> 1 ks[0]

TypeError: 'dict_keys' object does not support indexing
```

WARNING: WE CANNOT DIRECTLY MODIFY dict_keys

There aren't operations nor methods which allow us to change the elements of `dict_keys`, you can only act on the original dictionary.

QUESTION: Look at the following code fragments, and for each try guessing if it can work (or if it gives an error):

```
1. diz = {'a':4,  
         'b':5}
```

```
    ks = diz.keys()  
    ks.append('c')
```

```
2. diz = {'a':4,  
         'b':5}
```

```
    ks = diz.keys()  
    ks.add('c')
```

```
3. diz = {'a':4,  
         'b':5}
```

```
    ks = diz.keys()  
    ks['c'] = 3
```

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: None of the examples above can work, because we can't directly modify objects of type `dict_keys`. Operators like square brackets or methods like `.append`, `.add`, etc are not supported.

</div>

QUESTION: Look at the following code fragments, and for each try guessing which result it produces (or if it gives an error):

```
1. diz = {'a':1,'b':2}  
s = set(diz.keys())  
s.add('c',3))  
print(diz)  
print(s)
```

```
2. diz = {'a':3,'b':4}  
k = diz.keys()  
diz['c'] = 5  
print(len(k))
```

```
3. diz = {'a':'x',  
         'b':'y'}  
print('a' in diz.keys())
```

```
4. diz1 = {'a':1,'b':2}  
chiavi = diz1.keys()  
diz2 = dict(diz1)  
diz2['c'] = 3
```

(continues on next page)

(continued from previous page)

```
print('diz1=',diz1)
print('diz2=',diz2)
print('chiavi=',chiavi)
```

5. `diz1 = {'a':'b', 'c':'d'}
diz2 = {'a':'b', 'b':'c'}
print(set(diz1.keys()) - set(diz2.keys()))`

6. `diz1 = {'a':'b', 'c':'d'}
diz2 = {'e':'a', 'f':'c'}
ks = diz1.keys()
del diz1[diz2['e']]
del diz1[diz2['f']]
print(len(ks))`

Exercise - messy keys

⊕ PRINT a LIST with all the keys in the dictionary

- **NOTE 1:** it is NOT necessary for the list to be sorted
- **NOTE 2:** to convert any sequence to a list, use the predefined function `list`

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[13]: d = {'c':6, 'b':2, 'a':5}
# write here
list(d.keys())
[13]: ['c', 'b', 'a']
```

</div>

```
[13]: d = {'c':6, 'b':2, 'a':5}
# write here
```

Exercise - sorted keys

⊕ PRINT a LIST with all the dictionary keys

- **NOTE 1:** Now it IS necessary for the list to be sorted
- **NOTE 2:** to convert any sequence to a list, use the predefined function `list`

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[14]:

```
d = {'c':6, 'b':2, 'a':5}

# write here

my_list = list(d.keys())
my_list.sort()
print(my_list)

['a', 'b', 'c']
```

```
</div>
```

[14]:

```
d = {'c':6, 'b':2, 'a':5}

# write here
```

Exercise - keyring

Given the dictionaries d1 and d2, write some code which puts into a **list** ks all the keys in the two dictionaries, **without duplicates** and **alphabetically sorted**, and finally prints the list.

- your code must work with any d1 and d2

Example - given:

```
d1 = {
    'a':5,
    'b':9,
    'e':2,
}
d2 = {'a':9,
      'c':2,
      'e':2,
      'f':6}
```

after your code, it must result:

```
>>> print(keys)
['a', 'b', 'c', 'e', 'f']
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[15]:

```
d1 = {
    'a':5,
    'b':9,
    'e':2,
}
d2 = {'a':9,
      'c':2,
      'e':2,
```

(continues on next page)

(continued from previous page)

```
'f':6}

# write here
ks = list(set(d1.keys()) | set(d2.keys()))
ks.sort()
print(ks)

['a', 'b', 'c', 'e', 'f']
```

</div>

[15]:

```
d1 = {
    'a':5,
    'b':9,
    'e':2,
}
d2 = {'a':9,
      'c':2,
      'e':2,
      'f':6}

# write here
```

values method

Given a dictionary, we can obtain all the values by calling the method `.values()`

Imagine we have a dictionary `vehicles` which assigns an owner to each car plate:

[16]:

```
vehicles = {
    'AA111AA' : 'Mario',
    'BB222BB' : 'Lidia',
    'CC333CC' : 'Mario',
    'DD444DD' : 'Gino',
    'EE555EE' : 'Gino'
}

owners = vehicles.values()
```

WARNING: THE RETURNED SEQUENCE IS OF TYPE `dict_values`

`dict_values` may seem a list but it's not!

We've seen `dict_keys` is a view on the original dictionary, and so is `dict_values`, thus by adding an association to `vehicles` ...

[17]:

```
vehicles['FF666FF'] = 'Paola'
```

... the view `owners` will automatically result changed:

```
[18]: owners
[18]: dict_values(['Mario', 'Lidia', 'Mario', 'Gino', 'Gino', 'Paola'])
```

We also note that being *values* of a dictionary, duplicates are allowed.

WARNING: WE CANNOT USE INDEXES WITH `dict_values`

If we try, we will get an error:

```
>>> owners[0]
-----
TypeError                                     Traceback (most recent call last)
<ipython-input-90-c888bf602918> in <module>()
----> 1 owners[0]

TypeError: 'dict_values' object does not support indexing
```

WARNING: WE CANNOT DIRECTLY MODIFY `dict_values`

There aren't operations nor methods that allow us to change the elements of `dict_values`, we can only act on the original dictionary.

QUESTION: Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

```
1. diz = {'a':4,
          'b':5}

vals = diz.values()
vals.append(4)
```

```
2. d = {0:'a',
        1:'b',
        2:'b'}
vals = d.values()
d[2]='c'
print(vals)
```

```
3. diz = {'a':4,
          'b':5}

vals = diz.values()
vals.add(5)
```

```
4. diz = {0:1,
          1:2,
          2:3}

diz[list(diz.values())[0]-1]
```

```
5. diz = {'a':4,
          'b':5}

        vals = diz.values()
        vals['c'] = 6
```

```
6. diz = {'a':4,
          'b':5}

        vals = diz.values()
        vals[6] = 'c'
```

Exercise - one by one

Given a dictionary `my_dict`, write some code which prints `True` if each key is associated to a value *different* from the values of all other keys. Otherwise prints `False`.

Example 1 - given:

```
my_dict = {'a' : 3,
           'c' : 6,
           'g' : 8}
```

After your code, it must print `True` (because 3,6 and 8 are all different)

`True`

Example 2 - given:

```
my_dict = {'x' : 5,
           'y' : 7,
           'z' : 5}
```

it must print:

`False`

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[19]:

```
my_dict = {'a' : 3,
           'c' : 6,
           'g' : 8}

"""

my_dict= {'x' : 5,
           'y' : 7,
           'z' : 5}

"""

# write here

print(len(my_dict.keys()) == len(set(my_dict.values())))
```

```
True
```

```
</div>
```

```
[19]:
```

```
my_dict = {'a' : 3,
           'c' : 6,
           'g' : 8}

"""
my_dict= {'x' : 5,
          'y' : 7,
          'z' : 5}
"""

# write here
```

Exercise - bag

Given a dictionary `my_dict` of character associations, write some code which puts into the variable `bag` the sorted list of all the keys and values.

Example - given:

```
my_dict = {
    'a':'b',
    'b':'f',
    'c':'b',
    'd':'e'
}
```

After your code, it must print:

```
>>> print(bag)
['a', 'b', 'c', 'd', 'e', 'f']
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[20]:
```

```
my_dict = {
    'a':'b',
    'b':'f',
    'c':'b',
    'd':'e'
}

# write here

bag = list(set(my_dict.keys()) | set(my_dict.values()))
bag.sort()

print(bag)
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```

</div>

[20]:

```
my_dict = {
    'a':'b',
    'b':'f',
    'c':'b',
    'd':'e'
}

# write here
```

Exercise - common values

Given two dictionaries d1 and d2, write some code which PRINTS True if they have *at least* a value in common (without considering the keys)

Example 1 - given:

```
d1 = {
    'a':4,
    'k':2,
    'm':5
}

d2 = {
    'b':2,
    'e':4,
    'g':9,
    'h':1
}
```

after your code, it must print True (because they have the values 2 and 4 in common):

```
Common values? True
```

Example 2 - given:

```
d1 = {
    'd':1,
    'e':2,
    'f':6
}

d2 = {
    'a':3,
    'b':5,
    'c':9,
    'd':7
}
```

after your code, it must print:

Common values? False

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[21]:

```
d1 = {  
    'a': 4,  
    'k': 2,  
    'm': 5  
}  
  
d2 = {  
    'b': 2,  
    'e': 4,  
    'g': 9,  
    'h': 1  
}  
  
"""  
d1 = {  
    'd': 1,  
    'e': 2,  
    'f': 6  
}  
  
d2 = {  
    'a': 3,  
    'b': 5,  
    'c': 9,  
    'd': 7  
}  
"""  
  
# write here  
  
print('Common values?', len(set(d1.values()) & set(d2.values())) > 0)
```

Common values? True

</div>

[21]:

```
d1 = {  
    'a': 4,  
    'k': 2,  
    'm': 5  
}  
  
d2 = {  
    'b': 2,  
    'e': 4,  
    'g': 9,  
    'h': 1  
}  
  
"""  
d1 = {
```

(continues on next page)

(continued from previous page)

```

'd':1,
'e':2,
'f':6
}

d2 = {
    'a':3,
    'b':5,
    'c':9,
    'd':7
}
"""

# write here

```

Exercise - small big

Given a dictionary d which has integers as keys and values, print True if the smaller key is equal to the greatest value.

Example 1 - given:

```

d = {
    14:1,
    11:7,
    7:3,
    70:5
}

```

after your code, it must print True (because the smallest key is 7 which is equal to the greatest value 7):

True

Example 2 - given:

```

d = {
    12:1,
    11:9,
    7:3,
    2:5,
    9:1
}

```

after your code, it must print False (because the smallest key 2 is different from the greatest value 9):

False

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[22]:

```

d = {
    14:1,
    11:7,
}

```

(continues on next page)

(continued from previous page)

```
    7:3,
    70:5
}

"""
d = {
    12:1,
    11:9,
    7:3,
    2:5,
    9:1
}
"""

# write here

min(d.keys()) == max(d.values())
[22]: True
```

</div>

```
[22]: 
d = {
    14:1,
    11:7,
    7:3,
    70:5
}

"""
d = {
    12:1,
    11:9,
    7:3,
    2:5,
    9:1
}
"""

# write here
```

items method

We can extract all the key/value associations as a list of couples of type tuple with the method `.items()`. Let's see an example which associates attractions to the city they are in:

```
[23]: holiday = {'Piazza S.Marco' : 'Venezia',
                 'Fontana di Trevi': 'Roma',
                 'Uffizi'           : 'Firenze',
                 'Colosseo'         : 'Roma',
                 }
```

```
[24]: holiday.items()
[24]: dict_items([('Piazza S.Marco', 'Venezia'), ('Fontana di Trevi', 'Roma'), ('Uffizi',
    ↪'Firenze'), ('Colosseo', 'Roma'))]
```

In this case we see that an object of type `dict_items` is returned. As in previous cases, it is a **view** which we **can't** directly modify. If the original dictionary gets changed, the mutation will be reflected in the view:

```
[25]: attractions = holiday.items()
```

```
[26]: holiday['Palazzo Ducale'] = 'Venezia'
```

```
[27]: attractions
```

```
[27]: dict_items([('Piazza S.Marco', 'Venezia'), ('Fontana di Trevi', 'Roma'), ('Uffizi',
    ↪'Firenze'), ('Colosseo', 'Roma'), ('Palazzo Ducale', 'Venezia'))])
```

QUESTION: Look at the following code fragments, and for each try guessing which result it produces (or if it gives an error):

1. `{'a':7, 'b':9}.items()[0] = ('c',8)`

2. `dict({'a':7, 'b':5}).items()['a']`

3. `len(set({'a':'b', 'a':'B'}).items()))`

4. `{'a':2}.items().find(('a',2))`

5. `{'a':2}.items().index(('a',2))`

6. `list({'a':2}.items()).index(('a',2))`

7. `diz1 = {'a':7,
 'b':5}
diz2 = dict(diz1.items())
diz1['a'] = 6
print(diz1 == diz2)`

8. `('a', 'b') in {'a':('a','b'), 'b':('a','b')}.items()`

9. `('a', 'b') in list({'a':('a','b'), 'b':('a','b')}.items())[0]`

Exercise - union without update

Given the dictionaries `d1` and `d2`, write some code which creates a NEW dictionary `d3` containing all the key/value couples from `d1` and `d2`.

- we suppose all the key/value couples are distinct
- **DO NOT** use cycles
- **DO NOT** use `.update()`
- your code must work for *any* `d1` and `d2`

Example - given:

```
d1 = {'a':4,
      'b':7}
d2 = {'c':5,
      'd':8,
      'e':2}
```

after your code, it must result (order is not important):

```
>>> print(d3)
{'a': 4, 'e': 2, 'd': 8, 'c': 5, 'b': 7}
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[28]:

```
d1 = {'a':4,
      'b':7}
d2 = {'c':5,
      'd':8,
      'e':2}

# write here
diz3 = dict(list(d1.items()) + list(d2.items()))
#print(d3)
```

</div>

[28]:

```
d1 = {'a':4,
      'b':7}
d2 = {'c':5,
      'd':8,
      'e':2}

# write here
```

update method

Having a dictionary to start with, it is possible to MODIFY it by joining another with the method `.update()`:

[29]:

```
d1 = {'goats' : 6,
      'cabbage' : 9,
      'shepherds':1}

d2 = {'goats' :12,
      'cabbage':15,
      'benches':3,
      'hay' : 7}
```

[30]:

```
d1.update(d2)
```

[31]: d1

```
{'goats': 12, 'cabbage': 15, 'shepherds': 1, 'benches': 3, 'hay': 7}
```

Note how the common keys among the two dictionaries like 'goats' and 'cabbage' have values from the second.

If we will, it's also possible to pass a sequence of couples like this:

[32]: d1.update([('hay', 3), ('benches', 18), ('barns', 4)])

[33]: d1

```
{'goats': 12,
 'cabbage': 15,
 'shepherds': 1,
 'benches': 18,
 'hay': 3,
 'barns': 4}
```

Exercise - axby

Given a dictionary `dcc` which associates characters to characters and a string `s` formatted with couples of characters like `ax` separated by a semi-colon ;, substitute all the values in `dcc` with the corresponding values denoted in the string.

- your code must work for *any* dictionary `my_dict` and lists

Example - given:

```
dcc = {
    'a': 'x',
    'b': 'y',
    'c': 'z',
    'd': 'w'
}
s = 'bx;cw;ex'
```

after your code, it must result:

```
>>> dcc
{'a': 'x', 'b': 'y', 'c': 'z', 'd': 'w', 'e': 'x'}
```

[Show solution](#)

```
<a class="jupman-sol" href="#" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol" style="display:none">
```

[34]:

```
dcc = {
    'a': 'x',
    'b': 'y',
    'c': 'z',
    'd': 'w'
}
s = 'bx;cw;ex'

# write here

la = s.split(';')
```

(continues on next page)

(continued from previous page)

```
dcc.update(la)
dcc
[34]: {'a': 'x', 'b': 'x', 'c': 'w', 'd': 'w', 'e': 'x'}
```

</div>

```
[34]: 
dcc = {
    'a':'x',
    'b':'y',
    'c':'z',
    'd':'w'
}
s = 'bx;cw;ex'

# write here
```

Continue

Go on with Dictionaries 4¹⁸⁸

5.6.4 Dictionaries 4 - special classes

Download exercise zip

Browse online files¹⁸⁹

There are special classes we can use:

Class	Description
<i>OrderedDict</i>	Dictionary which allows to maintain the order of insertion of keys
<i>Counter</i>	Dictionary which allows to rapidly calculate histograms

What to do

1. Unzip exercises.zip in a folder, you should obtain something like this:

```
sets
dictionaries1.ipynb
dictionaries1-sol.ipynb
dictionaries2.ipynb
dictionaries2-sol.ipynb
dictionaries3.ipynb
dictionaries3-sol.ipynb
dictionaries4.ipynb
dictionaries4-sol.ipynb
```

(continues on next page)

¹⁸⁸ <https://en.softpython.org/dictionaries/dictionaries4-sol.html>

¹⁸⁹ <https://github.com/DavidLeoni/softpython-en/tree/master/dictionaries>

(continued from previous page)

```
dictionaries5-chal.ipynb
jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `dictionaries4.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press **Control + Enter**
- to execute Python code inside a Jupyter cell AND select next cell, press **Shift + Enter**
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press **Alt + Enter**
- If the notebooks look stuck, try to select **Kernel -> Restart**

OrderedDict

As we said before, when we print a dictionary with `print` or we leave the visualization to Jupyter, most of the times couples are not in insertion order. For the order to be predictable, you must use an `OrderedDict`

First you need to import it from the `collections` module:

```
[2]: from collections import OrderedDict
```

```
[3]: od = OrderedDict()
```

An `OrderedDict` appears and behaves like regular dictionaries:

```
[4]: od['some key'] = 5
od['some other key'] = 7
od[('an', 'immutable', 'tuple', 'as key')] = 3
od['Another key'] = 'now a string!'
od[123] = 'hello'
```

When visualizing with Jupyter, we see the insertion order:

```
[5]: od
[5]: OrderedDict([('some key', 5),
                 ('some other key', 7),
                 (('an', 'immutable', 'tuple', 'as key'), 3),
                 ('Another key', 'now a string!'),
                 (123, 'hello'))]
```

As we see it with a regular `print`:

```
[6]: print(od)
OrderedDict([('some key', 5), ('some other key', 7), (('an', 'immutable', 'tuple',
    ↴'as key'), 3), ('Another key', 'now a string!'), (123, 'hello'))]
```

Let's see how it appears in Python Tutor:

```
[7]: from collections import OrderedDict
od = OrderedDict()
od['some key'] = 5
od['some other key'] = 7
od[('an', 'immutable', 'tuple', 'as key')] = 3
od['Another key'] = 'now a string!'
od[123] = 'hello'

jupman.pytut()

[7]: <IPython.core.display.HTML object>
```

Exercise - phonebook

Write some code which given three tuples with names and phone numbers, PRINTS an `OrderedDict` which associates names to phone numbers, in the order in which are proposed

- Your code must work with *any* tuple
- Do not forget to import `OrderedDict` from `collections`

Example:

```
t1 = ('Alice', '143242903')
t2 = ('Bob', '417483437')
t3 = ('Charles', '423413213')
```

after your code, it should result:

```
OrderedDict([('Alice', '143242903'), ('Bob', '417483437'), ('Charles', '423413213')])
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[8]: t1 = ('Alice', '143242903')
t2 = ('Bob', '417483437')
t3 = ('Charles', '423413213')

# write here

# first we need to import some collection
from collections import OrderedDict

od = OrderedDict([t1, t2, t3])
print(od)

OrderedDict([('Alice', '143242903'), ('Bob', '417483437'), ('Charles', '423413213')])
```

</div>

```
[8]: t1 = ('Alice', '143242903')
t2 = ('Bob', '417483437')
t3 = ('Charles', '423413213')
```

(continues on next page)

(continued from previous page)

```
# write here
```

Exercise - OrderedDict copy

Given an OrderedDict od1 containing English to Italian translations, create a NEW OrderedDict called od2 which contains the same translations as input PLUS the translation 'water' : 'acqua'

- NOTE 1: your code should work with any ordered dict as input
- NOTE 2: od2 MUST be associated to a NEW OrderedDict !!

Example - given:

```
od1 = OrderedDict()
od1['dog'] = 'cane'
od1['home'] = 'casa'
od1['table'] = 'tavolo'
```

after your code, you should obtain:

```
>>> print(od1)
OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo')])
>>> print(od2)
OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo'), ('water', 'acqua
↪')])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[9]:

```
from collections import OrderedDict

od1 = OrderedDict()
od1['dog'] = 'cane'
od1['home'] = 'casa'
od1['table'] = 'tavolo'

# write here
od2 = OrderedDict(od1)
od2['water'] = 'acqua'

print("od1=", od1)
print("od2=", od2)

od1= OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo')])
od2= OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo'), ('water',
↪'acqua')))
```

</div>

[9]:

```
from collections import OrderedDict

od1 = OrderedDict()
```

(continues on next page)

(continued from previous page)

```
od1['dog'] = 'cane'  
od1['home'] = 'casa'  
od1['table'] = 'tavolo'  
  
# write here
```

Counter

If we need to know how many different elements there are in a sequence (in other words, if we need to calculate a frequency histogram), the class `Counter` from `collections` module comes useful. `Counter` is a special type of dictionary, and first of all, we must declare to Python our intention to use it:

```
[10]: from collections import Counter
```

Suppose we want to count how many different characters there are in this list:

```
[11]: my_seq = ['t', 'e', 'm', 'p', 'e', 'r', 'a', 'm', 'e', 'n', 't']
```

We can initialize `Counter` like this:

```
[12]: histogram = Counter(my_seq)
```

If we print it, we see that the first elements are the most frequent:

```
[13]: print(histogram)  
Counter({'e': 3, 't': 2, 'm': 2, 'p': 1, 'r': 1, 'a': 1, 'n': 1})
```

WARNING: IF WE DON'T USE `print` JUPYTER WILL PRINT IN ALPHABETICAL ORDER!

```
[14]: histogram # careful !  
[14]: Counter({'t': 2, 'e': 3, 'm': 2, 'p': 1, 'r': 1, 'a': 1, 'n': 1})
```

We can obtain a list with the `n` most frequent items by using the method `most_common`, which returns a list of tuples:

```
[15]: histogram.most_common(5)  
[15]: [('e', 3), ('t', 2), ('m', 2), ('p', 1), ('r', 1)]
```

`Counter` can be initialized with any sequence, for example with tuples:

```
[16]: ct = Counter((50, 70, 40, 60, 40, 50, 40, 70, 50, 50, 50, 60, 50, 30, 50, 30, 40, 50, 60, 70))  
print(ct)  
  
Counter({50: 8, 40: 4, 70: 3, 60: 3, 30: 2})
```

or strings:

```
[17]: cs = Counter('condonation')
```

```
[18]: print(cs)
Counter({'o': 3, 'n': 3, 'c': 1, 'd': 1, 'a': 1, 't': 1, 'i': 1})
```

For other methods we refer to [Python documentation](#)¹⁹⁰

Exercise - saddened

Given a string `s`, write some code which prints:

- the most frequent character
- the least frequent character
- how many and which different frequencies there are
- Your code must work with *any* string `s`
- Ignore the possibility there could be ties among the most/least frequent items
- remember to import `Counter` from `collections`

Example - given:

```
s = 'saddened'
```

your code must print:

```
Among the most frequent ones we find ('d', 3)
Among the least frequent ones we find ('a', 1)
There are 3 different frequencies: {1, 2, 3}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[19]: s = 'saddened'

# write here
from collections import Counter

c = Counter(s)

print("Among the most frequent ones we find", c.most_common()[0])
print("Among the least frequent ones we find", c.most_common()[-1])
print("There are", len(set(c.values())), "different frequencies:", set(c.values()))
```

```
Among the most frequent ones we find ('d', 3)
Among the least frequent ones we find ('n', 1)
There are 3 different frequencies: {1, 2, 3}
```

</div>

```
[19]: s = 'saddened'

# write here
```

(continues on next page)

¹⁹⁰ <https://docs.python.org/3/library/collections.html#collections.Counter>

(continued from previous page)

Continue

Go on with first challenges¹⁹¹

[]:

5.6.5 Dictionaries 5 - First challenges

Download exercises zip

Browse file online¹⁹²

We now propose some exercises without solution, do you accept the challenge?

Challenge - The Institute for Advanced Technotronics

⊕ The Institute for Advanced Technotronics builds computing machines which require thousands of cables. They are categorized according to their section diameter, in intervals of two numbers. Each interval is associated with a color. The Institute records everything in a journal, but now it's refurbishing the inventory and wants to build a better database. Write some code that given a list `diameters` of **exactly 6 numbers** and another list `colors` of **exactly 3 strings**, like

```
diameters = [5, 9, 13, 18, 27, 90]
colors = ['yellow', 'orange', 'red']
```

outputs a dictionary in this format:

```
{(5, 9) : 'yellow',
(13, 18): 'orange',
(27, 90): 'red'}
```

DO NOT write constant list elements in your code (i.e. no 27)

[1]:

```
diameters,colors = [5, 9, 13, 18, 27, 90], ['yellow', 'orange', 'red']

# write here
```

¹⁹¹ <https://en.softpython.org/dictionaries/dictionaries5-chal.html>

¹⁹² <https://github.com/DavidLeoni/softpython-en/dictionaries>

Challenge - Incredible Machines contest

⊕ Each week the Institute promotes a contest. The participants in turns must inspect a new incredible machine and understand how it works. During each competition, the `winner` gains `prize` points which are then recorded in a registry, adding them to points gained in previous competitions.

The machines are shiny new, and the Institute is aware the first participant may experience some issue with machines being run for the first time. To compensate for this, the `first` participant is always granted an `extra` amount of points.

- participants interact with the machine in alphabetical order

Write some code to MODIFY the registry with the winner prize, and extra amount of points for the first contestant. Print also the process.

- **DO NOT** use loops
- **DO NOT** write constant participants names (so no '`Carl`'...)
- Display participant order nicely

Example - given:

```
extra, winner, prize = 30, 'Marianne', 200

registry = { 'Lisa' : 10,
             'Robert' : 30,
             'Marianne': 20,
             'Carl' : 20,
             'Sara' : 60,
             'Suzanne' : 30}
```

your code should print

```
Participants order: Carl, Lisa, Marianne, Robert, Sara, Suzanne
Carl begins, receives extra grant of 30 points
Marianne won, receives 200 points

Updated registry is
{'Carl': 50,
 'Lisa': 10,
 'Marianne': 220,
 'Robert': 30,
 'Sara': 60,
 'Suzanne': 30}
```

HINT: to print nicely, use `pprint`:

```
from pprint import pprint
pprint(registry)
```

NOTE: `pprint` (and also jupyter) display keys in alphabetical order, but that's just aesthetic: the original order in memory remains unchanged.

[2]:

```
extra, winner, prize = 30, 'Marianne', 200
registry = { 'Lisa' : 10,
             'Robert' : 30,
             'Marianne': 20,
             'Carl' : 20,
```

(continues on next page)

(continued from previous page)

```
'Sara' : 60,  
'Suzanne' : 30}  
  
#extra, winner, prize, registry = 10, 'Sebastian', 300, {'Sebastian':40,'Alfio':20}  
  
# write here
```

Challenge - Going nuts

⊕ The Institute has a large storage room where component parts are kept. Sometimes mechanics come in a hurry yelling they need a given amount of some item.

Write some code which prints `True` if there are enough items in the storage, and `False` otherwise.

NOTE: a mechanic may ask for an item which is not recorded at all in the storage

- **DO NOT** write string constants in your code (so no '`knobs`' ...)
- **DO NOT** use loops
- **DO NOT** use `if` statements
- **HINT** if you don't know how to do it, have a look at boolean evaluation order¹⁹³

[3]:

```
item, amount = 'knobs', 15    # True  
#item, amount = 'knobs', 16    # False  
#item, amount = 'nuts', 9     # True  
#item, amount = 'rivets', 8    # False  
#item, amount = 'clamps', 1    # False  
#item, amount = 'pins', 1      # False  
  
storage = {'nuts':11,  
          'knobs':15,  
          'pins':0,  
          'bolts':7}  
  
# write here
```

Challenge - Galactic storm

⊕⊕ A micro black hole passed through the solar system, bringing chaos into all moon orbits. The probes sent by the Institute recorded **exactly 3** jumps of moons, where for each jump the last moon of one planet became the last moon of another.

Write some code which give given a dictionary `galaxy` mapping planets to their moons, **MODIFIES** `galaxy` according to another dictionary `jumps`

- assume there cannot be chains of jumps (i.e. no Jupyter -> Mars -> Neptune)
- **DO NOT** write constant planet names in your code (so no '`Jupyter`')

¹⁹³ <https://en.softpython.org/basics/basics2-bools-sol.html#Evaluation-order>

- **DO NOT** replace the assignments in `galaxy` (so no `galaxy[planet] = ...`)
- **DO NOT** use search methods (so no `.remove`, `.index`...)

Example - given:

(note for astrophiles: we didn't care putting all moons, nor used any particular order)

```
galaxy = {
    'Jupiter' : ['Io', 'Europa', 'Ganymede', 'Callisto'],
    'Saturn' : ['Mimas', 'Enceladus', 'Dione', 'Rhea', 'Titan', 'Hyperion'],
    'Mars' : ['Phobos', 'Deimos'],
    'Earth' : ['Moon'],
    'Neptune' : ['Triton', 'Proteus', 'Despina', 'Thalassa'],
    'Uranus' : ['Titania', 'Oberon']
}

jumps ={
    'Jupiter' : 'Mars',
    'Saturn' : 'Neptune',
    'Earth' : 'Uranus'
}
```

After your code, it must result:

```
>>> galaxy
{'Jupiter': ['Io', 'Europa', 'Ganymede'],
 'Saturn': ['Mimas', 'Enceladus', 'Dione', 'Rhea', 'Titan'],
 'Mars': ['Phobos', 'Deimos', 'Callisto'],
 'Earth': [],
 'Neptune': ['Triton', 'Proteus', 'Despina', 'Thalassa', 'Hyperion'],
 'Uranus': ['Titania', 'Oberon', 'Moon']}
```

[4]:

```
galaxy = {
    'Jupiter' : ['Io', 'Europa', 'Ganymede', 'Callisto'],
    'Saturn' : ['Mimas', 'Enceladus', 'Dione', 'Rhea', 'Titan', 'Hyperion'],
    'Mars' : ['Phobos', 'Deimos'],
    'Earth' : ['Moon'],
    'Neptune' : ['Triton', 'Proteus', 'Despina', 'Thalassa'],
    'Uranus' : ['Titania', 'Oberon']
}

jumps ={
    'Jupiter' : 'Mars',
    'Saturn' : 'Neptune',
    'Earth' : 'Uranus'
}

# write here
```

[]:

A2 CONTROL FLOW

6.1 If command

6.1.1 Conditionals - if else

[Download exercises zip](#)

Browse online files¹⁹⁴

We can use the conditional command `if` every time the computer must take a decision according to the value of some condition. If the condition is evaluated as true (that is, the boolean `True`), then a code block will be executed, otherwise execution will pass to another one.

References:

- [Basics - booleans¹⁹⁵](#)

What to do

1. Unzip [exercises zip](#) in a folder, you should obtain something like this:

```
if
  if1.ipynb
  if1-sol.ipynb
  if2-chal.ipynb
  jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `if1.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`

¹⁹⁴ <https://github.com/DavidLeoni/softpython-en/tree/master/if>

¹⁹⁵ <https://en.softpython.org/basics/basics2-bools-sol.html>

- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

The basic command if else

Let's see a small program which takes different decisions according to the value of a variable sweets:

```
[2]: sweets = 20

if sweets > 10 :
    print('We found...')
    print('Many sweets!')
else:
    print("Alas there are.. ")
    print('few sweets')

print()
print("Let's find other sweets!")
```

We found...
Many sweets!

Let's find other sweets!

The condition here is sweets > 10

```
[3]: sweets > 10
[3]: True
```

WARNING: Right after the condition you must place a colon :

```
if sweets > 10:
```

Since in the example above sweets is valued 20, the condition gets evaluated to True and so the code block following the if row gets executed.

Let's try instead to place a small number, like sweets = 5:

```
[4]: sweets = 5

if sweets > 10 :
    print('We found...')
    print('Many sweets!')
else:
    print("Alas there are.. ")
    print('Few sweets')

print()
print("Let's find other sweets!")

Alas there are..
Few sweets!

Let's find other sweets!
```

In this case, the code block after the `else:` row got executed

WARNING: Careful about block indentation!

As all code blocks in Python, they are preceded by spaces. Usually there are 4 spaces (in some Python projects you can find only 2, but official Python guidelines recommend 4)

else is optional

It is not mandatory to use `else`. If we omit it and the condition becomes `False`, the control directly pass to commands with the same indentation level of `if` (without errors):

```
[5]: sweets = 5

if sweets > 10 :
    print('We found...')
    print('Many sweets!')

print()
print("Let's find other sweets!")
```

Let's find other sweets!

QUESTION: Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

1. `x = 3
if x > 2 and if x < 4:
 print('ABBA')`

2. `x = 3
if x > 2 and x < 4
 print('ABBA')`

3. `x = 3
if x > 2 and x < 4:
 print('ABBA')`

4. `x = 2
if x > 1:
 print(x+1, x):`

5. `x = 3
if x > 5 or x:
 print('ACDC')`

6. `x = 7
if x == 7:
 print('GLAM')`

7. `x = 7
if x < 1:`

(continues on next page)

(continued from previous page)

```
    print('BIM')
else:
    print('BUM')
print('BAM')
```

8. `x = 30
if x > 8:
 print('DOH')
if x > 10:
 print('DUFF')
if x > 20:
 print('BURP')`

9. `if not True:
 print('upside down')
else:
 print('down upside')`

10. `if False:
else:
 print('ZORB')`

11. `if False:
 pass
else:
 print('ZORB')`

12. `if 0:
 print('Brandy')
else:
 print('Rum')`

13. `if False:
 print('illustrious')
else:
 print('distinguished')
else:
 print('excellent')`

14. `if 2 != 2:
 'BE'
else:
 'CAREFUL'`

15. `if 2 != 2:
 print('BE')
else:
 print('CAREFUL')`

16. `x = [1, 2, 3]
if 4 in x:
 x.append(4)
else:`

(continues on next page)

(continued from previous page)

```
x.remove(3)
print(x)
```

17. `if 'False':
 print('WATCH OUT FOR THE STRING!')
else:
 print('CRUEL')`

Exercise - no fuel

You want to do a car trip for which you need at least 30 litres of fuel. Write some code that:

- if the fuel variable is less than 30, prints 'Not enough fuel, I must fill up' and increments fuel of 20 litres
- Otherwise, prints Enough fuel!
- In any case, prints at the end 'We depart with ' followed by the final quantity of fuel

Example - given:

```
fuel = 5
```

After your code, it must print:

```
Not enough fuel, I must fill up
We depart with 25 litres
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```
fuel = 5
#fuel = 30

# write here

if fuel < 30:
    print('Not enough fuel, I must fill up')
    fuel += 20
else:
    print('Enough fuel!')

print('We depart with', fuel, 'litres')
```

```
Not enough fuel, I must fill up
We depart with 25 litres
```

</div>

[6]:

```
fuel = 5
#fuel = 30

# write here
```

The command *if - elif - else*

By examining the little sweets program we just saw, you may have wondered what it should print when there are no sweets at all. To handle many conditions, we could chain them with the command `elif` (abbreviation of *else if*):

```
[7]: sweets = 0 # WE PUT ZERO

if sweets > 10:
    print('We found...')
    print('Many sweets!')
elif sweets > 0:
    print("Alas there are... ")
    print('Few sweets!')
else:
    print("Too bad!")
    print('There are no sweets!')

print()
print("Let's find other sweets!")

Too bad!
There are no sweets!

Let's find other sweets!
```

EXERCISE: Try changing the values of `sweets` in the above cell and see what happens

The little program behaves exactly like the previous ones and when no condition is satisfied the last code block after the `else` is executed:

We can add as many `elif` as we want, so we could even put a specific `elif x == 0:` and handle in the `else` all other cases, even the unforeseen or absurd ones like for example placing a negative number of sweets. Why should we do it? Accidents can always happen, you surely found a good deal of *bugged* programs in your daily life... (we will see how to better handle these situations in the tutorial [Errors handling and testing¹⁹⁶](#))

```
[8]: sweets = -2 # LET'S TRY A NEGATIVE NUMBER

if sweets > 10:
    print('We found...')
    print('Many sweets!')
elif sweets > 0:
    print("Alas there are... ")
    print('Few sweets!')
elif sweets == 0:
    print("Too bad! ")
    print('There are no sweets!')
else:
    print('Something went VERY WRONG! We found', sweets, 'sweets')

print()
print("Let's find other sweets!")

Something went VERY WRONG! We found -2 sweets

Let's find other sweets!
```

EXERCISE: Try changing the values of `sweets` in the cell above and see what happens

¹⁹⁶ <https://en.softpython.org/errors-and-testing/errors-and-testing-sol.html>

Questions

Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

```
1. y = 2
   if y < 3:
       print('bingo')
   elif y <= 2:
       print('bango')
```

```
2. z = 'q'
   if not 'quando'.startswith(z):
       print('BAR')
   elif not 'spqr'[2] == z:
       print('WAR')
   else:
       print('ZAR')
```

```
3. x = 1
   if x < 5:
       print('SHIPS')
   elif x < 3:
       print('RAFTS')
   else:
       print('LIFEBOATS')
```

```
4. x = 5
   if x < 3:
       print('GOLD')
   else if x >= 3:
       print('SILVER')
```

```
5. if 0:
       print(0)
   elif 1:
       print(1)
```

Questions - Are they equivalent?

Look at the following code fragments: each contains two parts, A and B. For each value of the variables they depend on, try guessing whether part A will print exactly the same result printed by code in part B

- **FIRST** think about the answer
- **THEN** try executing with each of the values of suggested variables

Are they equivalent? - strawberries

Try changing the value of strawberries by removing the comments

```
strawberries = 5
#strawberries = 2
#strawberries = 10

print('strawberries =', strawberries)
print('A:')
if strawberries > 5:
    print("The strawberries are > 5")
elif strawberries > 5:
    print("I said the strawberries are > 5!")
else:
    print("The strawberries are <= 5")

print('B:')
if strawberries > 5:
    print("The strawberries are > 5")
if strawberries > 5:
    print("I said the strawberries are > 5!")
if strawberries <= 5:
    print("The strawberries are <= 5")
```

Are they equivalent? - max

```
x, y = 3, 5
#x, y = 5, 3
#x, y = 3, 3

print('x =', x)
print('y =', y)

print('A:')
if x > y:
    print(x)
else:
    print(y)

print('B:')
print(max(x,y))
```

Are they equivalent? - min

```
x, y = 3, 5
#x, y = 5, 3
#x, y = 3, 3

print('x =', x)
print('y =', y)

print('A:')
```

(continues on next page)

(continued from previous page)

```

if x < y:
    print(y)
else:
    print(x)

print('B:')
print(min(x,y))

```

Are they equivalent? - big small

```

x = 2
#x = 4
#x = 3

print('x =',x)

print('A:')
if x > 3:
    print('big')
else:
    print('small')

print('B:')
if x < 3:
    print('small')
else:
    print('big')

```

Are they equivalent? - Cippirillo

```

x = 3
#x = 10
#x = 11
#x = 15

print('x =', x)

print('A:')
if x % 5 == 0:
    print('cippirillo')
if x % 3 == 0:
    print('cippirillo')

print('B:')
if x % 3 == 0 or x % 5 == 0:
    print('cippirillo')

```

Exercise - farm

Given a string `s`, write some code which prints 'BARK!' if the string ends with `dog`, prints 'CROAK!' if the string ends with '`frog`' and prints '???' in all other cases

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[9]:

```
s = 'bulldog'
#s = 'bullfrog'
#s = 'frogbull'

print(s)

# write here

if s.endswith('dog'):
    print('BAU')
elif s.endswith('frog'):
    print('CROAK!')
else:
    print('????')

bulldog
BAU
```

</div>

[9]:

```
s = 'bulldog'
#s = 'bullfrog'
#s = 'frogbull'

print(s)

# write here
```

Exercise - accents

Write some code which prints whether a `word` ends or not with an accented character.

- To determine if a character is accented, use the strings of accents `acute` and `grave`
- Your code must work with any `word`

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[10]:

```
acute = "áéíóú"
grave = "àèìòù"

word = 'urrà'          # ends with an accent
#word = 'martello'   # does not end with an accent
```

(continues on next page)

(continued from previous page)

```
#word = 'ahó'          # ends with an accent
#word = 'però'         # ends with an accent
#word = 'capitaneria' # does not end with an accent
#word = 'viceré'       # ends with an accent
#word = 'cioè'          # ends with an accent
#word = 'chéto'         # does not end with an accent
#word = 'Chi dice che la verità è una sòla?' # does not end with an accent

# write here

if word[-1] in acute or word[-1] in grave:
    print(word, 'ends with an accent!')
else:
    print(word, 'does not end with an accent')

urrà ends with an accent!
```

</div>

[10]:

```
acute = "áéíóú"
grave = "àèìòù"

word = 'urrà'          # ends with an accent
#word = 'martello'    # does not end with an accent
#word = 'ahó'          # ends with an accent
#word = 'però'          # ends with an accent
#word = 'capitaneria' # does not end with an accent
#word = 'viceré'        # ends with an accent
#word = 'cioè'          # ends with an accent
#word = 'chéto'         # does not end with an accent
#word = 'Chi dice che la verità è una sòla?' # does not end with an accent

# write here
```

Exercise - Arcana

Given an arcana `x` expressed as a string and a list of `majors` and `minors` arcana, print to which category `x` belongs. If `x` does not belong to any category, prints is a `Mistery`.

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[11]:

```
x = 'Wheel of Fortune' # The Wheel of Fortune is a Major Arcana
#x = 'The Tower'        # major
#x = 'Ace of Swords'   # minor
#x = 'Two of Coins'    # minor
#x = 'Coding'           # mystery

majors = ['Wheel of Fortune', 'The Chariot', 'The Tower']
minors = ['Ace of Swords', 'Two of Coins', 'Queen of Cups']
```

(continues on next page)

(continued from previous page)

```
# write here
if x in majors:
    print('The', x, 'is a Major Arcana')
elif x in minors:
    print('The', x, 'is a Minor Arcana')
else:
    print(x, 'is a Mystery')
```

```
The Wheel of Fortune is a Major Arcana
```

```
</div>
```

```
[11]:
```

```
x = 'Wheel of Fortune' # The Wheel of Fortune is a Major Arcana
#x = 'The Tower'        # major
#x = 'Ace of Swords'   # minor
#x = 'Two of Coins'    # minor
#x = 'Coding'          # mystery

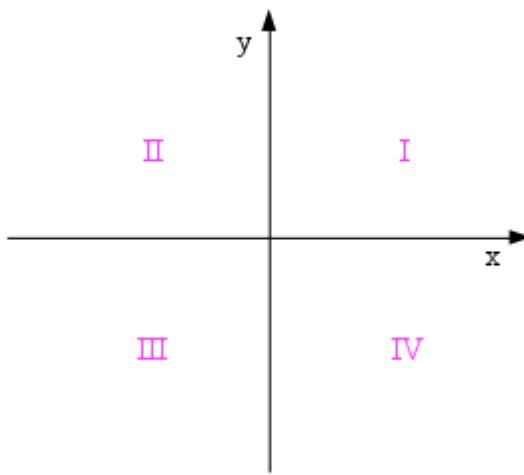
majors = ['Wheel of Fortune', 'The Chariot', 'The Tower']
minors = ['Ace of Swords', 'Two of Coins', 'Queen of Cups']

# write here
```

Nested ifs

`if` commands are *blocks* so they can be nested as any other block.

Let's make an example. Suppose you have a point at coordinates `x` and `y` and you want to know in which quadrant it lies:



You might write something like this:

```
[12]: x, y = 5, 9
#x, y = -5, 9
#x, y = -5, -9
```

(continues on next page)

(continued from previous page)

```
#x,y = 5,-9

print('x =',x,'y =', y)

if x >= 0:
    if y >= 0:
        print('first quadrant')
    else:
        print('fourth quadrant')
else:
    if y >= 0:
        print('second quadrant')
    else:
        print('third quadrant')

x = 5 y = 9
first quadrant
```

EXERCISE: try the various couples of suggested points by removing the comments and convince yourself the code is working as expected.

NOTE: Sometime the nested `if` can be avoided by writing sequences of `elif` with boolean expressions which verify two conditions at a time:

```
[13]: x,y = 5,9
#x,y = -5,9
#x,y = -5,-9
#x,y = 5,-9

print('x =',x,'y =', y)

if x >= 0 and y >= 0:
    print('first quadrant')
elif x >= 0 and y < 0:
    print('fourth quadrant')
elif x < 0 and y >= 0:
    print('second quadrant')
elif x < 0 and y < 0:
    print('third quadrant')

x = 5 y = 9
first quadrant
```

Exercise - abscissae and ordinates 1

The code above is not very precise, as doesn't consider the case of points which lie on axes. In these cases instead of the quadrant number it should print:

- ‘origin’ when `x` and `y` are equal to 0
- ‘ascissae’ when `y` is 0
- ‘ordinate’ when `x` is 0

Write down here a modified version of the code with nested ifs which takes into account also these cases, then test it by removing the comments from the various suggested point coordinates.

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[14]:

```
x,y = 0,0      # origin
#x,y = 0,5      # ordinate
#x,y = 5,0      # abscissa
#x,y = 5,9      # first
#x,y = -5,9     # second
#x,y = -5,-9    # third
#x,y = 5,-9     # fourth

print('x =',x,'y =', y)

# write here
if x == 0 and y == 0:
    print('origin')
elif x == 0:
    print('ordinate')
elif x > 0:
    if y == 0:
        print('abscissa')
    elif y > 0:
        print('first quadrant')
    else:
        print('fourth quadrant')
else:
    if y == 0:
        print('abscissa')
    elif y > 0:
        print('second quadrant')
    else:
        print('third quadrant')

x = 0 y = 0
origin
```

</div>

[14]:

```
x,y = 0,0      # origin
#x,y = 0,5      # ordinate
#x,y = 5,0      # abscissa
#x,y = 5,9      # first
#x,y = -5,9     # second
#x,y = -5,-9    # third
#x,y = 5,-9     # fourth

print('x =',x,'y =', y)

# write here
```

Esercise - abscissae and ordinates 2

If we wanted to be even more specific, instead of a generic ‘abscissa’ or ‘ordinate’, we might print:

- ‘abscissa between the first and fourth quadrant’
- ‘abscissa between the second and third quadrant’
- ‘ordinate between the first and the second quadrant’
- ‘ordinate between the third and the fourth quadrant’

Copy the code from the previous exercise, and modify it to also consider such cases.

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
    data-jupman-show="Show solution"
    data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[15]:

```
x,y = 0,0      # origin
#x,y = 0,5    # ordinate between the first and the second quadrant
#x,y = 0,-5   # ordinate between the third and the fourth quadrant
#x,y = 5,0    # abscissa between the first and the fourth quadrant
#x,y = -5,0   # abscissa between the second and the third quadrant
#x,y = 5,9    # first
#x,y = -5,9   # second
#x,y = -5,-9  # third
#x,y = 5,-9   # fourth

print('x =',x,'y =', y)

# write here
if x == 0 and y == 0:
    print('origin')
elif x == 0:
    if y > 0:
        print('ordinate between the first and the second quadrant')
    else:
        print('ordinate between the third and the fourth quadrant')
elif x > 0:
    if y == 0:
        print('abscissa between the first and the fourth quadrant')
    elif y > 0:
        print('first quadrant')
    else:
        print('fourth quadrant')
else:
    if y == 0:
        print('abscissa between the second and the third quadrant')
    elif y > 0:
        print('second quadrant')
    else:
        print('third quadrant')

x = 0 y = 0
origin
```

</div>

[15]:

```
x,y = 0,0      # origin
```

(continues on next page)

(continued from previous page)

```
#x,y = 0, 5      # ordinate between the first and the second quadrant
#x,y = 0, -5     # ordinate between the third and the fourth quadrant
#x,y = 5, 0      # abscissa between the first and the fourth quadrant
#x,y = -5, 0     # abscissa between the second and the third quadrant
#x,y = 5, 9      # first
#x,y = -5, 9     # second
#x,y = -5, -9    # third
#x,y = 5, -9     # fourth

print('x =',x,'y =', y)

# write here
```

Exercise - bus

You must catch the bus, and only have few minutes left. To do the trip:

- you need the backpack, otherwise you remain at home
- you also need money for the ticket or the transport card or both, otherwise you remain at home.

Write some code which given three variables `backpack`, `money` and `card`, prints what you see in the comments according to the various cases. Once you're done writing the code, test the results by removing comments from the assignments.

- **HINT:** to keep track of the found objects, try creating a list of strings which holds the objects

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[16]:

```
backpack, money, card = True, False, True
# I have no money !
# I've found: backpack,card
# I can go !

#backpack, money, card = False, False, True
# I don't have the backpack, I can't go !

#backpack, money, card = True, True, False
# I have no card !
# I've found: backpack,money
# I can go !

#backpack, money, card = True, True, True
# I've found: backpack,money,card
# I can go !

#backpack, money, card = True, False, False
# I have no money !
# I have no card !
# I don't have the card nor the money, I can't go !
```

(continues on next page)

(continued from previous page)

```
# write here

found = []
if backpack:
    found.append('backpack')
    if money:
        found.append('money')
    else:
        print('I have no money !')
    if card:
        found.append('card')
    else:
        print('I have no card !')
    if money or card:
        print("I've found:", ', '.join(found))
        print('I can go !')
    else:
        print("I don't have the card nor the money, I can't go !")
else:
    print("I don't have the backpack, I can't go !")
```

```
I have no money !
I've found: backpack,card
I can go !
```

</div>

[16]:

```
backpack, money, card = True, False, True
# I have no money !
# I've found: backpack,card
# I can go !

#backpack, money, card = False, False, True
# I don't have the backpack, I can't go !

#backpack, money, card = True, True, False
# I have no card !
# I've found: backpack,money
# I can go !

#backpack, money, card = True, True, True
# I've found: backpack,money,card
# I can go !

#backpack, money, card = True, False, False
# I have no money !
# I have no card !
# I don't have the card nor the money, I can't go !

# write here
```

Exercise - chronometer

A chronometer is counting the hours, minutes and seconds since the midnight of a certain day in a string chronometer, in which the numbers of hours, minutes and seconds are separated by colon :

Write some code which prints the day phase according to the number of passed hours:

- from 6:00 included to 12:00 excluded: prints morning
- from 12:00 included to 18:00 excluded: prints afternoon
- from 18:00 included to 21:00 excluded: prints evening
- from 21:00 included to 6:00 excluded: prints night
- **USE elif** with multiple boolean expressions
- Your code **MUST** work even if the chronometer goes beyond 23:59:59, see examples
- **HINT:** use the modulo operator % for having hours which only go from 0 to 23

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[17]:

```
chronometer = '10:23:43'    # morning
#chronometer = '12:00:00'    # afternoon
#chronometer = '15:56:02'    # afternoon
#chronometer = '19:23:27'    # evening
#chronometer = '21:45:15'    # night
#chronometer = '02:45:15'    # night
#chronometer = '27:45:30'    # night
#chronometer = '32:28:30'    # morning

# write here
hour = int(chronometer.split(':')[0]) % 24

if hour >= 6 and hour < 12:
    print('morning')
elif hour >=12 and hour < 18:
    print('afternoon')
elif hour >=18 and hour < 21:
    print('evening')
else:
    print('night')

morning
```

</div>

[17]:

```
chronometer = '10:23:43'    # morning
#chronometer = '12:00:00'    # afternoon
#chronometer = '15:56:02'    # afternoon
#chronometer = '19:23:27'    # evening
#chronometer = '21:45:15'    # night
#chronometer = '02:45:15'    # night
#chronometer = '27:45:30'    # night
#chronometer = '32:28:30'    # morning

# write here
```

(continues on next page)

(continued from previous page)

Questions - Are they equivalent?

Look at the following code fragments: each contains two parts, A and B. For each value of `x`, try guessing whether part A will print exactly the same result printed by code in part B

- **FIRST** think about the answer
- **THEN** try executing with each of the suggested values of `x`

Are they equivalent? - inside outside 1

```
x = 3
#x = 4
#x = 5

print('x =', x)

print('A:')
if x > 3:
    if x < 5:
        print('inside')
    else:
        print('outside')
else:
    print('outside')

print('B:')
if x > 3 and x < 5:
    print('inside')
else:
    print('outside')
```

Are they equivalent? - stars planets

```
x = 2
#x = 3
#x = 4

print('x =', x)

print('A:')
if not x > 3:
    print('stars')
else:
    print('planets')

print('B:')
if x > 3:
```

(continues on next page)

(continued from previous page)

```
    print('planets')
else:
    print('stars')
```

Are they equivalent? - green red

```
x = 10
#x = 5
#x = 0

print('x =', x)

print('A:')
if x >= 5:
    print('green')
    if x >= 10:
        print('red')

print('B:')
if x >= 10:
    if x >= 5:
        print('green')
    print('red')
```

Are they equivalent? - circles squares

```
x = 4
#x = 3
#x = 2
#x = 1
#x = 0

print('x =', x)

print('A:')
if x > 3:
    print('circles')
else:
    if x > 1:
        print('squares')
    else:
        print('triangles')

print('B:')
if x <= 1:
    print('triangles')
elif x <= 3:
    print('squares')
else:
    print('circles')
```

Are they equivalent? - inside outside 2

```
x = 7
#x = 0
#x = 15

print('x =', x)

print('A:')
if x > 5:
    if x < 10:
        print('inside')
    else:
        print('outside')
else:
    print('outside')

print('B:')
if not x > 5 and not x < 10:
    print('outside')
else:
    print('inside')
```

Are they equivalent? - Ciabanga

```
x = 4
#x = 5
#x = 6
#x = 9
#x = 10
#x = 11

print('x =', x)

print('A:')
if x < 6:
    print('Ciabanga!')
else:
    if x >= 10:
        print('Ciabanga!')

print('B:')
if x <= 5 or not x < 10:
    print('Ciabanga!')
```

Exercise - The maximum

Write some code which prints the maximum value among the numbers x, y and z

- use **nested ifs**
- **DO NOT** use the function `max`
- **DO NOT** create variables named `max` (it would violate the V Commandment¹⁹⁷: you shall never ever redefine system functions)

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[18]:

```
x,y,z = 1,2,3
#x,y,z = 1,3,2
#x,y,z = 2,1,3
#x,y,z = 2,3,1
#x,y,z = 3,1,2
#x,y,z = 3,2,1

# write here

if x > y:
    if x > z:
        print(x)
    else:
        print(z)
elif y > z:
    print(y)
else:
    print(z)
```

3

</div>

[18]:

```
x,y,z = 1,2,3
#x,y,z = 1,3,2
#x,y,z = 2,1,3
#x,y,z = 2,3,1
#x,y,z = 3,1,2
#x,y,z = 3,2,1

# write here
```

¹⁹⁷ <https://en.softpython.org/commandments.html#V-COMMANDMENT>

Ternary operator

In some cases, initializing a variable with different values according to a condition may result convenient.

Example:

The discount which is applied to a purchase depends on the purchased quantity. Create a variable `discount` by setting its value to 0 if the variable `expense` is less than 100€, or 10% if it is greater.

```
[19]: expense = 200
discount = 0

if expense > 100:
    discount = 0.1
else:
    discount = 0 # not necessary

print("expense:", expense, " discount:", discount)
expense: 200 discount: 0.1
```

The previous code can be written more concisely like this:

```
[20]: expense = 200
discount = 0.1 if expense > 100 else 0
print("expense:", expense, " discount:", discount)
expense: 200 discount: 0.1
```

The syntax of the ternary operator is:

```
VARIABLE = VALUE if CONDITION else ANOTHER_VALUE
```

which means that VARIABLE is initialized to VALUE if CONDITION is True, otherwise it is initialized to OTHER_VALUE

Questions ternary ifs

QUESTION: Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

1.

```
y = 3
x = 8 if y < 2 else 9
print(x)
```

2.

```
y = 1
z = 2 if y < 3
```

3.

```
y = 10
z = 2 if y < 3 elif y > 5 9
```

Exercise - shoes

Write some code which given the numerical variable `shoes`, if `shoes` is less than 10 it gets incremented by 1, otherwise it is decremented by 1

- USE ONLY the **ternary if**
- Your code must work for *any* value of `shoes`

Example 1 - given:

```
shoes = 2
```

After your code, it must result:

```
>>> print(shoes)
3
```

Example 2 - given:

```
shoes = 16
```

After your code, it must result:

```
>>> print(shoes)
15
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[21]:

```
shoes = 2
#shoes = 16

# write here

shoes = shoes + 1 if shoes < 10 else shoes - 1
print('shoes =', shoes)

shoes = 3
```

</div>

[21]:

```
shoes = 2
#shoes = 16

# write here
```

Exercise - the little train

Write some code which given 3 strings `sa`, `sb` and `sc` assigns the string `CHOO CHOO` to variable `x` if it is possible to compose `sa`, `sb` and `sc` to obtain the writing '`the little train`', otherwise assigns the string '`:-()`'

- **USE** a ternary if
- your code must work for **any** triplet of strings
- **NOTE:** we are only interested to know IF it is possible to compose writings like '`the little train`', we are NOT interested in which order they will get composed
- **HINT:** you are allowed to create a helper list

Example 1 - given:

```
sa, sb, sc = "little", "train", "the"
```

after your code, it must result:

```
>>> print(x)
CHOO CHOO
```

Example 2 - given:

```
sa, sb, sc = "quattro", "ni", "no"
```

after your code, it must result:

```
>>> print(x)
:-()
```

`<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"` data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[22]:

```
sa, sb, sc = "little", "train", "the"      # CHOO CHOO
#sa, sb, sc = "little", "the", "train"    # CHOO CHOO
#sa, sb, sc = "a", "little", "train"      # :-(
#sa, sb, sc = "train", "no", "no"         # :-()

# write here
words = [sa, sb, sc]
x = 'CHOO CHOO' if 'the' in words and 'little' in words and 'train' in words else ':-('
print(x)
```

CHOO CHOO

</div>

[22]:

```
sa, sb, sc = "little", "train", "the"      # CHOO CHOO
#sa, sb, sc = "little", "the", "train"    # CHOO CHOO
#sa, sb, sc = "a", "little", "train"      # :-(
#sa, sb, sc = "train", "no", "no"         # :-()
```

(continues on next page)

(continued from previous page)

```
# write here
```

Continue

Go on with **if** - first challenges¹⁹⁸

6.1.2 If 2 - Challenges

Download exercises zip

Browse online files¹⁹⁹

Treasure Island

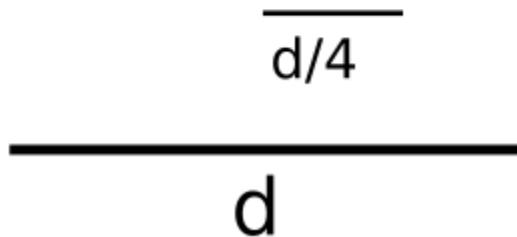
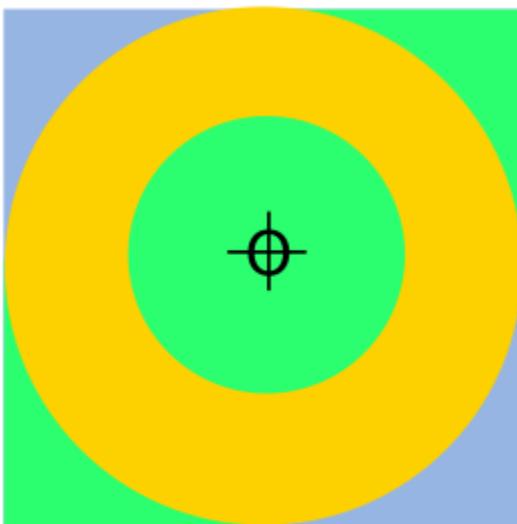
While reading ancient manuscripts you discovered there is an island in the pacific where incredible treasures are buried. In the manuscripts there is a map, and the zones where the gold could be are marked in green. You can ignore the other colors. You send in a drone which can land and drill the terrain. Strong winds and various factors could move the drone away from the target, so the drone at every moment needs to know whether or not is on a zone it should drill.

Write some code which given the map side length d and two coordinates x and y , RETURN True if the place is to drill (that is, the drone is on a green zone), otherwise return False.

ASSUME THAT THE ORIGIN (0,0) IS AT THE CENTRE OF THE MAP

¹⁹⁸ <https://en.softpython.org/if/if2-chal.html>

¹⁹⁹ <https://github.com/DavidLeoni/softpython-en/tree/master/if>



[1]:

```
import math

d = 10
x,y = 0,0 # True
#x,y = 0.2*d,0 # True
#x,y = 0,0.1*d # True
#x,y = 0,-0.03*d # True
#x,y = -0.01*d,-d*0.05 # True
# corona
#x,y = 0,-0.3*d # False
#x,y = 0.35*d,0 # False
#x,y = 0.4*d,0.27*d # False
#x,y = 0.31*d,-0.4*d # False
#x,y = -0.31*d,0.4*d # False
#x,y = -0.3*d,-0.38*d # False
# corners
#x,y = 0.49*d,0.49*d # True
#x,y = 0.45*d,-0.46*d # False
#x,y = -0.48*d,0.45*d # False
#x,y = -0.49*d,-0.47*d # True

# write here
```

6.2 For loops

6.2.1 For loops 1 - intro

[Download exercises zip](#)

Browse online files²⁰⁰

If we want to perform some actions for each element of a collection, we will need the so-called `for` loop, which allows to *iterate* any sequence.

What to do

1. Unzip [exercises zip](#) in a folder, you should obtain something like this:

```
for
    for1-intro.ipynb
    for1-intro-sol.ipynb
    for2-strings.ipynb
    for2-strings-sol.ipynb
    for3-lists.ipynb
    for3-lists-sol.ipynb
    for4-tuples.ipynb
    for4-tuples-sol.ipynb
    for5-sets.ipynb
    for5-sets-sol.ipynb
    for6-dictionaries.ipynb
    for6-dictionaries-sol.ipynb
    for7-nested.ipynb
    for7-nested-sol.ipynb
    for8-chal.ipynb
    jupman.py
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `for1-intro.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

²⁰⁰ <https://github.com/DavidLeoni/softpython-en/tree/master/for>

Iteration by element

If we have a sequence like this list:

```
[2]: sports = ['volleyball', 'tennis', 'soccer', 'swimming']
```

and we want to use every element of the list in some way (for example to print them), we can go through them (more precisely, *iterate*) with a `for` cycle:

```
[3]: for element in sports:
    print('Found an element!')
    print(element)

print('Done!')
```

```
Found an element!
volleyball
Found an element!
tennis
Found an element!
soccer
Found an element!
swimming
Done!
```

Let's see what happens in Python Tutor:

```
[4]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
#           (it's sufficient to execute it only once)

import jupman
```

```
[5]: sports = ['volleyball', 'tennis', 'soccer', 'swimming']
for element in sports:
    print('Found an element!')
    print(element)

print('Done!')

jupman.pytut()
```

```
Found an element!
volleyball
Found an element!
tennis
Found an element!
soccer
Found an element!
swimming
Done!
```

```
[5]: <IPython.core.display.HTML object>
```

Names of variables in `for`

At each iteration, an element of the list is assigned to the variable `element`.

As variable name we can choose whatever we like, for example this code is totally equivalent to the previous one:

```
[6]: sports = ['volleyball', 'tennis', 'soccer', 'swimming']
for name in sports:
    print('Found an element!')
    print(name)

print('Done!')

Found an element!
volleyball
Found an element!
tennis
Found an element!
soccer
Found an element!
swimming
Done!
```

We need to be careful about one thing:

II COMMANDMENT²⁰¹: Whenever you insert a variable in a `for` cycle, such variables must be new

If you defined the variable before, you shall not reintroduce it in a `for`, as this would bring confusion in the readers' mind.

For example:

```
[7]: sports = ['volleyball', 'tennis', 'soccer', 'swimming']
my_var = 'hello'

for my_var in sports: # you lose the original variable
    print(my_var)

print(my_var) # prints 'swimming' instead of 'hello'

volleyball
tennis
soccer
swimming
swimming
```

Iterating strings

Strings are sequences of characters, so we can iterate them with `for`:

```
[8]: for character in "hello":
    print(character)

h
e
```

(continues on next page)

²⁰¹ <https://en.softpython.org/commandments.html#II-COMMANDMENT>

(continued from previous page)

```
1
1
o
```

Iterating tuples

Tuples are also sequences so we can iterate them:

```
[9]: for word in ("I'm", 'visiting', 'a', 'tuple'):
    print(word)
```

```
I'm
visiting
a
tuple
```

Questions - iteration

Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

1. `for i in [1,2,3]:
 print(i)`

2. `for x in 7:
 print(x)`

3. `for x in [7]:
 print(x)`

4. `for x in ['a','b','c']:
 x`

5. `for i in []:
 print('GURB')`

6. `for i in [1,2,3]:
 print(type(i))`

7. `for i in '123':
 print(type(i))`

8. `for i in 'abc':
 print(i)`

9. `for x in ((4,5,6)):
 print(x)`

10. `for x in [[1],[2,3],[4,5,6]]:
 print(x)`

```
11. x = 5
    for x in ['a', 'b', 'c']:
        print(x)
    print(x)
```

```
12. for x in ['a', 'b', 'c']:
    pass
    print(x)
```

```
13. for x in [1, 2, 3, 4, 5, 6, 7, 8]:
    if x % 2 == 0:
        print(x)
```

```
14. la = [4, 5, 6]
    for x in la:
        print(x)
    la.reverse()
    for x in la[1:]:
        print(x)
```

Exercise - magic carpet

⊕ Months ago you bought a carpet from a pitchman. After some time, after a particularly stressful day, you say 'I wish I went on vacation to some exotic places, like say, *Marrakesh!*' To your astonishment, the carpet jumps in the air and answers: 'I hear and obey!'

Write some code which given the lists of places `trip1` and `trip2` prints all the visited stops.

Example - given:

```
trip1 = ['Marrakesh', 'Fez', 'Bazaar', 'Kasbah']
trip2 = ['Koutoubia', 'El Badii', 'Chellah']
```

Prints:

```
The first trip starts
    You: Let's go to Marrakesh !
    Carpet: I hear and obey
    You: Let's go to Fez !
    Carpet: I hear and obey
    You: Let's go to Bazaar !
    Carpet: I hear and obey
    You: Let's go to Kasbah !
    Carpet: I hear and obey
End of second trip
```

```
The second trip starts
    You: Let's go to Koutoubia !
    Carpet: I hear and obey
    You: Let's go to El Badii !
    Carpet: I hear and obey
    You: Let's go to Chellah !
    Carpet: I hear and obey
End of second trip
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[10]:

```
trip1 = ['Marrakesh', 'Fez', 'Bazaar', 'Kasbah']
trip2 = ['Koutoubia', 'El Badii', 'Chellah']

# write here
print('The first trip starts')
for place in trip1:
    print("    You: Let's go to",place,'!')
    print('    Carpet: I hear and obey')
print('End of second trip')

print()

print('The second trip starts')
for place in trip2:
    print("    You: Let's go to",place,'!')
    print('    Carpet: I hear and obey')
print('End of second trip')
```

```
The first trip starts
    You: Let's go to Marrakesh !
    Carpet: I hear and obey
    You: Let's go to Fez !
    Carpet: I hear and obey
    You: Let's go to Bazaar !
    Carpet: I hear and obey
    You: Let's go to Kasbah !
    Carpet: I hear and obey
End of second trip
```

```
The second trip starts
    You: Let's go to Koutoubia !
    Carpet: I hear and obey
    You: Let's go to El Badii !
    Carpet: I hear and obey
    You: Let's go to Chellah !
    Carpet: I hear and obey
End of second trip
```

</div>

[10]:

```
trip1 = ['Marrakesh', 'Fez', 'Bazaar', 'Kasbah']
trip2 = ['Koutoubia', 'El Badii', 'Chellah']

# write here
```

Esercise - evensum

⊕ Given the list `numbers`, write some code which calculates and prints the sum of the even **elements** (**not** the elements at even indexes !)

Example - given:

```
numbers = [3, 4, 1, 5, 12, 7, 9]
```

finds 4 and 12 so it must print:

```
16
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[11]:

```
numbers = [3, 4, 1, 5, 12, 7, 9]
```

```
# write here
```

```
s = 0
```

```
for x in numbers:
```

```
    if x % 2 == 0:
```

```
        s += x
```

```
print(s)
```

```
16
```

```
</div>
```

[11]:

```
numbers = [3, 4, 1, 5, 12, 7, 9]
```

```
# write here
```

Exercise - birbantello

⊕ Given a string in lowercase, write some code which prints each character in uppercase followed by the character as lowercase.

- **HINT:** to obtain uppercase characters use the `.upper()` method

Example - given:

```
s = "birbantello"
```

Prints:

```
B b  
I i  
R r  
B b  
A a  
N n  
T t
```

(continues on next page)

(continued from previous page)

```
E e
L l
L l
O o
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[12]:

```
s = "birbantello"

# write here
for x in s:
    print(x.upper(), x)
```

```
B b
I i
R r
B b
A a
N n
T t
E e
L l
L l
O o
```

</div>

[12]:

```
s = "birbantello"

# write here
```

Exercise - articulate

⊕ A new word is taught to a kid. He knows a lot of characters from the alphabet, but not all of them. To remember the known ones, he treats them as they were actors divided in three categories: the good, bad ad ugly. Write some code which given a word prints all the characters and for each of them tells whether it is good, bad or ugly. If a character is not recognized by the kid, prints 'not interesting'.

Example - given:

```
word = 'articulate'

good = 'abcde'
bad = 'ru'
ugly = 'ijklmn'
```

Prints:

```
a is good
r is bad
```

(continues on next page)

(continued from previous page)

```
t is not interesting
i is ugly
c is good
u is bad
l is ugly
a is good
t is not interesting
e is good
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
    data-jupman-show="Show solution"
    data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[13]:
```

```
word = 'articulate'

good = 'abcde'
bad = 'ru'
ugly = 'ijklmn'

# write here

for c in word:
    if c in good:
        print(c, 'is good')
    elif c in bad:
        print(c, 'is bad')
    elif c in ugly:
        print(c, 'is ugly')
    else:
        print(c, 'is not interesting')
```

```
a is good
r is bad
t is not interesting
i is ugly
c is good
u is bad
l is ugly
a is good
t is not interesting
e is good
```

```
</div>
```

```
[13]:
```

```
word = 'articulate'

good = 'abcde'
bad = 'ru'
ugly = 'ijklmn'

# write here
```

Exercise - gala

⊕ At a gala event, many high-society people are invited. At the beginning of the evening, doors are opened and guests enter a queue. Unfortunately, during these occasions uninvited guests always show up, so the concierge in the atrium is given a list of unwelcome ones. Whenever a guest is recognized as unwelcome, he will be taken care by the strong hands of Ferruccio the bouncer. Illustrious guests will be written instead in the list admitted.

Write some code which prints the various passages of the event.

Example - given:

```
queue = ['Consul', 'Notary', 'Skeleton', 'Dean', 'Goblin', 'Vampire', 'Jeweller']
unwelcome = {'Vampire', 'Goblin', 'Skeleton'}
admitted = []
```

Prints:

```
Open the doors!

Good evening Mr Consul
This way, Your Excellence
Next in line !
Good evening Mr Notary
This way, Your Excellence
Next in line !
Good evening Mr Skeleton
Ferruccio, would you please take care of Mr Skeleton ?
Next in line !
Good evening Mr Dean
This way, Your Excellence
Next in line !
Good evening Mr Goblin
Ferruccio, would you please take care of Mr Goblin ?
Next in line !
Good evening Mr Vampire
Ferruccio, would you please take care of Mr Vampire ?
Next in line !
Good evening Mr Jeweller
This way, Your Excellence
Next in line !

These guests were admitted: Consul, Notary, Dean, Jeweller
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[14]:

```
queue = ['Consul', 'Notary', 'Skeleton', 'Dean', 'Goblin', 'Vampire', 'Jeweller']
unwelcome = {'Vampire', 'Goblin', 'Skeleton'}
admitted = []

# write here
print('Open the doors!')
print()
for guest in queue:

    print('Good evening Mr', guest)
```

(continues on next page)

(continued from previous page)

```
if guest in unwelcome:
    print(" Ferruccio, would you please take care of Mr", guest, "?")
else:
    print(" This way, Your Excellence")
    admitted.append(guest)
print(' Next in line !')

print()
print('These guests were admitted:', ', '.join(admitted))
```

Open the doors!

Good evening Mr Consul
This way, Your Excellence
Next in line !

Good evening Mr Notary
This way, Your Excellence
Next in line !

Good evening Mr Skeleton
Ferruccio, would you please take care of Mr Skeleton ?
Next in line !

Good evening Mr Dean
This way, Your Excellence
Next in line !

Good evening Mr Goblin
Ferruccio, would you please take care of Mr Goblin ?
Next in line !

Good evening Mr Vampire
Ferruccio, would you please take care of Mr Vampire ?
Next in line !

Good evening Mr Jeweller
This way, Your Excellence
Next in line !

These guests were admitted: Consul, Notary, Dean, Jeweller

</div>

[14]:

```
queue = ['Consul', 'Notary', 'Skeleton', 'Dean', 'Goblin', 'Vampire', 'Jeweller']
unwelcome = {'Vampire', 'Goblin', 'Skeleton'}
admitted = []

# write here
```

Exercise - balance

⊕⊕ A crop of seeds has been harvested, and seeds will be poured in a certain number of bags of a given capacity each (i.e. 15 kilograms).

The seeds arrive in containers of variable capacity. Each container is placed on a weight scale and its content is poured in the current bag. As soon as the quantity capacity is reached, the scale weight is emptied, the bag is substituted with a new one which starts being filled from what remains from the previous fill. Write some code which prints the procedure.

Example - given:

```
containers = [5, 1, 7, 4, 3, 9, 5, 2, 7, 3]
capacity = 15
```

Prints:

```
Take 5 kg
The scale weight shows 5 kg
Take 1 kg
The scale weight shows 6 kg
Take 7 kg
The scale weight shows 13 kg
Take 4 kg
The scale weight shows 17 kg
We reached the capacity of 15 kg, there remain 2 kg

Take 3 kg
The scale weight shows 5 kg
Take 9 kg
The scale weight shows 14 kg
Take 5 kg
The scale weight shows 19 kg
We reached the capacity of 15 kg, there remain 4 kg

Take 2 kg
The scale weight shows 6 kg
Take 7 kg
The scale weight shows 13 kg
Take 3 kg
The scale weight shows 16 kg
We reached the capacity of 15 kg, there remain 1 kg

We filled 3 bags
```

Show solutionHide

[15]:

```
containers = [5, 1, 7, 4, 3, 9, 5, 2, 7, 3]
capacity = 15

# write here
bags = 0
k = 0
for n in containers:
    k += n
    print('Take', n, 'kg')
    print('The scale weight shows', k, 'kg')
```

(continues on next page)

(continued from previous page)

```
if k >= capacity:
    print('We reached the capacity of',capacity,'kg, there remain', k - capacity,
          'kg')
    print()
    k = k - capacity
    bags += 1

print('We filled', bags, 'bags')

Take 5 kg
The scale weight shows 5 kg
Take 1 kg
The scale weight shows 6 kg
Take 7 kg
The scale weight shows 13 kg
Take 4 kg
The scale weight shows 17 kg
We reached the capacity of 15 kg, there remain 2 kg

Take 3 kg
The scale weight shows 5 kg
Take 9 kg
The scale weight shows 14 kg
Take 5 kg
The scale weight shows 19 kg
We reached the capacity of 15 kg, there remain 4 kg

Take 2 kg
The scale weight shows 6 kg
Take 7 kg
The scale weight shows 13 kg
Take 3 kg
The scale weight shows 16 kg
We reached the capacity of 15 kg, there remain 1 kg

We filled 3 bags
```

</div>

[15]:

```
containers = [5,1,7,4,3,9,5,2,7,3]
capacity = 15

# write here
```

Counting with range

If we need to keep track of the iteration number, we can use the iterable sequence `range`, which produces a series of integer numbers from 0 INCLUDED until the specified number EXCLUDED:

```
[16]: for i in range(5):
    print(i)
```

0
1
2
3
4

Note it *did not* print the limit 5

When we call `range` we can also specify the starting index, which is INCLUDED in the generated sequence, while the arrival index is always EXCLUDED:

```
[17]: for i in range(3, 7):
    print(i)
```

3
4
5
6

Counting intervals: we can specify the increment to apply to the counter at each iteration by passing a third parameter, for example here we specify an increment of 2 (note the final 18 index is EXCLUDED from the sequence):

```
[18]: for i in range(4, 18, 2):
    print(i)
```

4
6
8
10
12
14
16

Reverse order: we can count in reverse by using a negative increment:

```
[19]: for i in range(5, 0, -1):
    print(i)
```

5
4
3
2
1

Note how the limit 0 *was not* reached, in order to arrive there we need to write

```
[20]: for i in range(5, -1, -1):
    print(i)
```

5
4
3

(continues on next page)

(continued from previous page)

```
2  
1  
0
```

Questions - range

Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

1. `for x in range(1):
 print(x)`

2. `for i in range(3):
 i`

3. `for i in range(3):
 print(i)`

4. `for x in range(-1):
 print(x)`

5. `for 'm' in range(3):
 print('m')`

6. `for i in range(3):
 i-1`

7. `for x in range(6, 4, -1):
 print(x)`

8. `for x in range(1, 0, -1):
 print(x)`

9. `for x in range(3, -3, -2):
 print(x)`

10. `for x in 3:
 print(x)`

11. `x = 3
for i in range(x):
 print(i)
for i in range(x, 2*x):
 print(i)`

12. `for x in range(range(3)):
 print(x)`

Exercise - printdoubles

⊕ Given a positive number n (i.e. n=4) write some code which prints:

```
The double of 0 is 0
The double of 1 is 2
The double of 2 is 4
The double of 3 is 6
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[21]:

```
n = 4
# write here
for i in range(n):
    print('The double of', i, 'is', i*2)
```

```
The double of 0 is 0
The double of 1 is 2
The double of 2 is 4
The double of 3 is 6
```

</div>

[21]:

```
n = 4
# write here
```

Exercise - multiples or not

⊕⊕ Write some code which given two integer positive numbers k and b:

- first prints all the numbers from k INCLUDED to b INCLUDED which are multiples of k
- the prints all the numbers from k EXCLUDED to b EXCLUDED which are NOT multiples of k

Example - given:

```
k,b = 3,15
```

it prints:

```
Multiples of 3
3
6
9
12
15

Not divisible by 3
4
5
7
8
```

(continues on next page)

(continued from previous page)

10
11
13
14

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[22]:

```
k,b = 3,15

# write here
print('Multiples of', k)
for i in range(k,b+1,k):
    print(i)
print()
print('Not divisible by', k)
for i in range(k+1,b):
    if i % k != 0:
        print(i)
```

```
Multiples of 3
3
6
9
12
15
```

```
Not divisible by 3
4
5
7
8
10
11
13
14
```

```
</div>
```

[22]:

```
k,b = 3,15

# write here
```

Exercise - ab interval

⊕⊕ Given two integers a and b greater or equal than zero, write some code which prints all the integer numbers among the two bounds INCLUDED.

- NOTE: a may be greater, equal or less than b , your code must handle all the cases.

Example 1 - given:

```
a, b = 5, 9
```

it must print:

```
5
6
7
8
9
```

Example 2 - given:

```
a, b = 8, 3
```

it must print:

```
3
4
5
6
7
8
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[23]:

```
a, b = 5, 9    # 5 6 7 8 9
#a, b = 8, 3    # 3 4 5 6 7 8
#a, b = 6, 6    # 6

# write here

mn = min(a, b)
mx = max(a, b)

for x in range(mn, mx + 1):
    print(x)
```

```
5
6
7
8
9
```

</div>

[23]:

```
a, b = 5, 9    # 5 6 7 8 9
```

(continues on next page)

(continued from previous page)

```
#a,b = 8,3 # 3 4 5 6 7 8  
#a,b = 6,6 # 6  
  
# write here
```

Exercise - FizzBuzz

Write some code which prints the numbers from 1 to 35 INCLUDED, but when a number is divisible by 3 prints instead FIZZ, when it is divisible by 5 prints BUZZ, and when it is divisible by 3 and 5 prints FIZZBUZZ.

Expected output:

```
1  
2  
FIZZ  
4  
BUZZ  
FIZZ  
7  
8  
FIZZ  
BUZZ  
11  
FIZZ  
13  
14  
FIZZBUZZ  
16  
17  
FIZZ  
19  
BUZZ  
FIZZ  
22  
23  
FIZZ  
BUZZ  
26  
FIZZ  
28  
29  
FIZZBUZZ  
31  
32  
FIZZ  
34  
BUZZ
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[24] :

(continues on next page)

(continued from previous page)

```
# write here
for i in range(1,36):
    if i % 15 == 0:
        print('FIZZBUZZ')
    elif i % 3 == 0:
        print('FIZZ')
    elif i % 5 == 0:
        print('BUZZ')
    else:
        print(i)
```

```
1
2
FIZZ
4
BUZZ
FIZZ
7
8
FIZZ
BUZZ
11
FIZZ
13
14
FIZZBUZZ
16
17
FIZZ
19
BUZZ
FIZZ
22
23
FIZZ
BUZZ
26
FIZZ
28
29
FIZZBUZZ
31
32
FIZZ
34
BUZZ
```

</div>

[24]:

```
# write here
```

Iterating by index

If we have a sequence like a list, sometimes during the iteration it is necessary to know in which cell position we are. We can generate the indexes with `range`, and use them to access a list:

```
[25]: sports = ['volleyball', 'tennis', 'soccer', 'swimming']

for i in range(len(sports)):
    print('position', i)
    print(sports[i])

position 0
volleyball
position 1
tennis
position 2
soccer
position 3
swimming
```

Note we passed to `range` the dimension of the list obtained with `len`.

Exercise - kitchen

⊕ Write some code which given a list of an even number of strings `kitchen`, prints the couples of elements we can find in sequences, one row at a time

Example - given:

```
kitchen = ['oil', 'soup', 'eggs', 'pie', 'tomato sauce', 'pasta', 'meat sauce',
↪'lasagna']
```

Prints:

```
oil, soup
eggs, pie
tomato sauce, pasta
meat sauce, lasagna
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[26]: kitchen = ['oil', 'soup', 'eggs', 'pie', 'tomato sauce', 'pasta', 'meat sauce',
↪'lasagna']

# write here
for i in range(0, len(kitchen)-1, 2):
    print(kitchen[i] + ',', kitchen[i+1])
```

```
oil, soup
eggs, pie
tomato sauce, pasta
meat sauce, lasagna
```

</div>

[26]:

```
kitchen = ['oil', 'soup', 'eggs', 'pie', 'tomato sauce', 'pasta', 'meat sauce',
↪'lasagna']

# write here
```

Exercise - neon

⊕ Given two lists `la` and `lb` of *equal length n*, write some code which prints their characters separated by a space on *n* rows

Example - given:

```
la = ['n', 'e', 'o', 'n']
lb = ['s', 'h', 'o', 'w']
```

prints:

```
n s
e h
o o
n w
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[27]:

```
la = ['n', 'e', 'o', 'n']
lb = ['s', 'h', 'o', 'w']

# write here

for i in range(len(la)):
    print(la[i], lb[i])
```

```
n s
e h
o o
n w
```

</div>

[27]:

```
la = ['n', 'e', 'o', 'n']
lb = ['s', 'h', 'o', 'w']

# write here
```

Exercise - emotions

⊕ Given the list of strings `emotions` and another one `grade` containing the numbers `-1` and `1`, write some code which prints the emotions followed with ‘positive’ if their corresponding grade is a number greater than zero or ‘negative’ otherwise

Example - given:

```
emotions = ['Fear', 'Anger', 'Sadness', 'Joy', 'Disgust', 'Ecstasy']
grade = [-1, -1, -1, 1, -1, 1]
```

prints:

```
Fear : negative
Anger : negative
Sadness : negative
Joy : positive
Disgust : negative
Ecstasy : positive
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[28]:

```
emotions = ['Fear', 'Anger', 'Sadness', 'Joy', 'Disgust', 'Ecstasy']
grade = [-1, -1, -1, 1, -1, 1]

# write here
for i in range(len(emotions)):
    if grade[i] > 0:
        print(emotions[i], ': positive')
    else:
        print(emotions[i], ': negative')
```

```
Fear : negative
Anger : negative
Sadness : negative
Joy : positive
Disgust : negative
Ecstasy : positive
```

</div>

[28]:

```
emotions = ['Fear', 'Anger', 'Sadness', 'Joy', 'Disgust', 'Ecstasy']
grade = [-1, -1, -1, 1, -1, 1]

# write here
```

Exercise - organetto

⊕ Given a string s , write some code which prints all the substrings you can obtain from the position of the character ' n ' and which terminates with the last character of s .

Example - given:

```
s = 'organetto'
```

Prints:

```
netto
etto
tto
to
o
```

[Show solution](#)

[29]:

```
s = 'organetto'

# write here
for i in range(s.index('n'), len(s)):
    print(s[i:])
```

```
netto
etto
tto
to
o
```

</div>

[29]:

```
s = 'organetto'

# write here
```

Exercise - sghiribizzo

Write some code which given the string s prints all the possible combinations of row couples such that a row begins with the first characters of s and the successive continues with the following characters.

Example - given:

```
s = 'sghiribizzo'
```

Prints:

```
s
ghiribizzo
sg
```

(continues on next page)

(continued from previous page)

```
hiribizzo
sgh
    iribizzo
sghi
    ribizzo
sghir
    ibizzo
sghiri
    bizzo
sghirib
    izzo
sghiribi
    zzo
sghiribiz
    zo
sghiribizz
    o
sghiribizzo
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[30]:

```
s = 'sghiribizzo'
# write here
for i in range(len(s)):
    print(s[:i+1])
    print(' '*i,s[i+1:])
```

```
s
ghiribizzo
sg
    hiribizzo
sgh
    iribizzo
sghi
    ribizzo
sghir
    ibizzo
sghiri
    bizzo
sghirib
    izzo
sghiribi
    zzo
sghiribiz
    zo
sghiribizz
    o
sghiribizzo
```

```
</div>
```

[30]:

```
s = 'sghiribizzo'
# write here
```

(continues on next page)

(continued from previous page)

Exercise - dna

Given two DNA strings `s1` and `s2` of equal length, write some code which prints among the first and second string another string made by spaces `` and pipe | where equal characters are found.

- **HINT:** create a list containing the characters space or the character |, and only at the end convert the string by using strings `join` method (doing so is much more efficient than keep generating strings with + operator)

Example - given:

```
s1 = "ATACATATAGGCCAATTATTATAAGTCAC"
s2 = "CGCCACTTAAGGCCCTGTATTAAAGTCGC"
```

Prints:

```
ATACATATAGGCCAATTATTATAAGTCAC
|| || | | | | | | | |
CGCCACTTAAGGCCCTGTATTAAAGTCGC
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[31]:

```
s1 = "ATACATATAGGCCAATTATTATAAGTCAC"
s2 = "CGCCACTTAAGGCCCTGTATTAAAGTCGC"

# write here
lst = []
for i in range(len(s1)):
    if(s1[i] == s2[i]):
        lst.append(' ')
    else:
        lst.append('|')

bars = ''.join(lst)

print(s1)
print(bars)
print(s2)
```

```
ATACATATAGGCCAATTATTATAAGTCAC
|| || | | | | | | | |
CGCCACTTAAGGCCCTGTATTAAAGTCGC
```

</div>

[31]:

```
s1 = "ATACATATAGGCCAATTATTATAAGTCAC"
s2 = "CGCCACTTAAGGCCCTGTATTAAAGTCGC"

# write here
```

(continues on next page)

(continued from previous page)

Exercise - sportello

⊕⊕ Given a string `s`, prints the first half of the characters as lowercase and the following half as uppercase.

- if the string is of odd length, the first half must have one character *more* than the second string.

Example - given:

```
s = 'sportello'
```

Your code must print:

```
s  
p  
o  
r  
t  
E  
L  
L  
O
```

(note that 'sportello' has odd length and there are *five* characters in the first half and *four* in the second

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[32]:

```
s = 'sportello' # sportELLO  
#s = 'maglia' # magLIA  
  
# write here  
  
if len(s) % 2 == 1:  
    midpoint = (len(s) // 2) + 1  
else:  
    midpoint = (len(s) // 2)  
  
for i in range(midpoint):  
    print(s[i])  
  
for i in range(midpoint, len(s)):  
    print(s[i].upper())
```

```
s  
p  
o  
r  
t  
E  
L  
L  
O
```

```
</div>
```

[32]:

```
s = 'sportello' # sportELLO
#s = 'maglia' # magLIA

# write here
```

Exercise - farm

⊕⊕ Given a dictionary `sounds` which associates animal names to the sounds they produce, and a list `rooms` of tuples of 2 elements containing the animal names, write some code that for each room prints the sounds you hear while passing in front of it.

- NOTE: the rooms to print are numbered **from 1**

Example - given:

```
sounds = {'dog':'Bark!',  
         'cat':'Mew!',  
         'cow':'Moo!',  
         'sheep':'Bleat!'}

rooms = [ ('dog', 'sheep'),  
         ('cat', 'cow'),  
         ('cow', 'dog') ]
```

Prints:

```
In the room 1 we hear Bark! and Bleat!  
In the room 2 we hear Mew! and Moo!  
In the room 3 we hear Moo! and Bark!
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[33]:

```
sounds = {'dog':'Bark!',  
         'cat':'Mew!',  
         'cow':'Moo!',  
         'sheep':'Bleat!'}

rooms = [ ('dog', 'sheep'),  
         ('cat', 'cow'),  
         ('cow', 'dog') ]

# write here

for i in range(len(rooms)):  
    room = rooms[i]  
    print('In the room',i+1,'we hear',sounds[room[0]], 'and', sounds[room[1]])
```

```
In the room 1 we hear Bark! and Bleat!  
In the room 2 we hear Mew! and Moo!  
In the room 3 we hear Moo! and Bark!
```

</div>

[33]:

```
sounds = {'dog':'Bark!',  
         'cat':'Mew!',  
         'cow':'Moo!',  
         'sheep':'Bleat!'}  
  
rooms = [('dog', 'sheep'),  
          ('cat', 'cow'),  
          ('cow', 'dog')]  
  
# write here
```

Exercise - pokemon

⊗⊗⊗ Given a list `pokemon` and a number `g` of groups, write some code which prints `g` rows showing all the group components. Group the pokemons in the order you find them in the list.

- **HINT 1:** To obtain the number of group components you should use integer division `//`
- **HINT 2:** to print group components use the method `join` of strings

Example 1 - given:

```
#           0           1           2           3           4           5  
pokemon = ['Charizard', 'Gengar', 'Arcanine', 'Bulbasaur', 'Blaziken', 'Umbreon',  
#           6           7           8           9           10          11  
          'Lucario', 'Gardevoir', 'Eevee', 'Dragonite', 'Volcarona', 'Sylveon' ]  
g = 3
```

prints:

```
group 1 : Charizard and Gengar and Arcanine and Bulbasaur  
group 2 : Blaziken and Umbreon and Lucario and Gardevoir  
group 3 : Eevee and Dragonite and Volcarona and Sylveon
```

Example 2 - given:

```
#           0           1           2           3           4           5  
pokemon = ['Charizard', 'Gengar', 'Arcanine', 'Bulbasaur', 'Blaziken', 'Umbreon',  
#           6           7           8           9           10          11  
          'Lucario', 'Gardevoir', 'Eevee', 'Dragonite', 'Volcarona', 'Sylveon' ]  
g = 4
```

prints:

```
group 1 : Charizard and Gengar and Arcanine  
group 2 : Bulbasaur and Blaziken and Umbreon  
group 3 : Lucario and Gardevoir and Eevee  
group 4 : Dragonite and Volcarona and Sylveon
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[34]:

```
#          0      1      2      3      4      5
pokemon = ['Charizard', 'Gengar', 'Arcanine', 'Bulbasaur', 'Blaziken', 'Umbreon',
#          6      7      8      9      10     11
           'Lucario', 'Gardevoir', 'Eevee', 'Dragonite', 'Volcarona', 'Sylveon' ]
g = 3
#g = 4

# write here
k = len(pokemon) // g # pokemon in a group

for i in range(0, g):
    print('group', i+1, ':', ' and '.join(pokemon[i*k:(i+1)*k]))
```

group 1 : Charizard and Gengar and Arcanine and Bulbasaur
group 2 : Blaziken and Umbreon and Lucario and Gardevoir
group 3 : Eevee and Dragonite and Volcarona and Sylveon

</div>

[34]:

```
#          0      1      2      3      4      5
pokemon = ['Charizard', 'Gengar', 'Arcanine', 'Bulbasaur', 'Blaziken', 'Umbreon',
#          6      7      8      9      10     11
           'Lucario', 'Gardevoir', 'Eevee', 'Dragonite', 'Volcarona', 'Sylveon' ]
g = 3
#g = 4

# write here
```

Modifying during iteration

Suppose you have a list `lst` containing characters, and you are asked to duplicate all the elements, for example if you have

```
lst = ['a', 'b', 'c']
```

after your code it must result

```
>>> print(lst)
['a', 'b', 'c', 'a', 'b', 'c']
```

Since you gained such great knowledge about iteration, you might be tempted to write something like this:

```
for char in lst:
    lst.append(char) # WARNING !
```

QUESTION: Do you see any problem?

Show answer<div class="jupman-sol jupman-sol-question" style="display:none">

ANSWER: if we go through the list and in *the meanwhile* we keep adding pieces, there is a concrete risk we will never terminate examining the list! Read carefully what follows:

</div>

X COMMANDMENT²⁰²: You shall never ever add nor remove elements from a sequence you are iterating with a `for`!

Falling into such temptations **would produce totally unpredictable behaviours** (do you know the expression *pulling the rug out from under your feet*?)

What about removing? We've seen that adding is dangerous, but so is removing. Suppose you have to eliminate all the elements from a list, you might be tempted to write something like this:

```
[35]: my_list = ['a', 'b', 'c', 'd', 'e']

for el in my_list:
    my_list.remove(el)      # VERY BAD IDEA
```

Have a close look at the code. Do you think we removed everything, uh?

```
[36]: my_list
[36]: ['b', 'd']
```

○○○ The absurd result is given by the internal implementation of Python, our version of Pyhton gives this result, yours might give a completely different one. **So be careful!**

If you really need to remove elements from a sequence you are iterating, use a `while` cycle²⁰³ or duplicate first a copy of the original sequence.

Exercise - duplicate

⊕ Try writing some code which MODIFIES a list `la` by duplicating the elements

- use a `for` cycle
- **DO NOT** use list multiplication

Example - given:

```
la = ['a', 'b', 'c']
```

after your code, it must result:

```
>>> la
['a', 'b', 'c', 'a', 'b', 'c']
```

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

```
[37]: la = ['a', 'b', 'c']

# write here
for element in list(la): # with list we create a *copy* of the original list, which
    ↪remains stable
    la.append(element)
print(la)
```

²⁰² <https://en.softpython.org/commandments.html#X-COMMANDMENT>

²⁰³ <https://en.softpython.org/while/while1-sol.html>

```
['a', 'b', 'c', 'a', 'b', 'c']
```

</div>

[37]:

```
la = ['a', 'b', 'c']

# write here
```

Exercise - hammers

⊕ Given a list of characters la, MODIFY the list by changing all the characters at even indeces with the character z

Example - given:

```
la = ['h', 'a', 'm', 'm', 'e', 'r', 's']
```

after your code, it must result:

```
>>> print(la)
['z', 'a', 'z', 'm', 'z', 'r', 'z']
```

- NOTE: here we *are not* adding nor removing cells from the list

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[38]:

```
la = ['h', 'a', 'm', 'm', 'e', 'r', 's']

# write here
for i in range(len(la)):
    if i % 2 == 0:
        la[i] = 'z'
print(la)
```

```
['z', 'a', 'z', 'm', 'z', 'r', 'z']
```

</div>

[38]:

```
la = ['h', 'a', 'm', 'm', 'e', 'r', 's']

# write here
```

Exercise - Orangutan

⊕⊕ Given two strings `sa` and `sb`, write some code which places in the string `sc` a string composed by alternating all the characters in `sa` and `sb`.

- if a string is shorter than the other one, at the end of `sc` put all the remaining characters from the other string.
- **HINT:** even if it is possible to augment a string a character at a time at each iteration, each time you do so a new string is created (because strings are immutable). So it's more efficient to keep augmenting a list, and then convert to string only at the very end.

Example - given:

```
sa, sb = 'gibbon', 'ORANGUTAN'
```

after your code it must result:

```
>>> print(sc)
gOiRbAbNoGnUTAN
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
  data-jupman-show="Show solution"
  data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

[39]:

```
sa, sb = 'gibbon', 'ORANGUTAN'      # gOiRbAbNoGnUTAN
#sa, sb = 'cruise ship', 'BOAT'    # cBrOuAiTse ship

# write here
temp = []

for i in range(len(sa)):
    temp.append(sa[i])
    if i < len(sb):
        temp.append(sb[i])

if i < len(sb):
    temp.extend(sb[i+1:])

sc = ''.join(temp)
print(sc)
```

```
gOiRbAbNoGnUTAN
```

```
</div>
```

[39]:

```
sa, sb = 'gibbon', 'ORANGUTAN'      # gOiRbAbNoGnUTAN
#sa, sb = 'cruise ship', 'BOAT'    # cBrOuAiTse ship

# write here
```

Exercise - basket

⊕⊕⊕ There is a basket full of fruits, which we represent as a list of strings. We want to take all the fruits and put them in a plate, in the same order we find them in the basket. We must take only the fruits contained in the set preferences.

- The basket may contain duplicates, if they are in the preferences you must take them all
- the fruits are to be taken **in the same order** in which they were found

Example - given:

```
basket = ['strawberry', 'melon', 'cherry', 'watermelon', 'apple', 'melon', 'watermelon'
          ↪, 'apple', ]
preferences = {'cherry', 'apple', 'strawberry'}
plate = []
```

after your code, it must result:

```
>>> print(basket)
['melon', 'watermelon', 'melon', 'watermelon']
>>> print(plate)
['strawberry', 'cherry', 'apple', 'apple']
```

You can solve the problem in two ways:

- Way 1 (simple and recommended): create a list new_basket and finally assign the variable basket to it
- Way 2 (hard, slow, not recommended but instructive): MODIFY the original basket list, using the **pop method**²⁰⁴ and without ever reassigning basket, so no rows beginning with basket =

Try solving the exercise in both ways.

Either way, always remember the sacred X COMMANDMENT²⁰⁵:

You shall never ever add nor remove elements from a sequence you are iterating with a for !

Show solution<div class="jupman-sol jupman-sol-code" style="display:none">

[40]:

```
# WAY 1

basket = ['strawberry', 'melon', 'cherry', 'watermelon', 'apple', 'melon', 'watermelon'
          ↪, 'apple', ]
preferences = {'cherry', 'apple', 'strawberry'}
plate = []

# write here
new_basket = []
for fruit in basket:
    if fruit in preferences:
        plate.append(fruit)
    else:
        new_basket.append(fruit)
```

(continues on next page)

²⁰⁴ <https://en.softpython.org/lists/lists3-sol.html#pop-method>

²⁰⁵ <https://en.softpython.org/commandments.html#X-COMMANDMENT>

(continued from previous page)

```
basket = new_basket # we substitute the original list
print('basket:', basket)
print('plate:', plate)

basket: ['melon', 'watermelon', 'melon', 'watermelon']
plate: ['strawberry', 'cherry', 'apple', 'apple']
```

```
</div>
```

```
[40]:
```

```
# WAY 1

basket = ['strawberry', 'melon', 'cherry', 'watermelon', 'apple', 'melon', 'watermelon',
          'apple', ]
preferences = {'cherry', 'apple', 'strawberry'}
plate = []

# write here
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
    data-jupman-show="Show solution"
    data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[41]:
```

```
# WAY 2

basket = ['strawberry', 'melon', 'cherry', 'watermelon', 'apple', 'melon', 'watermelon',
          'apple', ]
preferences = {'cherry', 'apple', 'strawberry'}
plate = []

# write here
copy = list(basket)
j = 0
# so we're sure to iterate on a different sequence from the one we're modifying
for i in range(len(copy)):
    fruit = copy[i]
    if fruit in preferences:
        plate.append(fruit)
        basket.pop(j)
    else:
        j += 1

print('basket:', basket)
print('plate:', plate)

basket: ['melon', 'watermelon', 'melon', 'watermelon']
plate: ['strawberry', 'cherry', 'apple', 'apple']
```

```
</div>
```

```
[41]:
```

```
# WAY 2
```

(continues on next page)

(continued from previous page)

```
basket = ['strawberry', 'melon', 'cherry', 'watermelon', 'apple', 'melon', 'watermelon
↪', 'apple', ]
preferences = {'cherry', 'apple', 'strawberry'}
plate = []

# write here
```

break and continue commands

We can use the commands `break` and `continue` to have even more control on loop execution.

NOTE: Please use sparingly!

When there is a lot of code in the cycle it's easy to 'forget' about their presence and introduce hard-to-discover bugs. On the other hand, in some selected cases these commands *may* increase code readability, so as everything use your judgement.

Terminate with `break`

To immediately exit a cycle you can use the `break` command:

```
[42]: for x in 'PARADE':
    if x == 'D':
        print('break, exits the loop!')
        break
        print('After the break')

    print(x)

print('Loop is over !')
```

P
A
R
A
break, exits the loop!
Loop is over !

Note how the instruction which prints 'After the break' was *not* executed

Jumping with `continue`

By calling `continue` execution is immediately brought to the next iteration , so we jump to the next element in the sequence without executing the instructions after the `continue`.

```
[43]: i = 1
for x in 'PARADE':
    if x == 'A':
        print("continue, jumps to next element")
        continue
    print(x)
print('Loop is over !')
```

```
P
continue, jumps to next element
R
continue, jumps to next element
D
E
Loop is over !
```

Combining `break` and `continue`

Let's see both in Python Tutor:

```
[44]: i = 1
for x in 'PARADE':
    if x == 'A':
        print("continue, jumps to next element")
        continue
    if x == 'D':
        print('break, exits loop!')
        break
    print(x)

print('Loop is over !')

jupman.pytut()
```

```
P
continue, jumps to next element
R
continue, jumps to next element
break, exits loop!
Loop is over !
```

```
[44]: <IPython.core.display.HTML object>
```

Questions - break and continue

Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

1. `for x in ['a', 'b', 'c']:
 print(x)
 break`

2. `for x in ['a', 'b', 'c']:
 print(x)
 break
 print('GLAM')`

3. `for x in ['a', 'b', 'c']:
 print(x)
 break
 break`

4. `for x in ['a', 'b', 'c']:
 break
 print(x)`

5. `break
for x in ['a', 'b', 'c']:
 print(x)`

6. `for x in ['a', 'b', 'c']:
 print(x)
 break`

7. `for x in ['a', 'b', 'c']:
 continue
 print(x)`

8. `for x in ['a', 'b', 'c']:
 print(x)
 continue`

9. `for x in ['a', 'b', 'c']:
 print(x)
 continue
 print('BAM')`

10. `continue
for x in ['a', 'b', 'c']:
 print(x)`

11. `for x in ['a', 'b', 'c']:
 print(x)
 continue`

12. `for x in ['a', 'b', 'c']:
 break`

(continues on next page)

(continued from previous page)

- ```
1/0
print('BAD KARMA')
```
13. `for x in ['a', 'b', 'c']:  
 1/0  
 break  
print('BAD KARMA')`
14. `for x in range(8):  
 if x < 4:  
 continue  
 print('ZAM', x)`
15. `for x in range(8):  
 if x >= 4:  
 break  
 print('ZUM', x)`
16. `for x in range(6):  
 if x % 2 == 0:  
 continue  
 print(x)`
17. `for x in ['M', 'C', 'M']:  
 print(x)  
 for y in ['S', 'P', 'Q', 'R']:  
 print(y)  
 break`
18. `for x in ['M', 'C', 'M']:  
 print(x)  
 break  
 for y in ['S', 'P', 'Q', 'R']:  
 print(y)`
19. `for x in ['M', 'C', 'M']:  
 print(x)  
 for y in ['S', 'P', 'Q', 'R']:  
 print(y)  
 continue`
20. `for x in ['M', 'C', 'M']:  
 print(x)  
 continue  
 for y in ['S', 'P', 'Q', 'R']:  
 print(y)`

## Exercise - autonomous walking

⊕ Write some code which given a string `phrase`, prints all the characters *except* the vocals.

Example - given:

```
phrase = 'autonomous walking'
```

prints:

```
t
n
m
s

w
l
k
n
g
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[45]:

```
phrase = 'autonomous walking'
#phrase='continuous'

write here
for x in phrase:
 if x in 'aeiou':
 continue
 else:
 print(x)
```

```
t
n
m
s

w
l
k
n
g
```

</div>

[45]:

```
phrase = 'autonomous walking'
#phrase='continuous'

write here
```

**Exercise - breaking bad**

⊕ Write some code which prints all the characters from string until it finds the string 'bad'.

Example - given:

```
string = 'cascapirillabadgnippobadzarpogno'
```

prints

```
c
a
s
c
a
p
i
r
i
l
l
a
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[46]:

```
string = 'cascapirillabadgnippobadzarpogno' # cascapiroilla
#string = 'sobad' # 'so'
#string = 'bad' # ''
#string = 'badso' # ''

write here
for i in range(len(string)):
 if string[i:i+3] == 'bad':
 break
 else:
 print(string[i])
```

```
c
a
s
c
a
p
i
r
i
l
l
a
```

</div>

[46]:

```
string = 'cascapirillabadgnippobadzarpogno' # cascapiroilla
#string = 'sobad' # 'so'
#string = 'bad' # ''
```

(continues on next page)

(continued from previous page)

```
#string = 'badso' # ''
write here
```

### Exercise - breaking point

⊗⊗ Given a phrase, prints all the words one per row *until* it finds a dot, and in that case it stops.

- **DO NOT** use `phrase.split('.')`. Splits on other characters are allowed.

Example - given:

```
phrase = 'At some point you must stop. Never go beyond the limit.'
```

prints:

```
At
some
point
you
must
stop
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[47]:

```
phrase = 'At some point you must stop. Never go beyond the limit.'
#phrase = "Respect the halt. Do you want to have us arrested?"
#phrase = 'Stop.'
#phrase = 'No stop'

write here

for word in phrase.split():
 if '.' in word:
 print(word[:-1])
 break
 else:
 print(word)
```

```
At
some
point
you
must
stop
```

</div>

[47]:

```
phrase = 'At some point you must stop. Never go beyond the limit.'
#phrase = "Respect the halt. Do you want to have us arrested?"
#phrase = 'Stop.'
```

(continues on next page)

(continued from previous page)

```
#phrase = 'No stop'

write here
```

### Exercise - breakdance

⊗⊗ As a skilled breakdancer, you're given `music` as a list of sounds. You will have to perform a couple of dances:

- during the first one, you will have to repeat the music sounds until you find exactly 3 sounds '`pa`', then you will shout `BREAKDANCE!`.
- during the second one, you will have to repeat the music sounds *in reverse* until you find exactly 3 sounds '`pa`', then you will shout `BREAKDANCE!`
- **DO NOT** modify `music`, so no `music.reverse()`

Example - given:

```
music = ['unz', 'pa', 'pa', 'tud', 'unz', 'pa', 'pa', 'tud', 'unz', 'boom', 'boom', 'tud']
```

Prints:

```
unz
pa
pa
tud
unz
pa
BREAKDANCE!

tud
boom
boom
unz
tud
pa
pa
unz
tud
pa
BREAKDANCE!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[48]: music = ['unz', 'pa', 'pa', 'tud', 'unz', 'pa', 'pa', 'tud', 'unz', 'boom', 'boom', 'tud']

write here

k = 0
for x in music:
 print(x)
 if x == 'pa':
 k += 1
```

(continues on next page)

(continued from previous page)

```

if k == 3:
 print('BREAKDANCE!')
 print()
 break

k = 0
for i in range(len(music)-1, -1, -1):
 print(music[i])
 if music[i] == 'pa':
 k += 1
 if k == 3:
 print('BREAKDANCE!')
 break

```

unz  
pa  
pa  
tud  
unz  
pa  
BREAKDANCE!

tud  
boom  
boom  
unz  
tud  
pa  
pa  
unz  
tud  
pa  
BREAKDANCE!

&lt;/div&gt;

```
[48]: music = ['unz','pa','pa','tud','unz','pa','pa','tud','unz','boom','boom','tud']
write here
```

unz  
pa  
pa  
tud  
unz  
pa  
BREAKDANCE!

tud  
boom  
boom  
unz  
tud  
pa  
pa  
unz

(continues on next page)

(continued from previous page)

```
tud
pa
BREAKDANCE !
```

### Continue

Go on with exercises on iterating strings<sup>206</sup>

## 6.2.2 For loops 2 - iterating strings

### Download exercises zip

Browse file online<sup>207</sup>

Let's see some exercise about strings.

### Exercise - Impertinence

Given the sequence of characters having a length multiple of 3, write some code which puts into variable `triplets` all the sub-sequences of three characters

Example - given:

```
sequence = "IMPERTINENCE"
IMPERTINENTE
```

after your code, it must result:

```
>>> print(triplets)
['IMP', 'ERT', 'INE', 'NCE']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2] :

```
sequence = "IMPERTINENCE" # ['IMP', 'ERT', 'INE', 'NCE']
#sequence = "CUTOUT" # ['CUT', 'OUT']
#sequence = "O_o" # ['O_o']

write here
triplets = []
for i in range(len(sequence)):
 if i % 3 == 0:
 triplets.append(sequence[i:i+3])
print(triplets)
```

```
['IMP', 'ERT', 'INE', 'NCE']
```

</div>

<sup>206</sup> <https://en.softpython.org/for/for2-strings-sol.html>

<sup>207</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/for>

[2]:

```
sequence = "IMPERTINENCE" # ['IMP', 'ERT', 'INE', 'NCE']
#sequence = "CUTOUT" # ['CUT', 'OUT']
#sequence = "O_o" # ['O_o']

write here
```

**Exercise - rosco**

⊗⊗ Given a string `word` and string `repetitions` containing only digits, write some code which puts in variable `result` a string containing all the characters of `word` repeated by the number of times reported in the corresponding position of `repetitions`.

Example - given:

```
word, repetitions = "rosco", "14323"
```

After your code it must result:

```
>>> result
'rooooosssccooo'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```
word, repetitions = "rosco", "14323" # 'rooooosssccooo'
word, repetitions = "chocolate", "144232312" # 'chhhhoooocccooollaatee'

write here
res = []

for i in range(len(word)):
 res.append(word[i]*int(repetitions[i]))

result = "".join(res)
print(result)

chhhhoooocccooollaatee
```

</div>

[3]:

```
word, repetitions = "rosco", "14323" # 'rooooosssccooo'
word, repetitions = "chocolate", "144232312" # 'chhhhoooocccooollaatee'

write here
```

### Continue

Go on with exercises about for loops with lists<sup>208</sup>

[ ]:

### 6.2.3 For loops 3 - iterating lists

#### Download exercises zip

Browse file online<sup>209</sup>

Let's see some exercise about lists.

#### Exercise - The contest

⊕ A list of participant has won a contest, and now we want to show on a display their rank. Write some code which MODIFIES the list by writing the rank of the participant next to the name.

Example - given:

```
partecipants = ['Marta', 'Peppo', 'Elisa', 'Gioele', 'Rosa']
```

After your code it must result:

```
>>> partecipants
['Marta-1', 'Peppo-2', 'Elisa-3', 'Gioele-4', 'Rosa-5']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]:

```
partecipants = ['Marta', 'Peppo', 'Elisa', 'Gioele', 'Rosa']
#partecipants = ['Gioele', 'Carmela', 'Rosario']

write here

for i in range(len(partecipants)):
 partecipants[i] = partecipants[i] + '-' + str(i+1)

partecipants
[2]: ['Marta-1', 'Peppo-2', 'Elisa-3', 'Gioele-4', 'Rosa-5']
```

```
</div>

[2]:
partecipants = ['Marta', 'Peppo', 'Elisa', 'Gioele', 'Rosa']
#partecipants = ['Gioele', 'Carmela', 'Rosario']

write here
```

---

<sup>208</sup> <https://en.softpython.org/for/for3-lists-sol.html>  
<sup>209</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/for>

### Exercise - babbà

⊕⊕ Write some code which given a character `search` to find and a phrase, produces a list with all the words containing that character.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution

[3]:

```
search = 's' # ['This', 'is', 'donuts,', 'croissant']
#search = 'f' # ['full', 'of', 'coffee']

phrase = "This city is full of donuts, croissant and coffee"

write here

res = []
for word in phrase.split():
 if search in word:
 res.append(word)
print(res)

['This', 'is', 'donuts,', 'croissant']
```

</div>

[3]:

```
search = 's' # ['This', 'is', 'donuts,', 'croissant']
#search = 'f' # ['full', 'of', 'coffee']

phrase = "This city is full of donuts, croissant and coffee"

write here
```

### Exercise - The Temple of Fortune

⊕⊕ While exploring a temple in the region of Uttar Pradesh, you found precious stones each one with a sacred number carved in it. You are tempted to take them all, but a threatening message looms over the stones, telling only the fools takes the numbers without first consult the Oracle.

To one side, you find the statue of a Buddha with crossed legs, which keeps a tray with some holes in sequence on his lap. Some hole is filled with a bean, others aren't.

Given a list `stones` of numbers and one `oracle` of booleans, write some code which MODIFIES the list `bag` by putting inside only the numbers of `stones` such that there is a `True` in a corresponding position of `oracle`.

- assume both the lists have exactly the same dimensions

Example - given:

[4]:

```
stones = [9, 7, 6, 8, 7]
oracle = [True, False, True, True, False]
```

After your code it must result:

```
>>> print(bag)
[9, 6, 8]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5]:

```
stones, oracle = [9, 7, 6, 8, 7], [True, False, True, True, False] # [9, 6, 8]
#stones, oracle = [3, 5, 2, 3, 4, 2, 4], [True, True, False, True, False, True, False] #_
↪ [3, 5, 3, 2]
bag = []

write here

for i in range(len(stones)):
 if oracle[i]:
 bag.append(stones[i])

print(bag)
```

```
[9, 6, 8]
```

</div>

[5]:

```
stones, oracle = [9, 7, 6, 8, 7], [True, False, True, True, False] # [9, 6, 8]
#stones, oracle = [3, 5, 2, 3, 4, 2, 4], [True, True, False, True, False, True, False] #_
↪ [3, 5, 3, 2]
bag = []

write here
```

### Exercise - the longest word

⊕⊕ Write some code which given a phrase, prints the **length** of the longest word.

- **NOTE:** we only want to know the length of the longest word, not the word itself!

Example - given:

```
phrase = "The hiker is climbing the brink of the mountain"
```

your code must print

```
8
```

which is the length of the most long word, in this case climbing and mountain in a tie.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```
phrase = "The hiker is climbing the brink of the mountain" # 8
#phrase = "The fearsome pirate Le Chuck ruled ruthlessly the South seas" # 10
#phrase = "Practically obvious" # 11
```

(continues on next page)

(continued from previous page)

```
write here

lengths = []

for word in phrase.split():
 lengths.append(len(word))
print(max(lengths))

8
```

&lt;/div&gt;

[6]:

```
phrase = "The hiker is climbing the brink of the mountain" # 8
#phrase = "The fearsome pirate Le Chuck ruled ruthlessly the South seas" # 10
#phrase = "Practically obvious" # 11

write here
```

## Exercise - desert

⊕⊕⊕ Write some code which given a string `trip` produces a list with all the words which *precede* the commas.

Example - given:

```
[7]: trip = "They crossed deserts, waded across rivers, clambered over the mountains, and
→finally arrived to the Temple"
```

your code must produce:

```
['deserts', 'rivers', 'mountains']
```

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol" jupman-sol-code" style="display:none">

[8]:

```
trip = "They crossed deserts, waded across rivers, clambered over the mountains, and
→finally arrived to the Temple"
['deserts', 'rivers', 'mountains']
#trip = "They walked with across the strees, the crowded markets, the alleys, the
→porches, until they found the cathedral."
['strees', 'markets', 'alleys', 'porches']
#trip = "The trip ended."
[]

write here
words = trip.split(',')

res = []

for phrase in words[:-1]:
 res.append(phrase.split()[-1])
res
```

```
[8]: ['deserts', 'rivers', 'mountains']
```

```
</div>
```

```
[8]: trip = "They crossed deserts, waded across rivers, clambered over the mountains, and
→finally arrived to the Temple"
['deserts', 'rivers', 'mountains']
#trip = "They walked with across the strees, the crowded markets, the alleys, the
→porches, until they found the cathedral."
['strees', 'markets', 'alleys', 'porches']
#trip = "The trip ended."
[]

write here
```

### Exercise - splash

⊕⊕⊕ Given a lst of odd length filled with zeros except the number in the middle, write some code which MODIFIES the list to write numbers which decrease according to the distance from the middle.

- the length of the list is always odd
- assume the list is always long enough to host a zero at each side
- a list of dimension 1 will only contain a zero

Example 1 - given:

```
lst = [0, 0, 0, 0, 4, 0, 0, 0, 0]
```

After your code, it must result:

```
>>> lst
[0, 1, 2, 3, 4, 3, 2, 1, 0]
```

Example 2 - given:

```
lst = [0, 0, 0, 3, 0, 0, 0]
```

after your code, it must result:

```
>>> lst
[0, 1, 2, 3, 2, 1, 0]
```

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol" jupman-sol-code" style="display:none">

```
[9]:
```

```
lst = [0, 0, 0, 0, 4, 0, 0, 0] # -> [0, 1, 2, 3, 4, 3, 2, 1, 0]
#lst = [0, 0, 0, 3, 0, 0, 0] # -> [0, 1, 2, 3, 2, 1, 0]
#lst = [0, 0, 2, 0, 0] # -> [0, 1, 2, 1, 0]
#lst = [0] # -> [0]

write here
```

(continues on next page)

(continued from previous page)

```
m = len(lst) // 2

for i in range(m):
 lst[m+i] = m - i

for i in range(m):
 lst[i] = i
lst

[9]: [0, 1, 2, 3, 4, 3, 2, 1, 0]
```

&lt;/div&gt;

[9]:

```
lst = [0, 0, 0, 0, 4, 0, 0, 0, 0] # -> [0, 1, 2, 3, 4, 3, 2, 1, 0]
#lst = [0, 0, 0, 3, 0, 0, 0] # -> [0, 1, 2, 3, 2, 1, 0]
#lst = [0, 0, 2, 0, 0] # -> [0, 1, 2, 1, 0]
#lst = [0] # -> [0]

write here
```

## Continue

Go on with exercises about [iterating tuples<sup>210</sup>](#)

### 6.2.4 For loops 4 - iterating tuples

#### [Download exercises zip](#)

[Browse file online<sup>211</sup>](#)

Let's see some exercise about tuples.

#### Exercise - double couples

⊕ Given a `lst` with `n` integer numbers, places in `res` a NEW list which contains `n` tuples having each two elements. Every tuple contains a number taken from the corresponding position of the initial list, and its double.

For example - given:

```
lst = [5, 3, 8]
```

After your code it must result:

```
>>> print(res)
[(5,10), (3,6), (8,16)]
```

<sup>210</sup> <https://en.softpython.org/for/for4-tuples-sol.html>

<sup>211</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/for>

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[2] :

```
lst = [5, 3, 8]
#lst = [2, 7] # [(2, 4), (7, 14)]

res = []

write here

for element in lst:
 res.append((element, element * 2))
print(res)

[(5, 10), (3, 6), (8, 16)]
```

</div>

[2] :

```
lst = [5, 3, 8]
#lst = [2, 7] # [(2, 4), (7, 14)]

res = []

write here
```

### Exercise - carpet

⊕⊕ Let's call a *tuple* a tuple with a couple of elements. Write some code which given a tuple  $t$ , produces a list having as elements *touples* each taken in alternation from  $t$ .

- if the input tuple  $t$  has an odd number of elements, the last tuple in the list to return will be made of only one element

Example 1 - given:

```
>>> t = ('c', 'a', 'r', 'p', 'e', 't') # even length
```

after your code it must result:

```
>>> print(res)
[('c', 'a'), ('r', 'p'), ('e', 't')]
```

Example 2 - given:

```
>>> t = ('s', 'p', 'i', 'd', 'e', 'r', 's') # odd length
```

After your code it must result:

```
>>> print(res)
[('s', 'p'), ('i', 'd'), ('e', 'r'), ('s',)]
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[3]: t = ('c', 'a', 'r', 'p', 'e', 't')
#t = ('s', 'p', 'i', 'd', 'e', 'r', 's')

write here
res = []
i = 0
for i in range(0, len(t)-1, 2):
 res.append((t[i], t[i+1]))
if len(t) % 2 == 1:
 res.append((t[-1],))
print(res)

[('c', 'a'), ('r', 'p'), ('e', 't')]
```

&lt;/div&gt;

```
[3]: t = ('c', 'a', 'r', 'p', 'e', 't')
#t = ('s', 'p', 'i', 'd', 'e', 'r', 's')

write here
```

## Continue

Go on with [iterating sets<sup>212</sup>](#)

```
[]:
```

### 6.2.5 For loops 5 - set iteration

[Download exercises zip](#)

[Browse file online<sup>213</sup>](#)

Given a set, we can examine the element sequence with a `for` cycle.

**WARNING:** sets iteration order is **not** predictable !

To better understand why, you can see again the tutorial on sets<sup>214</sup>

```
[2]: for word in {'this', 'is', 'a', 'set'}:
 print(word)

this
is
a
set
```

<sup>212</sup> <https://en.softpython.org/for/for5-sets-sol.html>

<sup>213</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/for>

<sup>214</sup> <https://en.softpython.org/sets/sets-sol.html#Creating-a-set>

```
[3]: s = set()
s.add('pan')
s.add('de')
s.add('mo')
s.add('nium')
print(s)

{'pan', 'mo', 'de', 'nium'}
```

### Questions - sets

Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

```
1. s = set()
s.add('pan')
s.add('de')
s.add('mo')
s.add('nium')
print(s)
```

```
2. for x in {'a',12,'34',56,34}[2:4]:
 print(x)
```

```
3. for x in set(['a']) | set(['b']):
 print(x)
```

```
4. for x in set(['a']) & set(['b']):
 print(x)
```

### Exercise - Screwed

The multinational ToxiCorp produces electrical appliances which are designed on purpose to break after a couple of years usage. When that happens, their components require very special tools only the corporation possess. Customers are then forced to go to repair workshops affiliated with ToxiCorp, and pay extra money. Over time the corporation has developed so many special shapes for screws that now its workshops have trouble managing all needed screwdrivers, so they ask you to devise a software to tell workshops which screwdrivers they are missing. You find it questionable, but they pay well, so you accept.

Each screw is star shaped, and is defined by a radius and a certain number of tips. We can represent it as a two elements list like [3, 7] where 3 is the radius and 7 the number of tips. Each screwdrivers is also defined as a two elements list with the values of the radius and tips it can screw.

A workshop has in store a list of screws and a list of screwdrivers: write some code that prints a sorted list of the screwdrivers which are missing in order to be able to handle all the screw types.

Example - given:

```
screws = [[5,8], [7,4], [2,9], [8,2], [7,4], [2,6], [8,3], [2,6], [8,3], [8,3], [5,8]]
screwdrivers = [[8,2], [1,3], [5,8], [2,5], [1,3]]
```

Your code must print:

```
Required screwdrivers: [(2, 6), (2, 9), (7, 4), (8, 3)]
```

- Notice input lists may have duplicates
- **DO NOT** use list methods or operators which search stuff
- so no `.index`, `.find`, `in` ... they're slow!
- **DO NOT** use nested loops... they would probably be slow !

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4]:

```
screws = [[5,8], [7,4], [2,9],[8,2], [7,4],[2,6], [8,3],
 [2,6], [8,3], [8,3], [5,8]]
screwdrivers = [[8,2], [1,3], [5,8], [2,5], [1,3]]
#Required screwdrivers: [(2, 6), (2, 9), (7, 4), (8, 3)]

#screws = [[7,2],[3,5],[1,9],[3,5]]
#screwdrivers = [[8,4],[3,5]]
#Required screwdrivers: [[1, 9], [7, 2]]

write here

screws_set = set()
screwdrivers_set = set()

for x,y in screws:
 screws_set.add((x,y))

for x,y in screwdrivers:
 screwdrivers_set.add((x,y))

temp = list(screws_set - screwdrivers_set)
temp.sort()
required = []
for x,y in temp:
 required.append([x,y])

print("Required screwdrivers:", required)
```

Required screwdrivers: [[2, 6], [2, 9], [7, 4], [8, 3]]

</div>

[4]:

```
screws = [[5,8], [7,4], [2,9],[8,2], [7,4],[2,6], [8,3],
 [2,6], [8,3], [8,3], [5,8]]
screwdrivers = [[8,2], [1,3], [5,8], [2,5], [1,3]]
#Required screwdrivers: [(2, 6), (2, 9), (7, 4), (8, 3)]

#screws = [[7,2],[3,5],[1,9],[3,5]]
#screwdrivers = [[8,4],[3,5]]
#Required screwdrivers: [[1, 9], [7, 2]]

write here
```

### Continue

Go on with `for` and dictionaries<sup>215</sup>

[ ]:

## 6.2.6 For loops 2 - iterating dictionaries

### Download exercises zip

Browse file online<sup>216</sup>

Given a dictionary, we can examine the sequence of its keys, values or both with a `for` cycle.

### Iterating keys

To iterate **only the keys** it is sufficient to use the `in` operator:

**WARNING:** keys iteration order is **not** predictable !

```
[2]: pastries = {
 'cream puff':5,
 'brioche':8,
 'donut':2
}
```

```
[3]: for key in pastries:
 print('Found key :', key)
 print(' with value:', pastries[key])

Found key : cream puff
 with value: 5
Found key : brioche
 with value: 8
Found key : donut
 with value: 2
```

At each iteration, the declared variable `key` is assigned to a key taken from the dictionary, in an order we cannot predict.

### Iterating key-value pairs

We can also directly obtain both the key and the associated value with this notation:

```
[4]: for key, value in pastries.items():
 print('Found key :', key)
 print(' with value:', pastries[key])
```

<sup>215</sup> <https://en.softpython.org/for/for6-dictionaries-sol.html>

<sup>216</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/for>

```
Found key : cream puff
with value: 5
Found key : brioche
with value: 8
Found key : donut
with value: 2
```

.items() return a list of key/value couples, and during each iteration a couple is assigned to the variable `key` and `value`.

## Iterating values

We can iterate the values calling the method `values()`

**WARNING:** values iteration order is also **not** predictable !

```
[5]: for value in pastries.values():
 print('Found value', value)

Found value 5
Found value 8
Found value 2
```

## Questions - iteration

Look at the following code fragments , and for each try guessing the result it produces (or if it gives an error):

**WARNING:** Remember the order is IMPOSSIBLE to foresee, so the important bit is to guess all the printed stuff

1. `for x in {'a':1,'b':2,'c':3}:
 print(x)`

2. `for x in {1:'a',2:'b',3:'c'}:
 print(x)`

3. `diz = {'a':1,'b':2,'c':3}
for x in diz:
 print(x[diz])`

4. `diz = {'a':1,'b':2,'c':3}
for x in diz:
 print(diz[x])`

5. `diz = {'a':1,'b':2,'c':3}
for x in diz:
 if x == 'b':
 print(diz[x])`

6. 

```
for k,v in {1:'a',2:'b',3:'c'}:
 print(k,v)
```
7. 

```
for x in {1:'a',2:'b',3:'c'}.values():
 print(x)
```
8. 

```
for x in {1:'a',2:'b',3:'c'}.keys():
 print(x)
```
9. 

```
for x in {1:'a',2:'b',3:'c'}.items():
 print(x)
```
10. 

```
for x,y in {1:'a',2:'b',3:'c'}.items():
 print(x,y)
```

### Questions - Are they equivalent?

Look at the following code fragments: each contains two parts, A and B. For each fragment, try guessing whether part A will print exactly the same result printed by code in part B

- FIRST think about the answer
- THEN try executing

### Are they equivalent ? postin

```
diz = {
 'p':'t',
 'o':'i',
 's':'n',
}

print('A:')
for x in diz.keys():
 print(x)

print('\nB:')
for y in diz:
 print(y)
```

### Are they equivalent ? corfel

```
diz = {
 'c':'t',
 'o':'e',
 'r':'l',
}

print('A:')
for p,q in diz.items():
 print(q)
```

(continues on next page)

(continued from previous page)

```
print ('\nB:')
for x in diz.values():
 print (x)
```

**Are they equivalent ? - gel**

```
diz = {
 'g':'l',
 'e':'e',
 'l':'g',
}

print ('A:')
for x in diz.values():
 print (x)

print ('\nB:')
for z in diz.items():
 print (z[0])
```

**Are they equivalent ? - giri**

```
diz = {
 'p':'g',
 'e':'i',
 'r':'r',
 'i':'i',
}

print ('A:')
for p,q in diz.items():
 if p == q:
 print (p)

print ('\nB:')
for x in diz:
 if x == diz[x]:
 print (x)
```

**Are they equivalent? - Found**

First think if they are equivalent, then check with all the proposed values of k.

**Be very careful about this exercise !**

Getting this means having *really* understood dictionaries ;-)

```
k = 'w'
#k = 'h'
#k = 'y'
#k = 'z'

dct = {
 'w':'s',
 'h':'o',
 'y':'?',
}

print('A:')
for x in dct:
 if x == k:
 print('Found', dct[x])

print('\nB:')
if k in dct:
 print('Found', dct[k])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** The two codes reported above are equivalent, with an important difference: code A will be executed in a time proportional to the dimension of `dct` (because it needs to go through all the dictionary), code B instead will be always executed in a short constant time which does *not* depend on the dimension of `dct`. Both the command `if k in dct`, and the expression `dct [k]` (which retrieves the value associated to key `k`) are extremely fast.

### WARNING: be sure to fully understand this point!

So many people write code as in part A, losing the main feature of dictionaries which is fast access. As long as data is small you may not notice, but when we have several megabytes of key/value couples you start feeling the time lost in pointless loops! For more you can read (or review) the section [Fast disorder](#)<sup>217</sup> in the dictionaries tutorial.

</div>

## Iteration exercises

### Exercise - color of hearts

⊕ Write some code which given a dictionary `suits`, for each suits prints its color.

Example - given:

```
suits = {
 'hearts':'red',
 'spades':'black',
 'diamonds':'red',
 'clubs':'black'
}
```

Prints:

<sup>217</sup> <https://en.softpython.org/dictionaries/dictionaries2-sol.html#Fast-disorder>

**WARNING:** do not care about the order in which values are printed!

On your computer you might see different results, the important bit is that all rows get printed.

```
The color of spades is black
The color of diamonds is red
The color of hearts is red
The color of clubs is black
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```
suits = {
 'hearts':'red',
 'spades':'black',
 'diamonds':'red',
 'clubs':'black'
}

write here

for k in suits.keys():
 print('The color of', k, 'is', suits[k])
```

```
The color of hearts is red
The color of spades is black
The color of diamonds is red
The color of clubs is black
```

</div>

[6]:

```
suits = {
 'hearts':'red',
 'spades':'black',
 'diamonds':'red',
 'clubs':'black'
}

write here
```

## Exercise - jewels

⊕ In the dictionary `jewels` some keys are equal to the respective values. Write some code which find such keys and prints them all.

Example - given:

```
jewels = {
 'rubies': 'jade',
 'opals': 'topazes',
 'gems': 'gems',
```

(continues on next page)

(continued from previous page)

```
'diamonds': 'gems',
'rubies': 'rubies'
}
```

prints:

```
couple of equal elements: gems and gems
couple of equal elements: rubies and rubies
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[7]:

```
jewels = {
 'rubies': 'jade',
 'opals': 'topazes',
 'gems': 'gems',
 'diamonds': 'gems',
 'rubies': 'rubies'
}

write here
for k,v in jewels.items():
 if k == v:
 print('couple of equal elements:',k, 'and', v)
```

```
couple of equal elements: rubies and rubies
couple of equal elements: gems and gems
```

</div>

[7]:

```
jewels = {
 'rubies': 'jade',
 'opals': 'topazes',
 'gems': 'gems',
 'diamonds': 'gems',
 'rubies': 'rubies'
}

write here
```

### Exercise - powers

⊕ Given a number n, write some code which creates a NEW dictionary d containing as keys the numbers from 1 a n INCLUDED, by associating keys to their squares.

Example - given:

```
n = 5
```

after your code, it must result:

```
>>> print(d)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8]:

```
n = 5

write here
d = {}
for i in range(1,n+1):
 d[i] = i*i

print(d)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

</div>

[8]:

```
n = 5

write here
```

## Exercise - flowers

⊕ Given a list `flowers`, write some code which creates a NEW dictionary `is_cap` which associates to each flower True if the flower name is written all uppercase, and False otherwise

- **HINT:** to verify whether a string is all uppercase, use `.isupper()` method

```
flowers = ['sunflower', 'GILLYFLOWER', 'tulip', 'PASSION FLOWER', 'ROSE', 'violet']
```

prints (they are in alphabetical order because we print with `pprint`):

```
>>> from pprint import pprint
>>> pprint(is_cap)
{'GILLYFLOWER': True,
 'PASSION FLOWER': True,
 'ROSE': True,
 'sunflower': False,
 'tulip': False,
 'violet': False}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[9]:

```
flowers = ['sunflower', 'GILLYFLOWER', 'tulip', 'PASSION FLOWER', 'ROSE', 'violet']

write here

is_cap = {}
```

(continues on next page)

(continued from previous page)

```
for el in flowers:
 is_cap[el] = el.isupper()

from pprint import pprint
pprint(is_cap)

{'GILLYFLOWER': True,
 'PASSION FLOWER': True,
 'ROSE': True,
 'sunflower': False,
 'tulip': False,
 'violet': False}
```

</div>

[9]:

```
flowers = ['sunflower', 'GILLYFLOWER', 'tulip', 'PASSION FLOWER', 'ROSE', 'violet']

write here
```

### Exercise - art

⊕ An artist painted a series of works with different techniques. In the dictionary `prices` he writes the price of each technique. The artist intend to promote a series of exhibitions, and in each of them he will present a particular technique. Supposing for each technique he produced `q` paintings, show how much he will earn in each exhibition (suppose he sells everything).

Example - given:

```
q = 20

exhibitions = ['watercolor', 'oil', 'mural', 'tempera', 'charcoal', 'ink']

prices = {'watercolor': 3000,
 'oil': 6000,
 'mural': 2000,
 'tempera': 4000,
 'charcoal': 7000,
 'ink': 1000
 }
```

Prints - **this time order matters!!**

```
Expected Income:
exhibition watercolor : 60000 €
exhibition oil : 120000 €
exhibition mural : 40000 €
exhibition tempera : 80000 €
exhibition charcoal : 140000 €
exhibition ink : 20000 €
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[10]:

```
q = 20

exhibitions = ['watercolor', 'oil', 'mural', 'tempera', 'charcoal', 'ink']

prices = {'watercolor': 3000,
 'oil' : 6000,
 'mural' : 2000,
 'tempera' : 4000,
 'charcoal' : 7000,
 'ink' : 1000
 }

write here

print('Expected Income:')
for i in range(len(exhibitions)):
 technique = exhibitions[i]
 print(' exhibition', technique, ":", prices[technique]*q, '€')

Expected Income:
exhibition watercolor : 60000 €
exhibition oil : 120000 €
exhibition mural : 40000 €
exhibition tempera : 80000 €
exhibition charcoal : 140000 €
exhibition ink : 20000 €
```

&lt;/div&gt;

[10]:

```
q = 20

exhibitions = ['watercolor', 'oil', 'mural', 'tempera', 'charcoal', 'ink']

prices = {'watercolor': 3000,
 'oil' : 6000,
 'mural' : 2000,
 'tempera' : 4000,
 'charcoal' : 7000,
 'ink' : 1000
 }

write here
```

### Exercise - stationery stores

⊕ An owner of two stationery shops, in order to reorganize the stores wants to know the materials which are in common among the shops. Given two dictionaries `store1` and `store2` which associates objects to their quantity, write some code which finds all the keys in common and for each prints the sum of the found quantities.

Example - given:

```
store1 = {'pens':10,
 'folders':20,
 'papers':30,
 'scissors':40}

store2 = {'pens':80,
 'folders':90,
 'goniometer':130,
 'scissors':110,
 'rulers':120,
 }
```

prints (order is **not** important):

```
materials in common:
 pens : 90
 folders : 110
 scissors : 150
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[11]:

```
store1 = {'pens':10,
 'folders':20,
 'papers':30,
 'scissors':40}

store2 = {'pens':80,
 'folders':90,
 'goniometer':130,
 'scissors':110,
 'rulers':120,
 }

write here

print('materials in common:')
for k in store1:
 if k in store2:
 print(' ',k, ':', store1[k] + store2[k])
```

```
materials in common:
 pens : 90
 folders : 110
 scissors : 150
```

</div>

[11]:

```
store1 = {'pens':10,
```

(continues on next page)

(continued from previous page)

```

'folders':20,
'papers':30,
'scissors':40}

store2 = {'pens':80,
 'folders':90,
 'goniometer':130,
 'scissors':110,
 'rulers':120,
 }
write here

```

## Exercise - legumes

⊕ A store has numbered shelves, each containing a number of legumes expressed in kilograms. We represent `store` as a list. There is also a `registry` available as a dictionary which associates to legume names the shelves number in which they are contained.

Write some code which given a list of legume names, shows the sum of kilograms in the store for those legumes.

Example - given:

```

legumes = ['lentils', 'soy']

0 1 2 3 4 5
store = [50,90,70,10,20,50]

registry = {'peas':3,
 'soy':1,
 'chickpeas':5,
 'lentils':4,
 'broad beans':2,
 'beans':0,
}

```

after your code, it must print (order does **not** matter):

```

Searching for lentils and soy ...
Found 20 kg of lentils
Found 90 kg of soy
Total: 110 kg

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[12]:

```

legumes = ['lentils', 'soy'] # 110
#legumes = ['beans', 'broad beans', 'chickpeas'] # 170

0 1 2 3 4 5
store = [50,90,70,10,20,50]

registry = {'peas':3,

```

(continues on next page)

(continued from previous page)

```

'soy':1,
'chickpeas':5,
'lentils':4,
'broad beans':2,
'beans':0,
}
write here
print('Searching for', ' and '.join(legumes), '...')
s = 0
for leg in legumes:
 print('Found', store[registry[leg]], 'kg of', leg)
 s += store[registry[leg]]

print('Total:', s, 'kg')

Searching for lentils and soy ...
Found 20 kg of lentils
Found 90 kg of soy
Total: 110 kg

```

&lt;/div&gt;

[12]:

```

legumes = ['lentils', 'soy'] # 110
#legumes = ['beans', 'broad beans', 'chickpeas'] # 170

0 1 2 3 4 5
store = [50,90,70,10,20,50]

registry = {'peas':3,
 'soy':1,
 'chickpeas':5,
 'lentils':4,
 'broad beans':2,
 'beans':0,
}
write here

```

### Exercise - smog

⊕ Write some code which given two dictionaries `smog` and `prepositions` which associate places to respectively values of smog and prepositions, prints all the places telling the smog is excessive if the value is greater than 30, otherwise is tolerable.

- **NOTE:** when printing the first preposition character must be capital: to transform the string you can use the method `.capitalize()`

Example - given:

```

smog = {'streets' : 40,
 'cities' : 20,
 'intersections' : 90,
 'trains' : 15,
 'lakes' : 5

```

(continues on next page)

(continued from previous page)

```

 }

propositions = {
 'streets' : 'on',
 'cities' : 'in',
 'lakes' : 'at',
 'trains' : 'on',
 'intersections': 'at',
}

```

prints (order **does not** matter):

```

On streets the smog level is excessive
In cities the smog level is tolerable
At intersections the smog level is excessive
On trains the smog level is tolerable
At lakes the smog level is tolerable

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[13]:

```

smog = { 'streets' : 40,
 'cities' : 20,
 'intersections': 90,
 'trains' : 15,
 'lakes' : 5
 }

propositions = {
 'streets' : 'on',
 'cities' : 'in',
 'lakes' : 'at',
 'trains' : 'on',
 'intersections': 'at',
}

write here
for x in smog:
 if smog[x] > 30:
 print(propositions[x].capitalize(), x, "the smog level is excessive")
 else:
 print(propositions[x].capitalize(), x, "the smog level is tolerable")

```

```

On streets the smog level is excessive
In cities the smog level is tolerable
At intersections the smog level is excessive
On trains the smog level is tolerable
At lakes the smog level is tolerable

```

</div>

[13]:

```

smog = { 'streets' : 40,
 'cities' : 20,
 'intersections': 90,
 'trains' : 15,
}

```

(continues on next page)

(continued from previous page)

```

 'lakes' : 5
}

prepositions = {
 'streets' : 'on',
 'cities' : 'in',
 'lakes' : 'at',
 'trains' : 'on',
 'intersections' : 'at',
}
write here

```

## Exercise - sports

⊕⊕ Write some code which given a dictionary `sports` in which people are associated to the favourite sport, create a NEW dictionary `counts` in which associates each sport to the number of people that prefer it.

Example - given:

```

sports = {
 'Gianni':'soccer',
 'Paolo':'tennis',
 'Sara':'volleyball',
 'Elena':'tennis',
 'Roberto':'soccer',
 'Carla':'soccer',
}

```

After your code, it must result:

```

>>> print(counts)
{'tennis': 2, 'soccer': 3, 'volleyball': 1}

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```

[14]: sports = {
 'Gianni':'soccer',
 'Paolo':'tennis',
 'Sara':'volleyball',
 'Elena':'tennis',
 'Roberto':'soccer',
 'Carla':'soccer',
}

write here

counts = {}

for k,v in sports.items():
 if v in counts:
 counts[v] += 1

```

(continues on next page)

(continued from previous page)

```

else:
 counts[v] = 1

print(counts)
{'soccer': 3, 'tennis': 2, 'volleyball': 1}

```

&lt;/div&gt;

```
[14]: sports = {
 'Gianni':'soccer',
 'Paolo':'tennis',
 'Sara':'volleyball',
 'Elena':'tennis',
 'Roberto':'soccer',
 'Carla':'soccer',
}

write here

{'soccer': 3, 'tennis': 2, 'volleyball': 1}
```

## Exercise - green lizard

⊕⊕ Write some code which given a set search of characters to find, counts for each how many are present in the string text and places the number in the dictionary counts.

Example - given:

```
[15]: search = {'i','t','r'}
text = "A diurnal lizard of green and brown color A pattern may also be present in
↪the form of dark slate grey streaks or spots. When found with a brown coloration,
↪sometimes with lighter stripe down the back."
counts = {}
```

After your code it must result:

```
>>> print(counts)
{'r': 5, 'i': 2, 't': 0}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[16]: #jupman-ignore-output
search = {'i','t','r'}
text = "A diurnal lizard of green and brown color."
counts = {}

write here

solution 1, most efficient
for char in search:
 counts[char] = 0
for char in text:
```

(continues on next page)

(continued from previous page)

```
if char in search:
 counts[char] += 1

print(counts)

solution 2, less efficient (scans text n times with count)
for char in search:
 counts[char] = text.count(char)

print(counts)
{'i': 2, 't': 0, 'r': 5}
{'i': 2, 't': 0, 'r': 5}
```

&lt;/div&gt;

```
[16]: #jupman-ignore-output
search = {'i','t','r'}
text = "A diurnal lizard of green and brown color."
counts = {}

write here

{'i': 2, 't': 0, 'r': 5}
{'i': 2, 't': 0, 'r': 5}
```

## Modifying a dictionary during iteration

Suppose you have a dictionary of provinces:

```
provinces = {
 'tn': 'Trento',
 'mi':'Milano',
 'na':'Napoli',
}
```

and you want to MODIFY it so that after your code the acronyms are added as capitalized:

```
>>> print(provinces)
{'tn': 'Trento',
 'mi':'Milano',
 'na':'Napoli',
 'TN': 'Trento',
 'MI':'Milano',
 'NA':'Napoli',
}
```

You might think to write something like this:

```
for key in provinces:
 provinces[key.upper()] = provinces[key] # WARNING !
```

**QUESTION:** Do you see any problem?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); " data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** if you go through a dictionary and in *the meanwhile* you keep adding pieces, there is a concrete risk we will never terminate examining the keys !

So carefully read what follows:

</div>

---

**X COMMANDMENT<sup>218</sup>:** You shall never ever add nor remove elements from a dictionary you are iterating with a for !

---

In this case, if we try executing the code, we will get an explicit error:

```

RuntimeError Traceback (most recent call last)
<ipython-input-26-9b20900057e8> in <module>()
----> 1 for key in provinces:
 2 provinces[key.upper()] = provinces[key] # WARNING !
RuntimeError: dictionary changed size during iteration
```

but in other cases (like for example lists) modifying stuff **may produce totally unpredictable behaviours** (do you know the expression *pulling the rug out from under your feet ?* )

**What about removing?** We've seen adding is dangerous, but so is removing.

Suppose we want to remove any couple having as value 'Trento'

```
provinces = {
 'tn': 'Trento',
 'mi':'Milano',
 'na':'Napoli',
}
```

to obtain:

```
>>> print(provinces)
{'mi':'Milano',
 'na':'Napoli'}
```

If we try executing something like this Python notices and raises an exception:

```
provinces = {
 'tn': 'Trento',
 'mi':'Milano',
 'na':'Napoli',
}

for key in provinces:
 if provinces[key] == 'Trento':
 del provinces[key] # VERY BAD IDEA
```

```

RuntimeError Traceback (most recent call last)
```

(continues on next page)

<sup>218</sup> <https://en.softpython.org/commandments.html#X-COMMANDMENT>

(continued from previous page)

```
<ipython-input-23-5df0fd659120> in <module>()
 5 'na':'Napoli'
 6 }
----> 7 for key in provinces:
 8 if provinces[key] == 'Trento':
 9 del provinces[key] # VERY BAD IDEA
```

```
RuntimeError: dictionary changed size during iteration
```

If you really need to remove elements from the sequence in which you are iterating, use a while cycle<sup>219</sup> or first copy the original sequence.

### Exercise - zazb

⊕⊕ Write some code which given a dictionary chars with characters as keys, MODIFY the dictionary so to add keys like the existing ones prefixed with character 'z' - new keys should be associated with the constant integer 10

Example - given:

```
chars = {
 'a':3,
 'b':8,
 'c':4
}
```

after your code, chars should result MODIFIED like this:

```
>>> chars
{
 'a':3,
 'b':8,
 'c':4,
 'za':10,
 'zb':10,
 'zc':10
}
```

**QUESTION:** Is it desirable to write a solution like the following one? Read carefully!

```
chars = {
 'a':3,
 'b':8,
 'c':4
}

for key in chars:
 chars['z'+key] = 10 # WARNING !! TROUBLE AHEAD !!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); " data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** Absolutely not - in this case we're lucky and we will obtain an explicit error, in other cases we might obtain infinite loops or incomprehensible results:

<sup>219</sup> <https://en.softpython.org/while/while1-sol.html>

```

RuntimeError Traceback (most recent call last)
<ipython-input-36-550c4c302120> in <module>()
 5 }
 6
----> 7 for key in chars:
 8 chars['z'+key] = 10

RuntimeError: dictionary changed size during iteration
```

**Do something better:** try now rewriting a version of the program without this bug.

</div>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[17]:

```
chars = {
 'a':3,
 'b':8,
 'c':4
}

write here

for el in list(chars.keys()): # list 'takes a picture' of the current keys state
 chars['z'+el] = 10

chars
```

[17]: {'a': 3, 'b': 8, 'c': 4, 'za': 10, 'zb': 10, 'zc': 10}

</div>

[17]:

```
chars = {
 'a':3,
 'b':8,
 'c':4
}

write here
```

## Exercise - DIY

⊗⊗ A depot for do-it-yourself hobbists has a catalog which associates object types to the shelves where to put them. Each day, a list of entries is populated with the newly arrived object types. Such types are placed in the depot, a dictionary which associates to each shelf the object type pointed by the catalog. Write some code which given the list entries and catalog, populates the dictionary depot

Example - given:

```
entries = ['chairs', 'lamps', 'cables']
```

(continues on next page)

(continued from previous page)

```
catalog = {'stoves' : 'A',
 'chairs' : 'B',
 'carafes' : 'D',
 'lamps' : 'C',
 'cables' : 'F',
 'gardening' : 'E'}

depot = {}
```

after your code, it must result:

```
>>> print(depot)
{'B': 'chairs', 'C': 'lamps', 'F': 'cables'}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[18]:

```
entries = ['chairs', 'lamps', 'cables'] # depot becomes: {'B': 'chairs', 'C': 'lamps'
entries = ['carafes', 'gardening'] # depot becomes: {'D': 'carafes', 'E':
entries = ['stoves'] # depot becomes: {'A': 'stoves'}
```

```
catalog = {'stoves' : 'A',
 'chairs' : 'B',
 'carafes' : 'D',
 'lamps' : 'C',
 'cables' : 'F',
 'gardening' : 'E'}
```

```
depot = {}

write here
```

```
depot = {}
```

```
for shipment in entries:
 depot[catalog[shipment]] = shipment
```

```
depot
```

[18]:

```
{'B': 'chairs', 'C': 'lamps', 'F': 'cables'}
```

```
</div>
```

[18]:

```
entries = ['chairs', 'lamps', 'cables'] # depot becomes: {'B': 'chairs', 'C': 'lamps'
entries = ['carafes', 'gardening'] # depot becomes: {'D': 'carafes', 'E':
entries = ['stoves'] # depot becomes: {'A': 'stoves'}
```

```
catalog = {'stoves' : 'A',
 'chairs' : 'B',
 'carafes' : 'D',
 'lamps' : 'C',
```

(continues on next page)

(continued from previous page)

```

 'cables' : 'F',
 'gardening' : 'E'}

depot = {}

write here

```

**Exercise - mine**

⊕⊕ Given a dictionary `mine` which associates keys to numbers, MODIFY the dictionary `extracted` associating the same keys of `mine` to lists with keys repeated the given number of times

Example - given:

```

mine = {'brass': 5,
 'iron' : 8,
 'copper' : 1}
extracted = {}

```

after your code it must result:

```

>>> print(extracted)
{'brass': ['brass', 'brass', 'brass', 'brass', 'brass'],
 'iron': ['iron', 'iron', 'iron', 'iron', 'iron', 'iron', 'iron', 'iron'],
 'copper': ['copper']}

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[19]:

```

mine = {'brass': 5,
 'iron' : 8,
 'copper' : 1}
extracted = {}

write here

```

```

for key in mine:
 extracted[key] = [key] * mine[key]

```

```
extracted
```

[19]:

```

{'brass': ['brass', 'brass', 'brass', 'brass', 'brass'],
 'iron': ['iron', 'iron', 'iron', 'iron', 'iron', 'iron', 'iron', 'iron'],
 'copper': ['copper']}

```

```
</div>
```

[19]:

```

mine = {'brass': 5,
 'iron' : 8,
 'copper' : 1}
extracted = {}

```

(continues on next page)

(continued from previous page)

```
write here
```

## Continue

Go on with nested for loops<sup>220</sup>

### 6.2.7 For loops 7 - nested loops

#### Download exercises zip

Browse file online<sup>221</sup>

It's possible to include a `for` cycle inside another one, for example we could visit all the words of a list of strings and for each word we could print all its characters:

```
[2]: lst = ["some",
 "light",
 "ahead"]

for string in lst:
 for char in string:
 print(char)
 print()
```

```
s
o
m
e

l
i
g
h
t

a
h
e
a
d
```

---

<sup>220</sup> <https://en.softpython.org/for/for7-nested-sol.html>  
<sup>221</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/for>

## Nested for

What we said previously about variable names is even more important with nested loops:

---

### II COMMANDMENT<sup>222</sup> Whenever you insert a variable in a `for` cycle, such variable must be new

---

If you defined a variable in an external `for`, you shall not reintroduce it in an internal `for`, because this would bring a lot of confusion. For example here `s` is introduced both in the external and in the internal loop:

```
[3]: for s in ['volleyball', 'tennis', 'soccer', 'swimming']:
 for s in range(3): # debugging hell, you lose the external cycle s
 print(s)

 print(s) # prints 2 instead of a sport!
```

```
0
1
2
2
0
1
2
2
0
1
2
2
0
1
2
2
```

## Questions - nested for

Look at the following code fragments , and for each try guessing the result it produces (or if it gives an error):

1. `for y in for x in range(3):
 print(x,y)`

2. `for y in for x in range(2) in range(3):
 print(x,y)`

3. `for y in range(3):
 for x in range(2):
 print(x,y)`

4. `for x in range(2):
 for x in range(3):
 print(x)
 print(x)`

---

<sup>222</sup> <https://en.softpython.org/commandments.html#II-COMMANDMENT>

```
5. for x in range(2):
 for y in range(3):
 print(x,y)
 print(x,y)
```

```
6. for x in range(1):
 for y in range(1):
 print(x,y)
```

```
7. for x in range(2):
 for y in range(3):
 print(x,y)
```

```
8. la = 'abc'
for x in la:
 for y in la:
 print(x)
```

```
9. for x in 'ab':
 for y in 'cd':
 print(x,y)
 for y in 'ef':
 print(x,y)
```

```
10. for x in 'abc':
 for y in 'abc':
 if x == y:
 print(x)
```

```
11. for x in 'abc':
 for y in 'abc':
 if x != y:
 print(x,y)
```

```
12. lst = []
for x in 'a':
 for y in 'bc':
 lst.append(x)
 lst.append(y)
print(lst)
```

```
13. lst = []
for x in 'abc':
 for y in 'de':
 lst.append('z')
print(len(lst))
```

```
14. c = 1
for x in range(1,4):
 s = ''
 for y in range(1,4):
 s = s + str(c)
 c += 1
 print(s)
```

## Exercise - casting

⊕ A new USA-Japanese videocultural production is going to be launched, so actors are called for casting. The director wants to try a scene with all the possible couples which can be formed among actors and actresses. Write some code which prints all the couples, also putting introduction messages.

- **NOTE:** the number of actors and actresses may be different

Example - given:

```
actresses = ['Leela', 'Wilma']
actors = ['Captain Harlock', 'Lupin', 'Kenshiro']
```

prints:

```
Leela enters the scene!
Captain Harlock enters the scene!
 Leela and Captain Harlock get ready ... ACTION!
 Thanks Captain Harlock - next one !
Lupin enters the scene!
 Leela and Lupin get ready ... ACTION!
 Thanks Lupin - next one !
Kenshiro enters the scene!
 Leela and Kenshiro get ready ... ACTION!
 Thanks Kenshiro - next one !
Thanks Leela - next one !
Wilma enters the scene!
 Captain Harlock enters the scene!
 Wilma and Captain Harlock get ready ... ACTION!
 Thanks Captain Harlock - next one !
 Lupin enters the scene!
 Wilma and Lupin get ready ... ACTION!
 Thanks Lupin - next one !
 Kenshiro enters the scene!
 Wilma and Kenshiro get ready ... ACTION!
 Thanks Kenshiro - next one !
Thanks Wilma - next one !

Casting is over for today!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4]:

```
actresses = ['Leela', 'Wilma']
actors = ['Captain Harlock', 'Lupin', 'Kenshiro']

write here
for actress in actresses:
 print(actress, 'enters the scene!')
 for actor in actors:
 print(' ', actor, 'enters the scene!')

 print(' ', actress, 'and', actor, 'get ready ... ACTION!')
 print(' Thanks', actor, '- next one !')
 print('Thanks', actress, '- next one !')
print()
print('Casting is over for today!')
```

```
Leela enters the scene!
Captain Harlock enters the scene!
 Leela and Captain Harlock get ready ... ACTION!
 Thanks Captain Harlock - next one !
Lupin enters the scene!
 Leela and Lupin get ready ... ACTION!
 Thanks Lupin - next one !
Kenshiro enters the scene!
 Leela and Kenshiro get ready ... ACTION!
 Thanks Kenshiro - next one !
Thanks Leela - next one !
Wilma enters the scene!
Captain Harlock enters the scene!
 Wilma and Captain Harlock get ready ... ACTION!
 Thanks Captain Harlock - next one !
Lupin enters the scene!
 Wilma and Lupin get ready ... ACTION!
 Thanks Lupin - next one !
Kenshiro enters the scene!
 Wilma and Kenshiro get ready ... ACTION!
 Thanks Kenshiro - next one !
Thanks Wilma - next one !

Casting is over for today!
```

</div>

[4]:

```
actresses = ['Leela', 'Wilma']
actors = ['Captain Harlock', 'Lupin', 'Kenshiro']

write here
```

### Exercise - cover the plane

⊕ Given the integers  $a$  and  $b$ , write some code which prints all the possible couples of numbers  $x$  and  $y$  such that  $1 \leq x \leq a$  and  $1 \leq y \leq b$

For example, given:

```
a, b = 5, 3
```

it must print:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
4 1
```

(continues on next page)

(continued from previous page)

```
4 2
4 3
5 1
5 2
5 3
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5]:

```
a,b = 5, 3

write here
for x in range(1,a+1):
 for y in range(1,b+1):
 print(x,y)
```

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
4 1
4 2
4 3
5 1
5 2
5 3
```

&lt;/div&gt;

[5]:

```
a,b = 5, 3

write here
```

### Exercise - triangular

- ⊕ Given the integer  $a$ , write some code which prints all the possible couples of numbers  $x$  and  $y$  such that  $0 \leq x \leq y < a$ .  
For example, for

```
a = 5
```

it must print:

```
0 0
0 1
0 2
0 3
```

(continues on next page)

(continued from previous page)

```
0 4
1 1
1 2
1 3
1 4
2 2
2 3
2 4
3 3
3 4
4 4
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```
a = 5
write here
for x in range(a):
 for y in range(x,a):
 print(x,y)
```

```
0 0
0 1
0 2
0 3
0 4
1 1
1 2
1 3
1 4
2 2
2 3
2 4
3 3
3 4
4 4
```

</div>

[6]:

```
a = 5
write here
```

## Exercise - port

⊕ Write some code which given a list `words` and a list `characters`, for each word calculates how many characters it contains

- **ONLY** count the characters present in `characters`
- **ONLY** print the result if the number is greater than zero

Example - given:

```
words = ['ships', 'pier', 'oar', 'fish trap', 'sails', 'trawling net']
characters = ['n', 'i', 's']
```

prints:

```
ships contains 1 i
ships contains 2 s
pier contains 1 i
fish trap contains 1 i
fish trap contains 1 s
sails contains 1 i
sails contains 2 s
trawling net contains 2 n
trawling net contains 1 i
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[7]:

```
words = ['ships', 'pier', 'oar', 'fish trap', 'sails', 'trawling net']
characters = ['n', 'i', 's']
```

# write here

```
for x in words:
 for y in characters:
 if y in x:
 print(x, 'contains', x.count(y), y)
```

```
ships contains 1 i
ships contains 2 s
pier contains 1 i
fish trap contains 1 i
fish trap contains 1 s
sails contains 1 i
sails contains 2 s
trawling net contains 2 n
trawling net contains 1 i
```

</div>

[7]:

```
words = ['ships', 'pier', 'oar', 'fish trap', 'sails', 'trawling net']
characters = ['n', 'i', 's']
```

# write here

(continues on next page)

(continued from previous page)

## Exercise - polygons

⊕⊕ Given a list `polygons` with polygon names ordered by sides number starting from a triangle, write some code which prints all the possible questions we can form regarding the number of sides. Start from a minimum of 3 sides until a maximum corresponding to the number of sides of the last polygon (remember names are ordered by number of sides!)

Example - given:

```
0 1 2 3
polygons = ["triangle", "square", "pentagon", "hexagon"]
```

prints:

```
Does the triangle have 3 sides? True
Does the triangle have 4 sides? False
Does the triangle have 5 sides? False
Does the triangle have 6 sides? False
Does the square have 3 sides? False
Does the square have 4 sides? True
Does the square have 5 sides? False
Does the square have 6 sides? False
Does the pentagon have 3 sides? False
Does the pentagon have 4 sides? False
Does the pentagon have 5 sides? True
Does the pentagon have 6 sides? False
Does the hexagon have 3 sides? False
Does the hexagon have 4 sides? False
Does the hexagon have 5 sides? False
Does the hexagon have 6 sides? True
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[8]:

```
0 1 2 3
polygons = ["triangle", "square", "pentagon", "hexagon"]

write here
for i in range(len(polygons)):
 for j in range(len(polygons)):
 print('Does the', polygons[i], 'have', j+3, 'sides?', i+3 == j+3)
```

```
Does the triangle have 3 sides? True
Does the triangle have 4 sides? False
Does the triangle have 5 sides? False
Does the triangle have 6 sides? False
Does the square have 3 sides? False
Does the square have 4 sides? True
Does the square have 5 sides? False
Does the square have 6 sides? False
Does the pentagon have 3 sides? False
Does the pentagon have 4 sides? False
```

(continues on next page)

(continued from previous page)

```
Does the pentagon have 5 sides? True
Does the pentagon have 6 sides? False
Does the hexagon have 3 sides? False
Does the hexagon have 4 sides? False
Does the hexagon have 5 sides? False
Does the hexagon have 6 sides? True
```

&lt;/div&gt;

[8]:

```
0 1 2 3
polygons = ["triangle", "square", "pentagon", "hexagon"]

write here
```

### Exercise - bon jour

⊕⊕⊕ Given two strings `sa` and `sb` in lowercase, write some code which prints single letters from `sa` as upper case, followed by all possible combinations of `sb` where ONLY ONE character is uppercase.

Example - given:

```
sa = 'bon'
sb = 'jour'
```

Must print:

```
B Jour
B jOur
B joUr
B jouR
O Jour
O jOur
O joUr
O jouR
N Jour
N jOur
N joUr
N jouR
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[9]:

```
sa = 'bon'
sb = 'jour'

write here

for c1 in sa:
 for i in range(len(sb)):
 print(c1.upper() + ' ' + sb[:i] + sb[i].upper() + sb[i+1:])
```

```
B Jour
B jOur
B joUr
B jouR
O Jour
O jOur
O joUr
O jouR
N Jour
N jOur
N joUr
N jouR
```

```
</div>
```

```
[9]:
```

```
sa = 'bon'
sb = 'jour'

write here
```

### Continue

Go on with [for challenges](#)<sup>223</sup>

```
[]:
```

## 6.2.8 for loops 8 - Challenge

### Download exercises zip

[Browse file online](#)<sup>224</sup>

We now propose some exercises without solution, do you accept the challenge?

### Challenge - Faceborg

The social network Faceborg wants to assimilate your soul by ingesting your personal data. Every time you like or share a post, Faceborg knows it. Every time you see a like on a website, Faceborg knows you've been there, even without clicking the icon.

Faceborg already assimilated billions of people personal data, either because users explicitly inserted it into the system by completing their profiles, or because it was inferred by their online behaviours.

Faceborg wants even more, and sends you an automated email asking to improve its clustering algorithm to generate more revenue from advertisers. You firmly deny, to no avail: Faceborg puts a subliminal order into a flashing ad, and takes complete control of your mind.

**Data model:** A person in Faceborg is modelled as a dictionary holding fields of sensitive personal information. Field values are normalized in the range `-1.0` to the opposite `1.0`, for example a person might be represented as this:

---

<sup>223</sup> <https://en.softpython.org/for/for8-chal.html>

<sup>224</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/for>

```
person = {
 'income' : 0.9,
 'health' : 0.5,
 'religiosity' : -0.4,
 'politics' : -0.1
}
```

In order to perform fast clustering, Faceborg creates summaries of people dividing each field values into five categories:

```
-1.0----1.0
12345

1: < -0.6
2: -0.6 <= x < -0.2
3: -0.2 <= x < 0.2
4: 0.2 <= x < 0.6
5: >= 0.6
```

For example, after being summarized, the person summary will look like this tuple:

```
>>> summary
(4, 5, 3, 2)
```

Notice labels in this representation are taken in sorted order, so the values above correspond to:

```
'health', 'income', 'politics', 'religiosity'
```

## Faceborg 1. Ingestion

Write some code that given a `person` dictionary produces a summary tuple using labels in **sorted** order.

[1]:

```
person = {
 'income' : 0.9,
 'health' : 0.5,
 'religiosity' : -0.4,
 'politics' : -0.1
}

write here
```

## Faceborg 2. Selling your soul

The meaning of the various labels is stored in a dictionary which maps a field name to a two-elements tuple with the meanings of respectively the minimal (-1.0) and maximal value (1.0). NOTE: being negative or positive by itself has NO particular meaning:

```
labels = {
 'income' : ('poor', 'wealthy'),
 'health' : ('healthy', 'ill'),
 'religiosity' : ('religious', 'atheist'),
```

(continues on next page)

(continued from previous page)

```
'politics' : ('left', 'right')
}
```

Given a person summary as a tuple and a dictionary of labels, Faceborg wants to show advertisers the summary as a nice printout, so it commands you to write the code.

Example - given:

```
summary=(4, 5, 3, 2)
labels = {
 'income' : ('poor', 'wealthy'),
 'health' : ('healthy', 'ill'),
 'religiosity': ('religious', 'atheist'),
 'politics' : ('left', 'right')
}
```

Your code must print this:

```
12345
health : healthy * ill
income : poor * wealthy
politics : left * right
religiosity : religious * atheist
```

**HINT:** you may want to use methods `str.ljust`<sup>225</sup> and `str.rjust`<sup>226</sup>

[2]:

```
summary=(4, 5, 3, 2)
#summary=(1, 3, 4, 1)

labels = {
 'income' : ('poor', 'wealthy'),
 'health' : ('healthy', 'ill'),
 'religiosity': ('religious', 'atheist'),
 'politics' : ('left', 'right')
}

write here
```

### Faceborg 3. Go where the money is

Given a person summary, advertisers want to immediately know how many people there are for that particular summary.

Suppose Faceborg holds this current model for fast retrieval, which associates the tuples of person summaries to the number of people sharing that same summary:

```
model = {
 (2, 1, 1, 3): 50000000,
 (1, 3, 1, 2): 90000000,
 (3, 3, 1, 2): 40000000,
 (4, 5, 3, 2): 20000000,
```

(continues on next page)

<sup>225</sup> <https://www.programiz.com/python-programming/methods/string/ljust>

<sup>226</sup> <https://www.programiz.com/python-programming/methods/string/rjust>

(continued from previous page)

```
(1, 3, 1, 1): 40000000,
(5, 3, 2, 3): 70000000,
(1, 3, 3, 2): 30000000,
}
```

Write some code that given a list of dictionaries `people`, updates the model with all their summaries.

Example - given:

```
people = [
 {
 'income' : 0.9,
 'health' : 0.5,
 'religiosity': -0.4,
 'politics' : -0.1
 }, # corresponds to (4, 5, 3, 2), summary already present in the model
 {
 'income' : 0.1,
 'health' : -0.6,
 'religiosity': -0.8,
 'politics' : 0.5
 } # corresponds to (2, 3, 4, 1), summary not present in the model
]
```

After your code, Faceborg model should become like this (order **does not** matter):

```
>>> model
{(1, 3, 1, 1): 40000000,
 (1, 3, 1, 2): 90000000,
 (1, 3, 3, 2): 30000000,
 (2, 1, 1, 3): 50000000,
 (2, 3, 4, 1): 1,
 (3, 3, 1, 2): 40000000,
 (4, 5, 3, 2): 20000001, # note the 1
 (5, 3, 2, 3): 70000000}
```

[7]:

```
model = {
 (2, 1, 1, 3): 50000000,
 (1, 3, 1, 2): 90000000,
 (3, 3, 1, 2): 40000000,
 (4, 5, 3, 2): 20000000,
 (1, 3, 1, 1): 40000000,
 (5, 3, 2, 3): 70000000,
 (1, 3, 3, 2): 30000000,
}

people = [
 {
 'income' : 0.9,
 'health' : 0.5,
 'religiosity': -0.4,
 'politics' : -0.1
 }, # corresponds to (4, 5, 3, 2), summary already present in the model
 {
 'income' : 0.1,
```

(continues on next page)

(continued from previous page)

```
'health' : -0.6,
'religiosity': -0.8,
'politics' : 0.5
} # corresponds to (2, 3, 4, 1), summary not present in the model
]

write here
```

## Challenge - The informant

The FBI Crime unit is trying to infiltrate a powerful mob organization - so far, they managed to obtain the services of an informant who now and then gives precious tips. All the investigations are then recorded in classified documents, which need to be anonymized according to the reader's clearance level. In order to mark possibly sensitive information, detectives place symbols such as > and < to delimit pieces of text to anonymize.

Write some code which anonymizes the text, substituting words in sensitive sequences with the proper number of asterisks.

Example - given:

```
text = """Our > informant Mr Big Ears <, who's operating within > the Organization, <_
< told us
 about a possible encounter among suspects we're following. The > suspect Mr_
< Wrong Do
 (also known as Mr Cut Throat) < was in fact seen last night near > Vice_
< Palace < while
 talking with > Mr So Bad <. They nervously glanced around, and > after a_
< while,
 they quickly exchanged two suitcases. < The details of the deal are yet to_
< be discovered."""
punctuation = [',', '.', ';', ',', '']
```

your code should print:

```
Our ***** * *** ****, who's operating within *** ******, told
us about a possible encounter among suspects we're following. The ***** *
***** ** (***** ***** ** * *** *****) was in fact seen last night near ****
***** while talking with ** ** **. They nervously glanced around, and
***** * ***** , ***** ***** ***** *** *****. The details of the deal
are yet to be discovered.
```

**NOTE 1:** Sometimes detectives place > < with punctuation around, your program should handle those cases as well, preserving punctuation in the output. I.e.

Our > informant Mr Big Ears <, who's

should become:

Our \*\*\*\*\* \* \*\*\* \*\*\*\*, who's

**NOTE 2:** Your program should also preserve punctuation in anonymized words, i.e.

suspect Mr Wrong Do (also known as Mr Cut Throat)

should become:

```
***** * * ***** (* * * * * *)
```

### HAVE YOU READ THE ABOVE NOTES?

Making a rough version of the program should be relatively straightforward, making the details also work may be more challenging

```
[1]:
text = """Our > informant Mr Big Ears <, who's operating within > the Organization, <
→told us
 about a possible encounter among suspects we're following. The > suspect Mr
→Wrong Do
 (also known as Mr Cut Throat) < was in fact seen last night near > Vice
→Palace < while
 talking with > Mr So Bad <. They nervously glanced around, and > after a
→while,
 they quickly exchanged two suitcases. < The details of the deal are yet to
→be discovered."""

punctuation = [',', '.', ';', ' ', '']

write here
```

```
[]:
```

## 6.3 While loops

### 6.3.1 While loops 1 - introduction

[Download exercises zip](#)

Browse online files<sup>227</sup>

Let's see how to repeat instructions by executing them inside `while` loops.

The main feature of `while` loop is to allow explicit control when the loop should end. Typically, such loops are used when we must *iterate* on a sequence we don't know the dimension of in advance, or the dimension can vary over time, or when several conditions might determine the cycle stop.

<sup>227</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/while>

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
while
 while1.ipynb
 while1-sol.ipynb
 while2-chal.ipynb
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `while.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

### Counting with a while

A `while` cycle is a code block which is executed when a certain boolean condition is verified. The code block is repeatedly executed as long as the condition is true.

Let's see an example:

```
[2]: i = 1

while i < 4:
 print('Counted', i)
 i += 1

print('Loop is over!')
```

Counted 1  
Counted 2  
Counted 3  
Loop is over!

In the example, the boolean condition is

```
i < 4
```

the block to keep executing is

```
print('Counted', i)
i += 1
```

Like any Python code blocks, the block is indented with spaces (usually 4).

Have a better look at the execution in Python Tutor and read the following comment.

```
[3]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
(it's sufficient to execute it only once)

import jupman
```

```
[4]: i = 1
while i < 4:
 print('Counted', i)
 i += 1

print('Loop is over !')

jupman.pytut()
```

```
Counted 1
Counted 2
Counted 3
Loop is over !
```

```
[4]: <IPython.core.display.HTML object>
```

In the example we used a variable we called `i` and initialized it to zero.

At the beginning of the cycle `i` is valued 1, so the boolean expression `i < 4` is evaluated as `True`. Since it's `True`, execution continues inside the block with the `print` and finally MODIFIES `i` by incrementing `i += 1`.

Now the execution goes to `while` row, and condition `i < 4` is evaluated again. At this second iteration `i` is valued 2, so the boolean expression `i < 4` is again evaluated to `True` and the execution remains inside the block. A new `print` is done and `i` gets incremented.

Another loop is done until `i` is valued 4. A that point `i < 4` produces `False` so in that moment execution *exits* the `while` block and goes on with the commands at the same indentation level as the `while`

## Terminating while

When we have a `while` cycle, typically sooner or later we want it to terminate (programs which hang aren't users' favourites ...). To guarantee termination, we need:

1. initializing a variable outside the cycle
2. a condition after the `while` command which evaluates that variable (and optionally other things)
3. at least one instruction in the internal block which MODIFIES the variable, so that sooner or later condition 2 is going to be satisfied

If any of these points is omitted, we will have problems. Let's try forgetting them on purpose:

**Error 1: omit initialization.** As in those cases in Python where we forgot to initialize a variable (let's try `j` in this case), the execution is interrupted as soon we try using the variable:

```
print("About to enter the cycle .")
while j < 4:
 print('Counted', j)
 j += 1

print('Loop is over !')
```

```
About to enter the cycle ..

NameError Traceback (most recent call last)
<ipython-input-277-3f311955204d> in <module>()
 1 print("About to enter the cycle ..")
----> 2 while j < 4:
 3 print('Counted', j)
 4 j += 1
 5

NameError: name 'j' is not defined
```

**Error 2: omit using the variable in the condition.** If we forget to evaluate the variable, for example by using a wrong one (say `x`), the loop will never stop:

```
i = 1
x = 1
print('About to enter the cycle ..')
while x < 4: # evaluates x instead of i
 print('Counted', i)
 i += 1

print('Loop is over !')
```

```
About to enter the cycle ..
Counted 1
Counted 2
Counted 3
Counted 4
Counted 5
Counted 6
. .
```

**Error 3: Omit to MODIFY the variable in the internal block.** If we forget to place at least one instruction which MODIFIES the variable used in the condition, whenever the condition is evaluated it will always produce the same boolean value `False` preventing the cycle from exiting:

```
i = 1
print('About to enter the cycle ..')
while i < 4:
 print('Counted', i)

print('Loop is over !')
```

```
About to enter the cycle ..
Counted 1
Counted 1
Counted 1
Counted 1
Counted 1
. .
```

## Non-terminating while

**QUESTION:** Can you imagine a program which *never* terminates?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** if you live nearby a hydropower or nuclear plant, what happens if the program regulating the water stops?

Or: suppose you are inside an airplane and the program which checks the fuel flux to the engine suddenly stops. Could this be a problem?

All programs if well written must foresee termination, but some software are executed for such a long time that termination is to be considered an exceptional event.

</div>

## Questions

**QUESTION:** Look at the following code fragments , and for each try guessing the result it produces (or if it gives an error):

1. 

```
i = 0
while i < 3:
 print(i)
```

2. 

```
k = 0
while k < 5:
 print(k)
 k + 1
```

3. 

```
i = 0
while i < 3:
 print(i)
i += 1
```

4. 

```
i = 0
while False:
 print(i)
 i += 1
print('Done !')
```

5. 

```
i = 0
while i < 3:
 print(i)
 i += 1
```

6. 

```
k = 0
while k < 2:
 print(i)
 k += 1
```

7. 

```
i = 0
while i < 3:
 print('GAM')
 i = i + 1
```

```
8. while zanza < 2
 print('ZANZA')
 zanza += 1
```

```
9. i = 0
while False:
 print(i)
 i = i + 1
print('DARK')
```

```
10. i = 0
while True:
 print(i)
 i = i + 1
print('LIGHT')
```

```
11. while 2 + 3:
 print('z')
 print('')
```

```
12. i = 10
while i > 0:
 if i > 5:
 print(i)
 i -= 1
print('WAM')
```

```
13. i = 10
while i > 0:
 if i > 5:
 print(i)
 i -= 1
print('MAW')
```

```
14. import random
x = 0
while x < 7:
 x = random.randint(1, 10)
 print(x)

print('LUCK')
```

```
15. x, y = 0, 0
while x + y < 4:
 x += 1
 y += 1
 print(x, y)
```

```
16. x, y = 0, 3
while x < y:
 print(x, y)
 x += 1
 y -= 1
```

## Esercises

### Exercise - printeven

⊕ Write some code to print all the odd numbers from 1 to k in a while cycle

- for  $k < 1$  prints nothing

Example - given:

```
k = 5
```

after your code it must print:

```
1
3
5
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5] :

```
k = 5 # 1 3 5
#k = 1 # 1
#k = 0 # no print

write here
i = 1
while i <= k:
 if i % 2 == 1:
 print(i)
 i += 1
```

```
1
3
5
```

</div>

[5] :

```
k = 5 # 1 3 5
#k = 1 # 1
#k = 0 # no print

write here
```

### Exercise - average

⊕ Write some code that given a list `numbers`, calculates the average of values using a `while` and then prints it.

- if the list is not empty, the average is supposed to be `0.0`
- **DO NOT** use the function `sum`
- **DO NOT** create variables called `sum` (would violate the **V COMMANDMENT**<sup>228</sup>: you shall never ever redefine system functions)

Example - given:

```
numbers = [8, 6, 5, 9]
```

prints

```
7.0
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```
numbers = [8, 6, 5, 9] # 7.0
#numbers = [3, 1, 2] # 2.0
#numbers = [] # 0

write here
s = 0.0
i = 0
while i < len(numbers):
 s += numbers[i]
 i += 1

if len(numbers) > 0:
 print(s / len(numbers))
else:
 print(0.0)
```

```
7.0
```

</div>

[6]:

```
numbers = [8, 6, 5, 9] # 7.0
#numbers = [3, 1, 2] # 2.0
#numbers = [] # 0

write here
```

<sup>228</sup> <https://en.softpython.org/commandments.html#V-COMMANDMENT>

## break and continue commands

For getting even more control on cycle execution we can use the commands `break` and `continue`

---

### NOTE: Use them sparingly!

When there is a lot of code in the cycle it's easy to 'forget' about their presence and introduce hard-to-discover bugs. On the other hand, in some selected cases these commands may increase code readability, so as everything use your judgement.

---

## Terminate with a break

The scheme we've just seen is the recommended one to properly terminate a `while`, but if we have a condition which does NOT evaluate the variable we are incrementing (like for example the constant expression `True`), as an alternative we can use the command `break` to immediatly exit the cycle:

```
[7]: i = 1
while True:

 print('Counted', i)

 if i > 3:
 print('break! Exiting the loop!')
 break
 print('After the break')

 i += 1

print('Loop is over !')
```

Counted 1  
Counted 2  
Counted 3  
Counted 4  
break! Exiting the loop!  
Loop is over !

Note After the `break` is *not* shown.

## Jumping with continue

We can bring the execution immediately to the next iteration by calling `continue`, which directly jumps to the condition check without executing the instructions after the `continue`.

### WARNING: `continue` instructions can cause infinite loops if used carelessly!

When using `continue` ensure it doesn't jump the instruction which modifies the variable used in the termination condition (or it doesn't jump a `break` needed for exiting the cycle)!

To avoid problems here we incremented `i` before the `if` with a `continue`:

```
[8]: i = 1
while i < 5:
 print('Counted', i)

 i += 1

 if i % 2 == 1:
 print('continue, jumping to condition check')
 continue
 print('After the continue')

 print('arrived till the end')

print('Loop is over !')

Counted 1
arrived till the end
Counted 2
continue, jumping to condition check
Counted 3
arrived till the end
Counted 4
continue, jumping to condition check
Loop is over !
```

Let's try combining break and continue, and see what happens in Python Tutor:

```
[9]: i = 1
while i < 5:
 print('Counted', i)
 if i > 3:
 print('break! Exiting the cycle!')
 break
 print('After the break')
 i += 1
 if i % 2 == 1:
 print('continue, jumping to next condition check')
 continue
 print('After the continue')
 print('arrived till the end')

print('Loop is over !')

jupman.pytut()

Counted 1
arrived till the end
Counted 2
continue, jumping to next condition check
Counted 3
arrived till the end
Counted 4
break! Exiting the cycle!
Loop is over !

[9]: <IPython.core.display.HTML object>
```

### Questions about break and continue

**QUESTION:** Look at the following code fragments , and try guessing for each the result it produces (or if it gives an error):

```
1. i = 1
while i < 4:
 print('Counted', i)
 i += 1
 continue

print('Loop is over !')
```

```
2. i = 1
while i < 4:
 print('Counted', i)
 continue
 i += 1

print('Loop is over !')
```

```
3. i = 3
while i > 0:
 print('Counted', i)
 if i == 2:
 print('continue, jumping to condition check')
 continue
 i -= 1
 print('arrived till the end')

print('Loop is over !')
```

```
4. i = 0
while True:
 i += 1
 print(i)
 if i > 3:
 break

print('BONG')
```

```
5. i = 0
while True:
 if i < 3:
 continue
 else:
 break
 i += 1

print('ZONG')
```

```
6. i = 0
while True:
 i += 1
 if i < 3:
```

(continues on next page)

(continued from previous page)

```
 continue
else:
 break

print('ZANG')
```

### Questions - Are they equivalent?

Look at the following code fragments: each contains two parts, A and B. For each value of the variables they depend on, try guessing whether part A will print exactly the same result printed by code in part B

- **FIRST** think about the answer and **write down the expected output**
- **THEN** try executing with each of the values of suggested variables

### Are they equivalent? - BORG

```
print('A:')
while True:
 print('BORG')
 break

print('\nB:')
while False:
 pass
print('BORG')
```

### Are they equivalent? - until 3

```
print('A:')
x = 0
while x < 3:
 print(x)
 x += 1

print('\nB:')
x = 1
while x <= 3:
 print(x-1)
 x += 1
```

### Are they equivalent? - by chance

Remember `randint(a, b)` gives back a random integer N such that  $a \leq N \leq b$

```
print('A:')
x = 0
while x < 3:
 x += 1
print(x)

print('\nB:')
x = 0
import random
while x != 3:
 x = random.randint(1,5)
print(x)
```

### Are they equivalent? - until six

```
print('A:')
i = 0
while i < 3:
 print(i)
 i += 1
while i < 6:
 print(i)
 i += 1

print('\nB:')
i = 0
while i < 6:
 print(i)
 i += 1
```

### Are they equivalent? - countdown 1

```
print('A:')
i = 2
print(i)
while i > 0:
 i -= 1
 print(i)

print('\nB:')
i = 2
while i > 0:
 print(i)
 i -= 1
```

### Are they equivalent? - countdown 2

```
print('A:')
i = 2
print(i)
while i > 0:
 i -= 1
 print(i)

print('\nB:')
i = 2
while i > 0:
 print(i)
 i -= 1
print(i)
```

```
[10]: print('A:')
i = 2
print(i)
while i > 0:
 i -= 1
 print(i)

print('\nB:')
i = 2
while i > 0:
 print(i)
 i -= 1
print(i)
```

```
A:
2
1
0
```

```
B:
2
1
0
```

### Are they equivalent? - sorcery

```
print('A:')
s = 'sorcery'
i = 0
while s[i] != 'e':
 i += 1
print(s[i:])

print('B:')
s = 'sorcery'
i = len(s)
while s[i] != 'e':
 i -= 1
print(s[i:])
```

### Are they equivalent? - ping pong

```

print('A:')
ping,pong = 0,3
while ping < 3 or pong > 0:
 print(ping,pong)
 ping += 1
 pong -= 1

print('\nB:')
ping,pong = 0,3
while not(ping >= 3 and pong <= 0):
 print(ping,pong)
 ping += 1
 pong -= 1

```

### Are they equivalent? - zanna

```

print('A:')
n,i,s = 0,0,'zanna'
while i < len(s):
 if s[i] == 'n':
 n += 1
 i += 1
print(n)

print('\nB:')
n,i,s = 0,0,'zanna'
while i < len(s):
 i += 1
 if s[i-1] == 'n':
 n += 1
print(n)

```

### Are they equivalent? - pasticcio

```

print('A:')
c,i,s = 0,0,'pasticcio'
while i < len(s):
 if s[i] == 'c':
 c += 1
 i += 1
print(c)

print('\nB:')
no,k,s = 0,0,'pasticcio'
while k < len(s):
 if s[k] != 'c':
 no += 1
 else:
 k += 1
print(len(s) - no)

```

## Exercises - counters

### Exercise - don't break 1

⊕ Look at the following code, and write in the following cell some code which produces the same result with a `while` and **without using** `break`

```
[11]: x = 3
while True:
 print(x)
 if x == 0:
 break
 x -= 1
```

```
3
2
1
0
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[12]: x = 3

write here

while x >= 0:
 print(x)
 x -= 1
```

```
3
2
1
0
```

</div>

```
[12]: x = 3

write here
```

### Exercise - don't break 2

⊕ Look at the following code, and write in the following cell some code which produces the same result with a `while` and **without using** `break`

```
[13]: la = [2, 3, 7, 5, 6]
k = 7 # 2 3 7
#k = 5 # 2 3 7 5 6
#k = 13 # 2 3 7 5 6

i = 0
```

(continues on next page)

(continued from previous page)

```

while True:
 print(la[i])
 if i >= len(la)-1 or la[i] == k:
 break
 else:
 i += 1
2
3
7

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[14]:

```

la = [2, 3, 7, 5, 6]
k = 7 # 2 3 7
#k = 6 # 2 3 7 5 6
#k = 13 # 2 3 7 5 6

i = 0

write here

while i < len(la) and la[i] != k:
 print(la[i])
 i += 1
if i < len(la) and la[i] == k:
 print(la[i])
2
3
7

```

&lt;/div&gt;

[14]:

```

la = [2, 3, 7, 5, 6]
k = 7 # 2 3 7
#k = 6 # 2 3 7 5 6
#k = 13 # 2 3 7 5 6

i = 0

write here

```

**Exercise - Give me a break**

⊕ Look at the following code, and write in the next cell some code which produces the same result with a `while` **this time using a break**

[15]:

```
x,y = 1,5 # (1,5) (2,4)
#x,y = 2,8 # (2, 8) (3, 7) (4, 6)

while x < y or x == 4:
 print((x,y))
 x += 1
 y -= 1

(1, 5)
(2, 4)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[16]:

```
x,y = 1,5 # (1,5) (2,4)
#x,y = 2,8 # (2, 8) (3, 7) (4, 6)

write here
while True:

 if x >= y or x == 4:
 break
 else:
 print((x,y))
 x += 1
 y -= 1

if x < y or x == 4:
 print((x,y))

(1, 5)
(2, 4)
```

&lt;/div&gt;

[16]:

```
x,y = 1,5 # (1,5) (2,4)
#x,y = 2,8 # (2, 8) (3, 7) (4, 6)

write here
```

### Exercise - paperboard

⊕ Prints integer numbers from 0 to k INCLUDED using a `while`, and for each number prints to its side one among the strings 'PA', 'PER' and 'BOARD' alternating them

Ex - for k=8 prints

```
0 PA
1 PER
2 BOARD
3 PA
4 PER
5 BOARD
6 PA
7 PER
8 BOARD
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[17] :

```
k = 8

write here
x = 0
while x <= k:
 if x % 3 == 0:
 print(x, 'PA')
 elif x % 3 == 1:
 print(x, 'PER')
 else:
 print(x, 'BOARD')
 x += 1
```

```
0 PA
1 PER
2 BOARD
3 PA
4 PER
5 BOARD
6 PA
7 PER
8 BOARD
```

</div>

[17] :

```
k = 8

write here
```

### Exercise - until ten

- ⊕ Given two numbers  $x$  and  $y$ , write some code with a `while` which prints and increments the numbers, stopping as soon as one of them reaches ten.

```
x, y = 5, 7
```

after your code it must result:

```
5 7
6 8
7 9
8 10
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[18]:

```
x, y = 5, 7
#x, y = 8, 4

write here
while x <= 10 and y <= 10:
 print(x,y)
 x += 1
 y += 1
```

```
5 7
6 8
7 9
8 10
```

</div>

[18]:

```
x, y = 5, 7
#x, y = 8, 4

write here
```

### Exercise - cccc

- ⊕ Write some code using a `while` which given a number  $y$ , prints  $y$  rows containing the character `c` as many times as the row number.

Example - given:

```
y = 4
```

Prints:

```
c
cc
ccc
cccc
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[19]:

```
y = 4

write here
x = 0
while x <= y:
 print('c'*x)
 x += 1
```

```
c
cc
ccc
cccc
```

</div>

[19]:

```
y = 4

write here
```

## Exercise - converge

⊕ Given two numbers  $x$  and  $k$ , using a `while` modify and print  $x$  until it reaches  $k$  included

- **NOTE:**  $k$  can either be greater or lesser than  $x$ , you must handle both cases

Example 1 - given:

```
x, k = 3, 5
```

prints:

```
3
4
5
```

Example 2 - given:

```
x, k = 6, 2
```

prints:

```
6
5
4
3
2
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[20]:

```
x, k = 3, 5 # 3 4 5
#x, k = 6, 2 # 6 5 4 3 2
#x, k = 4, 4 # 4

write here

while x != k:
 print(x)
 if x < k:
 x += 1
 else:
 x -= 1
print(x)
```

```
3
4
5
```

</div>

[20]:

```
x, k = 3, 5 # 3 4 5
#x, k = 6, 2 # 6 5 4 3 2
#x, k = 4, 4 # 4

write here
```

## Searching a sequence

We are at the airport, and we've been told to reach the gate of our trusted airline company *Turbulenz*. We don't remember exactly the gate, but we know we have to stop at the first *Turbulenz* sign we find. If by mistake we went further, we might encounter other gates for international flights, and who knows where we would end up.

If we have to perform searches in potentially long sequences, and we don't always need a complete visit, using a `while` loop is more convenient and efficient than a `for`.

We could represent the example above as a list:

[21]:

```
0 1 2 3 4 5 ...
˓→ 6 7
airport = ['Flyall', 'PiercedWings', 'PigeonJet', 'Turbolenz', 'BoingBoing', 'Jettons',
˓→ 'Turbulenz', 'BoingBoing']
```

Once the element is found, we would like the program to print the position in which it was found, in this case 3.

Naturally, if you read well the `list` search methods<sup>229</sup> you already know there is a handy method `.index('Turbulenz')`, but in this notebook we adopt the philosophy of 'do it yourself', and will try building our search algorithms from scratch.

**QUESTION:** Can you think of some corner case where `index` method can also bring a problem?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

<sup>229</sup> <https://en.softpython.org/lists/lists4-sol.html>

**ANSWER:** if the list does *not* contain what we're looking for, calling `index` will raise an exception thus stopping the program.

</div>

## What we need

To build our search, we will need:

1. control variable
2. stop condition
3. control variable update

The control variable in this case could be an index `i`, the stop condition could evaluate whether we reached the end of the airport, and inside the cycle we will update the index to keep searching. But where should we evaluate whether or not we have found the gate? Furthermore, since we are expert programmers we believe in misfortune, and know horror scenarios could happen indeed, like Turbulenz company going bankrupt the very same day of our arrival! Thus, we also need to foresee the case our search may give no result, and decide what should happen in such situation.

## How to check

There are two ways to check a discovery:

- a) most direct way is to place an exit check inside the body of `while` itself: we could put an `if` statement which controls when the element is found, and in such case performs the execution of a `break` command. It's by no means elegant, yet it could be a first approach.
- b) a better option would be performing the check in the boolean condition of the `while`, but devising a program which works in all cases could be slightly trickier.

Let's try them both in the following exercises.

### Exercise - Turbulenz with a break

⊕⊕ Write some code which uses a `while` to search the list `airport` for the FIRST occurrence of `company`: as soon as it is found, stops searching and PRINTS the index where it was found.

- If the company is not found, PRINTS 'Not found'
- USE a `break` to stop the search
- REMEMBER to test your code with all the suggested airports

Example 1 - given:

```
company='Turbolenz'
airport = ['Flyall','PiercedWings','PigeonJet','Turbolenz', 'BoingBoing','Jettons',
↪'Turbulenz','BoingBoing']
```

after your code, it must print:

```
Found the first Turbolenz at index 3
```

Example 2 - given:

```
company = 'FlapFlap'
airport = ['PiercedWings', 'BoingBoing', 'Turbolenz', 'PigeonJet']
```

it must print:

```
FlapFlap was not found
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[22]:

```
company = 'Turbolenz'
airport = ['Flyall', 'PiercedWings', 'PigeonJet', 'Turbolenz', 'BoingBoing', 'Jettons',
 ↵ 'Turbulenz', 'BoingBoing']

#company = 'FlapFlap'
#airport = ['PiercedWings', 'BoingBoing', 'Turbolenz', 'PigeonJet']
#airport = []
#airport = ['FlapFlap']
#airport = ['Turbolenz', 'FlapFlap']

write here
i = 0
while i < len(airport):
 if airport[i] == company:
 print("Found the first", company, "at index", i)
 break
 i += 1
if i == len(airport):
 print(company, 'was not found')

Found the first Turbolenz at index 3
```

```
</div>
```

[22]:

```
company = 'Turbolenz'
airport = ['Flyall', 'PiercedWings', 'PigeonJet', 'Turbolenz', 'BoingBoing', 'Jettons',
 ↵ 'Turbulenz', 'BoingBoing']

#company = 'FlapFlap'
#airport = ['PiercedWings', 'BoingBoing', 'Turbolenz', 'PigeonJet']
#airport = []
#airport = ['FlapFlap']
#airport = ['Turbolenz', 'FlapFlap']

write here
```

### Exercise - Turbulenz without break

⊕⊕ Try now to rewrite the previous program **without** using `break` nor `continue`: to verify the finding, you will need to enrich the termination condition.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[23]:

```
company = 'Turbolenz'
airport = ['Flyall', 'PiercedWings', 'PigeonJet', 'Turbolenz', 'BoingBoing', 'Jettons',
 ↵'Turbulenz', 'BoingBoing']

#company = 'FlapFlap'
airport = ['PiercedWings', 'BoingBoing', 'Turbolenz', 'PigeonJet']
#airport = []
#airport = ['FlapFlap']
#airport = ['Turbolenz', 'FlapFlap']

write here

i = 0
while i < len(airport) and airport[i] != company:
 i += 1

if i == len(airport):
 print(company, 'was not found')
else:
 print("Found the first", company, "at index", i)
```

Found the first Turbolenz at index 2

</div>

[23]:

```
company = 'Turbolenz'
airport = ['Flyall', 'PiercedWings', 'PigeonJet', 'Turbolenz', 'BoingBoing', 'Jettons',
 ↵'Turbulenz', 'BoingBoing']

#company = 'FlapFlap'
airport = ['PiercedWings', 'BoingBoing', 'Turbolenz', 'PigeonJet']
#airport = []
#airport = ['FlapFlap']
#airport = ['Turbolenz', 'FlapFlap']

write here
```

**QUESTION:** you probably used two conditions in the `while`. By exchanging the order of the conditions in the proposed solution, would the program work fine? If not, in which cases could it fail?

- **HINT:** If you have doubts try reading the chapter `booleans - evaluation order`<sup>230</sup>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

<sup>230</sup> <https://en.softpython.org/basics/basic2-bools-sol.html#Evaluation-order>

**ANSWER:** the comparison between the index with the airport length must be done first, because when it gives `False` the evaluation of the `and` expression stops immediately without proceeding with the dangerous `airport[i] != company` which in an airport without FlapFlap company would become `airport[4] != company` and thus produce an index error:

```
company = 'FlapFlap'
0 1 2 3
airport = ['PiercedWings', 'BoingBoing', 'Turbolenz', 'PigeonJet']

write here
i = 0
WARNING: WRONG ORDER!
while airport[i] != company and i < len(airport):
 i += 1

if i == len(airport):
 print(company, 'was not found')
else:
 print("Found the first", company, "at index", i)
```

</div>

### Exercise - hangar

⊕⊕ Our plane just landed but now it must reach the hangar, dodging all the extraneous objects on the track!

Write some code which given a string `track` with a certain number of non-alphanumeric characters at the beginning, PRINTS the word which follows these characters.

Example - given:

```
track = '★☆♦♦♦hangar★★'
```

your code must print:

hangar★★

- **YOU CAN'T** know beforehand which extra characters you will find in the string
- **DO NOT** write characters like ★♦♦-\_ in the code

**HINT:** to determine if you have found alphanumerical characters or numbers, use `.isalpha()` and `.isdigit()` methods

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution

</a><div class="jupman-sol jupman-sol-code" style="display:none">

[24]:

```
track = '★☆♦♦♦hangar★★' # hangar★★
#track = '♦♦twinengine' # twinengine
#track = '-♦---♦---747-♦' # 747-♦
#track = 'glider' # glider
#track = '_♦_♦_♦_' # prints nothing

write here

i = 0
while i < len(track) and not (track[i].isalpha() or track[i].isdigit()):
```

(continues on next page)

(continued from previous page)

```
i += 1
print(track[i:])

hangar★★★

</div>
```

[24]:

```
track = '★★◆◆◆◆◆◆hangar★★★' # hangar★★★
#track = '◆◆twinengine' # twinengine
#track = '-◆---◆---747-◆' # 747-◆
#track = 'glider' # glider
#track = '_◆_◆_◆_' # prints nothing

write here
```

## Exercise - Wild West

⊕⊕ The two outlaws Carson and Butch agreed to bury a treasure in the jolly town of Tombstone, ma now each of them wants to take back the treasure without sharing anything with the partner.

- there is a road from Santa Fe until Tombstone to arrive to the treasure, which we represent as a list of strings
- we use two indexes butch and carson to represent where the outlaws are on the road
- each outlaw starts from a different town
- at each turn Carson moves of **one** city
- at each turn Butch moves of **two** cities, because he has a fast Mustang horse

Write some code which prints the run and terminates as soon as one them arrives to the last city, telling who got the treasure.

- In the case both outlaws arrive to the last city at the same time, prints Final duel in Tombstone !
- your code must work for *any* road and initial position carson and butch

Example - 1 given:

```
0 1 2 3 4 5
road = ['Santa Fe', 'Denver', 'Dodge City', 'Silverton', 'Agua Caliente', 'Tombstone']
carson, butch = 3, 0
```

it must print:

```
Carson starts from Silverton
Butch starts from Santa Fe
Carson reaches Agua Caliente
Butch reaches Dodge City
Carson reaches Tombstone
Butch reaches Agua Caliente

Carson takes the treasure in Tombstone !
```

Example 2 - given:

```
0 1 2 3 4 5
road = ['Santa Fe', 'Denver', 'Dodge City', 'Silverton', 'Agua Caliente', 'Tombstone']
carson, butch = 3, 2
```

it must print:

```
Carson starts from Silverton
Butch starts from Dodge City
Carson reaches Agua Caliente
Butch reaches Agua Caliente
Carson reaches Tombstone
Butch reaches Tombstone

Final duel in Tombstone !
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[25]:

```
0 1 2 3 4 5
road = ['Santa Fe', 'Denver', 'Dodge City', 'Silverton', 'Agua Caliente', 'Tombstone']

carson, butch = 3, 0 # Carson takes the treasure in Tombstone !
#carson, butch = 0, 0 # Butch takes the treasure in Tombstone !
#carson, butch = 3, 2 # Final duel in Tombstone !

write here

print('Carson starts from', road[carson])
print('Butch starts from', road[butch])

while carson < len(road)-1 and butch < len(road)-1:
 carson = min(len(road)-1, carson + 1)
 butch = min(len(road)-1, butch + 2)
 print('Carson reaches', road[carson])
 print('Butch reaches', road[butch])

 print()
if carson == len(road)-1 and butch == len(road)-1:
 print('Final duel in ', road[-1], '!')
elif carson == len(road)-1:
 print('Carson takes the treasure in ', road[-1], '!')
else:
 print('Butch takes the treasure in ', road[-1], '!')

Carson starts from Silverton
Butch starts from Santa Fe
Carson reaches Agua Caliente
Butch reaches Dodge City
Carson reaches Tombstone
Butch reaches Agua Caliente

Carson takes the treasure in Tombstone !
```

</div>

[25]:

```
0 1 2 3 4 5
road = ['Santa Fe', 'Denver', 'Dodge City', 'Silverton', 'Agua Caliente', 'Tombstone']

carson,butch = 3, 0 # Carson takes the treasure in Tombstone !
#carson,butch = 0, 0 # Butch takes the treasure in Tombstone !
#carson,butch = 3, 2 # Final duel in Tombstone !

write here
```

### Exercise - The Balance of Language

⊕⊕ In the sacred writings of Zamfir the Prophet, it is predicted that when all Earth inhabitants speak a language with all the words of same length, universal harmony will be reached among human people. This event is probably far in time and by that epoch the vocabulary of humans will be so wide and varied that checking all the words will certainly require powerful calculations: you are asked to program the underwater servers of Atlantis to perform a check in the centuries to come.

Given a string of words `language`, write some code which prints `True` if all the words have the same length, `False` otherwise.

To have an efficient algorithm, you must use a `while`:

- stop the loop as soon you can determine with certainty the program result
- **DO NOT** use `break` nor `continue`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[26]:

```
language = "eternal harmony forever" # True
#language = "war and violence" # False
#language = "vi rt uo si ty" # True
#language = "deceit bullying" # False
#language = "harmony crashed today" # False
#language = "peace" # True
#language = "" # True

write here
li = language.split()

n = len(li[0]) if len(li) > 0 else 0

all_equal = True
i = 1
while i < len(li) and all_equal:
 if n != len(li[i]):
 all_equal = False
 i += 1

print(all_equal)

True
```

</div>

[26]:

```
language = "eternal harmony forever" # True
#language = "war and violence" # False
#language = "vi rt uo si ty" # True
#language = "deceit bullying" # False
#language = "harmony crashed today" # False
#language = "peace" # True
#language = "" # True

write here
```

### Exercise - the tree shaker

⊕⊕⊕ Giustino the farmer decides to radically improve his farm productivity with high-tech devices, and asks you to develop a ‘tree shaker’ (so he calls it..) to perturbate the trees and harvest the exotic fruits he planted in his highlands (thanks to climate change...)

The plantation is a sequence of fruit trees, elements of the landscape (stones, gravel, etc) and signs S. The beginning and end of a subsequence of trees is always marked by a sign.

The vehicle to design has a cargo\_bed of **capacity 7** where it can store the harvest.

Write some code to scan the plantation and harvests in cargo\_bed the fruits as they are found.

- USE a while, stopping as soon as the cargo\_bed is full
- DO NOT use break nor continue
- DO NOT write fruit names or landscape elements (no bananas nor rocks ..). You can still write 'S', though.

Example - given:

[27]:

```
plantation=['rocks','stones', 'S', 'bananas','oranges','mangos','S', 'sand',
 'stones','stones', 'S', 'avocados','S', 'weeds', 'S', 'kiwi', 'mangos', 'S',
 'S', 'S', 'rocks','S', 'lime','S', 'pebbles','S', 'oranges','coconuts
 , 'S', 'gravel']
```

after your code, it must result:

```
>>> print(cargo_bed)
['bananas', 'oranges', 'mangos', 'avocados', 'kiwi', 'mangos', 'lime']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[28]:

```
0 1 2 3 4 5 6 7 8 9
plantation=['rocks','stones', 'S', 'bananas','oranges','mangos','S', 'sand',
 'stones','stones', 'S', 'avocados','S', 'weeds', 'S', 'kiwi', 'mangos', 'S',
 'S', 'S', 'rocks','S', 'lime','S', 'pebbles','S', 'oranges','coconuts
 , 'S', 'gravel']
```

(continues on next page)

(continued from previous page)

```

'S', 'avocados', 'S', 'weeds', 'S', 'kiwi', 'mangos', 'S', ↵
↳ 'S', 'S', # 20 21 22 23 24 25 26 27 ↵
↳ 28 29
 'rocks', 'S', 'lime', 'S', 'pebbles', 'S', 'oranges', 'coconuts
↳ ', 'S', 'gravel']

#plantation = ['S', 'S'] # []
#plantation = ['S', 'lemons', 'S'] # ['lemons']
#plantation = ['sand', 'S', 'lemons', 'S'] # ['lemons']
#plantation = ['oranges'] # []
#plantation = ['S', '1', '2', '3', '4', '5', '6', '7', '8', 'S'] # ['1', '2', '3', '4', '5', '6',
↳ '7']
#plantation = ['S', '1', '2', 'S', 'x', 'S', '3', '4', '5', '6', '7', '8', 'S', '9'] # ['1', '2',
↳ '3', '4', '5', '6', '7']

cargo_bed = []

write here

harvesting = False
i = 0
while i < len(plantation) and len(cargo_bed) < 7:
 if plantation[i] == 'S':
 harvesting = not harvesting
 else:
 if harvesting: cargo_bed.append(plantation[i])
 i += 1

print(cargo_bed)
['bananas', 'oranges', 'mangos', 'avocados', 'kiwi', 'mangos', 'lime']

```

&lt;/div&gt;

```

[28]: # 0 1 2 3 4 5 6 7 ↵
↳ 8 9
plantation=['rocks', 'stones', 'S', 'bananas', 'oranges', 'mangos', 'S', 'sand', ↵
↳ 'stones', 'stones',
 # 10 11 12 13 14 15 16 17 ↵
↳ 18 19
 'S', 'avocados', 'S', 'weeds', 'S', 'kiwi', 'mangos', 'S', ↵
↳ 'S', 'S',
 # 20 21 22 23 24 25 26 27 ↵
↳ 28 29
 'rocks', 'S', 'lime', 'S', 'pebbles', 'S', 'oranges', 'coconuts
↳ ', 'S', 'gravel']

#plantation = ['S', 'S'] # []
#plantation = ['S', 'lemons', 'S'] # ['lemons']
#plantation = ['sand', 'S', 'lemons', 'S'] # ['lemons']
#plantation = ['oranges'] # []
#plantation = ['S', '1', '2', '3', '4', '5', '6', '7', '8', 'S'] # ['1', '2', '3', '4', '5', '6',
↳ '7']
#plantation = ['S', '1', '2', 'S', 'x', 'S', '3', '4', '5', '6', '7', '8', 'S', '9'] # ['1', '2',
↳ '3', '4', '5', '6', '7']

```

(continues on next page)

(continued from previous page)

```
cargo_bed = []

write here

['bananas', 'oranges', 'mangos', 'avocados', 'kiwi', 'mangos', 'lime']
```

### Exercise - the ghost castle

⊕⊕⊕ Given a string and two characters char1 and char2, write some code which PRINTS True if all occurrences of char1 in string are **always** followed by char2

Example - given:

```
string,char1,char2 = 'fantastic story of the ghost castle', 's','t'
```

prints True because all the occurrences of s are followed by t

```
string,char1,char2 = "enthusiastic dadaist", 's','t'
```

prints False, because the sequence si is found, where s is not followed by t

- **USE** a `while`, try to make it efficient by stopping as soon as possible.
- **DO NOT** use `break`
- **DO NOT** use any search method (no `index`, `find`, `replace`, `count`...)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[29]:

```
string,char1,char2 = 'fantastic story of the ghost castle', 's','t' # True
#string,char1,char2 = "enthusiastic dadaist", 's','t' # False
#string,char1,char2 = "beetroots", 'r','o' # True
#string,char1,char2 = "beetroots", 'e','o' # False
#string,char1,char2 = "a", 'a','b' # False
#string,char1,char2 = "ab", 'a','b' # True
#string,char1,char2 = "aa", 'a','b' # False

write here
i = 0

res = True

if len(string) == 1:
 res = False

while i + 1 < len(string) and res:
 if string[i] == char1 and string[i+1] != char2:
 res = False
 i += 1
```

(continues on next page)

(continued from previous page)

```

res
[29]: True
</div>

[29]:
string,char1,char2 = 'fantastic story of the ghost castle', 's','t' # True
#string,char1,char2 = "enthusiastic dadaist", 's','t' # False
#string,char1,char2 = "beetroots", 'r','o' # True
#string,char1,char2 = "beetroots", 'e','o' # False
#string,char1,char2 = "a", 'a','b' # False
#string,char1,char2 = "ab", 'a','b' # True
#string,char1,char2 = "aa", 'a','b' # False

write here

```

## Modifying sequences

In the tutorial on `for` loops we've seen an important warning we repeat here:

---

**X COMMANDMENT<sup>231</sup>:** You shall never ever add or remove elements from a sequence you are iterating with a `for`!

Falling into such temptations **would produce totally unpredictable behaviours** (do you know the expression *pulling the rug out from under your feet*?)

If you really need to remove elements from a sequence you are iterating, use a while cycle or duplicate first a copy of the original sequence.

---

**Note the advice is only about `for` cycles.** In case of necessity, at the end suggests to adopt `while` loops. Let's see when and how ot use them.

## Stack - Drawing from a card deck

Suppose having a deck of cards which we represent as a list of strings, and we want to draw all the cards, reading them one by one.

We can write a `while` that as long as the deck contains cards, keeps removing cards from the top with the `pop` method<sup>232</sup> and prints their name. Remember `pop` MODIFIES the list by removing the last element AND gives back the element as call result, which we can save in a variable we will call `card`:

```
[30]: deck = ['3 hearts', # ----- bottom
 '2 spades',
 '9 hearts',
 '5 diamonds',
```

(continues on next page)

<sup>231</sup> <https://en.softpython.org/commands.html#X-COMMANDMENT>

<sup>232</sup> <https://en.softpython.org/lists/lists3-sol.html#pop-method>

(continued from previous page)

```
'8 clubs'] # <---- top

while len(deck) > 0:
 card = deck.pop()
 print('Drawn', card)

print('No more cards!')

jupman.pytut()

Drawn 8 clubs
Drawn 5 diamonds
Drawn 9 hearts
Drawn 2 spades
Drawn 3 hearts
No more cards!
```

[30]: <IPython.core.display.HTML object>

Looking at the code, we can notice that:

1. the variable `deck` is initialized
2. we verify that `deck` dimension is greater than zero
3. at each step the list `deck` is MODIFIED by reducing its dimension
4. it returns to step 2

The first three points are the conditions which guarantee the `while` loop will sooner or later actually terminate.

### Stack - Drawing until condition

Suppose now to continue drawing cards until we find a heart suit. The situation is more complicated, because now the cycle can terminate in two ways:

1. we find hearts, and interrupt the search
2. there aren't heart cards, and the deck is exhausted

In any case, in the end we must tell a result to the user. To do so, it's convenient to initialize `card` at the beginning like an empty string, so we can handle the case when no hearts cards are found (or the deck is empty).

Let's try a first implementation which uses an internal `if` to verify whether we have found hearts, and in that case exits with a `break` command.

- Try executing the code by uncommenting the second deck which has no hearts cards, and check the different executions.

```
[31]: deck = ['3 hearts', '2 spades', '9 hearts', '5 diamonds', '8 clubs']
#deck = ['8 spades', '2 spades', '5 diamonds', '4 clubs'] # no hearts!

card = ''
while len(deck) > 0:
 card = deck.pop()
 print('Drawn', card)
 if 'hearts' in card:
 break
```

(continues on next page)

(continued from previous page)

```

if 'hearts' in card:
 print('Found hearts!')
else:
 print("Didn't find hearts!")

jupman.pytut()

Drawn 8 clubs
Drawn 5 diamonds
Drawn 9 hearts
Found hearts!

[31]: <IPython.core.display.HTML object>

```

### Exercise - Don't break my heart

⊕ Write some code which solves the same previous problem:

- this time **DO NOT** use `break`
- ensure the code works with a deck without hearts, and also with an empty deck
- **HINT:** put a multiple condition in the `while`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```

[32]:
deck = ['3 hearts', '2 spades', '9 hearts', '5 diamonds', '8 clubs']
#deck = ['8 spades', '2 spades', '5 diamonds', '4 clubs'] # no hearts!
#deck = [] # no hearts !

card = ''

write here

while len(deck) > 0 and 'hearts' not in card:
 card = deck.pop()
 print('Drawn', card)

if 'hearts' in card:
 print("Found hearts!")
else:
 print("Didn't find hearts!")

Drawn 8 clubs
Drawn 5 diamonds
Drawn 9 hearts
Found hearts!

```

</div>

```

[32]:
deck = ['3 hearts', '2 spades', '9 hearts', '5 diamonds', '8 clubs']
#deck = ['8 spades', '2 spades', '5 diamonds', '4 clubs'] # no hearts!
#deck = [] # no hearts !

```

(continues on next page)

(continued from previous page)

```
card = ''

write here
```

### Questions - what happens?

**QUESTION:** Look at the following code fragments , and for each try guessing the result it produces (or if it gives an error):

1. `while []:  
 print('z')  
print('BIG')`

2. `while ['a']:  
 print('z')  
print('BUG')`

3. `la = []  
while len(la) < 3:  
 la.append('x')  
print(la)`

4. `la = ['x', 'y', 'z']  
while len(la) > 0:  
 print(la.pop())`

5. `la = ['x', 'y', 'z']  
while la:  
 print(la.pop(0))`

6. `la = [4, 5, 8, 10]  
while la.pop() % 2 == 0:  
 print(la)`

### Questions - are they equivalent?

Look at the following code fragments: each contains two parts, A and B. For each value of the variables they depend on, try guessing whether part A will print exactly the same result printed by code in part B

- **FIRST** think about the answer
- **THEN** try executing with each of the values of suggested variables

### Are they equivalent? - train

```
print('A:')
la = ['t', 'r', 'a', 'i', 'n']
while len(la) > 0:
 print(la.pop(0))

print('\nB:')
la = ['t', 'r', 'a', 'i', 'n']
la.reverse()
while len(la) > 0:
 print(la.pop(0))
```

### Are they equivalent? - append nx

```
print('A:')
x, n, la = 2, 0, []
while x not in la:
 la.append(n)
 n += 1
print(la)

print('\nB:')
x, la = 2, []
while len(la) < 3:
 la.append(x)
 x += 1
print(la)
```

## Exercises - stack

### Exercise - break sum

⊕ Look at the following code, and rewrite it in the following cell as while

- this time use command break

```
[33]: lst = []
i = 0
k = 10
while sum(lst) < k:
 lst.append(i)
 i += 1
 print(lst)
```

```
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[34]:

```
lst = []
i = 0

write here

while True:
 if sum(lst) >= k:
 break
 else:
 lst.append(i)
 i += 1
print(lst)
```

```
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
```

```
</div>
```

[34]:

```
lst = []
i = 0

write here
```

### Exercise - travelbook

⊕⊕ Suppose you visited the attic and found a stack of books, which we represent as a list of strings. Each string is prefixed by a label of one character indicating the category (D for Detective story, T for Travel, H for History)

```
stack = ['H-Middle Ages', # <---- bottom
 'T-Australia',
 'T-Scotland',
 'D-Suspects',
 'T-Caribbean'] # <---- top
```

Since we are passionate about travel books, we want to examine `stack` one book at a time to transfer books into another pile we call `travel`, which at the beginning is empty. We start from the top book in `stack`, and transfer into `travel` only the books starting with the label T like ('T-Australia')

```
travel = []
```

Write some code that produces the following print:

```
At the beginning:
 stack: ['H-Middle Ages', 'T-Australia', 'T-Scotland', 'D-Suspects', 'T-Caribbean'
→']
 travel: []
Taken T-Caribbean
 stack: ['H-Middle Ages', 'T-Australia', 'T-Scotland', 'D-Suspects']
```

(continues on next page)

(continued from previous page)

```

travel: ['T-Caribbean']
Discarded D-Suspects
stack: ['H-Middle Ages', 'T-Australia', 'T-Scotland']
travel: ['T-Caribbean']
Taken T-Scotland
stack: ['H-Middle Ages', 'T-Australia']
travel: ['T-Caribbean', 'T-Scotland']
Taken T-Australia
stack: ['H-Middle Ages']
travel: ['T-Caribbean', 'T-Scotland', 'T-Australia']
Discarded H-Middle Ages
stack: []
travel: ['T-Caribbean', 'T-Scotland', 'T-Australia']

```

- The non-travel books are not interesting and must be discarded
- Your code must work with *any* stack list

Show solution

<div class="jupman-sol-jupman-sol-code" style="display:none">

[35]:

```

stack = ['H-Middle Ages', # <---- bottom
 'T-Australia',
 'T-Scotland',
 'D-Suspects',
 'T-Caribbean'] # <---- top

travel = []

write here
print("At the beginning:")
print(' stack: ', stack)
print(' travel:', travel)

while len(stack) > 0:
 book = stack.pop()
 if book.startswith('T'):
 print('Taken', book)
 travel.append(book)
 else:
 print('Discarded', book)
 print(' stack: ', stack)
 print(' travel:', travel)

At the beginning:
stack: ['H-Middle Ages', 'T-Australia', 'T-Scotland', 'D-Suspects', 'T-Caribbean
→']
travel: []
Taken T-Caribbean
stack: ['H-Middle Ages', 'T-Australia', 'T-Scotland', 'D-Suspects']
travel: ['T-Caribbean']
Discarded D-Suspects
stack: ['H-Middle Ages', 'T-Australia', 'T-Scotland']
travel: ['T-Caribbean']
Taken T-Scotland
stack: ['H-Middle Ages', 'T-Australia']

```

(continues on next page)

(continued from previous page)

```

travel: ['T-Caribbean', 'T-Scotland']
Taken T-Australia
stack: ['H-Middle Ages']
travel: ['T-Caribbean', 'T-Scotland', 'T-Australia']
Discarded H-Middle Ages
stack: []
travel: ['T-Caribbean', 'T-Scotland', 'T-Australia']

```

&lt;/div&gt;

[35]:

```

stack = ['H-Middle Ages', # <---- bottom
 'T-Australia',
 'T-Scotland',
 'D-Suspects',
 'T-Caribbean'] # <---- top

travel = []

write here

```

**Exercise - BANG !**

⊕⊕ There are two stacks of objects `right_stack` and `left_stack` which we represent as lists of strings. As a pastime, a cowboy decides to shoot the objects at the top of the stacks, alternating the stack at each shoot. The cowboy is skilled and always hits the target, so each shot decreases a stack.

- Suppose the objects on top are the ones at the end of the list
- To keep track of which stack to hit, use a variable `shoot` holding either 'R' or 'L' character
- After each shot the cowboy if possible changes the stack , otherwise keeps shooting at the same stack until it's empty.
- your code must work for *any* stack and initial shot

Example - given:

```

left_stack = ['box', 'boot', 'horseshoe', 'bucket']
right_stack = ['bin', 'saddle', 'tin can']
shoot = 'R'

```

after your code, it must print:

```

Ready?
left_stack: ['box', 'boot', 'horseshoe', 'bucket']
right_stack: ['bin', 'saddle', 'tin can']
BANG! right: tin can
left_stack: ['box', 'boot', 'horseshoe', 'bucket']
right_stack: ['bin', 'saddle']
BANG! left: bucket
left_stack: ['box', 'boot', 'horseshoe']
right_stack: ['bin', 'saddle']

```

(continues on next page)

(continued from previous page)

```
BANG! right: saddle
 left_stack: ['box', 'boot', 'horseshoe']
 right_stack: ['bin']
BANG! left: horseshoe
 left_stack: ['box', 'boot']
 right_stack: ['bin']
BANG! right: bin
 left_stack: ['box', 'boot']
 right_stack: []
BANG! left: boot
 left_stack: ['box']
 right_stack: []
Nothing to shoot on the right!
 left_stack: ['box']
 right_stack: []
BANG! left: box
 left_stack: []
 right_stack: []
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[36]:

```
left_stack = ['box', 'boot', 'horseshoe', 'bucket']
right_stack = ['bin', 'saddle', 'tin can']
shoot = 'R'
#shoot = 'L'
#left_stack = ['bucket', 'box']

write here

print('Ready?')
print(' left_stack:', left_stack)
print(' right_stack:', right_stack)
while len(right_stack) > 0 or len(left_stack) > 0:
 if shoot == 'R':
 if len(right_stack) > 0:
 print('BANG! right: ', right_stack.pop())
 else:
 print('Nothing to shoot on the right!')
 shoot = 'L'
 else:
 if len(left_stack) > 0:
 print('BANG! left: ', left_stack.pop())
 else:
 print('Nothing to shoot on the left!')
 shoot = 'R'

 print(' left_stack:', left_stack)
 print(' right_stack:', right_stack)

Ready?
 left_stack: ['box', 'boot', 'horseshoe', 'bucket']
 right_stack: ['bin', 'saddle', 'tin can']
BANG! right: tin can
 left_stack: ['box', 'boot', 'horseshoe', 'bucket']
 right_stack: ['bin', 'saddle']
```

(continues on next page)

(continued from previous page)

```
BANG! left: bucket
 left_stack: ['box', 'boot', 'horseshoe']
 right_stack: ['bin', 'saddle']
BANG! right: saddle
 left_stack: ['box', 'boot', 'horseshoe']
 right_stack: ['bin']
BANG! left: horseshoe
 left_stack: ['box', 'boot']
 right_stack: ['bin']
BANG! right: bin
 left_stack: ['box', 'boot']
 right_stack: []
BANG! left: boot
 left_stack: ['box']
 right_stack: []
Nothing to shoot on the right!
 left_stack: ['box']
 right_stack: []
BANG! left: box
 left_stack: []
 right_stack: []
```

&lt;/div&gt;

[36]:

```
left_stack = ['box', 'boot', 'horseshoe', 'bucket']
right_stack = ['bin', 'saddle', 'tin can']
shoot = 'R'
#shoot = 'L'
#left_stack = ['bucket', 'box']

write here
```

## Exercise - Growing or degrowing?

⊕⊕ Write some code which given a list `la`, keeps MODIFYING the list according to this procedure:

- if the last element is odd (i.e. 7), attaches a new number at the end of the list obtained by multiplying by two the last element (i.e. attaches 14)
- if the last element is even, removes the last two elements
- **DO NOT** create a new list (so no rows starting with `la =`)
- **WARNING:** when we want both grow and degrow the sequence we are considering in a cycle, we must convince ourselves that sooner or later the termination condition will happen, it's easy to make mistakes and end up with an infinite cycle!
- **HINT:** to degrow the list, you can use the `pop` method<sup>233</sup>

Example - given:

```
la = [3, 5, 6, 7]
```

<sup>233</sup> <https://en.softpython.org/lists/lists3-sol.html#pop-method>

Executing the code, it must print:

```
Odd: attaching 14
 la becomes [3, 5, 6, 7, 14]
Even: removing 14
 removing 7
 la becomes [3, 5, 6]
Even: removing 6
 removing 5
 la becomes [3]
Odd: attaching 6
 la becomes [3, 6]
Even: removing 6
 removing 3
 la becomes []
Done! la is []
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[37]:

```
la = [3,5,6,7]

write here

i = 0
while len(la) > 0:
 if la[-1] % 2 == 1:
 new = la[-1]*2
 la.append(new)
 print(' Odd: attaching', new)
 else:
 print('Even: removing', la.pop())
 print(' removing', la.pop())
 print(' la becomes', la)
 i += 1
print('Done! la is', la)
```

```
Odd: attaching 14
 la becomes [3, 5, 6, 7, 14]
Even: removing 14
 removing 7
 la becomes [3, 5, 6]
Even: removing 6
 removing 5
 la becomes [3]
Odd: attaching 6
 la becomes [3, 6]
Even: removing 6
 removing 3
 la becomes []
Done! la is []
```

</div>

[37]:

```
la = [3,5,6,7]
```

(continues on next page)

(continued from previous page)

```
write here
```

### Continue

Go on with the challenges<sup>234</sup>

[ ]:

## 6.3.2 While loops 2 - Challenges

### Download exercises zip

Browse file online<sup>235</sup>

We now propose some exercises without solution, do you accept the challenge?

#### Challenge - Consecutive letters

⊗⊗ You are given a list of characters la. Write some code to discover the first letter which has a duplicate in the immediately next position.

- **USE** a while loop
- **STOP** as soon as you find a couple
- **DO NOT** use break
- **DO NOT** use search methods (so no .index, .find, nor strange string stuff like .replace ....)

Example 1 - given:

```
la = ['a', 'b', 'c', 'a', 'c', 'f', 'f', 'g', 'h', 'i', 'i', 'f', 'f', 'f', 'm'] # f 5
```

your code should output:

```
FOUND f at position 5
```

Example 2 - given:

```
la = ['c', 'b', 'a', 'b']
```

your code should output

```
DIDN'T FIND COUPLES!
```

<sup>234</sup> <https://en.softpython.org/while/while2-chal.html>

<sup>235</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/while>

[1]:

```

la = ['a', 'b', 'c', 'a', 'c', 'f', 'f', 'g', 'h', 'i', 'i', 'f', 'f', 'f', 'm'] # f 5
#la = ['a'] # nothing
#la = ['a', 'a'] # a 0
#la = ['a', 'b', 'a', 'a'] # a 2
#la = ['b', 'b', 'a'] # b 0
#la = ['c', 'b', 'a', 'b'] # nothing
#la = ['a', 'c', 'b', 'b', 'b'] # b 2

write here

```

## Challenge - Failed casinos

⊕⊕⊕ Casinos can make big money.... or fail big. You are asked to collect the giant characters from the falling billboards of some `casinos` which went bankrupt.

Your truck has only space to hold `capacity` characters. Visit `casinos` list **from last to first** order.

Write some code to:

- MODIFY `truck` list until it is filled with `capacity` characters
- MODIFY `casinos` removing ONLY the characters which were taken

Be careful:

- **USE** a while loop
- **STOP** as soon as you reach `capacity`
- **DO NOT** use `break`
- **DO NOT** use search methods (so no `.index`, `.find`, weird string stuff...)

Example - given:

```

truck = []
capacity = 20
casinos = ['The Claridge Hotel', 'Atlantic club', 'Camelot Hotel', 'Showboat', 'Le
 ↪Jardin', 'The Sands']

```

it should print:

```

Found The Sands
Collecting characters "The Sands"
Found Le Jardin
Collecting characters "Le Jardin"
Found Showboat
Collecting characters "Sh"
Collected 20 characters

truck variable is:
['T', 'h', 'e', ' ', 'S', ' ', 'a', ' ', 'n', ' ', 'd', ' ', 's', ' ', 'L', ' ', 'e', ' ', ' ', 'J', ' ', 'a', ' ', 'r', ' ', 'd', ' ', 'i',
 ↪'n', ' ', 'S', ' ', 'h']

casinos variable is:
['The Claridge Hotel', 'Atlantic club', 'Camelot Hotel', 'owboat']

```

**NOTICE** the 'S', 'h' at the end of truck and the missing Sh at the end of casinos

[2]:

```
truck = []
capacity = 20

casinos = ['The Claridge Hotel', 'Atlantic club', 'Camelot Hotel', 'Showboat', 'Le
↳Jardin', 'The Sands']
#capacity, casinos = 5, ['Regency Hotel'] # truck: ['R', 'e', 'g', 'e', 'n'] ↳
↳casinos: ['cy Hotel']
#capacity, casinos = 2, ['a', 'b', 'c'] # truck: ['c', 'b'] casinos: ['a']

write here
```

[ ]:

## 6.4 Sequences

### 6.4.1 Sequences and comprehensions

[Download exercises zip](#)

[Browse online files<sup>236</sup>](#)

We can write elegant and compact code with sequences. First we will see how to scan sequences with iterators, and then how to build them with comprehensions of lists.

#### What to do

1. Unzip `exercises zip` in a folder, you should obtain something like this:

```
sequences
 sequences1.ipynb
 sequences1-sol.ipynb
 sequences2-chal.ipynb
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `sequences.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter

<sup>236</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/sequences>

- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## Iterables - lists

When dealing with loops we often talk about *iterating* sequences, but what does it exactly mean for a sequence to be *iterable*? Concretely, it means we can call the function `iter` on that sequence.

Let's try for example with familiar lists:

```
[2]: iter(['a', 'b', 'c', 'd'])
[2]: <list_iterator at 0x7f8e886ef8d0>
```

We notice Python just created an object of type `list_iterator`.

**NOTE:** the list was not shown!

You can imagine an iterator as a sort of still machine, that each time it is activated it produces an element from the sequence, one at a time

Typically, an iterator only knows its *position* inside the sequence, and can provide us with the sequence elements one by one if we keep asking with calls to the function `next`:

```
[3]: iterator = iter(['a', 'b', 'c', 'd'])
[4]: next(iterator)
[4]: 'a'
[5]: next(iterator)
[5]: 'b'
[6]: next(iterator)
[6]: 'c'
[7]: next(iterator)
[7]: 'd'
```

Note how the iterator has a *state* to keep track of where it is in the sequence (in other words, it's *stateful*). The state is changed at each call of function `next`.

If we try asking more elements of the available ones, Python raises the exception `StopIteration`:

```
next(iterator)

StopIteration Traceback (most recent call last)
<ipython-input-65-4518bd5da67f> in <module>()
----> 1 next(iterator)

StopIteration:
```

---

V COMMANDMENT<sup>237</sup> You shall never ever redefine `next` and `iter` system functions.

DO NOT use them as variables !!

---

### iterables - range

We iterated a list, which is a completely materialized in memory sequence we scanned with the iterator object. There are also other peculiar sequences which are *not* materialized in memory, like for example `range`.

Previously we used `range` in for loops<sup>238</sup> to obtain a sequence of numbers, but exactly, what is `range` doing? Let's try calling it on its own:

```
[8]: range(4)
```

```
[8]: range(0, 4)
```

Maybe we expected a sequence of numbers, instead, Python is showing us an object of type `range` (with the lower range limit).

**NOTE:** No number sequence is currently present in memory

We only have a 'still' *iterable* object, which if we want can provide us with numbers

How can we ask for numbers?

We've seen we can use a `for` loop:

```
[9]: for x in range(4):
 print(x)
```

```
0
1
2
3
```

As an alternative, we can pass `range` to the function `iter` which produces an *iterator*.

**WARNING:** `range` is iterable but it is NOT an iterator !!

To obtain the iterator we must call the `iter` function on the `range` object

```
[10]: iterator = iter(range(4))
```

`iter` also produces a 'still' object, which hasn't materialized numbers in memory yet:

```
[11]: iterator
```

```
[11]: <range_iterator at 0x7f8e88783030>
```

In order to ask we must use the function `next`:

<sup>237</sup> <https://en.softpython.org/commandments.html#V-COMMANDMENT>

<sup>238</sup> <https://en.softpython.org/for/for1-intro-sol.html#Counting-with-range>

```
[12]: next(iterator)
```

```
[12]: 0
```

```
[13]: next(iterator)
```

```
[13]: 1
```

```
[14]: next(iterator)
```

```
[14]: 2
```

```
[15]: next(iterator)
```

```
[15]: 3
```

Note the iterator has a *state*, which is changed at each `next` call to keep track of where it is in the sequence.

If we try asking for more elements than actually available, Python raises a `StopIteration` exception:

```
next(iterator)
```

```

StopIteration Traceback (most recent call last)
<ipython-input-65-4518bd5da67f> in <module>()
 1 next(iterator)
```

```
StopIteration:
```

## Materializing a sequence

We said a `range` object does not physically materialize in memory all the numbers at the same time. We can get them one by one by only using the iterator. What if we wanted a list with all the numbers? In the tutorial [on lists<sup>239</sup>](#) we've seen that by passing a sequence to function `list`, a new list is created with all the sequence elements. We talked generically about a *sequence*, but the more correct term would have been *iterable*.

If we pass any *iterable* object to `list`, then a new list will be built - we've seen `range` is iterable so let's try:

```
[16]: list(range(4))
```

```
[16]: [0, 1, 2, 3]
```

Voilà ! Now the sequence is all physically present in memory.

**WARNING: `list` consumes the iterator!**

If you try calling twice `list` on the same iterator, you will get an empty list:

```
[17]:
```

```
sequence = range(4)
iterator = iter(sequence)
```

```
[18]:
```

```
new1 = list(iterator)
```

<sup>239</sup> <https://en.softpython.org/lists/lists1-sol.html#Convert-sequences-into-lists>

```
[19]: new1
[19]: [0, 1, 2, 3]
```

```
[20]: new2 = list(iterator)
```

```
[21]: new2
[21]: []
```

What if we wanted to directly access a specific position in the sequence generated by the iterator? Let's try extracting the character at index 2:

```
[22]: sequence = range(4)
iterator = iter(sequence)
```

```
iterator[2]

TypeError Traceback (most recent call last)
<ipython-input-129-3c080cc9e700> in <module>()
 1 sequence = range(4)
 2 iterator = iter(sequence)
----> 3 iterator[3]

TypeError: 'range_iterator' object is not subscriptable
```

... sadly we get an error!

We are left with only two alternatives. Either:

- a) First we convert to list and then use the squared brackets
- b) We call `next` 4 times (remember indexes start from zero)

Option a) very often looks handy, but careful: **converting an iterator into a list creates a NEW list in memory**. If the list is very big and/or this operation is repeated many times, you risk occupying memory for nothing.

Let's see the example in Python Tutor again:

```
[23]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
(it's sufficient to execute it only once)

import jupman
```

```
[24]: sequence = range(4)
iterator = iter(sequence)
new1 = list(iterator)
new2 = list(iterator)

jupman.pytut()
```

```
[24]: <IPython.core.display.HTML object>
```

**QUESTION:** Which object occupies more memory? a or b?

```
a = range(10)
b = range(10000000)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** they both occupy the same amount of memory.

</div>

**QUESTION:** Which object occupies more memory? a or b ?

```
a = list(range(10))
b = list(range(10000000))
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** b occupies more (the list is materialized)

</div>

### Questions - range

Look at the following expressions, and for each try guessing the result (or if it gives an error):

1. `range(3)`2. `range()`3. `list(range(-3))`4. `range(3, 6)`5. `list(range(5, 4))`6. `list(range(3, 3))`7. `range(3) + range(6)`8. `list(range(3)) + list(range(6))`9. `list(range(0, 6, 2))`10. `list(range(9, 6, -1))`

### reversed

`reversed` is a *function* which takes a sequence as parameter and PRODUCES a NEW *iterator* which allows to run through the sequence in reverse order.

WARNING: by calling `reversed` we directly obtain an *iterator* !So you do *not* need to make further calls to `iter` as done with `range`!

Let's have a better look with an example:

```
[25]: la = ['s', 'c', 'a', 'n']
```

```
[26]: reversed(la)
```

```
[26]: <list_reverseiterator at 0x7f8e886ad9d0>
```

We see `reversed` has produced an *iterator* as result (not a reversed list)

---

### INFO: iterators occupy a small amount of memory

Creating an iterator from a sequence only creates a sort of pointer, it *does not* create new memory regions.

---

Furthermore , we see the original list associated to `la` was *not* changed:

```
[27]: print(la)
```

```
['s', 'c', 'a', 'n']
```

**WARNING:** the function `reversed` is different from `reverse` method<sup>240</sup>

Note the final **d!** If we tried to call it as a method we would get an error:

```
>>> la.reversed()

AttributeError Traceback (most recent call last)
<ipython-input-182-c8d1eec57fdd> in <module>
----> 1 la.reversed()

AttributeError: 'list' object has no attribute 'reversed'
```

### Iterating with `next`

How can we obtain a reversed list in memory? In other words, how can we actionate the iterator machine?

We can ask the iterator for one element at a time with the function `next`:

```
[28]: la = ['a', 'b', 'c']
```

```
[29]: iterator = reversed(la)
```

```
[30]: next(iterator)
```

```
[30]: 'c'
```

```
[31]: next(iterator)
```

```
[31]: 'b'
```

---

<sup>240</sup> <https://en.softpython.org/lists/lists3-sol.html#reverse-method>

```
[32]: next(iterator)
[32]: 'a'
```

Once the iterator is exhausted, by calling `next` again we will get an error:

```
next(iterator)

StopIteration Traceback (most recent call last)
<ipython-input-248-4518bd5da67f> in <module>
----> 1 next(iterator)

StopIteration:
```

Let's try manually creating a destination list `lb` and adding elements we obtain one by one:

```
[33]: la = ['a', 'b', 'c']
iterator = reversed(la)
lb = []
lb.append(next(iterator))
lb.append(next(iterator))
lb.append(next(iterator))
print(lb)

jupman.pytut()
['c', 'b', 'a']

[33]: <IPython.core.display.HTML object>
```

### Exercise - sconcerto

Write some code which given a list of characters `la`, puts in a list `lb` all the characters at odd position taken from reversed list `la`.

- use `reversed` and `next`
- **DO NOT** modify `la`
- **DO NOT** use negative indexes
- **DO NOT** use list

Example - given:

```
8 7 6 5 4 3 2 1 0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []
```

After your code it must show:

```
>>> print(lb)
['t', 'e', 'n', 'c']
>>> print(la)
['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
```

We invite you to solve the problem in several ways:

**WAY 1 - without cycle:** Suppose the list length **is fixed**, and repeatedly call `next` *without using a loop*

**WAY 2 - while:** Suppose having a list of arbitrary length, and try generalizing previous code by *using a while cycle*, and calling `next` inside

- **HINT 1:** keep track of the position in which you are with a counter `i`
- **HINT 2:** you cannot call `len` on an iterator, so in the `while` conditions you will have to use the original list length

**WAY 3 - for:** this is the most elegant way. Suppose having a list of arbitrary length and *use a loop* like `for x in reversed(la)`

- **HINT:** you will still need to keep track of the position in which you are with an `i` counter

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[34]:

```
WAY 1: MANUAL

8 7 6 5 4 3 2 1 0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

write here

iterator = reversed(la)

next(iterator)
lb.append(next(iterator))
next(iterator)
lb.append(next(iterator))
next(iterator)
lb.append(next(iterator))
next(iterator)
lb.append(next(iterator))
print(lb)

#jupman.pytut()

['t', 'e', 'n', 'c']
```

</div>

[34]:

```
WAY 1: MANUAL

8 7 6 5 4 3 2 1 0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

write here
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[35]:

```
WAY 2: WHILE

8 7 6 5 4 3 2 1 0
```

(continues on next page)

(continued from previous page)

```

la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

write here

iterator = reversed(la)

i = 1
while i < len(la):
 if i % 2 == 1:
 next(iterator)
 lb.append(next(iterator))
 i += 2

print(lb)

['t', 'e', 'n', 'c']

```

&lt;/div&gt;

[35]:

```

WAY 2: WHILE

8 7 6 5 4 3 2 1 0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

write here

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[36]:

```

WAY 3: for

8 7 6 5 4 3 2 1 0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

write here
i = 0

for x in reversed(la):
 if i % 2 == 1:
 lb.append(x)
 i += 1
print(lb)

['t', 'e', 'n', 'c']

```

&lt;/div&gt;

[36]:

```
WAY 3: for
```

(continues on next page)

(continued from previous page)

```
8 7 6 5 4 3 2 1 0
la = ['s', 'c', 'o', 'n', 'c', 'e', 'r', 't', 'o']
lb = []

write here
```

## Materializing an iterator

Luckily enough, we can obtain a list from an iterator with a less laborious method.

We've seen that when we want to create a new list from a sequence, we can use `list` as if it were a function. We can also do it in this case, interpreting the iterator as if it were a sequence:

```
[37]: la = ['s', 'c', 'a', 'n']
 list(reversed(la))

[37]: ['n', 'a', 'c', 's']
```

Notice we generated a NEW list, the original one associated to `la` is always the same:

```
[38]: la

[38]: ['s', 'c', 'a', 'n']
```

Let's see what happens using Python Tutor (we created some extra variables to evidence relevant passages):

```
[39]: la = ['s', 'c', 'a', 'n']
 iterator = reversed(la)
 new = list(iterator)
 print("la is", la)
 print("new is", new)

jupman.pytut()

la is ['s', 'c', 'a', 'n']
new is ['n', 'a', 'c', 's']

[39]: <IPython.core.display.HTML object>
```

**QUESTION** Which effect is the following code producing?

```
la = ['b', 'r', 'i', 'd', 'g', 'e']
lb = list(reversed(reversed(la)))
```

## sorted

The **function** `sorted` takes as parameter a sequence and returns a NEW sorted list.

**WARNING:** `sorted` returns a LIST, not an iterator!

```
[40]: sorted(['g', 'a', 'e', 'd', 'b'])
[40]: ['a', 'b', 'd', 'e', 'g']
```

**WARNING:** `sorted` is a **function** different from `sort` method<sup>241</sup> !

Note the final **ed!** If we tried to call it with a different method we would get an error:

```
>>> la.sorted()

AttributeError Traceback (most recent call last)
<ipython-input-182-c8d1eec57fdd> in <module>
----> 1 la.reversed()

AttributeError: 'list' object has no attribute 'sorted'
```

## Exercise - reversort

⊕ Given a list of names, write some code to produce a list sorted in reverse

There are at least a couple of ways to do it in a single line of code, find them both

- INPUT: `['Maria', 'Paolo', 'Giovanni', 'Alessia', 'Greta']`
- OUTPUT: `['Paolo', 'Maria', 'Greta', 'Giovanni', 'Alessia']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[41]: # write here
list(sorted(['Maria', 'Paolo', 'Giovanni', 'Alessia', 'Greta'], reverse=True))

or
#list(reversed(sorted(['Maria', 'Paolo', 'Giovanni', 'Alessia', 'Greta'])))
```

```
[41]: ['Paolo', 'Maria', 'Greta', 'Giovanni', 'Alessia']
```

</div>

```
[41]: # write here
```

<sup>241</sup> <https://en.softpython.org/lists/lists3-sol.html#sort-method>

### zip

Suppose we have two lists `paintings` and `years`, with respectively names of famous paintings and the dates in which they were painted:

```
[42]: paintings = ["The Mona Lisa", "The Birth of Venus", "Sunflowers"]
years = [1503, 1482, 1888]
```

We want to produce a new list which contains some tuples which associate each painting with the year it was made:

```
[('The Mona Lisa', 1503),
 ('The Birth of Venus', 1482),
 ('Sunflowers', 1888)]
```

There are various ways to do it but certainly the most elegant is by using the **function** `zip` which produces an **iterator**:

```
[43]: zip(paintings, years)
[43]: <zip at 0x7f8e88550c80>
```

Even if you don't see written 'iterator' in the object name, we can still use it as such with `next`:

```
[44]: iterator = zip(paintings, years)
next(iterator)
[44]: ('The Mona Lisa', 1503)

[45]: next(iterator)
[45]: ('The Birth of Venus', 1482)

[46]: next(iterator)
[46]: ('Sunflowers', 1888)
```

As done previously, we can convert everything to a list with `list`:

```
[47]: paintings = ["The Mona Lisa", "The Birth of Venus", "Sunflowers"]
years = [1503, 1482, 1888]

list(zip(paintings, years))
[47]: [('The Mona Lisa', 1503), ('The Birth of Venus', 1482), ('Sunflowers', 1888)]
```

If the lists have different length, the sequence produced by `zip` will be as long as the shortest input sequence:

```
[48]: list(zip([1,2,3], ['a','b','c','d','e']))
[48]: [(1, 'a'), (2, 'b'), (3, 'c')]
```

If we will, we can pass an arbitrary number of sequences - for example, by passing three of them we will obtain triplets of values:

```
[49]: songs = ['Imagine', 'Hey Jude', 'Satisfaction', 'Yesterday']
authors = ['John Lennon', 'The Beatles', 'The Rolling Stones', 'The Beatles']
years = [1971, 1968, 1965, 1965]
list(zip(songs, authors, years))
```

```
[49]: [('Imagine', 'John Lennon', 1971),
 ('Hey Jude', 'The Beatles', 1968),
 ('Satisfaction', 'The Rolling Stones', 1965),
 ('Yesterday', 'The Beatles', 1965)]
```

### Exercise - ladder

Given a number  $n$ , create a list of tuples that for each integer number  $x$  such that  $0 \leq x \leq n$  associates the number  $n - x$

- INPUT:  $n=5$
- OUTPUT:  $[(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]$

Show solution  
Hide>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[50]: n = 5
write here
list(zip(range(n), reversed(range(n))))
```

```
[50]: [(0, 4), (1, 3), (2, 2), (3, 1), (4, 0)]
```

</div>

```
[50]: n = 5
write here
```

### List comprehensions

List comprehensions are handy when you need to generate a NEW list by executing the same operation on all the elements of a sequence. Comprehensions start and end with square brackets [ ] so their syntax reminds lists, but inside they contain a special `for` to loop inside a sequence:

```
[51]: numbers = [2, 5, 3, 4]

doubled = [x*2 for x in numbers]

doubled
```

```
[51]: [4, 10, 6, 8]
```

Note the variable `numbers` is still associated to the original list:

```
[52]: numbers
```

```
[52]: [2, 5, 3, 4]
```

What happened ? We wrote the name of a variable `x` we just invented, and we told Python to go through the list `numbers`: at each iteration, the variable `x` is associated to a different value of the list `numbers`. This value can be reused in the expression we wrote on left of the `for`, which in this case is `x*2`

As name for the variable we used `x`, but we could have used any other name, for example this code is equivalent to the previous one:

```
[53]: numbers = [2, 5, 3, 4]

doubled = [number * 2 for number in numbers]

doubled
[53]: [4, 10, 6, 8]
```

On the left of the `for` we can write any expression which produces a value, for example here we write `x + 1` to increment all the numbers of the original list:

```
[54]: numbers = [2, 5, 3, 4]

augmented = [x + 1 for x in numbers]

augmented
[54]: [3, 6, 4, 5]
```

**QUESTION:** What is this code going to produce? If we visualize it in Python Tutor, will `la` and `lb` point to different objects?

```
la = [7, 5, 6, 9]
lb = [x for x in la]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** When `[x for x in la]` is executed, during the first iteration `x` is valued 7, during the second 5, during the third one 6 and so on and so forth. In the expression on the left of the `for` we put only `x`, so as expression result we will get the same identical number taken from the original string.

The code will produce a NEW list `[7, 5, 6, 9]` and it will be associated to the variable `lb`.

</div>

```
[55]: la = [7, 5, 6, 9]
lb = [x for x in la]

jupman.pytut()

[55]: <IPython.core.display.HTML object>
```

### List comprehensions on strings

**QUESTION:** What is this code going to produce?

```
[x for x in 'question']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** It will produce `['q', 'u', 'e', 's', 't', 'i', 'o', 'n']`

Since `question` is a string, if we interpret it as a sequence each element of it is a character, so during the first iteration `x` is valued `'q'`, during the second `'u'`, during the third `'e'` and so on and so forth. In the expression on the left of the `for` we put only `x`, so as expression result we will obtain the same identical character taken from the original string.

</div>

Let's now suppose to have a list of animals and we want to produce another one with the same names as uppercase. We can do it in a compact way with a list comprehension like this:

```
[56]: animals = ['dogs', 'cats', 'squirrels', 'elks']

new_list = [animal.upper() for animal in animals]
```

```
[57]: new_list
[57]: ['DOGS', 'CATS', 'SQUIRRELS', 'ELKS']
```

In the left part reserved to the expression we used the method `.upper()` on the string variable `animal`. We know strings are immutable, so we're sure the method call produces a NEW string. Let's see what happened with Python Tutor:

```
[58]: animals = ['dogs', 'cats', 'squirrels', 'elks']

new_list = [animal.upper() for animal in animals]

jupman.pytut()

[58]: <IPython.core.display.HTML object>
```

⊕ **EXERCISE:** Try writing here a list comprehension to put all characters as lowercase (`.lower()` method)

Show solutionShow solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[59]: animals = ['doGS', 'caTS', 'SQUIrreLs', 'ELks']

write here

[animal.lower() for animal in animals]
[59]: ['dogs', 'cats', 'squirrels', 'elks']
```

</div>

```
[59]: animals = ['doGS', 'caTS', 'SQUIrreLs', 'ELks']

write here
```

## Questions - List comprehensions

Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

1. `[x for [4,2,5]]`

2. `x for x in range(3)`

3. `[x for y in 'cartoccio']`

4. [for x in 'zappa']
5. [for [3,4,5]]
6. [k + 1 for k in 'bozza']
7. [k + 1 for k in range(5)]
8. [k > 3 for k in range(7)]
9. [s + s for s in ['lam','pa','da']]
10. la = ['x','z','z']
   
[x for x in la] + [y for y in la]
11. [x.split('-') for x in ['a-b', 'c-d', 'e-f']]
12. ['@'.join(x) for x in [['a','b.com'],['c','d.org'],['e','f.net']] ]
13. ['z' for y in 'borgo'].count('z') == len('borgo')
14. m = [['a','b'],['c','d'],['e','f'] ]
   
la = [x.pop() for x in m] # not advisable - why ?
   
print('m:', m)
   
print('la:', la)

## Exercises - list comprehension

### Exercise - Bubble bubble

⊕ Given a list of strings, produce a sequence with all the strings replicated 4 times

- INPUT: ['chewing', 'gum', 'bubble']
- OUTPUT: ['chewingchewingchewingchewing', 'gumgumgumgum', 'bubblebubblebubblebubble']

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[60]:

```
import math

bubble_bubble = ['chewing', 'gum', 'bubble']

write here
[x*4 for x in bubble_bubble]

[60]: ['chewingchewingchewingchewing', 'gumgumgumgum', 'bubblebubblebubblebubble']
```

</div>

[60]:

```
import math

bubble_bubble = ['chewing', 'gum', 'bubble']

write here
```

**Exercise - root**

⊕ Given a list of numbers, produce a list with the square root of the input numbers

- INPUT: [16, 25, 81]
- OUTPUT: [4.0, 5.0, 9.0]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[61]:

```
import math

write here
[math.sqrt(x) for x in [16, 25, 81]]
```

[61]:

```
[4.0, 5.0, 9.0]
```

```
</div>
```

[61]:

```
import math

write here
```

**Exercise - When The Telephone Rings**

⊕ Given a list of strings, produce a list with the first characters of each string

- INPUT: ['When', 'The', 'Telephone', 'Rings']
- OUTPUT: ['W', 'T', 'T', 'R']

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[62]:

```
write here
[x[0] for x in ['When', 'The', 'Telephone', 'Rings']]
```

[62]:

```
['W', 'T', 'T', 'R']
```

```
</div>
```

[62]:

```
write here
```

### Exercise - don't worry

⊕ Given a list of strings, produce a list with the lengths of all the lists

- INPUT: ["don't", 'worry', 'and', 'be', 'happy']
- OUTPUT: [5, 5, 3, 2, 5]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[63]:

```
write here
[len(x) for x in ["don't", 'worry', 'and', 'be', 'happy']]
```

[63]:

```
[5, 5, 3, 2, 5]
```

```
</div>
```

[63]:

```
write here
```

### Exercise - greater than 3

⊕ Given a list of numbers, produce a list with True if the corresponding element is greater than 3, False otherwise

- INPUT: [4, 1, 0, 5, 0, 9, 1]
- OUTPUT: [True, False, False, True, False, True, False]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[64]:

```
write here
[x > 3 for x in [4, 1, 0, 5, 0, 9, 1]]
```

[64]:

```
[True, False, False, True, False, True, False]
```

```
</div>
```

[64]:

```
write here
```

### Exercise - even

⊕ Given a list of numbers, produce a list with True if the corresponding element is even

- INPUT: [3, 2, 4, 1, 5, 3, 2, 9]
- OUTPUT: [False, True, True, False, False, False, True, False]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[65]:

```
write here
[x % 2 == 0 for x in [3, 2, 4, 1, 5, 3, 2, 9]]
```

[65]:

```
[False, True, True, False, False, False, True, False]
```

</div>

[65]:

```
write here
```

### Exercise - both ends

⊕ Given a list of strings having at least two characters each, produce a list of strings with the first and last characters of each

- INPUT: ['departing', 'for', 'the', 'battlefront']
- OUTPUT: ['dg', 'fr', 'te', 'bt']

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[66]:

```
write here
[x[0] + x[-1] for x in ['departing', 'for', 'the', 'battlefront']]
```

[66]:

```
['dg', 'fr', 'te', 'bt']
```

</div>

[66]:

```
write here
```

**Exercise - dashes**

⊕ Given a list of lists of characters, produce a list of strings with characters separated by dashes

- INPUT: `[['a', 'b'], ['c', 'd', 'e'], ['f', 'g']]`
- OUTPUT: `['a-b', 'c-d-e', 'f-g']`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[67]:

```
write here
['-'.join(x) for x in [['a', 'b'], ['c', 'd', 'e'], ['f', 'g']]]
```

[67]:

```
['a-b', 'c-d-e', 'f-g']
```

```
</div>
```

[67]:

```
write here
```

**Exercise - lollosa**

⊕ Given a string `s`, produce a list of tuples having for each character the number of occurrences of that character in the string

- INPUT: `s = 'lollosa'`
- OUTPUT: `[('l', 3), ('o', 2), ('l', 3), ('l', 3), ('o', 2), ('s', 1), ('a', 1)]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[68]:

```
s = 'lollosa'
write here
[(car, s.count(car)) for car in s]
```

[68]:

```
[('l', 3), ('o', 2), ('l', 3), ('l', 3), ('o', 2), ('s', 1), ('a', 1)]
```

```
</div>
```

[68]:

```
s = 'lollosa'
write here
```

## Exercise - dog cat

⊕ Given a list of strings of at least two characters each, produce a list with the strings without intial and final characters

- INPUT: ['donkey', 'eagle', 'ox', 'dog']
- OUTPUT: ['onke', 'agl', '', 'o']

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[69]:

```
write here
[x[1:-1] for x in ['donkey', 'eagle', 'ox', 'dog']]
```

[69]:

```
['onke', 'agl', '', 'o']
```

</div>

[69]:

```
write here
```

## Exercise - smurfs

⊕ Given some names produce a list with the names sorted alphabetically and all in uppercase

- INPUT: ['Brainy', 'Hefty', 'Smurfette', 'Clumsy']
- OUTPUT: ['BRAINY', 'CLUMSY', 'HEFTY', 'SMURFETTE']

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[70]:

```
write here
[x.upper() for x in sorted(['Brainy', 'Hefty', 'Smurfette', 'Clumsy'])]
```

[70]:

```
['BRAINY', 'CLUMSY', 'HEFTY', 'SMURFETTE']
```

</div>

[70]:

```
write here
```

### Exercise - precious metals

⊕ Given two lists `values` and `metals` produce a list containing all the couples value-metal as tuples

INPUT:

```
values = [10, 25, 50]
metals = ['silver', 'gold', 'platinum']
```

OUTPUT: `[(10, 'silver'), (25, 'gold'), (50, 'platinum')]`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[71]:

```
values = [10, 25, 50]
metals = ['silver', 'gold', 'platinum']

write here
list(zip(values, metals))
```

[71]:

```
[(10, 'silver'), (25, 'gold'), (50, 'platinum')]
```

</div>

[71]:

```
values = [10, 25, 50]
metals = ['silver', 'gold', 'platinum']

write here
```

### Filtered list comprehensions

During the construction of a list comprehension we can filter the elements taken from the sequence by using an `if`. For example, the following expression takes from the sequence only numbers greater than 5:

[72]:

```
[x for x in [7, 4, 8, 2, 9] if x > 5]
```

[72]:

```
[7, 8, 9]
```

After the `if` we can put any expression which reuses the variable on which we are iterating, for example if we are iterating a string we can keep only the uppercase characters:

[73]:

```
[x for x in 'The World Goes Round' if x.isupper()]
```

[73]:

```
['T', 'W', 'G', 'R']
```

**WARNING: `else` is not supported**

For example, writing this generates an error:

```
[x for x in [7, 4, 8, 2, 9] if x > 5 else x + 1] # WRONG!
```

(continues on next page)

(continued from previous page)

```
File "<ipython-input-74-9ba5c135c58c>", line 1
 [x for x in [7,4,8,2,9] if x > 5 else x + 1]
 ^
SyntaxError: invalid syntax
```

## Questions - filtered list comprehensions

Look at the following code fragments, and for each try guessing the result it produces (or if it gives an error):

1. `[x for x in range(100) if False]`
2. `[x for x in range(3) if True]`
3. `[x for x in range(6) if x > 3 else 55]`
4. `[x for x in range(6) if x % 2 == 0]`
5. `[x for x in {'a','b','c'}] # careful about ordering`
6. `[x for x in [[5], [2,3], [4,2,3], [4]] if len(x) > 2]`
7. `[(x,x) for x in 'xyxyxxy' if x != 'x' ]`
8. `[x for x in ['abCdEFg'] if x.upper() == x]`
9. `la = [1,2,3,4,5]
[x for x in la if x > la[len(la)//2]]`

## Exercises - filtered list comprehensions

### Exercise - savannah

Given a list of strings, produce a list with only the strings of length greater than 6:

- INPUT: `['zebra', 'leopard', 'giraffe', 'gnu', 'rhinoceros', 'lion']`
- OUTPUT: `['leopard', 'giraffe', 'rhinoceros']`

Show solution  
 data-jupman-hide="Hide">>Show solution</a><div class="jupman-sol-jupman-sol-code" style="display:none">

[74]:

```
write here
[x for x in ['zebra', 'leopard', 'giraffe', 'gnu', 'rhinoceros', 'lion'] if len(x) >_
 ↪6]
[74]: ['leopard', 'giraffe', 'rhinoceros']
```

</div>

[74]:

```
write here
```

### Exercise - puZZled

Given a list of strings, produce a list with only the strings which contain at least a 'z'. The selected strings must be transformed so to place the z in uppercase.

- INPUT: ['puzzled', 'park', 'Aztec', 'run', 'mask', 'zodiac']
- OUTPUT: ['puZZled', 'AZtec', 'Zodiac']

[75]:

```
[x.replace('z', 'Z') for x in ['puzzled', 'park', 'Aztec', 'run', 'mask', 'zodiac'] if
↪'z' in x]
```

### Exercise - Data science

Produce a string with the words of the input string alternated uppercase / lowercase

- INPUT:

[76]: phrase = """Data science is an interdisciplinary field  
that uses scientific methods, processes, algorithms and systems  
to extract knowledge and insights from noisy, structured  
and unstructured data, and apply knowledge and actionable insights  
from data across a broad range of application domains."""

- OUTPUT (only one line):

```
DATA science IS an INTERDISCIPLINARY field THAT uses SCIENTIFIC methods,
↪PROCESSES, algorithms AND systems TO extract KNOWLEDGE and INSIGHTS from NOISY,
↪structured AND unstructured DATA, and APPLY knowledge AND actionable INSIGHTS
↪from DATA across A broad RANGE of APPLICATION domains.
```

⊕⊕⊕ WRITE ONLY ONE code line

⊕⊕⊕⊕ USE ONLY ONE list comprehension

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[77]:

```
phrase = """Data science is an interdisciplinary field
that uses scientific methods, processes, algorithms and systems
to extract knowledge and insights from noisy, structured
and unstructured data, and apply knowledge and actionable insights
from data across a broad range of application domains."""

write here

print(' '.join(
 [t[0].upper() + ' ' + t[1] for t in zip(phrase.split() [::2], phrase.split() [1::
↪2])))
```

(continues on next page)

(continued from previous page)

```
or
#print(' '.join([phrase.split()[i].upper() + ' ' + phrase.split()[i + 1] for i in
#range(0, len(phrase.split())-1, 2)]))

DATA science IS an INTERDISCIPLINARY field THAT uses SCIENTIFIC methods, PROCESSES,
algorithms AND systems TO extract KNOWLEDGE and INSIGHTS from NOISY, structured AND
unstructured DATA, and APPLY knowledge AND actionable INSIGHTS from DATA across A
broad RANGE of APPLICATION domains.
```

&lt;/div&gt;

[77]:

```
phrase = """Data science is an interdisciplinary field
that uses scientific methods, processes, algorithms and systems
to extract knowledge and insights from noisy, structured
and unstructured data, and apply knowledge and actionable insights
from data across a broad range of application domains."""
write here
```

## Continue

Go on with the challenges<sup>242</sup>

[ ]:

### 6.4.2 Sequences 2 - Challenges

[Download exercises zip](#)

Browse online files<sup>243</sup>

#### Challenge - ppulsar

Given a list of words universe, output a list which has first letters capitalized, double 'p's converted into a single one, and 'The' added at the beginning of each word

- USE list comprehension in one line of code

IN: universe = ['star', 'space', 'nebula', 'proton', 'pproppulsion', 'black hole', 'ppulsar']

OUT: ['The Star', 'The Space', 'The Nebula', 'The Proton', 'The Propulsion', 'The Black hole', 'The Pulsar']

<sup>242</sup> <https://en.softpython.org/sequences/sequences2-chal.html>

<sup>243</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/sequences>

### Challenge - broken wisdom

Given a list of words, produce a list of tuples of characters, with half the characters of each word.

- **USE** list comprehension in one line of code
- **BEWARE** of odd length words

IN: ['concepts', 'broken', 'wisdom', 'reality', 'unbroken', 'philosophy']  
OUT: [('c', 'o', 'n', 'c'), ('b', 'r', 'o'), ('w', 'i', 's'), ('r', 'e', 'a'), ('u', 'n', 'b', 'r'), ('p', 'h', 'i', 'l', 'o')]

[2]:

```
meditation = ['concepts', 'broken', 'wisdom', 'reality', 'unbroken', 'philosophy']
write here
```

### Challenge - rupture

Given a list of words `rupture` either lowercase or uppercase, and a list of words `gather` either uppercase or lowercase, collect only those words from `rupture` and separated the characters with dashes –

- **USE** list comprehension in one line of code

IN:

```
gather = ['TREMOR', 'hypocenter', 'DANGER', 'seismic']`
rupture = ['earthquake', 'tremor', 'temblor', 'litosphere', 'danger', 'TREMOR',
 'hypocenter', 'seismic', 'waves', 'LANDSLIDES', 'SEISMIC']
```

OUT: ['t-r-e-m-o-r', 'd-a-n-g-e-r', 't-r-e-m-o-r', 'h-y-p-o-c-e-n-t-e-r',  
's-e-i-s-m-i-c', 's-e-i-s-m-i-c']

[3]:

```
gather = ['TREMOR', 'hypocenter', 'DANGER', 'seismic']
rupture = ['earthquake', 'tremor', 'temblor', 'litosphere', 'danger', 'TREMOR',
 ↴'hypocenter', 'seismic', 'waves', 'LANDSLIDES', 'SEISMIC']

write here
```

## Challenge - cyclone

Given a list of words, outputs a list of tuples coupling a word position with the word itself

- **USE** list comprehension in one line of code
- **DO NOT** use search methods (no .index, .find, ...)

IN: ['rotating', 'storm', 'pressure', 'tropical', 'typhoon', 'strong winds', 'severe']

OUT:

```
[(0, 'rotating'), (1, 'storm'), (2, 'pressure'), (3, 'tropical'), (4, 'pressure
↔'),
(5, 'typhoon'), (6, 'typhoon'), (7, 'strong winds'), (8, 'severe'), (9, 'typhoon
↔')]
```

[4]:

```
cyclone = ['rotating', 'storm', 'pressure', 'tropical', 'pressure', 'typhoon', 'typhoon
↔', 'strong winds', 'severe', 'typhoon']

write here
```



## A3 BASIC ALGORITHMS

### 7.1 Functions, error handling and testing

#### 7.1.1 Functions 1 - introduction

**Download exercises zip**

Browse files online<sup>244</sup>

#### Introduction

A function is some code which takes some parameters and uses them to produce or report some result.

In this notebook we will see how to define functions to reuse code, and talk about variables scope.

**WARNING: this tutorial is not really complete**

For more info see:

- Andrea Passerini slides A04<sup>245</sup>

#### What to do

- unzip exercises in a folder, you should get something like this:

```
functions
 fun1-intro.ipynb
 fun1-intro-sol.ipynb
 fun2-errors-and-testing.ipynb
 fun2-errors-and-testing-sol.ipynb
 fun3-strings.ipynb
 fun3-strings-sol.ipynb
 fun4-lists.ipynb
 fun4-lists-sol.ipynb
 fun5-tuples.ipynb
 fun5-tuples-sol.ipynb
 fun6-sets.ipynb
```

(continues on next page)

<sup>244</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/functions>

<sup>245</sup> <http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A04-functions.pdf>

(continued from previous page)

```
fun6-sets-sol.ipynb
fun7-dictionaries.ipynb
fun7-dictionaries-sol.ipynb
fun8-chal.ipynb
jupman.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `functions/fun1-intro.ipynb`
- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

### Why functions?

We may need functions for a lot of reasons, including:

1. *Reduce code duplication:* put in functions parts of code that are needed several times in the whole program, so you don't need to repeat the same code over and over again;
2. *Decompose a complex task:* make the code easier to write and understand by splitting the whole program in several easier functions;

### Function definition - questions

For each of the following expressions, try guessing the result it produces (or if it gives error)

1. `def f():  
 print('car')  
 print(f())`

2. `def f():  
 print('car')  
 print(f())`

3. `def f():  
 return 3  
 print(f())`

4. `def f():  
 return 3  
 print(f())`

```
5. def f():
 return 3
print(f())
```

```
6. def f():
 return 3
print(f() f())
```

```
7. def f():
 return 3
print(f() *f())
```

```
8. def f():
 pass
print(f())
```

```
9. def f(x):
 return x
print(f())
```

```
10. def f(x):
 return x
print(f(5))
```

```
11. def f():
 print('fire')
x = f()
print(x)
```

```
12. def f():
 return(print('fire'))
print(f())
```

```
13. def f(x):
 return 'x'
print(f(5))
```

```
14. def f(x):
 return x
print(f(5))
```

```
15. def etc():
 print('etc...')
 return etc()
etc()
```

```
16. def gu():
 print('GU')
 ru()
def ru():
 print('RU')
 gu()
gu()
```

### Different function kinds

You can roughly find 5 different function kinds in the wild:

1. PRODUCES *SIDE EFFECTS*: PRINTS / ASKS MANUAL INPUT / WRITES by modifying the environment in some way - examples: printing characters on the screen, asking interactively input from the user, writing into a file
2. RETURNS a value, either as NEW memory region or a pointer to an existing memory region
3. MODIFIES the input
4. MODIFIES the input and RETURNS it (allows for call *chaining*)
5. MODIFIES the input and RETURNS something derived from it

Let's try now to understand the differences with various examples.

### SIDE EFFECTS

Only PRINTS / ASKS INTERACTIVE INPUT / WRITES INTO A FILE

- DOES NOT modify the input!
- DOES NOT return anything!

Example:

```
[2]: def printola(lst):
 """PRINTS the first two elements of the given list
 """
 print('The first two elements are', lst[0], lst[1])

la = [8,5,6,2]

printola(la)
jupman.pyput()

The first two elements are 8 5
[2]: <IPython.core.display.HTML object>
```

### RETURN

RETURN some value, either as NEW memory region or a pointer to an existing memory region according to the function text

- DOES NOT modify the input
- DOES NOT print anything!

Example:

```
[3]: def returnola(lst):
 """RETURN a NEW list having all the numbers doubled
 """
 ret = []
 for el in lst:
 ret.append(el*2)
 return ret
```

(continues on next page)

(continued from previous page)

```
[3]: la = [5,2,6,3]
res = returnola(la)
print("la :", la)
print("res:", res)
jupman.pytut()

la : [5, 2, 6, 3]
res: [10, 4, 12, 6]

[3]: <IPython.core.display.HTML object>
```

## MODIFY

MODIFY the input. By MODIFYING, we typically mean changing data inside *existing* memory regions, limiting as much as possible the creation of new ones.

- DOES NOT return anything!
- DOES NOT print anything!
- DOES NOT create new memory regions (or limits the creation to the bare needed)

Example:

```
[4]: def modifanta(lst):
 """MODIFIES lst by ordering it in-place
 """
 lst.sort()
 la = [43434]

 la = [7,4,9,8]

 modifanta(la)

 print("la:", la)
 jupman.pytut()

la: [4, 7, 8, 9]

[4]: <IPython.core.display.HTML object>
```

## MODIFY and RETURN

MODIFIES the input and RETURNS a pointer to it

- DOES NOT print anything!
- DOES NOT create new memory regions (or limits the creation to the bare needed)

Note: allows call *chaining*

```
[5]: def modiret(lst):
 """MODIFY lst by doubling all its elements, and finally RETURNS it
 """
 for i in range(len(lst)):
```

(continues on next page)

(continued from previous page)

```
lst[i] = lst[i] * 2
return lst

la = [8, 7, 5]
res = modiret(la)
print("res :", res) # [16, 14, 10] RETURNED the modified input
print("la :", la) # [16, 14, 10] la input was MODIFIED !!

print()
lb = [7, 5, 6]
modiret(lb).reverse() # NOTE WE CAN CONCATENATE
print("lb :", lb) # [12, 10, 14] lb input was MODIFIED !!
#modiret(lb).reverse().append(16) # ... but this wouldn't work. Why?

jupman.pytut()

res : [16, 14, 10]
la : [16, 14, 10]

lb : [12, 10, 14]
```

[5]: <IPython.core.display.HTML object>

## MODIFY AND RETURN A PART

MODIFY the input and RETURN a part of it

- DOES NOT print anything!

```
[6]: def modirip(lst):
 """MODIFY lst by sorting it and removing the greatest element. Finally, RETURN
 ↪the removed element.
 """
 lst.sort()
 ret = lst[-1]
 lst.pop()
 return ret

la = ['b', 'c', 'a']
res = modirip(la)
print("res :", res) # 'c' RETURNED a piece of the input
print("la :", la) # ['a', 'b'] la was MODIFIED!!
jupman.pytut()

res : c
la : ['a', 'b']
```

[6]: <IPython.core.display.HTML object>

## Remember the commandments

### III COMMANDMENT

#### You shall never ever reassign function parameters

Never perform any of these assignments, as you risk losing the parameter passed during function call:

```
[7]: def sin(my_int):
 my_int = 666 # you lost the 5 passed from external call!
 print(my_int) # prints 666

x = 5
sin(x)

666
```

Same reasoning can be applied to all other types:

```
[8]: def evil(my_string):
 my_string = "666"
```

```
[9]: def disgrace(my_list):
 my_list = [666]
```

```
[10]: def delirium(my_dict):
 my_dict = {"evil":666}
```

For the sole case when you have composite parameters like lists or dictionaries, you can write like below IF AND ONLY IF the function description requires to MODIFY the internal elements of the parameter (like for example sorting a list in-place or changing the field of a dictionary).

```
[11]: # MODIFY my_list in some way
def allowed(my_list):
 my_list[2] = 9 # OK, function text requires it

outside = [8,5,7]
allowed(outside)
print(outside)

[8, 5, 9]
```

```
[12]: # MODIFY dictionary in some way
def ok(dictionary):
 dictionary["my field"] = 5 # OK, function text requires it
```

```
[13]: # MODIFY instance in some way
def fine(class_instance):
 class_instance.my_field = 7 # OK, function text requires it
```

On the other hand, if the function requires to RETURN a NEW object, you shall not fall into the temptation of modifying the input:

```
[14]: # RETURN a NEW sorted list
def pain(my_list):
 my_list.sort() # BAD, you are modifying the input list instead of creating a
 ↪new one!
 return my_list
```

```
[15]: # RETURN a NEW list
def crisis(my_list):
 my_list[0] = 5 # BAD, as above
 return my_list
```

```
[16]: # RETURN a NEW dictionary
def torment(my_dict):
 my_dict['a'] = 6 # BAD, you are modifying the input dictionary instead of
 ↪creating a new one!
 return my_dict
```

```
[17]: # RETURN a NEW class instance
def desperation(my_instance):
 my_instance.my_field = 6 # BAD, you are modifying the input object
 # instead of creating a new one!
 return my_instance
```

## IV COMMANDMENT

---

### You shall never ever reassign values to function calls or methods

---

*WRONG:*

```
my_function() = 666
my_function() = 'evil'
my_function() = [666]
```

*CORRECT:*

```
x = 5
y = my_fun()
z = []
z[0] = 7
d = dict()
d["a"] = 6
```

Function calls like `my_function()` return calculations results and store them in a box in memory which is only created for the purposes of the call, and Python will not allow us to reuse it like it were a variable.

Whenever you see `name()` in the left part, it *cannot* be followed by the equality sign `=` (but it can be followed by two equals sign `==` if you are doing a comparison).

## V COMMANDMENT

### You shall never ever redefine system functions

Python has several system defined functions. For example `list` is a Python type: as such, you can use it for example as a function to convert some type to a list:

```
[18]: list("ciao")
[18]: ['c', 'i', 'a', 'o']
```

When you allow the forces of evil to take the best of you, you might be tempted to use reserved words like `list` as a variable for your own miserable purposes:

```
list = ['my', 'pitiful', 'list']
```

Python allows you to do so, but we do **not**, for the consequences are disastrous.

For example, if you now attempt to use `list` for its intended purpose like casting to list, it won't work anymore:

```
list("ciao")

TypeError Traceback (most recent call last)
<ipython-input-4-c63add832213> in <module>()
 1 list("ciao")
----> 1 TypeError: 'list' object is not callable
```

In particular, we recommend to **not redefine** these precious functions:

- `bool, int, float, tuple, str, list, set, dict`
- `max, min, sum`
- `next, iter`
- `id, dir, vars, help`

## Immutable values

Basic types such integers, float, booleans are immutable, as well as some sequences like strings and tuples : when you are asked to RETURN one of these types, say a string, the only thing you can do is obtaining NEW strings based upon the parameters you receive. Let's see an example.

Suppose we are asked to implement this function:

Write a function `my_upper` which RETURNS the passed string as uppercase.

We could implement it like this:

```
[19]: # Run this cell to have Python Tutor working
import jupman;
```

```
[20]: external_string = "sailor"

def my_upper(s):
 ret = s.upper() # string methods create NEW string
 return ret

result = my_upper(external_string)

print(' result:', result)
print('external_string:', external_string)

jupman.pytut()

 result: SAILOR
external_string: sailor

[20]: <IPython.core.display.HTML object>
```

Notice some things:

- the `external_string` didn't change
- we didn't write `s =` inside the function body, as the **IV COMMANDMENT**<sup>246</sup> prescribes not to reassign parameters
- we didn't refer to `external_string` inside the function body: doing so would have defeated the purpose of functions, which is to isolate them from outside world.

### Changing the world: fail / 1

What if we actually did want to change the assignment of `external_string`?

You might be tempted to write something like an assignment `s =` right inside the function. The following code will **not** work.

**QUESTION:** Why? Try to answer before checking execution in Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

**ANSWER:**

`s = s.upper()` only MODIFIES the assignment `s` of the function call frame, it has no effect on outside world

`</div>`

```
[21]: external_string = "sailor"

def my_upper(s):
 s = s.upper()
 return s

result = my_upper(external_string)

print(' result:', result)
print('external_string:', external_string)

jupman.pytut()
```

<sup>246</sup> <https://en.softpython.org/commandments.html#IV-COMMANDMENT>

```

 result: SAILOR
external_string: sailor
[21]: <IPython.core.display.HTML object>

```

### Changing the world: fail / 2

Let's see another temptation. You might try to assign `external_string = right` inside the function. The following code again will **not** work

**QUESTION:** Why? Try to answer before checking execution in Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:**

```
external_string = s.upper() creates NEW variable external_string inside frame of function call
</div>
```

```

[22]: external_string = "sailor"

def my_upper(s):
 external_string = s.upper()
 return external_string

result = my_upper(external_string)

print(' result:', result)
print('external_string:', external_string)

jupman.pytut()
 result: SAILOR
external_string: sailor
[22]: <IPython.core.display.HTML object>

```

### Changing the world: success!

The proper way to tackle the problem is to create a NEW string inside the function, return it, and then **outside** the function perform the assignment `external_string = result`.

```

[23]: external_string = "sailor"

def my_upper(s):
 ret = s.upper()
 return ret

result = my_upper(external_string)
external_string = result # reassigues *outside*

print(' result:', result)
print('external_string:', external_string)

jupman.pytut()

```

```
result: SAILOR
external_string: SAILOR
[23]: <IPython.core.display.HTML object>
```

### global keyword

If we really wanted to modify `external_string` association inside the function, we could still do it with `global`<sup>247</sup> keyword, but typically it's best to avoid using it to keep the code clean.

### Mutable values

Sequences like lists, sets, dictionaries are *mutable* objects. When you call a function and pass one of these objects, Python actually gives the function only a *reference* to the object: a very small pointer which is just an arrow pointing to the memory region where the actual object resides. Since the function only receives a small pointer, calling the function is a fast operation. On the other side, we need to be aware that since no copy of the whole data structure is performed, inside the function it will be like operating on the original memory region which lives outside the function call.

All of this may feel like a bit of a mouthful. Let's see a practical example in Python Tutor.

Let's say we need to implement this function:

Write a function which takes a list and MODIFIES it by doubling all of its numbers

Note in the text we used the word MODIFIES, meaning we really want to change the original memory region of the external object we are given.

As simple as it might seem, there are many ways to get this wrong. Let's see some.

### Doubling: fail / 1

You might be tempted to solve the problem like the following code, but it will **not** work.

**QUESTION:** Why? Try to answer before checking execution in Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** element is created as a NEW variable in the function call frame, doesn't change original cells

</div>

```
[24]: external_numbers = [10, 20, 30]

def double(lst):
 for element in lst:
 element = element * 2

double(external_numbers)

jupman.pytut()

[24]: <IPython.core.display.HTML object>
```

<sup>247</sup> [https://www.w3schools.com/python/python\\_variables\\_global.asp](https://www.w3schools.com/python/python_variables_global.asp)

## Doubling: fail / 2

You might have another temptation to solve the problem like the following code, but again it will **not** work.

**QUESTION:** Why? Try to answer before checking execution in Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); " data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:**

- `tmp = []` creates a NEW list, but function text tells to MODIFY
- `lst = tmp` Violates **IV COMANDMENT**<sup>248</sup>: sets the variable `lst` in the call frame to point to `tmp`, but both will be lost after the function call is over

</div>

```
[25]: external_numbers = [10, 20, 30]

def double(lst):
 tmp = []
 for element in lst:
 tmp.append(element * 2)

 lst = tmp

double(external_numbers)

jupman.pytut()
```

[25]: <IPython.core.display.HTML object>

## Doubling: fail / 3

You might be tempted to solve the problem also like in the following code, but again it will **not** work.

**QUESTION:** Why? Try to answer before checking execution in Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); " data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:**

- `tmp = []` creates a NEW list, but function text tells to MODIFY
- `external_numbers = tmp` creates a NEW association `external_numbers` inside function call frame, but it will be lost after the function call is over

</div>

```
[26]: external_numbers = [10, 20, 30]

def double(lst):
 tmp = []
 for element in lst:
 tmp.append(element * 2)
```

(continues on next page)

<sup>248</sup> <https://en.softpython.org/commandments.html#IV-COMMANDMENT>

(continued from previous page)

```
external_numbers = tmp

double(external_numbers)

jupman.pytut()

[26]: <IPython.core.display.HTML object>
```

### Doubling: fail / 4

Let's see the final temptation, which yet again will **not** work.

**QUESTION:** Why? Try to answer before checking execution in Python Tutor.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:**

```
external_numbers = [10, 20, 30]

def double(lst):
 tmp = [] # WRONG: we are creating a NEW list, text tells to_
 ↪MODIFY
 for element in lst:
 tmp.append(element * 2) # WRONG: we are modifying a NEW list

 return tmp # WRONG: text didn't ask you to RETURN anything

external_numbers = double(external_numbers) # WRONG: even if external_numbers_
 ↪association will # actually point to a list of doubled_
 ↪numbers,
 ↪memory, # it will be a completely NEW region of_
 ↪one! # while we wanted to MODIFY the original_
```

</div>

```
[27]: external_numbers = [10, 20, 30]

def double(lst):
 tmp = []
 for element in lst:
 tmp.append(element * 2)

 return tmp

external_numbers = double(external_numbers)

jupman.pytut()

[27]: <IPython.core.display.HTML object>
```

Probably you are a bit confused about the previous attempt, which to the untrained eye might look successful. Let's try to rewrite it with one variable more saved which will point to exactly the same original memory region of exter-

nal\_numbers. You will see that at the end saved will point to [10, 20, 30], showing we didn't actually MODIFY the original region.

```
[28]: external_numbers = [10,20,30]
saved = external_numbers # we preserve a pointer

def double(lst):
 tmp = []
 for element in lst:
 tmp.append(element * 2)
 return tmp

external_numbers = double(external_numbers)
print('external_numbers:', external_numbers) # [20,40,60]
print(' saved:', saved) # [10,20,30]

jupman.pytut()

external_numbers: [20, 40, 60]
 saved: [10, 20, 30]
[28]: <IPython.core.display.HTML object>
```

## Doubling success!

Let's finally see the right way to do it: we need to consider we want to refer to original cells, so to do it properly we need to access them *by index*, and we will need a `for in range`.

```
[29]: external_numbers = [1,2,3,4,5]

def double(lst):
 for i in range(len(lst)):
 lst[i] = lst[i] * 2

double(external_numbers)

jupman.pytut()
[29]: <IPython.core.display.HTML object>
```

Notice that:

- when the function call frame is created, we see an arrow to the original data
- the `external_list` actually changed, without ever reassigning it (not even outside)
- we didn't reassign `lst =` inside the function body, as the IV COMMANDMENT<sup>249</sup> prescribes not to reassign parameters
- we didn't use `return`, as the function text told us nothing about returning
- we didn't referred to `external_list` inside the function body: doing so would have defeated the purpose of functions, which is to isolate them from outside world.

In general, in the case of mutable data data isolation is never tight, as we get pointers to data living outside the function frame. When we manipulate pointers it's really up to us to take special care.

<sup>249</sup> <https://en.softpython.org/commands.html#IV-COMMANDMENT>

### Modifying parameters - Questions

For each of the following expressions, try guessing the result it produces (or if it gives error)

1. `def zam(bal):  
 bal = 4  
x = 8  
zam(x)  
print(x)`

2. `def zom(y):  
 y = 4  
y = 8  
zom(y)  
print(y)`

3. `def per(la):  
 la.append('è')  
per(la)  
print(la)`

4. `def zeb(lst):  
 lst.append('d')  
la = ['a', 'b', 'c']  
zeb(la)  
print(la)`

5. `def beware(la):  
 la = ['?', '?']  
lb = ['d', 'a', 'm', 'n']  
beware(lb)  
print(lb)`

6. `def umpa(string):  
 string = "lompa"  
word = "gnappa"  
umpa(word)  
print(word)`

7. `def sporty(diz):  
 diz['sneakers'] = 2  
cabinet = {'rackets': 4,  
 'balls': 7}  
sporty(cabinet)  
print(cabinet)`

8. `def numma(lst):  
 lst + [4, 5]  
la = [1, 2, 3]  
print(numma(la))  
print(la)`

9. `def jar(lst):  
 return lst + [4, 5]  
lb = [1, 2, 3]`

(continues on next page)

(continued from previous page)

```
print(jar(lb))
print(lb)
```

## Exercises - Changing music

It's time to better understand what we're doing when we mess with variables and function calls.

An uncle of ours gave us a dusty album of songs (for some reason tens of years have passed since he last turned on the radio)

```
[30]: album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]
```

Songs are reported with the group, a dash – and finally the name. Strong with our new knowledge about functions, we decide to put in practice modern software development practices to analyze these mysterious relics of the past.

In the following you will find several exercises which will ask you to develop functions: making something which *seems* to work is often easy, **the true challenge is following exactly what is asked in function text:** take particular care about capitalized words, like PRINT, MODIFY, RETURN, and to the desired outputs, trying to understand to which category the various functions belong to.

Exercises must all be solved following this scheme:

```
album = ...

def func(songs):
 # do something with songs, NOT with album
 #

func(album) # calls to test, external to function body
```

### DO NOT WRITE EXTERNAL VARIABLE NAMES INSIDE THE FUNCTION

In particular:

- **DO NOT** reassign `album =`
- **DO NOT** call its methods `album.some_method()`

A function must be typically seen as an isolated world, which should interact with the outworld ONLY through the given parameters. By explicitly writing `album`, you would override such isolation bringing great misfortune.

### ALWAYS USE A PARAMETER NAME DIFFERENT FROM EXTERNAL VARIABLES

For example, if external data is called `album`, you can call the parameter `songs`

**Exercise - show**

Write a function which given a list songs, PRINTS the group justified to the right followed by a : and the song name

**HINT:** to justify the text, use the string method .rjust(16)

```
>>> res = show(album) # only prints, implicitly returns None
```

```
Caterina Caselli: Cento giorni
 Delirium: Jesahel
 Jan Hammer: Crockett's Theme
Sonata Arctica: White Pearl, Black Oceans
 Lucio Dalla: 4 marzo 1943.mp3
The Wellermen: Wellerman
 Manu Chao: Por el Suelo
 Intillimani: El Pueblo Unido
```

```
>>> print(res)
None
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[31]:

```
album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]

write here

#category: side effects

def show(songs):
 for song in songs:
 parts = song.split(' - ')
 print(parts[0].rjust(16) + ':' + ' ' + parts[1])
```

```
show(album)

Caterina Caselli: Cento giorni
 Delirium: Jesahel
 Jan Hammer: Crockett's Theme
Sonata Arctica: White Pearl, Black Oceans
 Lucio Dalla: 4 marzo 1943.mp3
The Wellermen: Wellerman
 Manu Chao: Por el Suelo
 Intillimani: El Pueblo Unido
```

```
</div>
```

[31]:

```
album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]

write here
```

### Exercise - authors

Write a function which given a list of songs, RETURN a NEW list with only the authors

```
>>> autors(album)
['Caterina Caselli', 'Delirium', 'Jan Hammer', 'Sonata Arctica', 'Lucio Dalla', 'The
 ↪Wellermen', 'Manu Chao', 'Intillimani']

>>> album
['Caterina Caselli - Cento giorni',
 'Delirium - Jesahel',
 'Jan Hammer - Crockett's Theme',
 'Sonata Arctica - White Pearl, Black Oceans',
 'Lucio Dalla - 4 marzo 1943.mp3',
 'The Wellermen - Wellerman',
 'Manu Chao - Por el Suelo',
 'Intillimani - El Pueblo Unido']
```

[Show solution](#)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[32]:

```
album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]

write here

#category: RETURN

#NOTE: There aren't print commands in the function body!
```

(continues on next page)

(continued from previous page)

```
def authors(songs):
 ret = []
 for song in songs:
 ret.append(song.split(' - ')[0])
 return ret

print(authors(album))
album
['Caterina Caselli', 'Delirium', 'Jan Hammer', 'Sonata Arctica', 'Lucio Dalla', 'The
˓→Wellermen', 'Manu Chao', 'Intillimani']

[32]: ['Caterina Caselli - Cento giorni',
 'Delirium - Jesahel',
 "Jan Hammer - Crockett's Theme",
 'Sonata Arctica - White Pearl, Black Oceans',
 'Lucio Dalla - 4 marzo 1943.mp3',
 'The Wellermen - Wellerman',
 'Manu Chao - Por el Suelo',
 'Intillimani - El Pueblo Unido']
```

</div>

[32]:

```
album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]

write here
```

### Exercise - record

Write a function which given two lists `songsA` and `songsB`, MODIFIES `songsA` overwriting it with the content of `songsB`. If `songsB` has less elements than `songsS`, fill the remaining spaces with `None`

- **ASSUME** `songsB` has at most the same number of songs of `songsA`
- **DO NOT** reassign `album` (so no `album =`)

```
returns nothing!
>>> record(album, ["Toto Cotugno - L'Italiano vero", "Mia Martini - Minuetto", "Al
˓→Bano-Nel sole"])

>>> album # parameter was modified
["Toto Cotugno - L'Italiano vero",
 "Mia Martini - Minuetto",
 "Al Bano - Nel sole"]
```

(continues on next page)

(continued from previous page)

```
None,
None,
None,
None,
None]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[33]:

```
album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]

write here

#category: MODIFY

def registra(canzoniA, canzoniB):

 for i in range(len(canzoniB)):
 canzoniA[i] = canzoniB[i]
 i += 1
 while i < len(canzoniA):
 canzoniA[i] = None
 i += 1

registra(album, ["Toto Cotugno - L'Italiano vero", "Mia Martini - Minuetto", "Al Bano- ↵ Nel sole"])
album
```

[33]:

```
["Toto Cotugno - L'Italiano vero",
'Mia Martini - Minuetto',
'Al Bano- Nel sole',
None,
None,
None,
None,
None]
```

</div>

[33]:

```
album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
```

(continues on next page)

(continued from previous page)

```
"Manu Chao - Por el Suelo",
"Intillimani - El Pueblo Unido"
]

write here
```

### Exercise - great

Write a function `great` which given a list of songs MODIFIES the list by uppercasing all the characters, and then RETURNS it

- **DO NOT** reassign `album` (no `album =`)

Example:

```
>>> great(album) # return
['CATERINA CASELLI - CENTO GIORNI',
 'DELIRIUM - JESAHEL',
 "JAN HAMMER - CROCKETT'S THEME",
 'SONATA ARCTICA - WHITE PEARL, BLACK OCEANS',
 'LUCIO DALLA - 4 MARZO 1943.MP3',
 'THE WELLERMEN - WELLERMAN',
 'MANU CHAO - POR EL SUELO',
 'INTILLIMANI - EL PUEBLO UNIDO']

>>> album # parameter was modified
['CATERINA CASELLI - CENTO GIORNI',
 'DELIRIUM - JESAHEL',
 "JAN HAMMER - CROCKETT'S THEME",
 'SONATA ARCTICA - WHITE PEARL, BLACK OCEANS',
 'LUCIO DALLA - 4 MARZO 1943.MP3',
 'THE WELLERMEN - WELLERMAN',
 'MANU CHAO - POR EL SUELO',
 'INTILLIMANI - EL PUEBLO UNIDO']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[34]:

```
album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]

write here

#category: MODIFY and RETURN
```

(continues on next page)

(continued from previous page)

```

def great(songs):
 for i in range(len(songs)):
 songs[i] = songs[i].upper()
 return songs

print(great(album))
print()
album

['CATERINA CASELLI - CENTO GIORNI', 'DELIRIUM - JESAHEL', "JAN HAMMER - CROCKETT'S THEME", 'SONATA ARCTICA - WHITE PEARL, BLACK OCEANS', 'LUCIO DALLA - 4 MARZO 1943.MP3', 'THE WELLERMEN - WELLERMAN', 'MANU CHAO - POR EL SUELO', 'INTILLIMANI - EL PUEBLO UNIDO']

[34]: ['CATERINA CASELLI - CENTO GIORNI',
 'DELIRIUM - JESAHEL',
 "JAN HAMMER - CROCKETT'S THEME",
 'SONATA ARCTICA - WHITE PEARL, BLACK OCEANS',
 'LUCIO DALLA - 4 MARZO 1943.MP3',
 'THE WELLERMEN - WELLERMAN',
 'MANU CHAO - POR EL SUELO',
 'INTILLIMANI - EL PUEBLO UNIDO']

```

&lt;/div&gt;

```

[34]: album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]

write here

```

### Exercise - shorten

Write a function `shorten` which given a list of `songs` and a number `n`, MODIFIES `songs` so it has only `n` songs, then RETURNS a NEW list with all the removed elements.

- if `n` is too large, returns an empty list without modifying the album
- USE a parameter name different from `album`
- DO NOT reassign `album` (so no `album =`)

Example:

```
>>> shorten(album, 3) # returns
['Sonata Arctica - White Pearl, Black Oceans',
 'Lucio Dalla - 4 marzo 1943.mp3',
 'The Wellermen - Wellerman',
 'Manu Chao - Por el Suelo',
 'Intillimani - El Pueblo Unido']
>>> album # the parameter was modified
['Caterina Caselli - Cento giorni',
 'Delirium - Jesahel',
 "Jan Hammer - Crockett's Theme"]
>>> shorten(album, 7)
[]
>>> album
['Caterina Caselli - Cento giorni',
 'Delirium - Jesahel',
 "Jan Hammer - Crockett's Theme"]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[35]:

```
album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]

write here

#categoria: modifies and returns a part

def shorten(songs, n):
 ret = []
 if n >= len(songs):
 return ret
 for i in range(n):
 ret.append(songs.pop())
 ret.reverse()
 return ret

res1 = shorten(album, 3)
print('returned:\n', res1, '\n')
print("the album is:\n", album, '\n')
res2 = shorten(album, 5)
print('returned:\n', res2, '\n')
print("the album is:\n", album, '\n')

returned:
['The Wellermen - Wellerman', 'Manu Chao - Por el Suelo', 'Intillimani - El Pueblo
→Unido']

the album is:
['Caterina Caselli - Cento giorni', 'Delirium - Jesahel', "Jan Hammer - Crockett's
→Theme", 'Sonata Arctica - White Pearl, Black Oceans', 'Lucio Dalla - 4 (continues on next page)
→mp3']
```

(continued from previous page)

```

returned:
[]

the album is:
['Caterina Caselli - Cento giorni', 'Delirium - Jesahel', "Jan Hammer - Crockett's
 ↪Theme", 'Sonata Arctica - White Pearl, Black Oceans', 'Lucio Dalla - 4 marzo 1943.
 ↪mp3']

```

&lt;/div&gt;

[35]:

```

album = [
 "Caterina Caselli - Cento giorni",
 "Delirium - Jesahel",
 "Jan Hammer - Crockett's Theme",
 "Sonata Arctica - White Pearl, Black Oceans",
 "Lucio Dalla - 4 marzo 1943.mp3",
 "The Wellermen - Wellerman",
 "Manu Chao - Por el Suelo",
 "Intillimani - El Pueblo Unido"
]
write here

```

## Lambda functions

Lambda functions are functions which:

- have no name
- are defined on one line, typically right where they are needed
- their body is an expression, thus you need no `return`

Let's create a lambda function which takes a number `x` and doubles it:

[36]:

```
lambda x: x*2
```

[36]:

```
<function __main__.<lambda> (x)>
```

As you see, Python created a function object, which gets displayed by Jupyter. Unfortunately, at this point the function object got lost, because that is what happens to any object created by an expression that is not assigned to a variable.

To be able to call the function, we will thus convenient to assign such function object to a variable, say `f`:

[37]:

```
f = lambda x: x*2
```

[38]:

```
f
```

[38]:

```
<function __main__.<lambda> (x)>
```

Great, now we have a function we can call as many times as we want:

```
[39]: f(5)
```

```
[39]: 10
```

```
[40]: f(7)
```

```
[40]: 14
```

So writing

```
[41]: def f(x):
 return x*2
```

or

```
[42]: f = lambda x: x*2
```

are completely equivalent forms, the main difference being with `def` we can write functions with bodies on multiple lines. Lambdas may appear limited, so why should we use them? Sometimes they allow for very concise code. For example, imagine you have a list of tuples holding animals and their lifespan:

```
[43]: animals = [('dog', 12), ('cat', 14), ('pelican', 30), ('eagle', 25), ('squirrel', 6)]
```

If you want to sort them by lifespan, you can try the `.sort` method but it will not work:

```
[44]: animals.sort()
```

```
[45]: animals
```

```
[45]: [('cat', 14), ('dog', 12), ('eagle', 25), ('pelican', 30), ('squirrel', 6)]
```

Clearly, this is not what we wanted. To get proper ordering, we need to tell Python that when it considers a tuple for comparison, it should extract the lifespan number. To do so, Python provides us with `key` parameter, to which we must pass a function that takes as argument the sequence element under consideration (in this case a tuple) and returns a transformation of it which Python will use to perform the comparisons - in this case we want the life expectancy at the 1-th position in the tuple:

```
[46]: animals.sort(key=lambda t: t[1])
```

```
[47]: animals
```

```
[47]: [('squirrel', 6), ('dog', 12), ('cat', 14), ('eagle', 25), ('pelican', 30)]
```

Now we got the ordering we wanted. We could have written the thing as

```
[48]: def myf(t):
 return t[1]
```

```
animals.sort(key=myf)
animals
```

```
[48]: [('squirrel', 6), ('dog', 12), ('cat', 14), ('eagle', 25), ('pelican', 30)]
```

but lambdas clearly save some keyboard typing

Notice lambdas can take multiple parameters:

```
[49]: mymul = lambda x,y: x * y
mymul(2,5)
[49]: 10
```

### Exercise - apply\_borders

⊕ Write a function `apply_borders` which takes a function `f` as parameter and a sequence, and RETURN a tuple holding two elements:

- first element is obtained by applying `f` to the first element of the sequence
- second element is obtained by applying `f` to the last element of the sequence

Example:

```
>>> apply_borders(lambda x: x.upper(), ['the', 'river', 'is', 'very', 'long'])
('THE', 'LONG')
>>> apply_borders(lambda x: x[0], ['the', 'river', 'is', 'very', 'long'])
('t', 'l')
```

[Show solution](#)

```
[50]: # write here

def apply_borders(f, seq):
 return (f(seq[0]), f(seq[-1]))
```

</div>

```
[50]: # write here
```

```
[51]: print(apply_borders(lambda x: x.upper(), ['the', 'river', 'is', 'very', 'long']))
print(apply_borders(lambda x: x[0], ['the', 'river', 'is', 'very', 'long']))

('THE', 'LONG')
('t', 'l')
```

### Exercise - process

⊕⊕ Write a lambda expression to be passed as first parameter of the function `process` defined down here, so that a call to `process` generates a list as shown here:

```
>>> f = PUT_YOUR_LAMBDA_FUNCTION
>>> process(f, ['d', 'b', 'a', 'c', 'e', 'f'], ['q', 's', 'p', 't', 'r', 'n'])
['An', 'Bp', 'Cq', 'Dr', 'Es', 'Ft']
```

**NOTE:** `process` is already defined, you do not need to change it

[Show solution](#)

```
[52]: def process(f, lista, listb):
 ordA = list(sorted(lista))
 ordB = list(sorted(listb))
 ret = []
 for i in range(len(lista)):
 ret.append(f(ordA[i], ordB[i]))
 return ret

write here the f = lambda ...
f = lambda x,y: x.upper() + y
```

</div>

```
[52]: def process(f, lista, listb):
 ordA = list(sorted(lista))
 ordB = list(sorted(listb))
 ret = []
 for i in range(len(lista)):
 ret.append(f(ordA[i], ordB[i]))
 return ret

write here the f = lambda ...
```

```
[53]: process(f, ['d', 'b', 'a', 'c', 'e', 'f'], ['q', 's', 'p', 't', 'r', 'n'])
```

```
[53]: ['An', 'Bp', 'Cq', 'Dr', 'Es', 'Ft']
```

### Continue

Go on with error handling and testing<sup>250</sup>

```
[]:
```

## 7.1.2 Error handling and testing solutions

[Download exercises zip](#)

[Browse files online](#)<sup>251</sup>

---

<sup>250</sup> <https://en.softpython.org/functions/fun2-errors-and-testing-sol.html>

<sup>251</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/functions>

## Introduction

In this notebook we will try to understand what our program should do when it encounters unforeseen situations, and how to test the code we write.

For some strange reason, many people believe that computer programs do not need much error handling nor testing. Just to make a simple comparison, would you ever drive a car that did not undergo scrupulous checks? We wouldn't.

## What to do

1. unzip exercises in a folder, you should get something like this:

```
functions
 fun1-intro.ipynb
 fun1-intro-sol.ipynb
 fun2-errors-and-testing.ipynb
 fun2-errors-and-testing-sol.ipynb
 fun3-strings.ipynb
 fun3-strings-sol.ipynb
 fun4-lists.ipynb
 fun4-lists-sol.ipynb
 fun5-tuples.ipynb
 fun5-tuples-sol.ipynb
 fun6-sets.ipynb
 fun6-sets-sol.ipynb
 fun7-dictionaries.ipynb
 fun7-dictionaries-sol.ipynb
 fun8-chal.ipynb
jupman.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `functions/fun2-errors-and-testing.ipynb`
3. Go on reading that notebook, and follow instructions inside. Sometimes you will find cells marked with **Exercise** which will ask you to write Python commands in the following cells.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### Unforeseen situations

It is evening, there is to party for a birthday and they asked you to make a pie. You need the following steps:

1. take milk
2. take sugar
3. take flour
4. mix
5. heat in the oven

You take the milk, the sugar, but then you discover there is no flour. It is evening, and there aren't open shops. Obviously, it makes no sense to proceed to point 4 with the mixture, and you have to give up on the pie, telling the guest of honor the problem. You can only hope she/he decides for some alternative.

Translating everything in Python terms, we can ask ourselves if during the function execution, when we find an unforeseen situation, is it possible to:

1. **interrupt** the execution flow of the program
2. **signal** to whoever called the function that a problem has occurred
3. **allow to manage** the problem to whoever called the function

The answer is yes, you can do it with the mechanism of **exceptions** (Exception)

### make\_problematic\_pie

Let's see how we can represent the above problem in Python. A basic version might be the following:

```
[2]: def make_problematic_pie(milk, sugar, flour):
 """ Suppose you need 1.3 kg for the milk, 0.2kg for the sugar and 1.0kg for the
 ↵flour

 - takes as parameters the quantities we have in the sideboard
 """

 if milk > 1.3:
 print("take milk")
 else:
 print("Don't have enough milk !")

 if sugar > 0.2:
 print("take sugar")
 else:
 print("Don't have enough sugar!")

 if flour > 1.0:
 print("take flour")
 else:
 print("Don't have enough flour !")

 print("Mix")
 print("Heat")
 print("I made the pie!")
```

(continues on next page)

(continued from previous page)

```
make_problematic_pie(5,1,0.3) # not enough flour ...

print("Party")

take milk
take sugar
Don't have enough flour !
Mix
Heat
I made the pie!
Party
```

**QUESTION:** this above version has a serious problem. Can you spot it ??

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** the program above is partying even when we do not have enough ingredients !

</div>

### Check with the return

**EXERCISE:** We could correct the problems of the above pie by adding `return` commands. Implement the following function.

**WARNING: DO NOT move the `print ("Party")` inside the function**

The exercise goal is keeping it outside, so to use the value returned by `make_pie` for deciding whether to party or not.

If you have any doubts on functions with return values, check Chapter 6 of Think Python<sup>252</sup>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: def make_pie(milk, sugar, flour):
 """ - suppose we need 1.3 kg for milk, 0.2kg for sugar and 1.0kg for flour
 - takes as parameters the quantities we have in the sideboard
 IMPROVE WITH return COMMAND: RETURN True if the pie is doable,
 False otherwise

 OUTSIDE USE THE VALUE RETURNED TO PARTY OR NOT

 """
 # implement here the function

 if milk > 1.3:
 print("take milk")
 # return True # NO, it would finish right here
 else:
 print("Don't have enough milk !")
 return False
```

(continues on next page)

<sup>252</sup> <http://greenteapress.com/thinkpython2/html/thinkpython2007.html>

(continued from previous page)

```
if sugar > 0.2:
 print("take sugar")
else:
 print("Don't have enough sugar !")
 return False

if flour > 1.0:
 print("take flour")
else:
 print("Don't have enough flour !")
 return False

print("Mix")
print("Heat")
print("I made the pie !")
return True

now write here the function call, make_pie(5,1,0.3)
using the result to declare whether it is possible or not to party :-(

made_pie = make_pie(5,1,0.3)

if made_pie == True:
 print("Party")
else:
 print("No party !")

take milk
take sugar
Don't have enough flour !
No party !
```

&lt;/div&gt;

```
[3]: def make_pie(milk, sugar, flour):
 """ - suppose we need 1.3 kg for milk, 0.2kg for sugar and 1.0kg for flour

 - takes as parameters the quantities we have in the sideboard
 IMPROVE WITH return COMMAND: RETURN True if the pie is doable,
 False otherwise

 OUTSIDE USE THE VALUE RETURNED TO PARTY OR NOT

 """
 # implement here the function

now write here the function call, make_pie(5,1,0.3)
using the result to declare whether it is possible or not to party :-(
```

```
take milk
take sugar
Don't have enough flour !
No party !
```

## Exceptions

Real Python - Python Exceptions: an Introduction<sup>253</sup>

Using `return` we improved the previous function, but remains a problem: the responsibility to understand whether or not the pie is properly made is given to the caller of the function, who has to take the returned value and decide upon that whether to party or not. A careless programmer might forget to do the check and party even with an ill-formed pie.

So we ask ourselves: is it possible to stop the execution not just of the function, but of the whole program when we find an unforeseen situation?

To improve on our previous attempt, we can use the `exceptions`. To tell Python to **interrupt** the program execution in a given point, we can insert the instruction `raise` like this:

```
raise Exception()
```

If we want, we can also write a message to help programmers (who could be ourselves ...) to understand the problem origin. In our case it could be a message like this:

```
raise Exception("Don't have enough flour !")
```

Note: in professional programs, the exception messages are intended for programmers, verbose, and typically end up hidden in system logs. To final users you should only show short messages which are understandable by a non-technical public. At most, you can add an error code which the user might give to the technician for diagnosing the problem.

**EXERCISE:** Try to rewrite the function above by substituting the rows containing `return` with `raise Exception()`:

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[4]: def make_exceptional_pie(milk, sugar, flour):
 """ - suppose we need 1.3 kg for milk, 0.2kg for sugar and 1.0kg for flour
 - takes as parameters the quantities we have in the sideboard
 - if there are missing ingredients, raises Exception
 """
 # implement function

 if milk > 1.3:
 print("take milk")
 else:
 raise Exception("Don't have enough milk !")
 if sugar > 0.2:
 print("take sugar")
 else:
 raise Exception("Don't have enough sugar!")
```

(continues on next page)

<sup>253</sup> <https://realpython.com/python-exceptions/>

(continued from previous page)

```
if flour > 1.0:
 print("take flour")
else:
 raise Exception("Don't have enough flour!")
print("Mix")
print("Heat")
print("I made the pie !")
```

&lt;/div&gt;

```
[4]: def make_exceptional_pie(milk, sugar, flour):
 """ - suppose we need 1.3 kg for milk, 0.2kg for sugar and 1.0kg for flour

 - takes as parameters the quantities we have in the sideboard

 - if there are missing ingredients, raises Exception

 """
 # implement function
```

Once implemented, by writing

```
make_exceptional_pie(5,1,0.3)
print("Party")
```

you should see the following (note how “Party” is *not* printed):

```
take milk
take sugar

Exception Traceback (most recent call last)
<ipython-input-10-02c123f44f31> in <module>()
----> 1 make_exceptional_pie(5,1,0.3)
 2
 3 print("Party")

<ipython-input-9-030239f08ca5> in make_exceptional_pie(milk, sugar, flour)
 18 print("take flour")
 19 else:
---> 20 raise Exception("Don't have enough flour !")
 21 print("Mix")
 22 print("Heat")

Exception: Don't have enough flour !
```

We see the program got interrupted before arriving to mix step (inside the function), and it didn’t even arrived to party (which is outside the function). Let’s try now to call the function with enough ingredients in the sideboard:

```
[5]: make_exceptional_pie(5,1,20)
print("Party")

take milk
take sugar
take flour
```

(continues on next page)

(continued from previous page)

```
Mix
Heat
I made the pie !
Party
```

## Manage exceptions

Instead of brutally interrupting the program when problems are spotted, we might want to try some alternative (like go buying some ice cream). We could use some `try except` blocks like this:

```
[6]: try:
 make_exceptional_pie(5,1,0.3)
 print("Party")
except:
 print("Can't make the pie, what about going out for an ice cream?")
```

```
take milk
take sugar
Can't make the pie, what about going out for an ice cream?
```

If you note, the execution jumped the `print ("Party")` but no exception has been printed, and the execution passed to the row right after the `except`

## Particular exceptions

Until now we used a generic `Exception`, but, if you will, you can use more specific exceptions to better signal the nature of the error. For example, when you implement a function, since checking the input values for correctness is very frequent, Python gives you an exception called `ValueError`. If you use it instead of `Exception`, you allow the function caller to intercept only that particular error type.

If the function raises an error which is not intercepted in the catch, the program will halt.

```
[7]: def make_exceptional_pie_2(milk, sugar, flour):
 """ - suppose we need 1.3 kg for milk, 0.2kg for sugar and 1.0kg for flour
 - takes as parameters the quantities we have in the sideboard
 - if there are missing ingredients, raises Exception
 """

 if milk > 1.3:
 print("take milk")
 else:
 raise ValueError("Don't have enough milk !")
 if sugar > 0.2:
 print("take sugar")
 else:
 raise ValueError("Don't have enough sugar!")
 if flour > 1.0:
 print("take flour")
 else:
 raise ValueError("Don't have enough flour!")
```

(continues on next page)

(continued from previous page)

```
print("Mix")
print("Heat")
print("I made the pie !")

try:
 make_exceptional_pie_2(5,1,0.3)
 print("Party")
except ValueError:
 print()
 print("There must be a problem with the ingredients!")
 print("Let's try asking neighbors !")
 print("We're lucky, they gave us some flour, let's try again!")
 print("")
 make_exceptional_pie_2(5,1,4)
 print("Party")
except: # manages all exceptions
 print("Guys, something bad happened, don't know what to do. Better to go out and
→take an ice-cream !")
```

```
take milk
take sugar
```

```
There must be a problem with the ingredients!
Let's try asking neighbors !
We're lucky, they gave us some flour, let's try again!
```

```
take milk
take sugar
take flour
Mix
Heat
I made the pie !
Party
```

For more explanations about `try catch`, you can see [Real Python - Python Exceptions: an Introduction](#)<sup>254</sup>

### assert

They asked you to develop a program to control a nuclear reactor. The reactor produces a lot of energy, but requires at least 20 meters of water to cool down, and your program needs to regulate the water level. Without enough water, you risk a meltdown. You do not feel exactly up to the job, and start sweating.

Nervously, you write the code. You check and recheck the code - everything looks fine.

On inauguration day, the reactor is turned on. Unexpectedly, the water level goes down to 5 meters, and an uncontrolled chain reaction occurs. Plutonium fireworks follow.

Could we have avoided all of this? We often believe everything is good but then for some reason we find variables with unexpected values. The wrong program described above might have been written like so:

```
[8]: # we need water to cool our reactor

water_level = 40 # seems ok
```

(continues on next page)

<sup>254</sup> <https://realpython.com/python-exceptions/>

(continued from previous page)

```

print("water level: ", water_level)

a lot of code

water_level = 5 # forgot somewhere this bad row !

print("WARNING: water level low! ", water_level)

a lot of code

after a lot of code we might not know if there are the proper conditions so that ↵
everything works allright

print("turn on nuclear reactor")

water level: 40
WARNING: water level low! 5
turn on nuclear reactor

```

How could we improve it? Let's look at the `assert` command, which must be written by following it with a boolean condition.

`assert True` does absolutely nothing:

```
[9]: print("before")
 assert True
 print("after")
```

before  
after

Instead, `assert False` completely blocks program execution, by launching an exception of type `AssertionError` (Note how "after" is not printed):

```

print("before")
assert False
print("after")
```

before  
-----  
AssertionError Traceback (most recent call last)  
<ipython-input-7-a871fdc9ebee> in <module>()  
----> 1 assert False  
  
AssertionError:

To improve the previous program, we might use `assert` like this:

```
we need water to cool our reactor

water_level = 40 # seems ok

print("water level: ", water_level)

a lot of code

water_level = 5 # forgot somewhere this bad row !

print("WARNING: water level low! ", water_level)

a lot of code

after a lot of code we might not know if there are the proper conditions so that
everything works allright so before doing critical things, it is always a good idea
to perform a check ! if asserts fail (that is, the boolean expression is False),
the execution suddenly stops

assert water_level >= 20

print("turn on nuclear reactor")
```

```
water level: 40
WARNING: water level low! 5

AssertionError Traceback (most recent call last)
<ipython-input-3-d553a90d4f64> in <module>
 31 # the execution suddenly stops
 32
--> 33 assert water_level >= 20
 34
 35 print("turn on nuclear reactor")

AssertionError:
```

## When to use assert?

The case above is willingly exaggerated, but shows how a check more sometimes prevents disasters.

Asserts are a quick way to do checks, so much so that Python even allows to ignore them during execution to improve the performance (calling `python` with the `-O` parameter like in `python -O my_file.py`).

But if performance are not a problem (like in the reactor above), it's more convenient to rewrite the program using an `if` and explicitly raising an `Exception`:

```
we need water to cool our reactor

water_level = 40 # seems ok

print("water level: ", water_level)

a lot of code

water_level = 5 # forgot somewhere this bad row !

print("WARNING: water level low! ", water_level)

a lot of code

after a lot of code we might not know if there are the proper conditions so
that everything works all right. So before doing critical things, it is always
a good idea to perform a check !

if water_level < 20:
 raise Exception("Water level too low !") # execution stops here

print("turn on nuclear reactor")
```

```
water level: 40
WARNING: water level low! 5

Exception Traceback (most recent call last)
<ipython-input-30-4840536c3388> in <module>
 30
 31 if water_level < 20:
--> 32 raise Exception("Water level too low !") # execution stops here
 33
 34 print("turn on nuclear reactor")
```

(continues on next page)

(continued from previous page)

Exception: Water level too low !

Note how the reactor was *not* turned on.

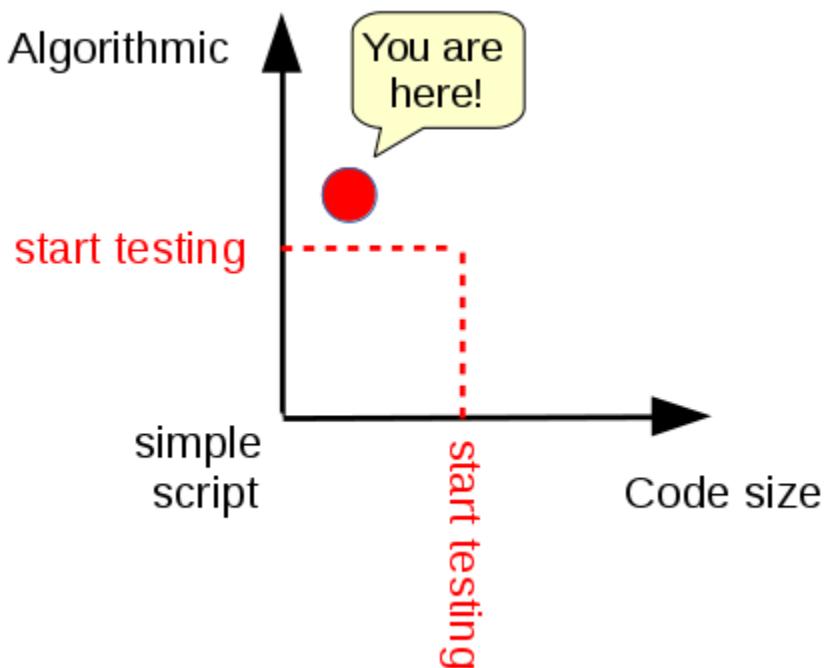
### Testing

- If it seems to work, then it actually works? *Probably not.*
- The devil is in the details, especially for complex algorithms.
- We will do a crash course on testing in Python

**WARNING:** Bad software can cause losses of million \$/€ or even harm people. Suggested reading: Software Horror Stories<sup>255</sup>

### Where Is Your Software?

As a data scientist, you might likely end up with code which is moderately complex from an algorithmic point of view, but maybe not too big in size. Either way, when red line is crossed you should start testing properly:



<sup>255</sup> <https://www.cs.tau.ac.il/~nachumd/horror.html>

## Testing with asserts

**NOTE: in this book we test with assert, but there are much better frameworks for testing!**

If you get serious about software development, please consider using something like [PyTest<sup>256</sup>](#) (recent and clean) or [Unittest<sup>257</sup>](#) (Python default testing suite, has more traditional approach)

In the part about [Foundations - A.3 Basic Algorithms<sup>258</sup>](#), we often use `assert` to perform tests, that is, to verify a function behaves as expected.

Look for example at this function:

```
[10]: def my_sum(x, y):
 s = x + y
 return s
```

We expect that `my_sum(2, 3)` gives 5. We can write in Python this expectation by using an `assert`:

```
[11]: assert my_sum(2, 3) == 5
```

Se `my_sum` is correctly implemented:

1. `my_sum(2, 3)` will give 5
2. the boolean expression `my_sum(2, 3) == 5` will give `True`
3. `assert True` will be executed without producing any result, and the program execution will continue.

Otherwise, if `my_sum` is NOT correctly implemented like in this case:

```
def my_sum(x, y):
 return 666
```

1. `my_sum(2, 3)` will produce the number 666
2. the boolean expression `my_sum(2, 3) == 5` will give `False`
3. `assert False` will interrupt the program execution, raising an exception of type `AssertionError`

## Exercise structure

Exercises in the [Foundations - A.3 Basic Algorithms<sup>259</sup>](#) are often structured in the following format:

```
def my_sum(x, y):
 """ RETURN the sum of numbers x and y
 """
 raise Exception("TODO IMPLEMENT ME!")

assert my_sum(2, 3) == 5
assert my_sum(3, 1) == 4
assert my_sum(-2, 5) == 3
```

<sup>256</sup> <https://docs.pytest.org/en/stable/>

<sup>257</sup> <https://docs.python.org/3/library/unittest.html>

<sup>258</sup> <https://en.softpython.org/index.html#basic-algorithms>

<sup>259</sup> <https://en.softpython.org/index.html#basic-algorithms>

If you attempt to execute the cell, you will see this error:

```

Exception Traceback (most recent call last)
<ipython-input-16-5f5c8512d42a> in <module>()
 6
 7
--> 8 assert my_sum(2, 3) == 5
 9 assert my_sum(3, 1) == 4
 10 assert my_sum(-2, 5) == 3

<ipython-input-16-5f5c8512d42a> in somma(x, y)
 3 """ RETURN the sum of numbers x and y
 4 """
--> 5 raise Exception("TODO IMPLEMENT ME!")
 6
 7

Exception: TODO IMPLEMENT ME!
```

To fix them, you will need to:

1. substitute the row `raise Exception("TODO IMPLEMENT ME!")` with the body of the function
2. execute the cell

If cell execution doesn't result in raised exceptions, perfect ! It means your function does what it is expected to do (the `assert` which succeed do not produce any output)

Otherwise, if you see some `AssertionError`, probably you did something wrong.

**NOTE:** The `raise Exception("TODO IMPLEMENT ME")` is put there to remind you that the function has a big problem, that is, it doesn't have any code !!! In long programs, it might happen you know you need a function, but in that moment you don't know what code put in the function body. So, instead of putting in the body commands that do nothing like `print()` or `pass` or `return None`, it is WAY BETTER to raise exceptions so that if by chance the program reaches the function, the execution is suddenly stopped and the user is signalled with the nature and position of the problem. Many editors for programmers, when automatically generating code, put inside function skeletons to implement some Exception like this.

Let's try to willingly write a wrong function body, which always return 5, independently from `x` and `y` given in input:

```
def my_sum(x, y):
 """ RETURN the sum of numbers x and y
 """
 return 5

assert my_sum(2, 3) == 5
assert my_sum(3, 1) == 4
assert my_sum(-2, 5) == 3
```

In this case the first assertion succeeds and so the execution simply passes to the next row, which contains another `assert`. We expect that `my_sum(3, 1)` gives 4, but our ill-written function returns 5 so this `assert` fails. Note how the execution is interrupted at the *second* assert:

```

AssertionError Traceback (most recent call last)
<ipython-input-19-e5091c194d3c> in <module>()
 6
--> 7 assert my_sum(2, 3) == 5
```

(continues on next page)

(continued from previous page)

```
----> 8 assert my_sum(3,1) == 4
 9 assert my_sum(-2,5) == 3

AssertionError:
```

If we implement well the function and execute the cell we will see no output: this means the function successfully passed the tests and we can conclude that it is *correct with reference to the tests* !

**ATTENTION:** always remember that these kind of tests are *never* exhaustive ! If tests pass it is only an indication the function *might* be correct, but it is never a certainty !

[12]:

```
def my_sum(x,y):
 """ RETURN the sum of numbers x and y
 """
 return x + y

assert my_sum(2,3) == 5
assert my_sum(3,1) == 4
assert my_sum(-2,5) == 3
```

**EXERCISE:** Try to write the body of the function `multiply`:

- substitute `raise Exception("TODO IMPLEMENT ME")` with `return x * y` and execute the cell. If you have written correctly, nothing should happen. In this case, congratulations! The code you have written is *correct with reference to the tests* !
- Try to substitute instead with `return 10` and see what happens.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[13]:

```
def my_mul(x,y):
 """ RETURN the multiplication of numbers x and y
 """

 return x * y

assert my_mul(2,5) == 10
assert my_mul(0,2) == 0
assert my_mul(3,2) == 6
```

</div>

[13]:

```
def my_mul(x,y):
 """ RETURN the multiplication of numbers x and y
 """

 raise Exception('TODO IMPLEMENT ME !')

assert my_mul(2,5) == 10
assert my_mul(0,2) == 0
assert my_mul(3,2) == 6
```

### Exercise - gre3

⊕⊕ Write a function `gre3` which takes three numbers and RETURN the greatest among them

Examples:

```
>>> gre3(1, 2, 4)
4

>>> gre3(5, 7, 3)
7

>>> gre3(4, 4, 4)
4
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[14] :

```
def gre3(a,b,c):
 if a > b:
 if a>c:
 return a
 else:
 return c
 else:
 if b > c:
 return b
 else:
 return c

assert gre3(1, 2, 4) == 4
assert gre3(5, 7, 3) == 7
assert gre3(4, 4, 4) == 4
```

</div>

[14] :

```
assert gre3(1, 2, 4) == 4
assert gre3(5, 7, 3) == 7
assert gre3(4, 4, 4) == 4
```

### Exercise - final\_price

⊕⊕ The cover price of a book is € 24,95, but a library obtains 40% of discount. Shipping costs are € 3 for first copy and 75 cents for each additional copy. How much `n` copies cost ?

Write a function `final_price(n)` which RETURN the price.

**ATTENTION 1:** For numbers Python wants a dot, NOT the comma !

**ATTENTION 2:** If you ordered zero books, how much should you pay ?

**HINT:** the 40% of 24,95 can be calculated by multiplying the price by 0.40

```
>>> p = final_price(10)
>>> print(p)

159.45

>>> p = final_price(0)
>>> print(p)

0
```

Show solution[data-jupman-show="Show solution"](#)[data-jupman-hide="Hide"](#)

```
[15]: def final_price(n):

 if n == 0:
 return 0
 else:
 return n* 24.95*0.6 + 3 +(n-1)*0.75

assert final_price(10) == 159.45
assert final_price(0) == 0

</div>

[15]: def final_price(n):
 raise Exception('TODO IMPLEMENT ME !')

assert final_price(10) == 159.45
assert final_price(0) == 0
```

### Exercise - arrival\_time

⊕⊕⊕ By running slowly you take 8 minutes and 15 seconds per mile, and by running with moderate rhythm you take 7 minutes and 12 seconds per mile.

Write a function `arrival_time(n, m)` which, supposing you start at 6:52, given `n` miles run with slow rhythm and `m` with moderate rhythm, PRINTs arrival time.

- **HINT 1:** to calculate an integer division, use `/`
- **HINT 2:** to calculate the remainder of integer division, use the module operator `%`

```
>>> arrival_time(2,2)
7:22
```

Show solution[data-jupman-show="Show solution"](#)[data-jupman-hide="Hide"](#)

```
[16]: def arrival_time(n,m):

 start_hour = 6
 start_minutes = 52

 # past time
```

(continues on next page)

(continued from previous page)

```
seconds = start_hour*60*60 + start_minutes*60 + n * (8*60+15) + m * (7*60+12)
minutes = seconds // 60
hours = minutes // 60

hours_display = hours % 24
minutes_display = minutes % 60

return "%s:%s" % (hours_display, minutes_display)

assert arrival_time(0,0) == '6:52'
assert arrival_time(2,2) == '7:22'
assert arrival_time(2,5) == '7:44'
assert arrival_time(8,5) == '8:34'
assert arrival_time(40,5) == '12:58'
assert arrival_time(100,25) == '23:37'
assert arrival_time(100,40) == '1:25'
assert arrival_time(700,305) == '19:43' # Forrest Gump
```

&lt;/div&gt;

```
[16]: def arrival_time(n,m):
 raise Exception('TODO IMPLEMENT ME !')

assert arrival_time(0,0) == '6:52'
assert arrival_time(2,2) == '7:22'
assert arrival_time(2,5) == '7:44'
assert arrival_time(8,5) == '8:34'
assert arrival_time(40,5) == '12:58'
assert arrival_time(100,25) == '23:37'
assert arrival_time(100,40) == '1:25'
assert arrival_time(700,305) == '19:43' # Forrest Gump
```

## Continue

Go on with exercises about functions and strings<sup>260</sup>

[ ]:

### 7.1.3 Functions 3 - exercises with strings

[Download exercises zip](#)[Browse files online](#)<sup>261</sup>

---

<sup>260</sup> <https://en.softpython.org/functions/fun3-strings-sol.html>

<sup>261</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/functions>

## First functions

### length

⊕ a. Write a function `length1(s)` in which, given a string, RETURN the length of the string. Use `len` function. For example, with "ciao" string your function should return 4 while with "hi" it should return 2

```
>>> x = length1("ciao")
>>> x
4
```

⊕ b. Write a function `length2` that like before calculates the string length, this time without using `len` (instead, use a `for` cycle)

```
>>> y = length2("mondo")
>>> y
5
```

Show solution

</div>

```
[2]: # write here

version with len, faster because python with a string always maintains in memory
the number of length immediately available

def length1(s):
 return len(s)

version with counter, slower
def length2(s):
 counter = 0
 for character in s:
 counter = counter + 1
 return counter
```

</div>

```
[2]: # write here
```

### contains

⊕ Write the function `contains(word, character)`, which RETURN True if the string contains the given character, otherwise RETURN False

- Use `in` operator

```
>>> x = contains('ciao', 'a')
>>> x
True
>>> y = contains('ciao', 'z')
>>> y
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: # write here

def contains(word, character):
 return character in word

</div>
```

```
[3]: # write here
```

### invertlet

⊕ Write the function `invertlet(first, second)` which takes in input two strings of length greater than 3, and RETURN a new string in which the words are concatenated and separated by a space, the last two characters in the words are inverted. For example, if you pass in input 'twist' and 'space', the function should RETURN 'twise spact'

- If the two strings are not of adequate length, the program PRINTS *error!*

**NOTE 1:** PRINTing is different from RETURNing !!! Whatever gets printed is shown to the user but Python cannot reuse it for calculations.

**NOTE 2:** if a function does not explicitly return anything, Python implicitly returns `None`.

**NOTE 3: Resorting to prints on error conditions is actually bad practice:** this is an invitation to think about what happens when you print something and do not return anything. You can read a discussion about it in Errors handling and testing page<sup>262</sup>

```
>>> x = invertlet("twist", "space")
>>> x
'twise spact'
>>> x = invertlet("fear", "me")
'error!'
>>> x
None
>>> x = invertlet("so", "bad")
'error!'
>>> x
None
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[4]: # write here

def invertlet(first,second):
 if len(first) <= 3 or len(second) <= 3:
 print("error!")
 else:
 return first[:-1] + second[-1] + " " + second[:-1] + first[-1]
```

(continues on next page)

<sup>262</sup> <https://en.softpython.org/functions/fun2-errors-and-testing-sol.html#Unforeseen-situations>

(continued from previous page)

```
print(invertlet("twist", "space"))
print(invertlet("fear", "me"))
print(invertlet("so", "bad"))

twise spact
error!
None
error!
None
```

&lt;/div&gt;

[4]:

# write here

**nspac**

⊕ Write a function nspace that given a string s in input, RETURN a new string in which the n-character is a space.

- if the number is too big, raise the exception ValueError - in the exception message state clearly what the problem was and the input.

**NOTE:** This time instead of printing the error we raise the exception, which will prevent the program from continuing further. This is a much better way to react to erroneous conditions.

```
>>> x = nspace('allegory', 5)
>>> x
'alleg ry'

>>> x = nspace('toy', 9)

ValueError Traceback (most recent call last)
2610223641.py in <module>
---> 12 nspace("toy", 9)
ValueError: index 9 is larger than word toy

>>> x = nspace('rack', 4)

ValueError Traceback (most recent call last)
2610223641.py in <module>
---> 12 nspace("rack", 4)
ValueError: index 4 is larger than word rack
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5]:

```
write here

def nspace(word, index):
 if index >= len(word):
```

(continues on next page)

(continued from previous page)

```

raise ValueError("index %s is larger than word %s" % (index, word))
return word[:index] + ' ' + word[index+1:]

nspac("allegory", 5)

#nspac("toy", 9)
#nspac("rack", 4)
[5]: 'alleg ry'

```

</div>

[5]:

```
write here
```

### startend

⊕ Write a function which takes a string s and RETURN the first and last two characters

- if length is less than 4, raises ValueError - in the exception message state clearly what the problem was and the input

```

>>> startend('robust pack')
rock

>>> startend('sig')

ValueError Traceback (most recent call last)
230230193.py in <module>
----> 8 startend('sig')

ValueError: I need at least 4 characters, got instead: sig

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```

write here

def startend(s):
 if len(s) < 4:
 raise ValueError("I need at least 4 characters, got instead: %s" % s)
 return s[:2] + s[-2:]

startend('robust pack')
#startend('sig')

```

[6]: 'rock'

</div>

[6]:

```
write here
```

(continues on next page)

(continued from previous page)

**swap**

Write a function that given a string, swaps the first and last character and RETURN the result.

- if the string is empty, raise `ValueError` - in the exception message state clearly the cause of the problem

```
>>> swap('dream')
mread
>>> swap('c')
c
>>> swap('')

ValueError Traceback (most recent call last)
2089609385.py in <module>
--> 11 swap('')
ValueError: Empty string!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[7]:

```
write here

def swap(s):
 if s == '':
 raise ValueError("Empty string!")
 return s[-1] + s[1:-1] + s[0]

print(swap('dream'))
print(swap('c'))
#swap('')
```

mread  
cc

&lt;/div&gt;

[7]:

# write here

## Verify comprehension

### ATTENTION

The following exercises contain tests with asserts. To understand how to solve them, read first [Error handling and testing](#)<sup>263</sup>

### has\_char

⊕ RETURN True if word contains char, False otherwise

- USE a while cycle
- DON'T use in operator nor methods such as .count (too easy!)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[8]: def has_char(word, char):

 index = 0 # initialize index
 while index < len(word):
 if word[index] == char:
 return True # we found the character, we can stop search
 index += 1 # it is like writing index = index + 1
 # if we arrive AFTER the while, there is only one reason:
 # we found nothing, so we have to return False
 return False
```

```
TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError'

assert has_char("ciao", 'a')
assert not has_char("ciao", 'A')
assert has_char("ciao", 'c')
assert not has_char("", 'a')
assert not has_char("ciao", 'z')
```

```
TEST END
```

```
</div>
```

```
[8]: def has_char(word, char):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError'

assert has_char("ciao", 'a')
assert not has_char("ciao", 'A')
```

(continues on next page)

<sup>263</sup> <https://en.softpython.org/functions/fun2-errors-and-testing-sol.html>

(continued from previous page)

```

assert has_char("ciao", 'c')
assert not has_char("", 'a')
assert not has_char("ciao", 'z')

TEST END

```

**count**

⊕ RETURN the number of occurrences of char in word

- USE a `for` in cycle
- DON'T use `count` method (too easy!)
- DON'T PRINT, IT MUST RETURN THE VALUE !

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[9]:

```

def count(word, char):

 occurrences = 0
 for c in word:
 #print("current character = ", char) # debugging prints are allowed
 if c == char:
 #print("found occurrence !") # debugging prints are allowed
 occurrences += 1
 return occurrences # THE IMPORTANT IS TO _RETURN_ THE VALUE AS THE EXERCISE_
 ↪ TEXT REQUIRES !!

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise_
 ↪ `AssertionError`

assert count("ciao", "z") == 0
assert count("ciao", "c") == 1
assert count("babbo", "b") == 3
assert count("", "b") == 0
assert count("ciao", "C") == 0
TEST END

```

&lt;/div&gt;

[9]:

```

def count(word, char):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise_
 ↪ `AssertionError`

assert count("ciao", "z") == 0
assert count("ciao", "c") == 1
assert count("babbo", "b") == 3

```

(continues on next page)

(continued from previous page)

```
assert count("", "b") == 0
assert count("ciao", "C") == 0
TEST END
```

## has\_lower

⊕ RITORNA True if the word contains *at least* one lowercase character, otherwise return False

- USE a while cycle

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[10]:

```
def has_lower(s):

 i = 0
 while i < len(s):
 if s[i] == s[i].lower():
 return True
 i += 1
 return False

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`

assert has_lower("David")
assert has_lower("daviD")
assert not has_lower("DAVID")
assert not has_lower("")
assert has_lower("a")
assert not has_lower("A")
```

</div>

[10]:

```
def has_lower(s):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`

assert has_lower("David")
assert has_lower("daviD")
assert not has_lower("DAVID")
assert not has_lower("")
assert has_lower("a")
assert not has_lower("A")
```

## dialect

⊕⊕ There exist a dialect in which all the "a" must be always preceded by a "g". In case a word contains an "a" *not* preceded by a "g", we can say with certainty that this word *does not* belong to the dialect. Write a function that given a word, RETURN True if the word respects the rules of the dialect, False otherwise.

```
>>> dialect("ammot")
False
>>> print(dialect("paganog"))
False
>>> print(dialect("pgaganog"))
True
>>> print(dialect("ciao"))
False
>>> dialect("cigao")
True
>>> dialect("zogava")
False
>>> dialect("zogavga")
True
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[11]:

```
def dialect(word):

 n = 0
 for i in range(0, len(word)):
 if word[i] == "a":
 if i == 0 or word[i - 1] != "g":
 return False
 return True

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`

assert dialect("a") == False
assert dialect("ab") == False
assert dialect("ag") == False
assert dialect("ag") == False
assert dialect("ga") == True
assert dialect("gga") == True
assert dialect("gag") == True
assert dialect("gaa") == False
assert dialect("gaga") == True
assert dialect("gabga") == True
assert dialect("gabgac") == True
assert dialect("gabbgac") == True
assert dialect("gabbgagag") == True
TEST END
```

</div>

[11]:

```
def dialect(word):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`

assert dialect("a") == False
assert dialect("ab") == False
assert dialect("ag") == False
assert dialect("ag") == False
assert dialect("ga") == True
assert dialect("gga") == True
assert dialect("gag") == True
assert dialect("gaa") == False
assert dialect("gaga") == True
assert dialect("gabga") == True
assert dialect("gabgac") == True
assert dialect("gabbgac") == True
assert dialect("gabbgagag") == True
TEST END
```

### countvoc

⊗⊗ Given a string, write a function that counts the number of vocals. If the vocals number is even, RETURN the number of vocals, otherwise raises exception ValueError

```
>>> countvoc("arco")
2
>>> count_voc("ciao")

ValueError Traceback (most recent call last)
<ipython-input-15-058310342431> in <module>()
 16 countvoc("arco")
--> 19 countvoc("ciao")

ValueError: Odd vocals !
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[12]:

```
def countvoc(word):

 n_vocals = 0

 vocals = ["a", "e", "i", "o", "u"]

 for char in word:
 if char.lower() in vocals:
 n_vocals = n_vocals + 1

 if n_vocals % 2 == 0:
```

(continues on next page)

(continued from previous page)

```

 return n_vocals
 else:
 raise ValueError("Odd vocals !")

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`

assert countvoc("arco") == 2
assert countvoc("scaturire") == 4

try:
 countvoc("ciao") # with this string we expect it raises exception ValueError
 raise Exception("I shouldn't arrive until here !")
except ValueError: # if it raises the exception ValueError, it is behaving as
expected and we do nothing
 pass

try:
 countvoc("aiuola") # with this string we expect it raises exception ValueError
 raise Exception("I shouldn't arrive until here !")
except ValueError: # if it raises the exception ValueError, it is behaving as
expected and we do nothing
 pass

```

&lt;/div&gt;

[12]:

```

def countvoc(word):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`

assert countvoc("arco") == 2
assert countvoc("scaturire") == 4

try:
 countvoc("ciao") # with this string we expect it raises exception ValueError
 raise Exception("I shouldn't arrive until here !")
except ValueError: # if it raises the exception ValueError, it is behaving as
expected and we do nothing
 pass

try:
 countvoc("aiuola") # with this string we expect it raises exception ValueError
 raise Exception("I shouldn't arrive until here !")
except ValueError: # if it raises the exception ValueError, it is behaving as
expected and we do nothing
 pass

```

### extract\_email

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[13]: def extract_email(s):
 """ Takes a string s formatted like

 "lun 5 nov 2018, 02:09 John Doe <john.doe@some-website.com>"

 and RETURN the email "john.doe@some-website.com"

 NOTE: the string MAY contain spaces before and after, but your function must
 ↪be able to extract email anyway.

 If the string for some reason is found to be ill formatted, raises ValueError
 """
 stripped = s.strip()
 i = stripped.find('<')
 return stripped[i+1:len(stripped)-1]

assert extract_email("lun 5 nov 2018, 02:09 John Doe <john.doe@some-website.com>") ==
 ↪"john.doe@some-website.com"
assert extract_email("lun 5 nov 2018, 02:09 Foo Baz <mrfoo.baz@blabla.com>") ==
 ↪"mrfoo.baz@blabla.com"
assert extract_email(" lun 5 nov 2018, 02:09 Foo Baz <mrfoo.baz@blabla.com> ") ==
 ↪"mrfoo.baz@blabla.com" # with spaces
```

</div>

```
[13]: def extract_email(s):
 """ Takes a string s formatted like

 "lun 5 nov 2018, 02:09 John Doe <john.doe@some-website.com>"

 and RETURN the email "john.doe@some-website.com"

 NOTE: the string MAY contain spaces before and after, but your function must
 ↪be able to extract email anyway.

 If the string for some reason is found to be ill formatted, raises ValueError
 """
 raise Exception('TODO IMPLEMENT ME !')

assert extract_email("lun 5 nov 2018, 02:09 John Doe <john.doe@some-website.com>") ==
 ↪"john.doe@some-website.com"
assert extract_email("lun 5 nov 2018, 02:09 Foo Baz <mrfoo.baz@blabla.com>") ==
 ↪"mrfoo.baz@blabla.com"
assert extract_email(" lun 5 nov 2018, 02:09 Foo Baz <mrfoo.baz@blabla.com> ") ==
 ↪"mrfoo.baz@blabla.com" # with spaces
```

## canon\_phone

⊕ Implement a function that canonicalize canonicalize a phone number as a string. It must RETURN the canonical version of phone as a string.

For us, a canonical phone number:

- contains no spaces
- contains no international prefix, so no +39 nor 0039: we assume all calls where placed from Italy (even if they have international prefix)

For example, all of these are canonicalized to "0461123456":

```
+39 0461 123456
+390461123456
0039 0461 123456
00390461123456
```

These are canonicalized as the following:

```
328 123 4567 -> 3281234567
0039 328 123 4567 -> 3281234567
0039 3771 1234567 -> 37711234567
```

### REMEMBER: strings are immutable !!!!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[14]: def phone_canon(phone):

 p = phone.replace(' ', '')
 if p.startswith('0039'):
 p = p[4:]
 if p.startswith('+39'):
 p = p[3:]
 return p

assert phone_canon('+39 0461 123456') == '0461123456'
assert phone_canon('+390461123456') == '0461123456'
assert phone_canon('0039 0461 123456') == '0461123456'
assert phone_canon('00390461123456') == '0461123456'
assert phone_canon('003902123456') == '02123456'
assert phone_canon('003902120039') == '02120039'
assert phone_canon('0039021239') == '021239'
```

</div>

```
[14]: def phone_canon(phone):
 raise Exception('TODO IMPLEMENT ME !')

assert phone_canon('+39 0461 123456') == '0461123456'
assert phone_canon('+390461123456') == '0461123456'
assert phone_canon('0039 0461 123456') == '0461123456'
assert phone_canon('00390461123456') == '0461123456'
assert phone_canon('003902123456') == '02123456'
```

(continues on next page)

(continued from previous page)

```
assert phone_canon('003902120039') == '02120039'
assert phone_canon('0039021239') == '021239'
```

## phone\_prefix

⊕⊕ We now want to extract the province prefix from phone numbers (see previous exercise) - the ones we consider as valid are in province\_prefixes list.

Note some numbers are from mobile operators and you can distinguish them by prefixes like 328 - the ones we consider are in mobile\_prefixes list.

Implement a function that RETURN the prefix of the phone as a string. Remember first to make it canonical !!

- If phone is mobile, RETURN string 'mobile'. If it is not a phone nor a mobile, RETURN the string 'unrecognized'
- To determine if the phone is mobile or from province, use province\_prefixes and mobile\_prefixes lists.
- DO USE THE PREVIOUSLY DEFINED FUNCTION phone\_canon(phone)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[15]: province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']
```

```
def phone_prefix(phone):

 c = phone_canon(phone)
 for m in mobile_prefixes:
 if c.startswith(m):
 return 'mobile'
 for p in province_prefixes:
 if c.startswith(p):
 return p
 return 'unrecognized'

assert phone_prefix('0461123') == '0461'
assert phone_prefix('+39 0461 4321') == '0461'
assert phone_prefix('0039011 432434') == '011'
assert phone_prefix('328 432434') == 'mobile'
assert phone_prefix('+39340 432434') == 'mobile'
assert phone_prefix('00666011 432434') == 'unrecognized'
assert phone_prefix('12345') == 'unrecognized'
assert phone_prefix('+39 123 12345') == 'unrecognized'
```

```
</div>
```

```
[15]: province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']
```

(continues on next page)

(continued from previous page)

```
def phone_prefix(phone):
 raise Exception('TODO IMPLEMENT ME !')

assert phone_prefix('0461123') == '0461'
assert phone_prefix('+39 0461 4321') == '0461'
assert phone_prefix('0039011 432434') == '011'
assert phone_prefix('328 432434') == 'mobile'
assert phone_prefix('+39340 432434') == 'mobile'
assert phone_prefix('00666011 432434') == 'unrecognized'
assert phone_prefix('12345') == 'unrecognized'
assert phone_prefix('+39 123 12345') == 'unrecognized'
```

## palindrome

⊕⊕⊕ A word is palindrome if it exactly the same when you read it in reverse

Write a function the RETURN True if the given word is palindrome, False otherwise

- assume that the empty string is palindrome

Example:

```
>>> x = palindrome('radar')
>>> x
True
>>> x = palindrome('abstruse')
>>> x
False
```

There are various ways to solve this problems, some actually easy & elegant. Try to find at least a couple of them (don't need to bang your head with the recursive one ..).

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[16]:

```
def palindrome(word):

 for i in range(len(word) // 2):
 if word[i] != word[len(word) - i - 1]:
 return False

 return True # note it is OUTSIDE for: after passing all controls,
 # we can conclude that the word it is actually palindrome

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`

assert palindrome('') == True # we assume the empty string is palindrome
assert palindrome('a') == True
assert palindrome('aa') == True
assert palindrome('ab') == False
```

(continues on next page)

(continued from previous page)

```
assert palindrome('aba') == True
assert palindrome('bab') == True
assert palindrome('bba') == False
assert palindrome('abb') == False
assert palindrome('abba') == True
assert palindrome('baab') == True
assert palindrome('abbb') == False
assert palindrome('bbba') == False
assert palindrome('radar') == True
assert palindrome('abstruse') == False
```

&lt;/div&gt;

[16]:

```
def palindrome(word):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`

assert palindrome('') == True # we assume the empty string is palindrome
assert palindrome('a') == True
assert palindrome('aa') == True
assert palindrome('ab') == False
assert palindrome('aba') == True
assert palindrome('bab') == True
assert palindrome('bba') == False
assert palindrome('abb') == False
assert palindrome('abba') == True
assert palindrome('baab') == True
assert palindrome('abbb') == False
assert palindrome('bbba') == False
assert palindrome('radar') == True
assert palindrome('abstruse') == False
```

## Continue

Go on with exercises about functions and lists<sup>264</sup>

### 7.1.4 Functions 4 - exercises with lists

[Download exercises zip](#)

[Browse files online](#)<sup>265</sup>

---

<sup>264</sup> <https://en.softpython.org/functions/fun4-lists-sol.html>

<sup>265</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/functions>

## Introduction

### Exercise - printwords

⊕ Write a function `printwords` that PRINTS all the words in a phrase

```
>>> printwords("ciao come stai?")
```

```
ciao
come
stai?
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]:

```
write here

phrase = "ciao come stai?"

def printwords(f):

 my_list = f.split() # DO *NOT* create a variable called 'list' !!!!!
 for word in my_list:
 print(word)

printwords(phrase)
```

```
ciao
come
stai?
```

</div>

[2]:

```
write here
```

### Exercise - printeven

⊕ Write a function `printeven`(`numbers`) that PRINTS all even numbers in a list of numbers `xs`

```
>>> printeven([1,2,3,4,5,6])
```

```
2
4
6
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```
write here
```

(continues on next page)

(continued from previous page)

```
def printeven(xs):
 for x in xs:
 if x % 2 == 0:
 print(x)

numbers = [1, 2, 3, 4, 5, 6]
printeven(numbers)
```

2  
4  
6

</div>

[3] :

```
write here
```

### Exercise - find26

⊕ Write a function that RETURN True if the number 26 is contained in a list of numbers

```
>>> find26([1, 26, 143, 431, 53, 6])
True
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4] :

```
write here

def find26(xs):
 return (26 in xs)

numbers = [1, 26, 143, 431, 53, 6]
find26(numbers)
```

[4] : True

</div>

[4] :

```
write here
```

## Exercise - firstsec

⊕ Write a function `firstsec(s)` that PRINTS the first and second word of a phrase.

- to find a list of words, you can use `.split()` method

```
>>> firstsec("ciao come stai?")
```

```
ciao come
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5]:

```
write here

def firstsec(s):

 my_list = phrase.split() # DO *NOT* create a variable called 'list' !!!!
 print(my_list[0], my_list[1])

phrase = "ciao come stai?"
firstsec(phrase)
```

```
ciao come
```

</div>

[5]:

```
write here
```

## Exercise - threeeven

⊕ Write a function that PRINTS "yes" if first three elements of a list are even numbers. Otherwise, the function must PRINT "no". In case the list contains less than three elements, PRINT "not good"

```
>>> threeeven([6, 4, 8, 4, 5])
yes
>>> threeeven([2, 5, 6, 3, 4, 5])
no
>>> threeeven([4])
not good
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```
write here

def threeeven(xs):
 if len(xs) >= 3:
 if xs[0] % 2 == 0 and xs[1] % 2 == 0 and xs[2] % 2 == 0:
 print("yes")
```

(continues on next page)

(continued from previous page)

```
 else:
 print("no")
 else:
 print("not good")

threeeven([6,4,8,4,5])
threeeven([2,5,6,3,4,5])
threeeven([4])
```

yes  
no  
not good

</div>

[6]:  
# write here

### Exercise - separate\_ip

⊕ An IP address is a string with four sequences of numbers (of max length 3), separated by a dot .. For example, 192.168.19.34 and 255.31.1.0 are IP addresses.

Write a function that given an IP address as input, PRINTS the numbers inside the IP address

- NOTE: do NOT use .replace method !

```
>>> separate_ip("192.168.0.1")

192
168
0
1
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[7]:  
# write here

```
def separate_ip(s):
 separated = s.split(".")
 for element in separated:
 print(element)
```

```
separate_ip("192.168.0.1")

192
168
0
1
```

</div>

[7]:

```
write here
```

### Exercise - average

⊕ Given a list of integer numbers, write a function `average(xs)` that RETURNS the arithmetic average of the numbers it contains. If the given list is empty, RETURN zero.

```
>>> x = average([3,4,2,3]) # (10/4 => 2.5)
>>> x
2.5
>>> y = average([])
>>> y
0
>>> z = average([30, 28 , 20, 29])
>>> z
26.75
```

Show solutionShow solution

>

[8]:

```
write here
```

```
def average(xs):

 if len(xs) == 0:
 return 0
 else:
 total = 0
 for x in xs:
 total = total + x

 return(total / len(xs))

av = average([])
print(av)
average([30,28,20,29])
0
26.75
```

[8]:

```
write here
```

### Exercise - Fake news generator

Functional illiteracy<sup>266</sup> is reading and writing skills that are inadequate “to manage daily living and employment tasks that require reading skills beyond a basic level”

⊕⊕ Knowing that functional illiteracy is on the rise, a news agency wants to fire obsolete human journalists and attract customers by feeding them with automatically generated *fake news*. You are asked to develop the algorithm for producing the texts: while ethically questionable, the company pays well, so you accept.

Typically, a *fake news* starts with a real subject, a real fact (the *antecedent*), and follows it with some invented statement (the *consequence*). You are provided by the company three databases, one with subjects, one with antecedents and one of consequences. To each antecedent and consequence is associated a topic.

Write a function `fake_news` which takes the databases and RETURN a list holding strings with all possible combinations of subjects, antecedents and consequences where the topic of antecedent matches the one of consequence. See desired output for more info.

**NOTE:** Your code MUST work with *any* database

**Expected output:**

```
>>> fake_news(db_subjects, db_antecedents, db_consequences)

['Government passed fiscal reform, now spending is out of control',
 'Government passed fiscal reform, this increased taxes by 10%',
 'Government passed fiscal reform, this increased deficit by a staggering 20%',
 'Government passed fiscal reform, as a consequence our GDP has fallen dramatically',
 'Government passed jobs act, now spending is out of control',
 'Government passed jobs act, this increased taxes by 10%',
 'Government passed jobs act, this increased deficit by a staggering 20%',
 'Government passed jobs act, as a consequence our GDP has fallen dramatically',
 'Government regulated pollution emissions, businesses had to fire many employees',
 'Government regulated pollution emissions, businesses are struggling to meet law\u2014requirements',
 'Government restricted building in natural areas, businesses had to fire many\u2014employees',
 'Government restricted building in natural areas, businesses are struggling to meet\u2014law requirements',
 'Government introduced more controls in agrifood production, businesses had to fire\u2014many employees',
 'Government introduced more controls in agrifood production, businesses are\u2014struggling to meet law requirements',
 'Government changed immigration policy, immigrants are stealing our jobs',
 'Party X passed fiscal reform, now spending is out of control',
 'Party X passed fiscal reform, this increased taxes by 10%',
 'Party X passed fiscal reform, this increased deficit by a staggering 20%',
 'Party X passed fiscal reform, as a consequence our GDP has fallen dramatically',
 'Party X passed jobs act, now spending is out of control',
 'Party X passed jobs act, this increased taxes by 10%',
 'Party X passed jobs act, this increased deficit by a staggering 20%',
 'Party X passed jobs act, as a consequence our GDP has fallen dramatically',
 'Party X regulated pollution emissions, businesses had to fire many employees',
 'Party X regulated pollution emissions, businesses are struggling to meet law\u2014requirements',
 'Party X restricted building in natural areas, businesses had to fire many employees',
 'Party X restricted building in natural areas, businesses are struggling to meet law\u2014requirements']
```

(continues on next page)

<sup>266</sup> [https://en.wikipedia.org/wiki/Functional\\_illiteracy](https://en.wikipedia.org/wiki/Functional_illiteracy)

(continued from previous page)

```
'Party X introduced more controls in agrifood production, businesses had to fire
many employees',
'Party X introduced more controls in agrifood production, businesses are struggling
to meet law requirements',
'Party X changed immigration policy, immigrants are stealing our jobs']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[9]:

```
db_subjects = [
 'Government',
 'Party X',
]

db_antecedents = [
 ("passed fiscal reform", "economy"),
 ("passed jobs act", "economy"),
 ("regulated pollution emissions", "environment"),
 ("restricted building in natural areas", "environment"),
 ("introduced more controls in agrifood production", "environment"),
 ("changed immigration policy", "foreign policy"),
]

db_consequences = [
 ("economy", "now spending is out of control"),
 ("economy", "this increased taxes by 10%"),
 ("economy", "this increased deficit by a staggering 20%"),
 ("economy", "as a consequence our GDP has fallen dramatically"),
 ("environment", "businesses had to fire many employees"),
 ("environment", "businesses are struggling to meet law requirements"),
 ("foreign policy", "immigrants are stealing our jobs"),
]

def fake_news(subjects, antecedents, consequences):
 ret = []
 for subject in subjects:
 for ant in antecedents:
 for con in consequences:
 if ant[1] == con[0]:
 ret.append(subject + ' ' + ant[0] + ', ' + con[1])
 return ret

fake_news(db_subjects, db_antecedents, db_consequences)
```

[9]: ['Government passed fiscal reform, now spending is out of control',
'Government passed fiscal reform, this increased taxes by 10%',
'Government passed fiscal reform, this increased deficit by a staggering 20%',
'Government passed fiscal reform, as a consequence our GDP has fallen dramatically',
'Government passed jobs act, now spending is out of control',
'Government passed jobs act, this increased taxes by 10%',
'Government passed jobs act, this increased deficit by a staggering 20%',

(continues on next page)

(continued from previous page)

```
'Government passed jobs act, as a consequence our GDP has fallen dramatically',
'Government regulated pollution emissions, businesses had to fire many employees',
'Government regulated pollution emissions, businesses are struggling to meet law\u2192
requirements',
'Government restricted building in natural areas, businesses had to fire many\u2192
employees',
'Government restricted building in natural areas, businesses are struggling to meet\u2192
law requirements',
'Government introduced more controls in agrifood production, businesses had to fire\u2192
many employees',
'Government introduced more controls in agrifood production, businesses are\u2192
struggling to meet law requirements',
'Government changed immigration policy, immigrants are stealing our jobs',
'Party X passed fiscal reform, now spending is out of control',
'Party X passed fiscal reform, this increased taxes by 10%',
'Party X passed fiscal reform, this increased deficit by a staggering 20%',
'Party X passed fiscal reform, as a consequence our GDP has fallen dramatically',
'Party X passed jobs act, now spending is out of control',
'Party X passed jobs act, this increased taxes by 10%',
'Party X passed jobs act, this increased deficit by a staggering 20%',
'Party X passed jobs act, as a consequence our GDP has fallen dramatically',
'Party X regulated pollution emissions, businesses had to fire many employees',
'Party X regulated pollution emissions, businesses are struggling to meet law\u2192
requirements',
'Party X restricted building in natural areas, businesses had to fire many employees
',
'Party X restricted building in natural areas, businesses are struggling to meet law\u2192
requirements',
'Party X introduced more controls in agrifood production, businesses had to fire\u2192
many employees',
'Party X introduced more controls in agrifood production, businesses are struggling\u2192
to meet law requirements',
'Party X changed immigration policy, immigrants are stealing our jobs']
```

&lt;/div&gt;

[9]:

```
db_subjects = [
 'Government',
 'Party X',
]

db_antecedents = [
 ("passed fiscal reform", "economy"),
 ("passed jobs act", "economy"),
 ("regulated pollution emissions", "environment"),
 ("restricted building in natural areas", "environment"),
 ("introduced more controls in agrifood production", "environment"),
 ("changed immigration policy", "foreign policy"),
]

db_consequences = [
 ("economy", "now spending is out of control"),
 ("economy", "this increased taxes by 10%"),
 ("economy", "this increased deficit by a staggering 20%"),
 ("economy", "as a consequence our GDP has fallen dramatically"),
```

(continues on next page)

(continued from previous page)

```

("environment", "businesses had to fire many employees"),
("environment", "businesses are struggling to meet law requirements"),
("foreign policy", "immigrants are stealing our jobs"),
]

def fake_news(subjects, antecedents, consequences):
 raise Exception('TODO IMPLEMENT ME !')

fake_news(db_subjects, db_antecedents, db_consequences)

```

## Functions with assert

We will discuss differences between *modifying* a list and *returning a new one*, and look into basic operations like transform, filter, mapping.

**ATTENTION:** Following exercises contain require to know tests with asserts, do understand how to carry them out, you can read first [Error handling and testing](#)<sup>267</sup>

## Mapping

Generally speaking, mapping (or transform) operations take something in input and gives back the same type of thing with elements somehow changed.

In these cases, pay attention if it is required to give back a NEW list or MODIFY the existing list.

## Exercise - newdoublef

⊕ Takes a list of integers in input and RETURN a NEW one with all the numbers of lst doubled.

- USE a for loop

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[10]: def newdoublef(lst):

 ret = []
 for x in lst:
 ret.append(x*2)
 return ret

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`:
assert newdoublef([]) == []
assert newdoublef([3]) == [6]
assert newdoublef([3, 7, 1]) == [6, 14, 2]
```

(continues on next page)

<sup>267</sup> <https://en.softpython.org/functions/fun2-errors-and-testing-sol.html#Testing-with-asserts>

(continued from previous page)

```
l = [3, 7, 1]
assert newdoublef(l) == [6, 14, 2]
assert l == [3, 7, 1]
TEST END
```

</div>

```
[10]: def newdoublef(lst):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`
assert newdoublef([]) == []
assert newdoublef([3]) == [6]
assert newdoublef([3, 7, 1]) == [6, 14, 2]

l = [3, 7, 1]
assert newdoublef(l) == [6, 14, 2]
assert l == [3, 7, 1]
TEST END
```

### Exercise - doublemod

⊕⊕ Takes a list of integers in input and MODIFIES it by doubling all the numbers.

Show solution  
Hide

```
[11]: def doublemod(lst):

 for i in range(len(lst)):
 lst[i] = lst[i] * 2

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`
l = []
doublemod(l)
assert l == []

l = [3]
doublemod(l)
assert l == [6]

l = [3, 7, 1]
doublemod(l)
assert l == [6, 14, 2]
TEST END
```

</div>

```
[11]: def doublemod(lst):

 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
→`AssertionError`
l = []
doublemod(l)
assert l == []

l = [3]
doublemod(l)
assert l == [6]

l = [3, 7, 1]
doublemod(l)
assert l == [6, 14, 2]
TEST END
```

### Exercise - newdoublec

⊕ Takes a list of integers in input and RETURN a NEW one with all the numbers of lst doubled.

- USE a list comprehension

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[12]: def newdoublec(lst):

 return [x*2 for x in lst]

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
→`AssertionError`
assert newdoublec([]) == []
assert newdoublec([3]) == [6]
assert newdoublec([3, 7, 1]) == [6, 14, 2]

l = [3, 7, 1]
assert newdoublec(l) == [6, 14, 2]
assert l == [3, 7, 1]
TEST END
```

</div>

```
[12]: def newdoublec(lst):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
→`AssertionError`
assert newdoublec([]) == []
assert newdoublec([3]) == [6]
```

(continues on next page)

(continued from previous page)

```
assert newdoublec([3,7,1]) == [6,14,2]

l = [3,7,1]
assert newdoublec(l) == [6,14,2]
assert l == [3,7,1]
TEST END
```

## Exercise - up

⊕ Takes a list of strings and RETURN a NEW list having all the strings in lst in capital

- USE a list comprehension

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[13]: def up(lst):
```

```
 return [x.upper() for x in lst]

assert up([]) == []
assert up(['']) == ['']
assert up(['a']) == ['A']
assert up(['aA']) == ['AA']
assert up(['Ba']) == ['BA']
assert up(['Ba', 'aC']) == ['BA', 'AC']
assert up(['Ba dA']) == ['BA DA']

l = ['ciAo']
assert up(l) == ['CIAO']
assert l == ['ciAo']
```

```
</div>
```

```
[13]: def up(lst):
 raise Exception('TODO IMPLEMENT ME !')
```

```
assert up([]) == []
assert up(['']) == ['']
assert up(['a']) == ['A']
assert up(['aA']) == ['AA']
assert up(['Ba']) == ['BA']
assert up(['Ba', 'aC']) == ['BA', 'AC']
assert up(['Ba dA']) == ['BA DA']

l = ['ciAo']
assert up(l) == ['CIAO']
assert l == ['ciAo']
```

## Filtering

Generally speaking, filter operations take something in input and give back the same type of thing with elements somehow filtered out.

In these cases, pay attention if it is required to RETURN a NEW list or MODIFY the existing list.

### Exercise - remall

⊗⊗ RETURN a NEW list which has the elements from list2 except the elements in list1

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[14]: def remall(list1, list2):

 list3 = list2[:]
 for x in list1:
 if x in list3:
 list3.remove(x)

 return list3

assert remall([], []) == []
assert remall(['a'], []) == []
assert remall([], ['a']) == ['a']
assert remall(['a'], ['a']) == []
assert remall(['b'], ['a']) == ['a']
assert remall(['a', 'b'], ['a', 'c', 'b']) == ['c']

orig_l1, orig_l2 = ['a', 'd'], ['a', 'c', 'd', 'b']
assert remall(orig_l1, orig_l2) == ['c', 'b']
assert orig_l1 == ['a', 'd'] # checks it doesn't modify the original ones
assert orig_l2 == ['a', 'c', 'd', 'b']
```

</div>

```
[14]: def remall(list1, list2):
 raise Exception('TODO IMPLEMENT ME !')

assert remall([], []) == []
assert remall(['a'], []) == []
assert remall([], ['a']) == ['a']
assert remall(['a'], ['a']) == []
assert remall(['b'], ['a']) == ['a']
assert remall(['a', 'b'], ['a', 'c', 'b']) == ['c']

orig_l1, orig_l2 = ['a', 'd'], ['a', 'c', 'd', 'b']
assert remall(orig_l1, orig_l2) == ['c', 'b']
assert orig_l1 == ['a', 'd'] # checks it doesn't modify the original ones
assert orig_l2 == ['a', 'c', 'd', 'b']
```

### Exercise - only\_capital\_for

⊕ Takes a list of strings lst and RETURN a NEW list which only contains the strings of lst which are all in capital letters (so keeps 'AB' but not 'aB')

- USE a for loop

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[15]: def only_capital_for(lst):

 ret = []
 for el in lst:
 if el.isupper():
 ret.append(el)
 return ret

assert only_capital_for(["CD"]) == ["CD"]
assert only_capital_for(["ab"]) == []
assert only_capital_for(["dE"]) == []
assert only_capital_for(["De"]) == []
assert only_capital_for(["ab", "DE"]) == ["DE"]
orig = ["ab", "CD", "Hb", "EF"]
assert only_capital_for(orig) == ["CD", "EF"]
assert orig == ["ab", "CD", "Hb", "EF"]
```

</div>

```
[15]: def only_capital_for(lst):
 raise Exception('TODO IMPLEMENT ME !')

assert only_capital_for(["CD"]) == ["CD"]
assert only_capital_for(["ab"]) == []
assert only_capital_for(["dE"]) == []
assert only_capital_for(["De"]) == []
assert only_capital_for(["ab", "DE"]) == ["DE"]
orig = ["ab", "CD", "Hb", "EF"]
assert only_capital_for(orig) == ["CD", "EF"]
assert orig == ["ab", "CD", "Hb", "EF"]
```

### Exercise - only\_capital\_comp

⊕ Takes a list of strings lst and RETURN a NEW list which only contains the strings of lst which are all in capital letters (so keeps 'AB' but not 'aB')

- USE a list comprehension

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[16]: def only_capital_comp(lst):

 return [el for el in lst if el.isupper()]
```

(continues on next page)

(continued from previous page)

```

assert only_capital_comp(["CD"]) == ["CD"]
assert only_capital_comp(["ab"]) == []
assert only_capital_comp(["dE"]) == []
assert only_capital_comp(["De"]) == []
assert only_capital_comp(["ab", "DE"]) == ["DE"]
orig = ["ab", "CD", "Hb", "EF"]
assert only_capital_comp(orig) == ["CD", "EF"]
assert orig == ["ab", "CD", "Hb", "EF"]

```

&lt;/div&gt;

```
[16]: def only_capital_comp(lst):
 raise Exception('TODO IMPLEMENT ME !')

assert only_capital_comp(["CD"]) == ["CD"]
assert only_capital_comp(["ab"]) == []
assert only_capital_comp(["dE"]) == []
assert only_capital_comp(["De"]) == []
assert only_capital_comp(["ab", "DE"]) == ["DE"]
orig = ["ab", "CD", "Hb", "EF"]
assert only_capital_comp(orig) == ["CD", "EF"]
assert orig == ["ab", "CD", "Hb", "EF"]
```

## Reducing

Generally speaking, *reduce* operations involve operating on sets of elements and giving back an often smaller result.

In these cases, we operate on lists. Pay attention if it is required to RETURN a NEW list or MODIFY the existing list.

### Exercise - sum\_all

⊕ RETURN the sum of all elements in lst

- Implement it as you like.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[17]: def sum_all(lst):

 return sum(lst)

assert sum_all([]) == 0
assert sum_all([7, 5]) == 12
assert sum_all([9, 5, 8]) == 22
```

&lt;/div&gt;

```
[17]: def sum_all(lst):
 raise Exception('TODO IMPLEMENT ME !')

assert sum_all([]) == 0
```

(continues on next page)

(continued from previous page)

```
assert sum_all([7,5]) == 12
assert sum_all([9,5,8]) == 22
```

### Exercise - sumevenf

⊕ RETURN the sum of all even elements in lst

- USE a for loop

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: def sumevenf(lst):

 ret = 0
 for el in lst:
 if el % 2 == 0:
 ret += el
 return ret

assert sumevenf([]) == 0
assert sumevenf([9]) == 0
assert sumevenf([4]) == 4
assert sumevenf([7,2,5,8]) == 10
```

</div>

```
[18]: def sumevenf(lst):
 raise Exception('TODO IMPLEMENT ME !')

assert sumevenf([]) == 0
assert sumevenf([9]) == 0
assert sumevenf([4]) == 4
assert sumevenf([7,2,5,8]) == 10
```

### Exercise - sumevenc

⊕ RETURN the sum of all even elements in lst

- USE a list comprehension
- WRITE only one line of code

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[19]: def sumevenc(lst):

 return sum([el for el in lst if el % 2 == 0])

assert sumevenc([]) == 0
```

(continues on next page)

(continued from previous page)

```
assert sumevenc([9]) == 0
assert sumevenc([4]) == 4
assert sumevenc([7,2,5,8]) == 10
```

&lt;/div&gt;

```
[19]: def sumevenc(lst):
 raise Exception('TODO IMPLEMENT ME !')

assert sumevenc([]) == 0
assert sumevenc([9]) == 0
assert sumevenc([4]) == 4
assert sumevenc([7,2,5,8]) == 10
```

## Other exercises

### Exercise - contains

⊕ RETURN True if elem is present in list, otherwise RETURN False

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[20]: def contains(xs, x):

 return x in xs

assert contains([], 'a') == False
assert contains(['a'], 'a') == True
assert contains(['a', 'b', 'c'], 'b') == True
assert contains(['a', 'b', 'c'], 'z') == False
```

&lt;/div&gt;

```
[20]: def contains(xs, x):
 raise Exception('TODO IMPLEMENT ME !')

assert contains([], 'a') == False
assert contains(['a'], 'a') == True
assert contains(['a', 'b', 'c'], 'b') == True
assert contains(['a', 'b', 'c'], 'z') == False
```

### Exercise - firstn

⊕ RETURN a list with the first numbers from 0 included to n excluded

- For example, `firstn(3)` must RETURN `[0, 1, 2]`
- if n is strictly negative, RETURN an empty list

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol" jupman-sol-code" style="display:none">

[21]:

```
def firstn(n):

 return list(range(n))

assert firstn(-1) == []
assert firstn(-2) == []
assert firstn(0) == []
assert firstn(1) == [0]
assert firstn(2) == [0, 1]
assert firstn(3) == [0, 1, 2]
```

</div>

[21]:

```
def firstn(n):
 raise Exception('TODO IMPLEMENT ME !')

assert firstn(-1) == []
assert firstn(-2) == []
assert firstn(0) == []
assert firstn(1) == [0]
assert firstn(2) == [0, 1]
assert firstn(3) == [0, 1, 2]
```

### Exercise - firstlast

⊕ RETURN True if the first element of a list is equal to the last one, otherwise RETURN False

NOTE: you can assume the list always contains at least one element.

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol" jupman-sol-code" style="display:none">

[22]:

```
def firstlast(xs):

 return xs[0] == xs[-1]

 # note: the comparation xs[0] == xs[-1] is an EXPRESSION which generates a
 ↪ boolean,
 # in this case True if the first character is equal to the last one and
 ↪ False otherwise
```

(continues on next page)

(continued from previous page)

```
so we can directly return the result of the expression
```

```
assert firstlast(['a']) == True
assert firstlast(['a', 'a']) == True
assert firstlast(['a', 'b']) == False
assert firstlast(['a', 'b', 'a']) == True
assert firstlast(['a', 'b', 'c', 'a']) == True
assert firstlast(['a', 'b', 'c', 'd']) == False
```

```
</div>
```

[22]:

```
def firstlast(xs):
 raise Exception('TODO IMPLEMENT ME !')

assert firstlast(['a']) == True
assert firstlast(['a', 'a']) == True
assert firstlast(['a', 'b']) == False
assert firstlast(['a', 'b', 'a']) == True
assert firstlast(['a', 'b', 'c', 'a']) == True
assert firstlast(['a', 'b', 'c', 'd']) == False
```

### Exercise - dup

⊕ RETURN a NEW list, in which each list element in input is duplicated. Example:

```
>>> dup(['hello', 'world', 'python'])
['hello', 'hello', 'world', 'world', 'python', 'python']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[23]:

```
def dup(xs):

 ret = []
 for x in xs:
 ret.append(x)
 ret.append(x)
 return ret

assert dup([]) == []
assert dup(['a']) == ['a', 'a']
assert dup(['a', 'b']) == ['a', 'a', 'b', 'b']
assert dup(['a', 'b', 'c']) == ['a', 'a', 'b', 'b', 'c', 'c']
assert dup(['a', 'a']) == ['a', 'a', 'a', 'a']
```

(continues on next page)

(continued from previous page)

```
assert dup(['a','a','b','b']) == ['a','a','a','a','b','b','b','b']
orig = ['a','a','b','b']
assert dup(orig) == ['a','a','a','a','b','b','b','b']
assert orig == ['a','a','b','b'] # it shouldn't MODIFY the original
```

&lt;/div&gt;

```
[23]: def dup(xs):
 raise Exception('TODO IMPLEMENT ME !')

assert dup([]) == []
assert dup(['a']) == ['a','a']
assert dup(['a','b']) == ['a','a','b','b']
assert dup(['a','b','c']) == ['a','a','b','b','c','c']
assert dup(['a','a']) == ['a','a','a','a']
assert dup(['a','a','b','b']) == ['a','a','a','a','b','b','b','b']
orig = ['a','a','b','b']
assert dup(orig) == ['a','a','a','a','b','b','b','b']
assert orig == ['a','a','b','b'] # it shouldn't MODIFY the original
```

## Exercise - hasdup

⊕⊕ RETURN True if xs contains element x more than once, otherwise RETURN False.

- DO NOT use .count method, too easy!

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[24]: def hasdup(x, xs):

 counter = 0

 for y in xs:
 if y == x:
 counter += 1
 if counter > 1:
 return True
 return False

assert hasdup("a", []) == False
assert hasdup("a", ["a"]) == False
assert hasdup("a", ["a", "a"]) == True
assert hasdup("a", ["a", "a", "a"]) == True
assert hasdup("a", ["b", "a", "a"]) == True
assert hasdup("a", ["b", "a", "a", "a"]) == True
assert hasdup("b", ["b", "a", "a", "a"]) == False
assert hasdup("b", ["b", "a", "b", "a"]) == True
```

&lt;/div&gt;

```
[24]: def hasdup(x, xs):
 raise Exception('TODO IMPLEMENT ME !')

 assert hasdup("a", []) == False
 assert hasdup("a", ["a"]) == False
 assert hasdup("a", ["a", "a"]) == True
 assert hasdup("a", ["a", "a", "a"]) == True
 assert hasdup("a", ["b", "a", "a"]) == True
 assert hasdup("a", ["b", "a", "a", "a"]) == True
 assert hasdup("b", ["b", "a", "a", "a"]) == False
 assert hasdup("b", ["b", "a", "b", "a"]) == True
```

### Exercise - ord3

⊕⊕ RETURN True if provided list has first three elements increasingly ordered, False otherwise

- if xs has less than three elements, RETURN False

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[25]: def ord3(xs):

 if len(xs) >= 3:
 return xs[0] <= xs[1] and xs[1] <= xs[2]
 else:
 return False

 assert ord3([5]) == False
 assert ord3([4, 7]) == False
 assert ord3([4, 6, 9]) == True
 assert ord3([4, 9, 7]) == False
 assert ord3([9, 5, 7]) == False
 assert ord3([4, 8, 9, 1, 5]) == True # first 3 elements increasing
 assert ord3([9, 4, 8, 10, 13]) == False # first 3 elements NOT increasing
```

</div>

```
[25]: def ord3(xs):
 raise Exception('TODO IMPLEMENT ME !')

 assert ord3([5]) == False
 assert ord3([4, 7]) == False
 assert ord3([4, 6, 9]) == True
 assert ord3([4, 9, 7]) == False
 assert ord3([9, 5, 7]) == False
 assert ord3([4, 8, 9, 1, 5]) == True # first 3 elements increasing
 assert ord3([9, 4, 8, 10, 13]) == False # first 3 elements NOT increasing
```

### Exercise - filterab

⊕⊕ Takes as input a list of characters, and RETURN a NEW list containing only the characters 'a' and 'b' found in the input list.

Example:

```
>>> filterab(['c', 'a', 'c', 'd', 'b', 'a', 'c', 'a', 'b', 'e'])
['a', 'b', 'a', 'a', 'b']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[26]: def filterab(xs):

 ret = []
 for x in xs:
 if x == 'a' or x == 'b':
 ret.append(x)
 return ret

assert filterab([]) == []
assert filterab(['a']) == ['a']
assert filterab(['b']) == ['b']
assert filterab(['a', 'b']) == ['a', 'b']
assert filterab(['a', 'b', 'c']) == ['a', 'b']
assert filterab(['a', 'c', 'b']) == ['a', 'b']
assert filterab(['c', 'a', 'b']) == ['a', 'b']
assert filterab(['c', 'a', 'c', 'd', 'b', 'a', 'c', 'a', 'b', 'e']) == ['a', 'b', 'a', 'a', 'b']

l = ['a', 'c', 'b']
assert filterab(l) == ['a', 'b'] # verify a NEW list is returned
assert l == ['a', 'c', 'b'] # verify original list was NOT modified
```

</div>

```
[26]: def filterab(xs):
 raise Exception('TODO IMPLEMENT ME !')

assert filterab([]) == []
assert filterab(['a']) == ['a']
assert filterab(['b']) == ['b']
assert filterab(['a', 'b']) == ['a', 'b']
assert filterab(['a', 'b', 'c']) == ['a', 'b']
assert filterab(['a', 'c', 'b']) == ['a', 'b']
assert filterab(['c', 'a', 'b']) == ['a', 'b']
assert filterab(['c', 'a', 'c', 'd', 'b', 'a', 'c', 'a', 'b', 'e']) == ['a', 'b', 'a', 'a', 'b']

l = ['a', 'c', 'b']
assert filterab(l) == ['a', 'b'] # verify a NEW list is returned
assert l == ['a', 'c', 'b'] # verify original list was NOT modified
```

## Exercise - hill

⊕⊕ RETURN a list having as first elements the numbers from 1 to n increasing, and after n the decrease until 1 included.

- NOTE: n is contained only once.

Example:

```
>>> hill(4)
[1, 2, 3, 4, 3, 2, 1]
```

[Show solution](#)

</div>

[27]: `def hill(n):`

```
 ret = []
 for i in range(1,n):
 ret.append(i)
 for i in range(n,0,-1):
 ret.append(i)
 return ret

assert hill(0) == []
assert hill(1) == [1]
assert hill(2) == [1, 2, 1]
assert hill(3) == [1, 2, 3, 2, 1]
assert hill(4) == [1, 2, 3, 4, 3, 2, 1]
assert hill(5) == [1, 2, 3, 4, 5, 4, 3, 2, 1]
```

</div>

[27]: `def hill(n):`

```
 raise Exception('TODO IMPLEMENT ME !')

assert hill(0) == []
assert hill(1) == [1]
assert hill(2) == [1, 2, 1]
assert hill(3) == [1, 2, 3, 2, 1]
assert hill(4) == [1, 2, 3, 4, 3, 2, 1]
assert hill(5) == [1, 2, 3, 4, 5, 4, 3, 2, 1]
```

## Exercise - peak

⊕⊕ Suppose in a list are saved the heights of a mountain road taking a measure every 3 km (we assume the road constantly goes upward). At a certain point, you will arrive at the mountain peak where you will measure the height with respect to the sea. Of course, there is also a road to go down hill (constantly downward) and here also the height will be measured every 3 km.

A measurement example is [100, 400, 800, 1220, 1600, 1400, 1000, 300, 40]

Write a function that RETURNS the *value* from the list which corresponds to the measurement taken at the peak.

- if the list contains less than three elements, raise exception ValueError

```
>>> peak([100, 400, 800, 1220, 1600, 1400, 1000, 300, 40])
1600
```

- **USE** a `while` cycle and terminate the function as soon as you reach the peak
- **DO NOT** use `max` function (too easy!)

[Show solution](#)

</div>

[28]:

```
def peak(xs):

 if len(xs) < 3:
 raise ValueError("Empty list !")

 i = 0
 while i < len(xs) - 1:
 if xs[i] > xs[i+1]:
 return xs[i]
 i += 1

 return xs[-1]

try:
 peak([]) # with this anomalous list we expect the exception ValueError is
 ↪raised

 raise Exception("Shouldn't arrive here!")
except ValueError: # if exception is raised, it is behaving as expected and we do
 ↪nothing
 pass
assert peak([5,40,7]) == 40
assert peak([5,30,4]) == 30
assert peak([5,70,70, 4]) == 70
assert peak([5,10,80,25,2]) == 80
assert peak([100,400, 800, 1220, 1600, 1400, 1000, 300, 40]) == 1600
```

</div>

[28]:

```
def peak(xs):
 raise Exception('TODO IMPLEMENT ME !')

try:
 peak([]) # with this anomalous list we expect the exception ValueError is
 ↪raised

 raise Exception("Shouldn't arrive here!")
except ValueError: # if exception is raised, it is behaving as expected and we do
 ↪nothing
 pass
assert peak([5,40,7]) == 40
assert peak([5,30,4]) == 30
assert peak([5,70,70, 4]) == 70
```

(continues on next page)

(continued from previous page)

```
assert peak([5,10,80,25,2]) == 80
assert peak([100,400, 800, 1220, 1600, 1400, 1000, 300, 40]) == 1600
```

### Exercise - even

⊕⊕ RETURN a list containing the elements at even position, starting from zero which is considered even

- assume the input list always contains an even number of elements
- HINT:** remember that `range` can take three parameters

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[29]: def even(xs):

 ret = []
 for i in range(0, len(xs), 2):
 ret.append(xs[i])
 return ret

assert even([]) == []
assert even(['a', 'b']) == ['a']
assert even(['a', 'b', 'c', 'd']) == ['a', 'c']
assert even(['a', 'b', 'a', 'c']) == ['a', 'a']
assert even(['a', 'b', 'c', 'd', 'e', 'f']) == ['a', 'c', 'e']
```

</div>

```
[29]: def even(xs):
 raise Exception('TODO IMPLEMENT ME !')

assert even([]) == []
assert even(['a', 'b']) == ['a']
assert even(['a', 'b', 'c', 'd']) == ['a', 'c']
assert even(['a', 'b', 'a', 'c']) == ['a', 'a']
assert even(['a', 'b', 'c', 'd', 'e', 'f']) == ['a', 'c', 'e']
```

### Exercise - mix

⊕⊕ RETURN a NEW list in which the elements are taken in alternation from `lista` and `listb`

- assume that `lista` and `listb` contain the same number of elements

Example:

```
>>> mix(['a', 'b', 'c'], ['x', 'y', 'z'])
['a', 'x', 'b', 'y', 'c', 'z']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[30]:

```
def mix(lista, listb):

 ret = []
 for i in range(len(lista)):
 ret.append(lista[i])
 ret.append(listb[i])
 return ret

assert mix([], []) == []
assert mix(['a'], ['x']) == ['a', 'x']
assert mix(['a'], ['a']) == ['a', 'a']
assert mix(['a', 'b'], ['x', 'y']) == ['a', 'x', 'b', 'y']
assert mix(['a', 'b', 'c'], ['x', 'y', 'z']) == ['a', 'x', 'b', 'y', 'c', 'z']
```

</div>

[30]:

```
def mix(lista, listb):
 raise Exception('TODO IMPLEMENT ME !')

assert mix([], []) == []
assert mix(['a'], ['x']) == ['a', 'x']
assert mix(['a'], ['a']) == ['a', 'a']
assert mix(['a', 'b'], ['x', 'y']) == ['a', 'x', 'b', 'y']
assert mix(['a', 'b', 'c'], ['x', 'y', 'z']) == ['a', 'x', 'b', 'y', 'c', 'z']
```

### Exercise - fill

⊕⊕ Takes a list lst1 of n elements and a list lst2 of m elements, and MODIFIES lst2 by copying all lst1 elements in the first n positions of lst2

- If n > m, raises a ValueError

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[31]:

```
def fill(lst1, lst2):

 if len(lst1) > len(lst2):
 raise ValueError("List 1 is bigger than list 2 ! lst_a = %s, lst_b = %s" %_
 ↪(len(lst1), len(lst2)))
 j = 0
 for x in lst1:
 lst2[j] = x
 j += 1

try:
```

(continues on next page)

(continued from previous page)

```

fill(['a','b'], [None])
 raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
 "Test passed"

try:
 fill(['a','b','c'], [None,None])
 raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
 "Test passed"

L1 = []
R1 = []
fill(L1, R1)
assert L1 == []
assert R1 == []

L = []
R = ['x']
fill(L, R)
assert L == []
assert R == ['x']

L = ['a']
R = ['x']
fill(L, R)
assert L == ['a']
assert R == ['a']

L = ['a']
R = ['x', 'y']
fill(L, R)
assert L == ['a']
assert R == ['a', 'y']

L = ['a', 'b']
R = ['x', 'y']
fill(L, R)
assert L == ['a', 'b']
assert R == ['a', 'b']

L = ['a', 'b']
R = ['x', 'y', 'z']
fill(L, R)
assert L == ['a', 'b']
assert R == ['a', 'b', 'z']

L = ['a']
R = ['x', 'y', 'z']
fill(L, R)
assert L == ['a']
assert R == ['a', 'y', 'z']

```

```
</div>

[31]: def fill(lst1, lst2):

 raise Exception('TODO IMPLEMENT ME !')

try:
 fill(['a', 'b'], [None])
 raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
 "Test passed"

try:
 fill(['a', 'b', 'c'], [None, None])
 raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
 "Test passed"

L1 = []
R1 = []
fill(L1, R1)
assert L1 == []
assert R1 == []

L = []
R = ['x']
fill(L, R)
assert L == []
assert R == ['x']

L = ['a']
R = ['x']
fill(L, R)
assert L == ['a']
assert R == ['a']

L = ['a']
R = ['x', 'y']
fill(L, R)
assert L == ['a']
assert R == ['a', 'y']

L = ['a', 'b']
R = ['x', 'y']
fill(L, R)
assert L == ['a', 'b']
assert R == ['a', 'b']

L = ['a', 'b']
R = ['x', 'y', 'z']
fill(L, R)
assert L == ['a', 'b']
```

(continues on next page)

(continued from previous page)

```
assert R == ['a', 'b', 'z']

L = ['a']
R = ['x', 'y', 'z',]
fill(L, R)
assert L == ['a']
assert R == ['a', 'y', 'z']
```

### Exercise - nostop

⊕⊕ When you analyze a phrase, it might be useful processing it to remove very common words, for example articles and prepositions: "a book on Python" can be simplified in "book Python"

The 'not so useful' words are called *stopwords*. For example, this process is done by search engines to reduce the complexity of input string provided by the user.

Implement a function which takes a string and RETURN the input string without stopwords

Implementa una funzione che prende una stringa e RITORNA la stringa di input senza le stopwords

**HINT 1:** Python strings are *immutable* ! To remove words you need to create a *new* string from the original string

**HINT 2:** create a list of words with:

```
words = stringa.split(" ")
```

**HINT 3:** transform the list as needed, and then build the string to return with " ".join(lista)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"< data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[32]:

```
def nostop(s, stopwords):

 words = s.split(" ")
 for s in stopwords:
 if s in words:
 words.remove(s)
 return " ".join(words)

assert nostop("a", ["a"]) == ""
assert nostop("a", []) == "a"
assert nostop("", []) == ""
assert nostop("", ["a"]) == ""
assert nostop("a book", ["a"]) == "book"
assert nostop("a book on Python", ["a", "on"]) == "book Python"
assert nostop("a book on Python for beginners", ["a", "the", "on", "at", "in", "of", "for", ""]) == "book Python beginners"
```

</div>

[32]:

```
def nostop(s, stopwords):
 raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```

assert nostop("a", ["a"]) == ""
assert nostop("a", []) == "a"
assert nostop("", []) == ""
assert nostop("", ["a"]) == ""
assert nostop("a book", ["a"]) == "book"
assert nostop("a book on Python", ["a", "on"]) == "book Python"
assert nostop("a book on Python for beginners", ["a", "the", "on", "at", "in", "of", "for", ""])
assert == "book Python beginners"

```

### Exercise - threez

⊕⊕ MODIFY the given lst by placing the string 'z' at the indeces divisible by 3.

```

>>> lst = ['f', 'c', 's', 'g', 'a', 'w', 'a', 'b']
>>> trez(lst)
>>> lst
>>> ['z', 'c', 's', 'z', 'a', 'w', 'z', 'b']

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```

[33]: def threez(lst):

 ret = []
 for i in range(len(lst)):
 if i % 3 == 0:
 lst[i] = 'z'

l1 = []
threez(l1)
assert l1 == []
l2 = ['a']
threez(l2)
assert l2 == ['z']
l3 = ['a', 'b']
assert threez(['a', 'b']) == None # returns nothing!
threez(l3)
assert l3 == ['z', 'b']
l4 = ['a', 'b', 'c']
threez(l4)
assert l4 == ['z', 'b', 'c']
l5 = ['a', 'b', 'c', 'd']
threez(l5)
assert l5 == ['z', 'b', 'c', 'z']
l6 = ['f', 'c', 's', 'g', 'a', 'w', 'a', 'b']
threez(l6)
assert l6 == ['z', 'c', 's', 'z', 'a', 'w', 'z', 'b']

```

</div>

```

[33]: def threez(lst):
 raise Exception('TODO IMPLEMENT ME !')

```

(continues on next page)

(continued from previous page)

```

l1 = []
threez(l1)
assert l1 == []
l2 = ['a']
threez(l2)
assert l2 == ['z']
l3 = ['a', 'b']
assert threez(['a', 'b']) == None # returns nothing!
threez(l3)
assert l3 == ['z', 'b']
l4 = ['a', 'b', 'c']
threez(l4)
assert l4 == ['z', 'b', 'c']
l5 = ['a', 'b', 'c', 'd']
threez(l5)
assert l5 == ['z', 'b', 'c', 'z']
l6 = ['f', 'c', 's', 'g', 'a', 'w', 'a', 'b']
threez(l6)
assert l6 == ['z', 'c', 's', 'z', 'a', 'w', 'z', 'b']

```

## Exercises with numbers

### Exercise - listoint

⊕⊕ Given a non-empty list of digits representing a non-negative integer, return a proper python integer

The digits are stored such that the most significant digit is at the head of the list, and each element in the list is a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

Example:

```

>>> listoint([3, 7, 5])
375
>>> listoint([2, 0])
20
>>> listoint([0])
0

```

**DO NOT** try hacks like converting the whole list to string, dirty tricks always bring undesired consequences!

The proper way is to follow rules of math, keeping in mind that in mind that

$$5746 = 5 * 1000 + 7 * 100 + 4 * 10 + 6 * 1$$

For our purposes, it is better to rewrite the formula like this:

$$5746 = 6 * 1 + 4 * 10 + 7 * 100 + 5 * 1000$$

Basically, we are performing a sum 4 times. Each time and starting from the least significant digit, the digit in consideration is multiplied for a progressively bigger power of 10, starting from  $10^0 = 1$  up to  $10^4 = 1000$ .

To understand how it could work in Python, we might progressively add stuff to a cumulator variable `c` like this:

```
c = 0

c = c + 6*1
c = c + 4*10
c = c + 7*100
c = c + 5*1000
```

In a more pythonic and concise way, we would write:

```
c = 0

c += 6*1
c += 4*10
c += 7*100
c += 5*1000
```

So first of all to get the 6,4,7,5 it might help to try scanning the list in reverse order using the function `reversed` (notice the `ed` at the end!)

```
[34]: for x in reversed([5, 7, 4, 6]):
 print(x)

6
4
7
5
```

Once we have such sequence, we need a way to get a sequence of progressively increasing powers of 10. To do so, we might use a variable `power`:

```
[35]: power = 1

for x in reversed([5, 7, 4, 6]):
 print(power)
 power = power * 10

1
10
100
1000
```

Now you should have the necessary elements to implement the required function by yourself.

---

**PLEASE REMEMBER:** if you can't find a general solution, keep trying with constants and write down all the passages you do. Then in new cells try substituting the constants with variables and keep experimenting - it's the best method to spot patterns !

---

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[36]: def listoint(lst):
 """ RETURN a Python integer which is represented by the provided list of digits,
 ↪which always
 represent a number >= 0 and has no trailing zeroes except for special case of
 ↪number 0
 """

```

(continues on next page)

(continued from previous page)

```

power = 1
num = 0
for digit in reversed(lst):
 num += power * digit
 power = power * 10
return num

assert listoint([0]) == 0
assert listoint([1]) == 1
assert listoint([2]) == 2
assert listoint([92]) == 92
assert listoint([90]) == 90
assert listoint([5,7,4]) == 574

```

&lt;/div&gt;

```
[36]: def listoint(lst):
 """ RETURN a Python integer which is represented by the provided list of digits,
 ↪which always
 represent a number >= 0 and has no trailing zeroes except for special case of
 ↪number 0
 """
 raise Exception('TODO IMPLEMENT ME !')

assert listoint([0]) == 0
assert listoint([1]) == 1
assert listoint([2]) == 2
assert listoint([92]) == 92
assert listoint([90]) == 90
assert listoint([5,7,4]) == 574
```

## Exercise - intolist

⊕⊕ Let's now try the inverse operation, that is, going from a proper Python number like 574 to a list [5, 7, 4]

Example:

```

>>> intolist(375)
[3,7,5]

>>> intolist(20)
[2,0]

>>> intolist(0)
[0]

```

To do so, we must exploit integer division // and reminder operator %.

Let's say we want to get the final digit 4 out of 574. To do so, we can notice that 4 is the remainder of integer division between 547 and 10:

```
[37]: 574 % 10
```

[37]: 4

This extracts the four, but if we want to find an algorithm for our problem, we must also find a way to progressively reduce the problem size. To do so, we can exploit the integer division operator `//`:

[38]: 574 // 10

[38]: 57

Now, given any integer number, you know how to

- a. extract last digit
- b. reduce the problem for the next iteration

This should be sufficient to proceed. Pay attention to special case for input 0.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);  
data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[39]: def intolist(num):
 """ Takes an integer number >= 0 and RETURN a list of digits representing the
 ↪number in base 10.
 """

 if num == 0:
 return [0]
 else:
 ret = []
 d = num
 while d > 0:
 digit = d % 10 # remainder of d divided by 10
 ret.append(digit)
 d = d // 10

 return list(reversed(ret))

assert intolist(0) == [0]
assert intolist(1) == [1]
assert intolist(2) == [2]
assert intolist(92) == [9, 2]
assert intolist(90) == [9, 0]
assert intolist(574) == [5, 7, 4]
```

</div>

```
[39]: def intolist(num):
 """ Takes an integer number >= 0 and RETURN a list of digits representing the
 ↪number in base 10.
 """
 raise Exception('TODO IMPLEMENT ME !')

assert intolist(0) == [0]
assert intolist(1) == [1]
assert intolist(2) == [2]
assert intolist(92) == [9, 2]
assert intolist(90) == [9, 0]
assert intolist(574) == [5, 7, 4]
```

## Exercise - add one

Given a non-empty list of digits representing a non-negative integer, adds one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the list is a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

For example:

```
Input: [1,2,3]
Output: [1,2,4]
```

```
Input: [3,6,9,9]
Output: [3,7,0,0]
```

```
Input: [9,9,9,9]
Output: [1,0,0,0,0]
```

There are two ways to solve this exercise: you can convert to a proper integer, add one, and then convert back to list which you will do in `add_one_conv`. The other way is to directly operate on a list, using a carry variable, which you will do in `add_one_carry`

## Exercise - add\_one\_conv

⊕⊕⊕ You need to do three steps:

1. Convert to a proper python integer
2. add one to the python integer
3. convert back to a list and return it

[Show solution](#)[Hide](#)</div>

```
[40]: def add_one_conv(lst):
 """
 Takes a list of digits representing an integer >= 0 without trailing zeroes
 ↪except zero itself
 and RETURN a NEW a list representing the value of lst plus one.

 Implement by calling already used implemented functions.
 """

 power = 1
 num = listoint(lst)

 return intolist(num + 1)

assert add_one_conv([0]) == [1]
assert add_one_conv([1]) == [2]
assert add_one_conv([2]) == [3]
assert add_one_conv([9]) == [1, 0]
assert add_one_conv([5, 7]) == [5, 8]
assert add_one_conv([5, 9]) == [6, 0]
assert add_one_conv([9, 9]) == [1, 0, 0]
```

```
</div>

[40]: def add_one_conv(lst):
 """
 Takes a list of digits representing an integer >= 0 without trailing zeroes
 ↵except zero itself
 and RETURN a NEW a list representing the value of lst plus one.

 Implement by calling already used implemented functions.
 """
 raise Exception('TODO IMPLEMENT ME !')

assert add_one_conv([0]) == [1]
assert add_one_conv([1]) == [2]
assert add_one_conv([2]) == [3]
assert add_one_conv([9]) == [1, 0]
assert add_one_conv([5, 7]) == [5, 8]
assert add_one_conv([5, 9]) == [6, 0]
assert add_one_conv([9, 9]) == [1, 0, 0]
```

### Exercise - add\_one\_carry

⊕⊕⊕ Given a non-empty array of digits representing a non-negative integer, adds one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

For example:

```
>>> add_one_carry([1, 2, 3])
[1, 2, 4]

>>> add_one_carry([3, 6, 9, 9])
[3, 7, 0, 0]

>>> add_one_carry([9, 9, 9, 9])
[1, 0, 0, 0, 0]
```

To implement it, directly operate on the list, using a `carry` variable.

Just follow addition as done in elementary school. Start from the last digit and sum one:

If you get a number  $\leq 9$ , that is the result of summing last two digits, and the rest is easy:

596+	carry=0
001	
----	
7	6 + 1 + carry = 7

596+	carry=0
001	
----	
97	9 + 0 + carry = 9

```
596+ carry=0
001

07 5 + 0 + carry = 5
```

If you get a number bigger than 9, then you put zero and set carry to one:

```
3599+ carry=0
0001

0 9 + 1 + carry = 10 # >9, will write zero and set carry to 1
```

```
3599+ carry=1
0001

00 9 + 0 + carry = 10 # >9, will write zero and set carry to 1
```

```
3599+ carry=1
0001

600 5 + 0 + carry = 6 # <= 9, will write result and set carry to zero
```

```
3599+ carry=0
0001

3600 3 + 0 + carry = 3 # <= 9, will write result and set carry to zero
```

[Show solution](#)</div>

```
[41]: def add_one_carry(lst):
 """
 Takes a list of digits representing a >= 0 integer without trailing zeroes
 ↪except zero itself
 and RETURN a NEW a list representing the value of lst plus one.
 """

 ret = []
 carry = 1
 for digit in reversed(lst):
 new_digit = digit + carry
 if new_digit == 10:
 ret.append(0)
 carry = 1
 else:
 ret.append(new_digit)
 carry = 0
 if carry == 1:
 ret.append(carry)
 ret.reverse()
 return ret

assert add_one_carry([0]) == [1]
assert add_one_carry([1]) == [2]
assert add_one_carry([2]) == [3]
```

(continues on next page)

(continued from previous page)

```
assert add_one_carry([9]) == [1, 0]
assert add_one_carry([5, 7]) == [5, 8]
assert add_one_carry([5, 9]) == [6, 0]
assert add_one_carry([9, 9]) == [1, 0, 0]
```

&lt;/div&gt;

```
[41]: def add_one_carry(lst):
 """
 Takes a list of digits representing a >= 0 integer without trailing zeroes
 ↪except zero itself
 and RETURN a NEW a list representing the value of lst plus one.
 """
 raise Exception('TODO IMPLEMENT ME !')

assert add_one_carry([0]) == [1]
assert add_one_carry([1]) == [2]
assert add_one_carry([2]) == [3]
assert add_one_carry([9]) == [1, 0]
assert add_one_carry([5, 7]) == [5, 8]
assert add_one_carry([5, 9]) == [6, 0]
assert add_one_carry([9, 9]) == [1, 0, 0]
```

## Exercise - collatz

⊕⊕⊕⊕ The Collatz conjecture<sup>268</sup> says that starting from any  $n$ , by performing these calculations recursively you obtain a sequence which finally ends up to 1:

- if  $n$  is even, divide  $n$  by 2
- if  $n$  is odd, multiply it by 3 and add 1
- Repeat until you reach the value of 1

Example: for  $n = 3$ , the sequence is  $[3, 10, 5, 16, 8, 4, 2, 1]$ .

Write a program that creates a list `seq`, such that for each value  $n$  between 1 and 50, `seq[n]` contains the length of the sequence so generated. In case of  $n = 3$ , the length is 8. In case of  $n = 27$ , the length is 111.

If you need to check your results, you can also try this nice online tool<sup>269</sup>

```
[42]: def collatz():
 raise Exception("TODO IMPLEMENT ME!")
```

<sup>268</sup> [https://en.wikipedia.org/wiki/Collatz\\_conjecture](https://en.wikipedia.org/wiki/Collatz_conjecture)

<sup>269</sup> <https://www.dcode.fr/collatz-conjecture>

**Continue**

Go on with exercises about functions and tuples<sup>270</sup>

[ ]:

### 7.1.5 Functions 5 - exercises with tuples

#### Download exercises zip

Browse files online<sup>271</sup>

#### Exercise - joined

⊕⊕ Write a function which given two tuples of characters `ta` and `tb` having each different characters (may also be empty), return a tuple made like this:

- if the tuple `ta` terminates with the same character `tb` begins with, RETURN the concatenation of `ta` and `tb` WITHOUT the join character duplicated.
- otherwise RETURN an empty tuple

Example:

```
>>> joined(('a', 'b', 'c'), ('c', 'd', 'e', 'e', 'f'))
('a', 'b', 'c', 'd', 'e', 'e', 'f')
>>> joined(('a', 'b'), ('b', 'c', 'd'))
('a', 'b', 'c', 'd')
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]:

```
def joined(ta, tb):
 if len(ta) > 0 and len(tb) > 0:
 if ta[-1] == tb[0]:
 return ta[:-1] + tb

 return ()

assert joined(('a', 'b', 'c'), ('c', 'd', 'e', 'e', 'f')) == ('a', 'b', 'c', 'd', 'e', 'e', 'f')
assert joined(('a', 'b'), ('b', 'c', 'd')) == ('a', 'b', 'c', 'd')
assert joined((), ('e', 'f', 'g')) == ()
assert joined((('a',), ('e', 'f', 'g'))) == ()
assert joined((('a', 'b', 'c'), ())) == ()
assert joined((('a', 'b', 'c'), ('d', 'e'))) == ()
```

</div>

<sup>270</sup> <https://en.softpython.org/functions/fun5-tuples-sol.html>

<sup>271</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/functions>

[2]:

```
def joined(ta,tb):
 raise Exception('TODO IMPLEMENT ME !')

assert joined(('a','b','c'), ('c','d','e','e','f')) == ('a', 'b', 'c', 'd', 'e', 'e', 'f'
 ↵')
assert joined(('a','b'), ('b','c','d')) == ('a', 'b', 'c', 'd')
assert joined((), ('e','f','g')) == ()
assert joined(('a',), ('e','f','g')) == ()
assert joined(('a','b','c'), ()) == ()
assert joined(('a','b','c'), ('d','e')) == ()
```

### nasty

⊕⊕⊕ Given two tuples `ta` and `tb`, `ta` made of characters and `tb` of positive integer numbers , write a function `nasty` which RETURNS a tuple having two character strings: the first character is taken from `ta`, the second is a number taken from the corresponding position in `tb`. The strings are repeated for a number of times equal to that number.

```
>>> nasty((‘u’, ‘r’, ‘g’), (4, 2, 3))
(‘u4’, ‘u4’, ‘u4’, ‘u4’, ‘r2’, ‘r2’, ‘g3’, ‘g3’)

>>> nasty((‘g’, ‘a’, ‘s’, ‘p’), (2, 4, 1, 3))
(‘g2’, ‘g2’, ‘a4’, ‘a4’, ‘a4’, ‘s1’, ‘p3’, ‘p3’)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]: # write here

```
def nasty(ta, tb):

 i = 0
 ret = []
 while i < len(tb):
 s = ta[i]+str(tb[i])
 ret.extend((s,) * tb[i])
 i += 1
 return tuple(ret)

TEST START - DO NOT TOUCH !
assert nasty((‘a’,), (3,)) == (‘a3’, ‘a3’, ‘a3’)
assert nasty((‘a’, ‘b’), (3, 1)) == (‘a3’, ‘a3’, ‘a3’, ‘b1’)
assert nasty((‘u’, ‘r’, ‘g’), (4, 2, 3)) == (‘u4’, ‘u4’, ‘u4’, ‘u4’, ‘r2’, ‘r2’, ‘g3’, ‘g3’)
assert nasty((‘g’, ‘a’, ‘s’, ‘p’), (2, 4, 1, 3)) == (‘g2’, ‘g2’, ‘a4’, ‘a4’, ‘a4’, ‘a4’, ‘s1’,
 ↵, ‘p3’, ‘p3’, ‘p3’)
TEST END
```

</div>

[3]: # write here

**Continue**

Go on with exercises about functions and sets<sup>272</sup>

[ ]:

## 7.1.6 Functions 6 - exercises with sets

### Download exercises zip

Browse files online<sup>273</sup>

### Exercise - syllabs

Write a function `syllabs` which given a string `word` made by only bisyllabs and a set `found`, finds all the distinct bisyllabs and puts them into the set `found`.

- NOTE: the function `syllabs` return NOTHING !

Example 1:

```
>>> found = set()
>>> syllabs("banana", found)
>>> found
{'an', 'ba'}
```

Example 2:

```
>>> found = set()
>>> syllabs("parariraparara", found)
>>> found
{'ri', 'ra', 'pa'}
```

[Show solution](#)  
[Hide](#)>

[6]:

```
write here
def syllabs(word, t):
 for i in range(0, len(word), 2):
 t.add(word[i:i+2])

found = set()
syllabs("banana", found)
print(found)

found = set()
syllabs("parariraparara", found)
print(found)

{'ba', 'na'}
{'ra', 'pa', 'ri'}
```

<sup>272</sup> <https://en.softpython.org/functions/fun6-sets-sol.html>

<sup>273</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/functions>

</div>

```
[6]: # write here
```

### Exercise - distinguish

⊕⊕ Write a function `distinguish` which given a list `big_list` containing sublists of *two* characters each, RETURN a NEW LIST containing all the *distinct* sublists (ignoring the duplicated sublists)

- the returned list must have the elements *in the same order* in which they were found in `big_list`
- to know fast whether a sublist was already found, **use a set**
- **DO NOT** search in lists (so no `count`, `index`, `in` in lists - they're slow!)
- **DO NOT** remove from lists (so no `remove` from lists - it's slow!)
- **HINT:** lists are *mutable*, can we place them in a set? If it's not possible, what can we do?

Example:

```
>>> big_list = [['d', 'd'], ['a', 'b'], ['d', 'd'], ['c', 'a'], ['c', 'a'], ['d', 'd'], ['a', 'b'
 ↵']]
>>> distinguish(big_list)
[[['d', 'd'], ['a', 'b'], ['c', 'a']]]
#NOTE: variable big_list MUST NOT be modified:
>>> big_list
[['d', 'd'], ['a', 'b'], ['d', 'd'], ['c', 'a'], ['c', 'a'], ['d', 'd'], ['a', 'b']]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: # write here

def distinguish(blist):
 s = set()
 ret = []

 for sublist in blist:
 # In sets we can't place lists because they are mutable,
 # but we can insert tuples
 tup = tuple(sublist)

 # Checking whether an element belongs to a set it's very fast:
 # it is independent from the set dimension!

 if tup not in s:
 ret.append(sublist)
 # Adding an element to a set is very fast:
 # it is independent from the set dimension!
 s.add(tup)

 return ret
```

big\_list = [ ['d', 'd'], ['a', 'b'], ['d', 'd'], ['c', 'a'], ['c', 'a'], ['d', 'd'], ['a', 'b'] ]

(continues on next page)

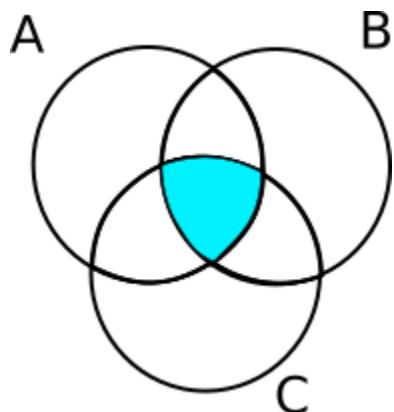
(continued from previous page)

```
#print('distincts:', distinguish(big_list))
#print('big_list:', big_list)
```

&lt;/div&gt;

```
[3]: # write here
```

### Exercise - intersection



Given a list `sets` containing an arbitrary number of sets, RETURN a NEW set which contains the elements common to all sets.

To solve the exercise, you can intersect a set at a time with a `for` cycle (slow) or with the technique described here<sup>274</sup> (short and fast).

- try to solve it in **both** ways
- **BEWARE** of the empty list!
- your code must work with **any** number of sets (the image is just an example)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[4]: def inter_for(sets):
```

```
 if len(sets) == 0:
 return set()

 first = True

 for el in sets:
 if first:
 ret = set(el)
 first = False
 else:
```

(continues on next page)

<sup>274</sup> <https://stackoverflow.com/a/2541814>

(continued from previous page)

```

 ret.intersection_update(el)
 return ret

TEST START - DO NOT TOUCH !
assert inter_for([]) == set()
assert inter_for([set(),set()]) == set()
assert inter_for([set(),set(),set()]) == set()
assert inter_for([{a},{a},{a}]) == {'a'}
assert inter_for([{a,b},{b,c},{c,a}]) == {'b'}
assert inter_for([{a,b},{b,c},{a,c}]) == {'a'}
assert inter_for([{c},{c},{b,c}]) == {'c'}
assert inter_for([{a,b},{a,b},{a,b}]) == {'a','b'}
assert inter_for([{a,b},{a,c},{b,c},{b,c}]) == {'b',
 ↪'c'}
check we didn't modify the input sets
s = {'a','b'}
assert inter_for([s,{b,c}]) == {'b'}
assert s == {'a','b'}
TEST END

```

&lt;/div&gt;

```
[4]: def inter_for(sets):

 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH !
assert inter_for([]) == set()
assert inter_for([set(),set()]) == set()
assert inter_for([set(),set(),set()]) == set()
assert inter_for([{a},{a},{a}]) == {'a'}
assert inter_for([{a,b},{b,c},{c,a}]) == {'b'}
assert inter_for([{a,b},{b,c},{a,c}]) == {'a'}
assert inter_for([{c},{c},{b,c}]) == {'c'}
assert inter_for([{a,b},{a,b},{a,b}]) == {'a','b'}
assert inter_for([{a,b},{a,c},{b,c},{b,c}]) == {'b',
 ↪'c'}
check we didn't modify the input sets
s = {'a','b'}
assert inter_for([s,{b,c}]) == {'b'}
assert s == {'a','b'}
TEST END

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[5]: def inter_fast(sets):

 if len(sets) == 0:
 return set()

 return set.intersection(*sets)
```

(continues on next page)

(continued from previous page)

```

TEST START - DO NOT TOUCH !
assert inter_fast([]) == set()
assert inter_fast([set(), set()]) == set()
assert inter_fast([set(), set(), set()]) == set()
assert inter_fast([{ 'a' }, { 'a' }, { 'a' }]) == { 'a' }
assert inter_fast([{ 'a' }, { 'b' }, { 'b' }, { 'b' }]) == { 'b' }
assert inter_fast([{ 'a' }, { 'a' }, { 'b' }, { 'a' }]) == { 'a' }
assert inter_fast([{ 'c' }, { 'c' }, { 'c' }, { 'b' }]) == { 'c' }
assert inter_fast([{ 'a' }, { 'b' }, { 'a' }, { 'b' }]) == { 'a', 'b' }
assert inter_fast([{ 'a' }, { 'b' }, { 'c' }, { 'a' }, { 'b' }, { 'c' }, { 'd' }, { 'b' }, { 'c' }]) == { 'b',
 ↪ 'c' }
check we didn't modify the input sets
s = { 'a', 'b' }
assert inter_fast([s, { 'b' }, { 'c' }]) == { 'b' }
assert s == { 'a', 'b' }
TEST END

```

&lt;/div&gt;

```

[5]: def inter_fast(sets):

 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH !
assert inter_fast([]) == set()
assert inter_fast([set(), set()]) == set()
assert inter_fast([set(), set(), set()]) == set()
assert inter_fast([{ 'a' }, { 'a' }, { 'a' }]) == { 'a' }
assert inter_fast([{ 'a' }, { 'b' }, { 'b' }, { 'b' }]) == { 'b' }
assert inter_fast([{ 'a' }, { 'a' }, { 'b' }, { 'a' }]) == { 'a' }
assert inter_fast([{ 'c' }, { 'c' }, { 'c' }, { 'b' }]) == { 'c' }
assert inter_fast([{ 'a' }, { 'b' }, { 'a' }, { 'b' }]) == { 'a', 'b' }
assert inter_fast([{ 'a' }, { 'b' }, { 'c' }, { 'a' }, { 'b' }, { 'c' }, { 'd' }, { 'b' }, { 'c' }]) == { 'b',
 ↪ 'c' }
check we didn't modify the input sets
s = { 'a', 'b' }
assert inter_fast([s, { 'b' }, { 'c' }]) == { 'b' }
assert s == { 'a', 'b' }
TEST END

```

[ ]:

## 7.2 Matrices of lists

### 7.2.1 Matrices: list of lists

[Download exercises zip](#)

Browse files online<sup>275</sup>

<sup>275</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/matrices-lists>

### Introduction

There are a couple of ways in Python to represent matrices: as lists of lists, or with the external library [Numpy<sup>276</sup>](#). The most used is surely Numpy but we see both representations anyway. Let's see the reason and main differences:

Lists of lists - as in this notebook:

1. native in Python
2. not efficient
3. lists are pervasive in Python, you will probably encounter matrices expressed as lists of lists anyway
4. you get an idea of how to construct a nested data structure
5. we can discuss memory references and copies along the way

Numpy - see other tutorial [Numpy matrices<sup>277</sup>](#)

1. not natively available in Python
2. efficient
3. used by many scientific libraries (scipy, pandas)
4. the syntax to access elements is slightly different from lists of lists
5. in rare cases it might bring installation problems and/or conflicts (implementation is not pure Python)

### What to do

- unzip exercises in a folder, you should get something like this:

```
matrices-lists
 matrices-lists1.ipynb
 matrices-lists1-sol.ipynb
 matrices-lists2.ipynb
 matrices-lists2-sol.ipynb
 matrices-lists3-chal.ipynb
 jupman.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `matrices-lists/matrices-lists1.ipynb`
- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press **Control + Enter**
- to execute Python code inside a Jupyter cell AND select next cell, press **Shift + Enter**
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press **Alt + Enter**
- If the notebooks look stuck, try to select **Kernel -> Restart**

<sup>276</sup> <https://www.numpy.org/>

<sup>277</sup> <https://en.softpython.org/matrices-numpy/matrices-numpy-sol.html>

## Overview

Let's see these lists of lists. Consider the following a matrix with 3 rows and 2 columns, or in short 3x2 matrix:

```
[2]: m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e']
]
```

For convenience, we assume as input to our functions there won't be matrices with no rows, nor rows with no columns.

Going back to the example, in practice we have a big external list:

```
m = [
]
```

and each of its elements is another list which represents a row:

```
m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e']
]
```

So, to access the whole first row ['a', 'b'], we would simply access the element at index 0 of the external list m:

```
[3]: m[0]
[3]: ['a', 'b']
```

To access the second whole second row ['c', 'd'], we would access the element at index 1 of the external list m:

```
[4]: m[1]
[4]: ['c', 'd']
```

To access the second whole third row ['c', 'd'], we would access the element at index 2 of the external list m:

```
[5]: m[2]
[5]: ['a', 'e']
```

To access the first element 'a' of the first row ['a', 'b'] we would add another subscript operator with index 0:

```
[6]: m[0][0]
[6]: 'a'
```

To access the second element 'b' of the first row ['a', 'b'] we would use instead index 1 :

```
[7]: m[0][1]
[7]: 'b'
```

**WARNING:** When a matrix is a list of lists, you can only access values with notation `m[i][j]`, **NOT** with `m[i, j]` !!

```
[8]: # write here the wrong notation m[0,0] and see which error you get:
```

## Matrix dimensions

⊕ **EXERCISE:** For getting matrix dimensions, we can use normal list operations. Which ones? You can assume the matrix is well formed (all rows have equal length) and has at least one row and at least one column

```
[9]: m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e']
]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[10]: # write here code for printing row and column dimensions

the outer list is a list of rows, so to count item we just use len(m)

print("rows")
print(len(m))

if we assume the matrix is well formed and has at least one row and column,
we can directly check the length of the first row

print("columns")
print(len(m[0]))
```

rows  
3  
columns  
2

</div>

```
[10]: # write here code for printing row and column dimensions
```

## Visiting with style

Suppose we want to visit all the cells of a matrix from left to right, and print them one by one.

If you want to keep track of the coordinates where you are in the traversal, you can use a nested `for` in `range` like so:

```
[11]: m = [
 ['a', 'b', 'c'],
 ['d', 'e', 'f'],
 ['g', 'h', 'i'],
 ['m', 'n', 'o'],
]
```

(continues on next page)

(continued from previous page)

```

]

for i in range(len(m)):
 for j in range(len(m[0])):
 print('i:', i, ' j:', j, ' m[i][j]:', m[i][j])
 print("ROW END!")

i: 0 j: 0 m[i][j]: a
i: 0 j: 1 m[i][j]: b
i: 0 j: 2 m[i][j]: c
ROW END!
i: 1 j: 0 m[i][j]: d
i: 1 j: 1 m[i][j]: e
i: 1 j: 2 m[i][j]: f
ROW END!
i: 2 j: 0 m[i][j]: g
i: 2 j: 1 m[i][j]: h
i: 2 j: 2 m[i][j]: i
ROW END!
i: 3 j: 0 m[i][j]: m
i: 3 j: 1 m[i][j]: n
i: 3 j: 2 m[i][j]: o
ROW END!

```

The algorithm is pretty simple, yet a couple of things are worth noting:

- we used `i` as *integer index* for the *rows*
- we used `j` as *integer index* for the *columns*

Those names and types are important, as they are basically standard in math books.

You could maybe dismiss name choices as a pure matter of style, yet:

## WHEN DEALING WITH MATRICES, STYLE \*IS\* SUBSTANCE

**Please adopt the above naming style.**

Those who don't, spend quite a lot of time in *Debugging Hell...* trust us.

### Question - A matter of style 1

Look at the following code that prints again all the cells from left to right (suppose we don't care about printing the coordinates).

Is this *good style* or not? Why?

```
[12]: m = [
 ['a', 'b', 'c'],
 ['d', 'e', 'f'],
 ['g', 'h', 'i'],
 ['m', 'n', 'o'],
]

for i in m:
 for j in i:
```

(continues on next page)

(continued from previous page)

```
 print(j)
 print("ROW END!")

a
b
c
ROW END!
d
e
f
ROW END!
g
h
i
ROW END!
m
n
o
ROW END!
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** that's bad style! We said we reserve *i* and *j* for *integer indexes*, but in the case above, *i* is a pointer to an entire *row* (thus a list), and *j* is the *content* of a cell (thus a string).

</div>

## Question - A matter of style 2

Look at the following code that prints again alle the cells from left to right (suppose we don't care about printing the coordinates).

Is this *good style* or not? Why?

```
[13]: m = [
 ['a', 'b', 'c'],
 ['d', 'e', 'f'],
 ['g', 'h', 'i'],
 ['m', 'n', 'o'],
]

for row in range(len(m)):
 for column in range(len(m[0])):
 print(m[row][column])
 print("ROW END!")
```

a  
b  
c  
ROW END!  
d  
e  
f  
ROW END!  
g  
h

(continues on next page)

(continued from previous page)

```
i
ROW END !
m
n
o
ROW END !
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** that's bad style! We said whenever possible we should use *i* and *j* for *integer indexes*. Instead, in this case we used the variable *row* for an integer index but typically with the name *row* you would expect to refer to the *list* holding the data. Same goes for the *column* variable.

</div>

### Question - A matter of style 3

Look at the following code that prints again alle the cells from left to right (suppose we don't care about printing the coordinates).

Is this *good style* or not? Why?

```
[14]: m = [
 ['a', 'b', 'c'],
 ['d', 'e', 'f'],
 ['g', 'h', 'i'],
 ['m', 'n', 'o'],
]

for row in m:
 for cell in row:
 print(cell)
 print("ROW END!")
```

```
a
b
c
ROW END !
d
e
f
ROW END !
g
h
i
ROW END !
m
n
o
ROW END !
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** The above style is ok. Since we don't explicitly need the position, if we want we can skip holding integer indeces *i* and *j* and directly obtain pointers to the internal lists, *provided we choose proper names for them*, like in this

case with the `row` variable. Notice we named the internal variable `cell` to mean we are referring to the *string content* of it.

```
</div>
```

### How to solve the exercises

All the following exercises are proposed as functions to implement.

---

**REMEMBER:** if the cell is executed and nothing happens, it's because all the assert tests have worked! In such case you probably wrote correct code but careful, these kind of tests are never exhaustive so you could have still made some error.

---

---

**III COMMANDMENT<sup>278</sup>: You shall never reassign function parameters**

---

---

**VI COMMANDMENT<sup>279</sup> You shall use `return` command only if you see written RETURN in the function description!**

---

## Extracting rows and columns

### How to extract a row

One of the first things you might want to do is to extract the `i`-th row. If you're implementing a function that does this, you have basically two choices. Either:

1. return a *pointer* to the *original* row
2. return a *copy* of the row.

Since a copy consumes memory, why should you ever want to return a copy? Sometimes you just don't know which use will be done of the data structure. For example, suppose you got a book of exercises which has empty spaces to write exercises in. It's such a great book everybody in the classroom wants to read it - but you are afraid if the book starts changing hands some careless guy might write on it. To avoid problems, you make a copy of the book and distribute it (let's leave copyright infringement matters aside :-)

### Extracting row pointers

So first let's see what happens when you just return a *pointer* to the *original* row.

**NOTE:** For convenience, at the end of the cell we put a magic call to `jupman.pytut()` which shows the code execution like in Python tutor (for further info about `jupman.pytut()`, [see here<sup>280</sup>](#)). If you execute all the code in Python tutor, you will see that at the end you have two arrow pointers to the row `['a', 'b']`, one starting from `m` list and one from `row` variable.

```
[15]: # WARNING: FOR PYTHON TUTOR TO WORK, REMEMBER TO EXECUTE THIS CELL with Shift+Enter
(it's sufficient to execute it only once)
import jupman
```

<sup>278</sup> <https://en.softpython.org/commandments.html#III-COMMANDMENT>

<sup>279</sup> <https://en.softpython.org/commandments.html#VI-COMMANDMENT>

<sup>280</sup> <https://en.softpython.org/tools/tools-sol.html#Visualizing-the-execution-with-Python-Tutor>

```
[16]: def extrowp(mat, i):
 """ RETURN the ith row from mat
 """
 return mat[i]

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]
row = extrowp(m, 0)
jupman.pytut()

[16]: <IPython.core.display.HTML object>
```

### Extract row with a for

⊕ Now try to implement a version which returns a **copy** of the row.

**QUESTION:** You might be tempted to implement something like this - but it wouldn't work. Why?

```
[17]: # WARNING: WRONG CODE!!!!

def extrow_wrong(mat, i):
 """ RETURN the ith row from mat, as a NEW list"""

 row = []
 row.append(mat[i])
 return row

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]
row = extrow_wrong(m, 0)
jupman.pytut()

[17]: <IPython.core.display.HTML object>
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** The code above adds a LIST as element to another empty list. In other words, it is wrapping the row (which is already a list) into another list. If you check the problem in Python Tutor, you will see an arrow going from row to a list of one element which will contain exactly one arrow to the original row.

</div>

You can build an actual copy in several ways, with a `for`, a slice or a list comprehension. Try to implement all versions, starting with the `for` here. Be sure to check your result with Python Tutor - to visualize python tutor inside the cell output, you might use the special command `jupman.pytut()` at the end of the cell as we did before. If you run the

code with Python Tutor, you should only see *one* arrow going to the original `['a', 'b']` row in `m`, and there should be *another* `['a', 'b']` copy somewhere, with `row` variable pointing to it.

⊗ EXERCISE: Implement the function `extrowf` which RETURNS the `i`-th row from `mat` as a NEW list.

- NOTE: To create a new list use a for cycle which iterates over the elements, *not* the indeces (so don't use range!)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: def extrowf(mat, i):

 row = []
 for x in mat[i]:
 row.append(x)
 return row

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extrowf(m, 0) == ['a', 'b']
assert extrowf(m, 1) == ['c', 'd']
assert extrowf(m, 2) == ['a', 'e']

check it didn't change the original matrix !
r = extrowf(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

uncomment to visualize execution here
#jupman.pytut()
```

</div>

```
[18]: def extrowf(mat, i):
 raise Exception('TODO IMPLEMENT ME !')

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extrowf(m, 0) == ['a', 'b']
assert extrowf(m, 1) == ['c', 'd']
assert extrowf(m, 2) == ['a', 'e']

check it didn't change the original matrix !
r = extrowf(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

uncomment to visualize execution here
#jupman.pytut()
```

## Extract row with range

Let's first rapidly see `range(n)`. Maybe you think it should return a sequence of integers, from zero to  $n - 1$ . Is it really like this?

```
[19]: range(5)
```

```
[19]: range(0, 5)
```

Maybe you expected something like a list `[0, 1, 2, 3, 4]`, instead we discovered that Python is quite lazy here: as a matter of fact, `range(n)` returns an *iterable* object, which is not a real sequence materialized in memory.

To take a real integer list, we must explicitly ask this iterable object to give us the objects one by one.

When you write `for i in range(s)` the for loop is doing exactly this, at each round it asks the object `range` to generate a number from the sequence. If we want the whole sequence materialized in memory, we can generate it by converting the `range` into a list object:

```
[20]: list(range(5))
```

```
[20]: [0, 1, 2, 3, 4]
```

Be careful, though. According to the sequence dimension, this might be dangerous. A billion elements list might saturate your computer RAM (in 2020 notebooks often have 4 gigabytes of RAM, that is, 4 billion bytes).

⊕ **EXERCISE:** Now implement the `extrowr` iterating over a range of row indexes:

- **NOTE 1:** To create a new list use a `for` loop
- **NOTE 2:** remember to use a new name for the column index!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[21]: def extrowr(mat, i):
 """ RETURN the ith row from mat as a NEW list
 """

 row = []
 for j in range(len(mat[0])):
 row.append(mat[i][j])
 return row

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extrowr(m, 0) == ['a', 'b']
assert extrowr(m, 1) == ['c', 'd']
assert extrowr(m, 2) == ['a', 'e']

check it didn't change the original matrix !
r = extrowr(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'
```

(continues on next page)

(continued from previous page)

```
uncomment to visualize execution here
#jupman.pytut()
```

```
</div>
```

```
[21]: def extrowr(mat, i):
 """ RETURN the ith row from mat as a NEW list
 """
 raise Exception('TODO IMPLEMENT ME !')

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extrowr(m, 0) == ['a', 'b']
assert extrowr(m, 1) == ['c', 'd']
assert extrowr(m, 2) == ['a', 'e']

check it didn't change the original matrix !
r = extrowr(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

uncomment to visualize execution here
#jupman.pytut()
```

## Extract row with a slice

⊕ Remember slices return a *copy* of a list? Now try using them.

Implement `extrows`, which RETURN the i-th row from `mat` as a NEW list.

- **NOTE:** To create it, use slices

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[22]: def extrows(mat, i):

 return mat[i][:] # if you omit start end indexes, you get a copy of the
 ↵whole list

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extrows(m, 0) == ['a', 'b']
assert extrows(m, 1) == ['c', 'd']
assert extrows(m, 2) == ['a', 'e']
```

(continues on next page)

(continued from previous page)

```
check it didn't change the original matrix !
r = extrows(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

uncomment to visualize execution here
#jupman.pytut()
```

&lt;/div&gt;

```
[22]: def extrows(mat, i):
 raise Exception('TODO IMPLEMENT ME !')

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extrows(m, 0) == ['a', 'b']
assert extrows(m, 1) == ['c', 'd']
assert extrows(m, 2) == ['a', 'e']

check it didn't change the original matrix !
r = extrows(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

uncomment to visualize execution here
#jupman.pytut()
```

## Extract row with list comprehension

⊕ Implement `extrowc`, which RETURNS the `i`-th row from `mat` as a NEW list, using a *list comprehension*.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[23]: def extrowc(mat, i):

 return [x for x in mat[i]]

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extrowc(m, 0) == ['a', 'b']
assert extrowc(m, 1) == ['c', 'd']
assert extrowc(m, 2) == ['a', 'e']

check it didn't change the original matrix !
```

(continues on next page)

(continued from previous page)

```
r = extrowc(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

#jupman.pytut()
```

</div>

```
[23]: def extrowc(mat, i):
 raise Exception('TODO IMPLEMENT ME !')

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extrowc(m, 0) == ['a', 'b']
assert extrowc(m, 1) == ['c', 'd']
assert extrowc(m, 2) == ['a', 'e']

check it didn't change the original matrix !
r = extrowc(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'

#jupman.pytut()
```

### Extract column with a for

⊕⊕ Now try extracting a column at  $j$ th position. This time we will be forced to create a new list, so we don't have to wonder if we need to return a pointer or a copy.

Implement `extcolf`, which RETURN the  $j$ -th column from `mat`. To create it, use a `for` loop.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[24]: def extcolf(mat, j):

 ret = []
 for row in mat:
 ret.append(row[j])
 return ret

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extcolf(m, 0) == ['a', 'c', 'a']
assert extcolf(m, 1) == ['b', 'd', 'e']
```

(continues on next page)

(continued from previous page)

```
check returned column does not modify m
c = extcolf(m, 0)
c[0] = 'z'
assert m[0][0] == 'a'

#jupman.pytut()
```

&lt;/div&gt;

```
[24]: def extcolf(mat, j):
 raise Exception('TODO IMPLEMENT ME !')

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extcolf(m, 0) == ['a', 'c', 'a']
assert extcolf(m, 1) == ['b', 'd', 'e']

check returned column does not modify m
c = extcolf(m, 0)
c[0] = 'z'
assert m[0][0] == 'a'

#jupman.pytut()
```

## Extract column with a list comprehension

⊕⊕ Implement `extcolc`, which RETURNS the  $j$ -th column from `mat`: to create it, use a list comprehension.

[Show solution](#)  
[Hide](#)

```
[25]: def extcolc(mat, j):

 return [row[j] for row in mat]

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extcolc(m, 0) == ['a', 'c', 'a']
assert extcolc(m, 1) == ['b', 'd', 'e']

check returned column does not modify m
c = extcolc(m, 0)
c[0] = 'z'
assert m[0][0] == 'a'

#jupman.pytut()
```

```
</div>

[25]: def extcolc(mat, j):
 raise Exception('TODO IMPLEMENT ME !')

m = [
 ['a', 'b'],
 ['c', 'd'],
 ['a', 'e'],
]

assert extcolc(m, 0) == ['a', 'c', 'a']
assert extcolc(m, 1) == ['b', 'd', 'e']

check returned column does not modify m
c = extcolc(m, 0)
c[0] = 'z'
assert m[0][0] == 'a'

#jupman.pytut()
```

### Creating new matrices

#### empty matrix

⊕⊕ There are several ways to create a new empty 3x5 matrix as lists of lists which contains zeros.

Implement `empty_matrix`, which RETURN a NEW matrix nxn as a list of lists filled with zeroes

- use two nested `for` cycles:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[26]: def empty_matrix(n, m):

 ret = []
 for i in range(n):
 row = []
 ret.append(row)
 for j in range(m):
 row.append(0)
 return ret

assert empty_matrix(1, 1) == [[0]]

assert empty_matrix(1, 2) == [[0, 0]]

assert empty_matrix(2, 1) == [[0],
 [0]]

assert empty_matrix(2, 2) == [[0, 0],
 [0, 0]]

assert empty_matrix(3, 3) == [[0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
[0, 0, 0],
[0, 0, 0]]
```

&lt;/div&gt;

```
[26]: def empty_matrix(n, m):
 raise Exception('TODO IMPLEMENT ME !')

assert empty_matrix(1, 1) == [[0]]

assert empty_matrix(1, 2) == [[0, 0]]

assert empty_matrix(2, 1) == [[0],
 [0]]

assert empty_matrix(2, 2) == [[0, 0],
 [0, 0]]

assert empty_matrix(3, 3) == [[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
```

### **empty\_matrix the elegant way**

To create a new list of 3 elements filled with zeros, you can write like this:

```
[27]: [0]*3
[27]: [0, 0, 0]
```

The \* is kind of multiplying the elements in a list

Given the above, to create a 5x3 matrix filled with zeros, which is a list of seemingly equal lists, you might then be tempted to write like this:

```
[28]: # WRONG
[[0]*3]*5
[28]: [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Why is that (possibly) wrong? Let's try to inspect it in Python Tutor:

```
[29]: bad = [[0]*3]*5
jupman.pyput()
[29]: <IPython.core.display.HTML object>
```

If you look closely, you will see many arrows pointing to the same list of 3 zeros. This means that if we change one number, we will apparently change 5 of them in the whole column !

The right way to create a matrix as list of lists with zeroes is the following:

```
[30]: # CORRECT
[[0]*3 for i in range(5)]
[30]: [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

**EXERCISE:** Try creating a matrix with 7 rows and 4 columns and fill it with 5.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[31]: # write here
```

```
[[5]*4 for i in range(7)]
```

```
[31]: [[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5]]
```

</div>

```
[31]: # write here
```

```
[31]: [[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5]]
```

### deep\_clone

⊗⊗ Let's try to produce a *complete* clone of the matrix, also called a *deep clone*, by creating a copy of the external list *and* also the internal lists representing the rows.

**QUESTION:** You might be tempted to write code like this, but it will not work. Why?

```
[32]:
```

```
WARNING: WRONG CODE
def deep_clone_wrong(mat):
 """ RETURN a NEW list of lists which is a COMPLETE DEEP clone
 of mat (which is a list of lists)
 """
 return mat[:]

m = [
 ['a', 'b'],
 ['b', 'd']
]

res = deep_clone_wrong(m)

Notice you will have arrows in res list going to the _original_ mat. We don't want
this !
jupman.pytut()
```

```
[32]: <IPython.core.display.HTML object>
```

```
Show answer<div class="jupman-sol jupman-sol-question" style="display:none">
```

**ANSWER:** `return mat[:]` is not sufficient, because it's a SHALLOW clone, and only copies the *external* list and not also the internal ones ! Note you will have rows in the `res` list which goes to the original matrix. We don't want this!

</div>

To fix the above code, you will need to iterate through the rows and *for each* row create a copy of that row.

⊕⊕ **EXERCISE:** Implement `deep_clone`, which RETURNS a NEW list as a complete DEEP CLONE of `mat` (which is a list of lists)

**NOTE:** the exercise can be solved very quickly by using the function `deepcopy`<sup>281</sup> but we invite you to solve it without for now.

```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[33]:

```
def deep_clone(mat):

 ret = []
 for row in mat:
 ret.append(row[:])
 return ret

m = [['a', 'b'],
 ['b', 'd']]

res = [['a', 'b'],
 ['b', 'd']]

verify the copy
c = deep_clone(m)
assert c == res

verify it is a DEEP copy (that is, it created also clones of the rows!)
c[0][0] = 'z'
assert m[0][0] == 'a'
```

</div>

[33]:

```
def deep_clone(mat):
 raise Exception('TODO IMPLEMENT ME !')

m = [['a', 'b'],
 ['b', 'd']]

res = [['a', 'b'],
 ['b', 'd']]

verify the copy
c = deep_clone(m)
assert c == res
```

(continues on next page)

<sup>281</sup> <https://en.softpython.org/lists/lists3-sol.html#deepcopy-function>

(continued from previous page)

```
verify it is a DEEP copy (that is, it created also clones of the rows!)
c[0][0] = 'z'
assert m[0][0] == 'a'
```

## Modifying matrices

### fillc

⊕⊕ Implement the function `fillc` which takes as input `mat` (a list of lists with dimension `nrows x ncol`) and MODIFIES it by placing the character `c` inside all the matrix cells.

- to visit the matrix use for in range cycles

Ingredients:

- find matrix dimension
- two nested fors
- use range

---

#### NOTE: This function returns nothing!

If in the function text it is not mentioned to return values, DO NOT place the `return`. If by chance you put it anyway it is not the world's end, but to avoid confusion is much better having a behaviour consistent with the text.

---

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[34]: def fillc(mat, c):
```

```
 nrows = len(mat)
 ncols = len(mat[0])

 for i in range(nrows):
 for j in range(ncols):
 mat[i][j] = c

m1 = [['a']]
m2 = [['z']]
fillc(m1, 'z')
assert m1 == m2

m3 = [['a']]
m4 = [['y']]
fillc(m3, 'y')
assert m3 == m4

m5 = [['a', 'b']]
m6 = [['z', 'z']]
fillc(m5, 'z')
assert m5 == m6
```

(continues on next page)

(continued from previous page)

```

m7 = [['a','b','c'],
 ['d','e','f'],
 ['g','h','i']]

m8 = [['y','y','y'],
 ['y','y','y'],
 ['y','y','y']]
fillc(m7,'y')
assert m7 == m8

j 0 1
m9 = [['a','b'], # 0
 ['c','d'], # 1
 ['e','f']] # 2

m10 = [['x','x'], # 0
 ['x','x'], # 1
 ['x','x']] # 2
fillc(m9, 'x')
assert m9 == m10

```

&lt;/div&gt;

```
[34]: def fillc(mat, c):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a']]
m2 = [['z']]
fillc(m1,'z')
assert m1 == m2

m3 = [['a']]
m4 = [['y']]
fillc(m3,'y')
assert m3 == m4

m5 = [['a','b']]
m6 = [['z','z']]
fillc(m5,'z')
assert m5 == m6

m7 = [['a','b','c'],
 ['d','e','f'],
 ['g','h','i']]

m8 = [['y','y','y'],
 ['y','y','y'],
 ['y','y','y']]
fillc(m7,'y')
assert m7 == m8

j 0 1
m9 = [['a','b'], # 0
 ['c','d'], # 1
 ['e','f']] # 2

m10 = [['x','x'], # 0
 ['x','x'], # 1
 ['x','x']] # 2

```

(continues on next page)

(continued from previous page)

```
['x', 'x'], # 1
['x', 'x']] # 2
fillc(m9, 'x')
assert m9 == m10
```

## fillx

⊕⊕ Takes a matrix mat as list of lists and a column index j, and MODIFIES mat by placing the 'x' character in all cells of the j-th column.

Example:

```
m = [
 ['a', 'b', 'c', 'd'],
 ['e', 'f', 'g', 'h'],
 ['i', 'l', 'm', 'n']
]
```

After the call to

```
fillx(m, 2)
```

the matrix m will be changed like this:

```
>>> print(m)
[
 ['a', 'b', 'x', 'd'],
 ['e', 'f', 'x', 'h'],
 ['i', 'l', 'x', 'n']
]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[35]: def fillx(mat, j):

 for row in mat:
 row[j] = 'x'

m1 = [['a']]
fillx(m1, 0)
assert m1 == [['x']]

m2 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillx(m2, 0)
assert m2 == [['x', 'b'],
 ['x', 'd'],
 ['x', 'f']]

m3 = [['a', 'b'],
```

(continues on next page)

(continued from previous page)

```

 ['c', 'd'],
 ['e', 'f']]
fillx(m3, 1)
assert m3 == [['a', 'x'],
 ['c', 'x'],
 ['e', 'x']]

m4 = [['a', 'b', 'c', 'd'],
 ['e', 'f', 'g', 'h'],
 ['i', 'l', 'm', 'n']]
fillx(m4, 2)
assert m4 == [['a', 'b', 'x', 'd'],
 ['e', 'f', 'x', 'h'],
 ['i', 'l', 'x', 'n']]

```

&lt;/div&gt;

```
[35]: def fillx(mat, j):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a']]
fillx(m1, 0)
assert m1 == [['x']]

m2 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillx(m2, 0)
assert m2 == [['x', 'b'],
 ['x', 'd'],
 ['x', 'f']]

m3 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillx(m3, 1)
assert m3 == [['a', 'x'],
 ['c', 'x'],
 ['e', 'x']]

m4 = [['a', 'b', 'c', 'd'],
 ['e', 'f', 'g', 'h'],
 ['i', 'l', 'm', 'n']]
fillx(m4, 2)
assert m4 == [['a', 'b', 'x', 'd'],
 ['e', 'f', 'x', 'h'],
 ['i', 'l', 'x', 'n']]
```

### fillz

⊕⊕ Takes a matrix `mat` as list of lists and a row index `i`, and MODIFIES `mat` by placing the character '`z`' in all the cells of the `i`-th row.

Example:

```
m = [
 ['a', 'b'],
 ['c', 'd'],
 ['e', 'f'],
 ['g', 'h']
]
```

After the call to

```
>>> fillz(m, 2)
```

the matrix `m` will be changed like so:

```
>>> print(m)

[
 ['a', 'b'],
 ['c', 'd'],
 ['z', 'z'],
 ['g', 'h']
]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[36]: def fillz(mat, i):

 ncol=len(mat[0])
 for j in range(ncol):
 mat[i][j] = 'z'

m1 = [['a']]
fillz(m1,0)
assert m1 == [['z']]

m2 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillz(m2,0)
assert m2 == [['z', 'z'],
 ['c', 'd'],
 ['e', 'f']]

m3 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillz(m3,1)
assert m3 == [['a', 'b'],
 ['z', 'z'],
 ['e', 'f']]
```

(continues on next page)

(continued from previous page)

```

 ['e', 'f']]
m4 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillz(m4, 2)
assert m4 == [['a', 'b'],
 ['c', 'd'],
 ['z', 'z']]

```

&lt;/div&gt;

```
[36]: def fillz(mat, i):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a']]
fillz(m1, 0)
assert m1 == [['z']]

m2 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillz(m2, 0)
assert m2 == [['z', 'z'],
 ['c', 'd'],
 ['e', 'f']]

m3 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillz(m3, 1)
assert m3 == [['a', 'b'],
 ['z', 'z'],
 ['e', 'f']]

m4 = [['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]
fillz(m4, 2)
assert m4 == [['a', 'b'],
 ['c', 'd'],
 ['z', 'z']]
```

## stitch\_down

⊗⊗ Given matrices `mat1` and `mat2` as list of lists, with `mat1` of size  $u \times n$  and `mat2` of size  $d \times n$ , RETURN a NEW matrix of size  $(u+d) \times n$  as list of lists, by stitching second mat to the bottom of mat1

- **NOTE:** by NEW matrix we intend a matrix with no pointers to original rows (see previous deep clone exercise)
- for examples, see asserts

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[37]: def stitch_down(mat1, mat2):

 res = []
 for row in mat1:
 res.append(row[:])
 for row in mat2:
 res.append(row[:])
 return res

m1 = [['a']]
m2 = [['b']]
assert stitch_down(m1, m2) == [['a'],
 ['b']]

check we are giving back a deep clone
s = stitch_down(m1, m2)
s[0][0] = 'z'
assert m1[0][0] == 'a'

m1 = [['a','b','c'],
 ['d','b','a']]
m2 = [['f','b','h'],
 ['g','h','w']]
assert stitch_down(m1, m2) == [['a','b','c'],
 ['d','b','a'],
 ['f','b','h'],
 ['g','h','w']]
```

</div>

```
[37]: def stitch_down(mat1, mat2):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a']]
m2 = [['b']]
assert stitch_down(m1, m2) == [['a'],
 ['b']]

check we are giving back a deep clone
s = stitch_down(m1, m2)
s[0][0] = 'z'
assert m1[0][0] == 'a'

m1 = [['a','b','c'],
 ['d','b','a']]
m2 = [['f','b','h'],
 ['g','h','w']]
assert stitch_down(m1, m2) == [['a','b','c'],
 ['d','b','a'],
 ['f','b','h'],
 ['g','h','w']]
```

## stitch\_up

⊗⊗ Given matrices `mat1` and `mat2` as list of lists, with `mat1` of size  $u \times n$  and `mat2` of size  $d \times n$ , RETURN a NEW matrix of size  $(u+d) \times n$  as list of lists, by stitching first mat to the bottom of mat2

- **NOTE:** by NEW matrix we intend a matrix with no pointers to original rows (see previous `deep_clone` exercise)
- To implement this function, use a call to the method `stitch_down` you implemented before.
- For examples, see assert

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[38]: def stitch_up(mat1, mat2):
 return stitch_down(mat2, mat1)

m1 = [['a']]
m2 = [['b']]
assert stitch_up(m1, m2) == [['b'],
 ['a']]

check we are giving back a deep clone
s = stitch_up(m1, m2)
s[0][0] = 'z'
assert m1[0][0] == 'a'

m1 = [['a', 'b', 'c'],
 ['d', 'b', 'a']]
m2 = [['f', 'b', 'h'],
 ['g', 'h', 'w']]

assert stitch_up(m1, m2) == [['f', 'b', 'h'],
 ['g', 'h', 'w'],
 ['a', 'b', 'c'],
 ['d', 'b', 'a']]
```

</div>

```
[38]: def stitch_up(mat1, mat2):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a']]
m2 = [['b']]
assert stitch_up(m1, m2) == [['b'],
 ['a']]

check we are giving back a deep clone
s = stitch_up(m1, m2)
s[0][0] = 'z'
assert m1[0][0] == 'a'

m1 = [['a', 'b', 'c'],
 ['d', 'b', 'a']]
m2 = [['f', 'b', 'h'],
```

(continues on next page)

(continued from previous page)

```
['g', 'h', 'w']]\n\nassert stitch_up(m1, m2) == [['f', 'b', 'h'],\n ['g', 'h', 'w'],\n ['a', 'b', 'c'],\n ['d', 'b', 'a']]
```

## stitch\_right

⊕⊕⊕ Given matrices mata and matb as list of lists, with mata of size n x 1 and matb of size n x r, RETURN a NEW matrix of size n x (1 + r) as list of lists, by stitching second matb to the right end of mata

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[39]:

```
def stitch_right(mata,matb):\n\n ret = []\n for i in range(len(mata)):\n row_to_add = mata[i][:]\n row_to_add.extend(matb[i])\n ret.append(row_to_add)\n return ret\n\nma1 = [['a', 'b', 'c'],\n ['d', 'b', 'a']]\nmb1 = [['f', 'b'],\n ['g', 'h']]\n\nassert stitch_right(ma1, mb1) == [['a', 'b', 'c', 'f', 'b'],\n ['d', 'b', 'a', 'g', 'h']]
```

</div>

[39]:

```
def stitch_right(mata,matb):\n raise Exception('TODO IMPLEMENT ME !')\n\nma1 = [['a', 'b', 'c'],\n ['d', 'b', 'a']]\nmb1 = [['f', 'b'],\n ['g', 'h']]\n\nassert stitch_right(ma1, mb1) == [['a', 'b', 'c', 'f', 'b'],\n ['d', 'b', 'a', 'g', 'h']]
```

**insercol**

⊗⊗ Given a matrix `mat` as list of lists, a column index `j` and a list `new_col`, write a function `insercol(mat, j, new_col)` which MODIFIES `mat` inserting the new column at position `j`.

Example - given:

```
[40]: # 0 1 2
m = [
 [5, 4, 6],
 [4, 7, 1],
 [3, 2, 6],
]
```

By calling

```
>>> insercol(m, 2, [7, 9, 3])
```

`m` will be MODIFIED with the insertion of column `[7, 9, 3]` at position `j=2`

```
>>> m
 # 0 1 2 3
[
 [5, 4, 7, 6],
 [4, 7, 9, 1],
 [3, 2, 3, 6],
]
```

- for other examples, see asserts
- **HINT:** lists already have a handy method `.insert`, so there is isn't much code to write, just write the right one ;)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[41]: def insercol(mat, j, nuova_col):

 for i in range(len(mat)):
 mat[i].insert(j, nuova_col[i])

m1 = [
 [5]
]
assert insercol(m1, 1, [8]) == None # the function returns nothing!
assert m1 == [[5, 8]]

m2 = [[5]]
insercol(m2, 0, [8])
assert m2 == [[8, 5]]

m3 = [[5, 4, 2],
 [8, 9, 3]]
insercol(m3, 1, [7, 6])
assert m3 == [[5, 7, 4, 2],
```

(continues on next page)

(continued from previous page)

```
[8, 6, 9, 3]]

m4 = [[5, 4, 6],
 [4, 7, 1],
 [3, 2, 6]]
insercol(m4, 2, [7, 9, 3])

assert m4 == [[5, 4, 7, 6],
 [4, 7, 9, 1],
 [3, 2, 3, 6]]
```

&lt;/div&gt;

[41]:

```
def insercol(mat, j, nuova_col):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [
 [5]
]
assert insercol(m1, 1, [8]) == None # the function returns nothing!
assert m1 == [[5, 8]]

m2 = [[5]]
insercol(m2, 0, [8])
assert m2 == [[8, 5]]

m3 = [[5, 4, 2],
 [8, 9, 3]]
insercol(m3, 1, [7, 6])
assert m3 == [[5, 7, 4, 2],
 [8, 6, 9, 3]]

m4 = [[5, 4, 6],
 [4, 7, 1],
 [3, 2, 6]]
insercol(m4, 2, [7, 9, 3])

assert m4 == [[5, 4, 7, 6],
 [4, 7, 9, 1],
 [3, 2, 3, 6]]
```

## threshold

⊕⊕ Takes a matrix as a list of lists (every list has the same dimension) and RETURN a NEW matrix as list of lists where there is `True` if the corresponding input element is greater than `t`, otherwise return `False`

Ingredients:

- a variable for the matrix to return
- for each original row, we need to create a new list

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[42]: def threshold(mat, t):
 ret = []
 for row in mat:
 new_row = []
 ret.append(new_row)
 for el in row:
 new_row.append(el > t)

 return ret

morig = [[1, 4, 2],
 [7, 9, 3]]

m1 = [[1, 4, 2],
 [7, 9, 3]]

r1 = [[False, False, False],
 [True, True, False]]
assert threshold(m1, 4) == r1
assert m1 == morig # verify original didn't change

m2 = [[5, 2],
 [3, 7]]

r2 = [[True, False],
 [False, True]]
assert threshold(m2, 4) == r2
```

</div>

```
[42]: def threshold(mat, t):
 raise Exception('TODO IMPLEMENT ME !')

morig = [[1, 4, 2],
 [7, 9, 3]]

m1 = [[1, 4, 2],
 [7, 9, 3]]

r1 = [[False, False, False],
 [True, True, False]]
assert threshold(m1, 4) == r1
assert m1 == morig # verify original didn't change

m2 = [[5, 2],
 [3, 7]]

r2 = [[True, False],
 [False, True]]
assert threshold(m2, 4) == r2
```

**swap\_rows**

⊕⊕ We will try swapping a couple of rows of a matrix

There are several ways to proceed. Before continuing, make sure to know how to exchange two values by solving this simple exercise - check your result in Python Tutor

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[43]: x = 3
y = 7

write here the code to swap x and y (do not directly use the constants 3 and 7!)

k = x
x = y
y = k

#jupman.pytut()
```

</div>

```
[43]: x = 3
y = 7

write here the code to swap x and y (do not directly use the constants 3 and 7!)
```

⊕⊕ Takes a matrix mat as list of lists, and RETURN a NEW matrix where rows at indexes i1 and i2 are swapped

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[44]: def swap_rows(mat, i1, i2):
```

```
 # deep clones
 ret = []
 for row in mat:
 ret.append(row[:])
 #swaps
 s = ret[i1]
 ret[i1] = ret[i2]
 ret[i2] = s
 return ret
```

```
m1 = [['a', 'd'],
 ['b', 'e'],
 ['c', 'f']]
```

```
r1 = swap_rows(m1, 0, 2)
```

```
assert r1 == [['c', 'f'],
 ['b', 'e'],
 ['a', 'd']]
```

(continues on next page)

(continued from previous page)

```

['a', 'd']]

r1[0][0] = 'z'
assert m1[0][0] == 'a'

m2 = [['a', 'd'],
 ['b', 'e'],
 ['c', 'f']]

swap with itself should in fact generate a deep clone
r2 = swap_rows(m2, 0, 0)

assert r2 == [['a', 'd'],
 ['b', 'e'],
 ['c', 'f']]

r2[0][0] = 'z'
assert m2[0][0] == 'a'
```

&lt;/div&gt;

```
[44]: def swap_rows(mat, i1, i2):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a', 'd'],
 ['b', 'e'],
 ['c', 'f']]

r1 = swap_rows(m1, 0, 2)

assert r1 == [['c', 'f'],
 ['b', 'e'],
 ['a', 'd']]

r1[0][0] = 'z'
assert m1[0][0] == 'a'

m2 = [['a', 'd'],
 ['b', 'e'],
 ['c', 'f']]

swap with itself should in fact generate a deep clone
r2 = swap_rows(m2, 0, 0)

assert r2 == [['a', 'd'],
 ['b', 'e'],
 ['c', 'f']]

r2[0][0] = 'z'
assert m2[0][0] == 'a'
```

### swap\_cols

⊗⊗ Takes a matrix mat and two column indeces j1 and j2 and RETURN a NEW matrix where the columns j1 and j2 are swapped

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[45]: def swap_cols(mat, j1, j2):

 ret = []
 for row in mat:
 new_row = row[:]
 new_row[j1] = row[j2]
 new_row[j2] = row[j1]
 ret.append(new_row)
 return ret

m1 = [['a', 'b', 'c'],
 ['d', 'e', 'f']]

r1 = swap_cols(m1, 0, 2)

assert r1 == [['c', 'b', 'a'],
 ['f', 'e', 'd']]

r1[0][0] = 'z'
assert m1[0][0] == 'a'
```

</div>

```
[45]: def swap_cols(mat, j1, j2):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a', 'b', 'c'],
 ['d', 'e', 'f']]

r1 = swap_cols(m1, 0, 2)

assert r1 == [['c', 'b', 'a'],
 ['f', 'e', 'd']]

r1[0][0] = 'z'
assert m1[0][0] == 'a'
```

### Continue

Go on with [Matrices 2 - other exercises<sup>282</sup>](#)

[ ]:

<sup>282</sup> <https://en.softpython.org/matrices-lists/matrices-lists2-sol.html>

## 7.2.2 Matrices 2: list of lists - other exercises

### Download exercises zip

Browse files online<sup>283</sup>

As always the only true way to understand a topic is by doing more exercises, so here they are!

### What to do

- unzip exercises in a folder, you should get something like this:

```
matrices-lists
 matrices-lists1.ipynb
 matrices-lists1-sol.ipynb
 matrices-lists2.ipynb
 matrices-lists2-sol.ipynb
 matrices-lists3-chal.ipynb
 jupman.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `matrices-lists/matrices-lists2.ipynb`
- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### Exercise - diag

`diag` extracts the diagonal of a matrix. To do so, `diag` requires an  $n \times n$  matrix as input. To make sure we actually get an  $n \times n$  matrix, this time you will have to validate the input, that is check if the number of rows is equal to the number of columns (as always we assume the matrix has at least one row and at least one column). If the matrix is not  $n \times n$ , the function should stop raising an exception. In particular, it should raise a `ValueError`<sup>284</sup>, which is the standard Python exception to raise when the expected input is not correct and you can't find any other more specific error.

Just for illustrative purposes, we show here the index numbers `i` and `j` and avoid putting apices around strings:

```
\ j 0,1,2,3
i
 [
0 [a,b,c,d],
1 [e,f,g,h],
```

(continues on next page)

<sup>283</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/matrices-lists>

<sup>284</sup> <https://docs.python.org/3/library/exceptions.html#ValueError>

(continued from previous page)

```
2 [p,q,r,s],
3 [t,u,v,z]
]
```

Let's see a step by step execution:

```
\ j 0,1,2,3
i
[
extract from row at i=0 --> 0 [a,b,c,d], 'a' is extracted from mat[0][0]
 1 [e,f,g,h],
 2 [p,q,r,s],
 3 [t,u,v,z]
]
```

```
\ j 0,1,2,3
i
[
extract from row at i=1 --> 0 [a,b,c,d],
 1 [e,f,g,h], 'f' is extracted from mat[1][1]
 2 [p,q,r,s],
 3 [t,u,v,z]
]
```

```
\ j 0,1,2,3
i
[
extract from row at i=2 --> 0 [a,b,c,d],
 1 [e,f,g,h],
 2 [p,q,r,s], 'r' is extracted from mat[2][2]
 3 [t,u,v,z]
]
```

```
\ j 0,1,2,3
i
[
extract from row at i=3 --> 0 [a,b,c,d],
 1 [e,f,g,h],
 2 [p,q,r,s],
 3 [t,u,v,z] 'z' is extracted from mat[3][3]
]
```

From the above, we notice we need elements from these indeces:

```
i, j
1, 1
2, 2
3, 3
```

There are two ways to solve this exercise, one is to use a double `for` (a nested for to be precise) while the other method uses only one `for`. Try to solve it in both ways. How many steps do you need with double for? and with only one?

⊕⊕ **EXERCISE:** Implement the `diag` function, which given an  $n \times n$  matrix `mat` as a list of lists, RETURN a list which contains the elemets in the diagonal (top left to bottom right corner).

- if `mat` is not  $n \times n$  raise `ValueError`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]:

```
def diag(mat):

 if len(mat) != len(mat[0]):
 raise ValueError("Matrix should be nxn, found instead %s x %s" % (len(mat), len(mat[0])))
 ret = []
 for i in range(len(mat)):
 ret.append(mat[i][i])
 return ret

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`
m = [['a','b','c'],
 ['d','e','f'],
 ['g','h','i']]

assert diag(m) == ['a','e','i']

try:
 diag([['a','b']]) # 1x2 dimension, not square

 raise Exception("SHOULD HAVE FAILED !") # if diag raises an exception which is
ValueError as we
expect it to do, the code should never
arrive here

except ValueError: # this only catches ValueError. Other types of errors are not
caught
 pass # In an except clause you always need to put some code.
Here we put a placeholder just to fill in
TEST END
```

&lt;/div&gt;

[2]:

```
def diag(mat):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`
m = [['a','b','c'],
 ['d','e','f'],
 ['g','h','i']]

assert diag(m) == ['a','e','i']

try:
 diag([['a','b']]) # 1x2 dimension, not square

 raise Exception("SHOULD HAVE FAILED !") # if diag raises an exception which is
ValueError as we
```

(continues on next page)

(continued from previous page)

```
expect it to do, the code should never
→arrive here

except ValueError: # this only catches ValueError. Other types of errors are not
→caught
 pass # In an except clause you always need to put some code.
 # Here we put a placeholder just to fill in
TEST END
```

### Exercise - anti\_diag

⊕ Given an nxn matrix mat as a list of lists, RETURN a list which contains the elements in the antidiagonal (top right to bottom left corner).

- If mat is not nxn raise ValueError

Before implementing it, be sure to write down understand the required indeces as we did in the example for the *diag* function.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: def anti_diag(mat):

 n = len(mat)
 ret = []
 for i in range(n):
 ret.append(mat[i][n-i-1])
 return ret

m = [['a','b','c'],
 ['d','e','f'],
 ['g','h','i']]

assert anti_diag(m) == ['c','e','g']

If you have doubts about the indexes remember to try it in python tutor !
jupman.pytut()
```

</div>

```
[3]: def anti_diag(mat):
 raise Exception('TODO IMPLEMENT ME !')

m = [['a','b','c'],
 ['d','e','f'],
 ['g','h','i']]

assert anti_diag(m) == ['c','e','g']

If you have doubts about the indexes remember to try it in python tutor !
jupman.pytut()
```

### Exercise - is\_utriang

⊕⊕⊕ You will now try to iterate only the lower triangular half of a matrix. Let's look at an example:

```
[4]: m = [
 [3, 2, 5, 8],
 [0, 6, 2, 3],
 [0, 0, 4, 9],
 [0, 0, 0, 5]
]
```

Just for illustrative purposes, we show here the index numbers *i* and *j*:

```
\ j 0,1,2,3
i
[
0 [3,2,5,8],
1 [0,6,2,3],
2 [0,0,4,9],
3 [0,7,0,5]
]
```

Let's see a step by step execution an a non-upper triangular matrix:

```
\ j 0,1,2,3
i
[
0 [3,2,5,8],
start from row at index i=1 -> 1 [0,6,2,3], Check until column limit j=0 included
2 [0,0,4,9],
3 [0,7,0,5]
]
```

One zero is found, time to check next row.

```
\ j 0,1,2,3
i
[
0 [3,2,5,8],
1 [0,6,2,3],
check row at index i=2 ---> 2 [0,0,4,9], Check until column limit j=1 included
3 [0,7,0,5]
]
```

Two zeros are found. Time to check next row.

```
\ j 0,1,2,3
i
[
0 [3,2,5,8],
1 [0,6,2,3],
2 [0,0,4,9],
check row at index i=3 ---> 3 [0,7,0,5] Check until column limit j=2 included
] BUT can stop sooner at j=1 because
 number at j=1 is different from zero.
 As soon as 7 is found, can return
→False
In this case the matrix is not upper
```

(continues on next page)

(continued from previous page)

triangular
------------

---

**VII COMMANDMENT<sup>285</sup> You shall also write on paper!**

---

When you develop these algorithms, it is fundamental to write down a step by step example like the above to get a clear picture of what is happening. Also, if you write down the indeces correctly, you will easily be able to derive a generalization. To find it, try to further write the found indeces in a table.

For example, from above for each row index  $i$  we can easily find out which limit index  $j$  we need to reach for our hunt for zeros:

$i$	limit $j$ (included)	Notes
1	0	we start from row at index $i=1$
2	1	
3	2	

From the table, we can see the limit for  $j$  can be calculated in terms of the current row index  $i$  with the simple formula  $i - 1$

The fact you need to span through rows and columns suggest you need two `fors`, one for rows and one for columns - that is, a *nested for*.

- please use ranges of indexes to carry out the task (`no for row in mat ..`)
- please use letter  $i$  as index for rows,  $j$  as index of columns and in case you need it  $n$  letter as matrix dimension

**HINT 1:** remember you can set range to start from a specific index, like `range(3, 7)` will start from 3 and end to 6 *included* (last 7 is *excluded*!)

**HINT 2:** To implement this, it's best looking for numbers *different* from zero. As soon as you find one, you can stop the function and return `False`. Only after *all* the number checking is done you can return `True`.

Finally, be reminded of the following:

---

**II COMMANDMENT<sup>286</sup> Whenever you introduce a variable with a for cycle, such variable must be new**

---

If you defined a variable before, you shall not reintroduce it in a `for`, since it's confusing and error prone.

So avoid these sins:

```
[5]: i = 7
for i in range(3): # sin, you lose i variable
 print(i)

0
1
2
```

```
[6]: def f(i):
 for i in range(3): # sin again, you lose i parameter
 print(i)
```

<sup>285</sup> <https://en.softpython.org/commandments.html#VII-COMMANDMENT>

<sup>286</sup> <https://en.softpython.org/commandments.html#VII-COMMANDMENT>

```
[7]: for i in range(2):
 for i in range(3): # debugging hell, you lose i from outer for
 print(i)

0
1
2
0
1
2
```

⊕⊕⊕ **EXERCISE:** If you read *all* the above, start implementing the function `is_utriang`, which RETURN True if the provided nxn matrix is upper triangular, that is, has all the entries below the diagonal set to zero. Return False otherwise.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[8]: def is_utriang(mat):

 n = len(mat)
 m = len(mat[0])

 for i in range(1,n):
 for j in range(i): # notice it arrives until i *excluded*, that is, arrives
 ↪to i - 1 *included*
 if mat[i][j] != 0:
 return False
 return True

assert is_utriang([[1]]) == True
assert is_utriang([[3,2,5],
 [0,6,2],
 [0,0,4]]) == True

assert is_utriang([[3,2,5],
 [0,6,2],
 [1,0,4]]) == False

assert is_utriang([[3,2,5],
 [0,6,2],
 [1,1,4]]) == False

assert is_utriang([[3,2,5],
 [0,6,2],
 [0,1,4]]) == False

assert is_utriang([[3,2,5],
 [1,6,2],
 [1,0,4]]) == False
```

</div>

```
[8]: def is_utriang(mat):
 raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
assert is_utriang([[1]]) == True
assert is_utriang([[3,2,5],
 [0,6,2],
 [0,0,4]]) == True

assert is_utriang([[3,2,5],
 [0,6,2],
 [1,0,4]]) == False

assert is_utriang([[3,2,5],
 [0,6,2],
 [1,1,4]]) == False

assert is_utriang([[3,2,5],
 [0,6,2],
 [0,1,4]]) == False

assert is_utriang([[3,2,5],
 [1,6,2],
 [1,0,4]]) == False
```

### Exercise - stitch\_left\_mod

This time let's try to *modify mat1 in place*, by stitching mat2 *to the left* of mat1.

So this time **don't** put a return instruction.

You will need to perform list insertion, which can be tricky. There are many ways to do it in Python, one could be using the weird splice assignment insertion:

```
mylist[0:0] = list_to_insert
```

see here for more info: <https://stackoverflow.com/a/10623383>

⊕⊕⊕ **EXERCISE:** Implement `stitch_left_mod`, which given the matrices `mat1` and `mat2` as list of lists, with `mat1` of size  $n \times 1$  and `mat2` of size  $n \times r$ , **MODIFIES** `mat1` so that it becomes of size  $n \times (l + r)$ , by stitching second `mat2` to the left of `mat1`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[9]: def stitch_left_mod(mat1, mat2):

 for i in range(len(mat1)):
 mat1[i][0:0] = mat2[i]

m1 = [['a', 'b', 'c'],
 ['d', 'b', 'a']]
m2 = [['f', 'b'],
 ['g', 'h']]
res = [[f, 'b', 'a', 'b', 'c'],
 ['g', 'h', 'd', 'b', 'a']]
```

(continues on next page)

(continued from previous page)

```
stitch_left_mod(m1, m2)
assert m1 == res
```

&lt;/div&gt;

```
[9]: def stitch_left_mod(mat1,mat2):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a','b','c'],
 ['d','b','a']]
m2 = [['f','b'],
 ['g','h']]

res = [['f','b','a','b','c'],
 ['g','h','d','b','a']]

stitch_left_mod(m1, m2)
assert m1 == res
```

### Exercise - transpose\_1

Let's see how to transpose a matrix *in-place*. The transpose  $M^T$  of a matrix  $M$  is defined as

$$M^T[i][j] = M[j][i]$$

The definition is simple yet implementation might be tricky. If you're not careful, you could easily end up swapping the values twice and get the same original matrix. To prevent this, iterate only the upper triangular part of the matrix and remember `range` function can also have a start index:

```
[10]: list(range(3,7))
[10]: [3, 4, 5, 6]
```

Also, make sure you know how to swap just two values by solving first this very simple exercise - also check the result in Python Tutor

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[11]: x = 3
y = 7

write here code for swapping x and y (don't directly use the constants 3 and 7!)

k = x
x = y
y = k

#jupman.pytut()
```

&lt;/div&gt;

```
[11]: x = 3
y = 7
```

(continues on next page)

(continued from previous page)

```
write here code for swapping x and y (don't directly use the constants 3 and 7!)
```

Going back to the transpose, for now we will consider only an nxn matrix. To make sure we actually get an nxn matrix, we will validate the input as before.

---

#### IV COMMANDMENT<sup>287</sup> (adapted for matrices): You shall never ever reassign function parameters

---

```
def myfun(M):
 # M is a parameter, so you shall *not* do any of such evil:
 M = [
 [6661, 6662],
 [6663, 6664]
]

 # For the sole case of composite parameters like lists (or lists of lists ..)
 # you can write stuff like this IF AND ONLY IF the function specification
 # requires you to modify the parameter internal elements (i.e. transposing _in-
 ↪place_):
 M[0][1] = 6663
```

⊕⊕⊕ **EXERCISE** If you read *all* the above, you can now proceed implementing the `transpose_1` function, which MODIFIES the given nxn matrix `mat` by transposing it *in-place*.

- If the matrix is not nxn, raises a `ValueError`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[12]: def transpose_1(mat):

 if len(mat) != len(mat[0]):
 raise ValueError("Matrix should be nxn, found instead %s x %s" % (len(mat), len(mat[0])))
 for i in range(len(mat)):
 for j in range(i+1, len(mat[i])):
 el = mat[i][j]
 mat[i][j] = mat[j][i]
 mat[j][i] = el

 # let's try wrong matrix dimensions:
try:
 transpose_1([[3,5]])
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 pass
```

(continues on next page)

<sup>287</sup> <https://en.softpython.org/commandments.html#IV-COMMANDMENT>

(continued from previous page)

```
m1 = [['a']]

transpose_1(m1)
assert m1 == [['a']]

m2 = [['a', 'b'],
 ['c', 'd']]

transpose_1(m2)
assert m2 == [['a', 'c'],
 ['b', 'd']]
```

&lt;/div&gt;

```
[12]: def transpose_1(mat):
 raise Exception('TODO IMPLEMENT ME !')
```

```
let's try wrong matrix dimensions:
try:
 transpose_1([[3,5]])
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 pass

m1 = [['a']]

transpose_1(m1)
assert m1 == [['a']]

m2 = [['a', 'b'],
 ['c', 'd']]

transpose_1(m2)
assert m2 == [['a', 'c'],
 ['b', 'd']]
```

## Exercise - transpose\_2

⊕⊕ Now let's try to transpose a generic  $n \times m$  matrix. This time for simplicity we will return a whole new matrix.

RETURN a NEW  $m \times n$  matrix which is the transpose of the given  $n \times m$  matrix `mat` as list of lists.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[13]: def transpose_2(mat):

 n = len(mat)
 m = len(mat[0])
 ret = [[0]*n for i in range(m)]
 for i in range(n):
 for j in range(m):
 ret[j][i] = mat[i][j]
```

(continues on next page)

(continued from previous page)

```

 return ret

m1 = [['a']]

r1 = transpose_2(m1)

assert r1 == [['a']]
r1[0][0] = 'z'
assert m1[0][0] == 'a'

m2 = [['a','b','c'],
 ['d','e','f']]

assert transpose_2(m2) == [['a','d'],
 ['b','e'],
 ['c','f']]

```

</div>

```
[13]: def transpose_2(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a']]

r1 = transpose_2(m1)

assert r1 == [['a']]
r1[0][0] = 'z'
assert m1[0][0] == 'a'

m2 = [['a','b','c'],
 ['d','e','f']]

assert transpose_2(m2) == [['a','d'],
 ['b','e'],
 ['c','f']]
```

### Exercise - cirpillino

Given a string and an integer n, RETURN a NEW matrix as list of lists containing all the characters in string subdivided in rows of n elements each.

- if the string length is not exactly divisible by n, raises ValueError

Example:

```
>>> cirpillino('cirpillinozimpirelloulalimpo', 4)
[[['c', 'i', 'r', 'p'],
 ['i', 'l', 'l', 'i'],
 ['n', 'o', 'z', 'i'],
 ['m', 'p', 'i', 'r'],
 ['e', 'l', 'l', 'o'],
 ['u', 'l', 'a', 'l'],
 ['i', 'm', 'p', 'o']]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"; data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[14]:

```
def cirpillino(string, n):

 if len(string) % n != 0:
 raise ValueError('The string is not divisible by %s' % n)
 ret = []

 for i in range(len(string) // n):
 ret.append(list(string[i*n:(i+1)*n]))
 return ret

TEST
assert cirpillino('z', 1) == [['z']]

assert cirpillino('abc', 1) == [['a'],
 ['b'],
 ['c']]

assert cirpillino('abcdef', 2) == [[['a', 'b'],
 ['c', 'd'],
 ['e', 'f']]]

assert cirpillino('abcdef', 3) == [[['a', 'b', 'c'],
 ['d', 'e', 'f']]]

assert cirpillino('cirpillinozimpirelloulalimpo', 4) == [[['c', 'i', 'r', 'p'],
 ['i', 'l', 'l', 'i'],
 ['n', 'o', 'z', 'i'],
 ['m', 'p', 'i', 'r'],
 ['e', 'l', 'l', 'o'],
 ['u', 'l', 'a', 'l'],
 ['i', 'm', 'p', 'o']]]

try:
 cirpillino('abc', 5)
 raise Exception("I should have failed!")
except ValueError:
 pass
```

&lt;/div&gt;

[14]:

```
def cirpillino(string, n):
 raise Exception('TODO IMPLEMENT ME !')

TEST
assert cirpillino('z', 1) == [['z']]

assert cirpillino('abc', 1) == [['a'],
 ['b'],
 ['c']]

assert cirpillino('abcdef', 2) == [[['a', 'b'],
```

(continues on next page)

(continued from previous page)

```
['c', 'd'],
['e', 'f'])

assert cirpillino('abcdef', 3) == [['a', 'b', 'c'],
 ['d', 'e', 'f']]

assert cirpillino('cirpillinozimpirelloulalimpo', 4) == [['c', 'i', 'r', 'p'],
 ['i', 'l', 'l', 'i'],
 ['n', 'o', 'z', 'i'],
 ['m', 'p', 'i', 'r'],
 ['e', 'l', 'l', 'o'],
 ['u', 'l', 'a', 'l'],
 ['i', 'm', 'p', 'o']]

try:
 cirpillino('abc', 5)
 raise Exception("I should have failed!")
except ValueError:
 pass
```

### Exercise - flag

Given two integer numbers  $n$  and  $m$ , with  $m$  a multiple of 3, RETURN a matrix  $n \times m$  as a list of lists having cells filled with numbers from 0 to 2 divided in three vertical stripes.

- if  $m$  is not a multiple of 3, raises `ValueError`

Example:

```
>>> flag(5,12)
[[0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2]]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[15]: def flag(n,m):

 if m % 3 != 0:
 raise ValueError('The number of columns is not a multiple of 3: %s' % m)

 ret = []

 for i in range(n):
 row = []
 for j in range(m):
 num = j // (m // 3)
 row.append(num)
 ret.append(row)
 return ret
```

(continues on next page)

(continued from previous page)

```
TEST
assert flag(1,3) == [[0, 1, 2]]

assert flag(1,6) == [[0,0,1,1, 2,2]]

assert flag(4,6) == [[0, 0, 1, 1, 2, 2],
 [0, 0, 1, 1, 2, 2],
 [0, 0, 1, 1, 2, 2],
 [0, 0, 1, 1, 2, 2]]

assert flag(2,9) == [[0, 0, 1, 1, 1, 2, 2, 2],
 [0, 0, 1, 1, 1, 2, 2, 2]]

assert flag(5,12) == [[0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 1, 1, 1, 2, 2, 2, 2]]

try:
 flag(3,7)
 raise Exception("I should have failed!")
except ValueError:
 pass
```

&lt;/div&gt;

```
[15]: def flag(n,m):
 raise Exception('TODO IMPLEMENT ME !')

TEST
assert flag(1,3) == [[0, 1, 2]]

assert flag(1,6) == [[0,0,1,1, 2,2]]

assert flag(4,6) == [[0, 0, 1, 1, 2, 2],
 [0, 0, 1, 1, 2, 2],
 [0, 0, 1, 1, 2, 2],
 [0, 0, 1, 1, 2, 2]]

assert flag(2,9) == [[0, 0, 1, 1, 1, 2, 2, 2],
 [0, 0, 1, 1, 1, 2, 2, 2]]

assert flag(5,12) == [[0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 1, 1, 1, 2, 2, 2, 2],
 [0, 0, 0, 1, 1, 1, 2, 2, 2, 2]]

try:
 flag(3,7)
 raise Exception("I should have failed!")
except ValueError:
 pass
```

### Exercise - avoid\_diag

⊕⊕ Given a square matrix  $n \times n$  as a list of lists, RETURN a NEW list with the sum of all numbers of every row EXCEPT the diagonal

- if the matrix is not square, raise ValueError

Example:

```
>>> avoid_diag([[5,6,2],
 [4,7,9],
 [1,9,8]])
[8, 13, 10]
```

because

```
8 = 6 + 2
13 = 4 + 7
10 = 1 + 9
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[16]:
def avoid_diag(mat):

 if len(mat) != len(mat[0]):
 raise ValueError("Non square matrix: %s x %s" % (len(mat), len(mat[0])))
 ret = []
 i = 0
 for row in mat:
 ret.append(sum(row) - row[i])
 i += 1
 return ret

assert avoid_diag([[5]]) == [0]

m2 = [[5,7],
 [9,1]]
assert avoid_diag(m2) == [7,9]
assert m2 == [[5,7],
 [9,1]]

assert avoid_diag([[5,6,2],
 [4,7,9],
 [1,9,8]]) == [8, 13, 10]

try:
 avoid_diag([[2,3,5],
 [1,5,2]])
 raise Exception("I should have failed!")
except ValueError:
 pass
```

</div>

```
[16]: def avoid_diag(mat):
 raise Exception('TODO IMPLEMENT ME !')

assert avoid_diag([[5]]) == [0]

m2 = [[5, 7],
 [9, 1]]
assert avoid_diag(m2) == [7, 9]
assert m2 == [[5, 7],
 [9, 1]]

assert avoid_diag([[5, 6, 2],
 [4, 7, 9],
 [1, 9, 8]]) == [8, 13, 10]

try:
 avoid_diag([[2, 3, 5],
 [1, 5, 2]])
 raise Exception("I should have failed!")
except ValueError:
 pass
```

### Exercise - no\_diag

⊕⊕ Given a matrix  $n \times n$  as a list of lists, RETURN a NEW matrix  $n \times n-1$  having the same cells as the original one EXCEPT the cells in the diagonal.

- if the matrix is not squared, raises ValueError

Example:

```
>>> m = [[8, 5, 3, 4],
 [7, 2, 4, 1],
 [9, 8, 3, 5],
 [6, 0, 4, 7]]
>>> no_diag(m)
[[5, 3, 4],
 [7, 4, 1],
 [9, 8, 5],
 [6, 0, 4]]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[17]: def no_diag(mat):

 if len(mat) != len(mat[0]):
 raise ValueError("Non square matrix: %s x %s" % (len(mat), len(mat[0])))

 ret = []
 i = 0
 for row in mat:
 new_row = row[0:i] + row[i+1:]
 ret.append(new_row)
 i += 1
```

(continues on next page)

(continued from previous page)

```
 return ret

TEST
m1 = [[3, 4],
 [8, 7]]
assert no_diag(m1) == [[4],
 [8]]
assert m1 == [[3, 4], # verify the original was not changed
 [8, 7]]

m2 = [[9, 4, 3],
 [8, 5, 6],
 [0, 2, 7]]
assert no_diag(m2) == [[4, 3],
 [8, 6],
 [0, 2]]

m3 = [[8, 5, 3, 4],
 [7, 2, 4, 1],
 [9, 8, 3, 5],
 [6, 0, 4, 7]]
assert no_diag(m3) == [[5, 3, 4],
 [7, 4, 1],
 [9, 8, 5],
 [6, 0, 4]]

try:
 no_diag([[2, 3, 5],
 [1, 5, 2]])
 raise Exception("I should have failed!")
except ValueError:
 pass
```

&lt;/div&gt;

```
[17]: def no_diag(mat):
 raise Exception('TODO IMPLEMENT ME !')

TEST
m1 = [[3, 4],
 [8, 7]]
assert no_diag(m1) == [[4],
 [8]]
assert m1 == [[3, 4], # verify the original was not changed
 [8, 7]]

m2 = [[9, 4, 3],
 [8, 5, 6],
 [0, 2, 7]]
assert no_diag(m2) == [[4, 3],
 [8, 6],
 [0, 2]]

m3 = [[8, 5, 3, 4],
 [7, 2, 4, 1],
 [9, 8, 3, 5],
 [6, 0, 4, 7]]
assert no_diag(m3) == [[5, 3, 4],
```

(continues on next page)

(continued from previous page)

```
[7,4,1],
[9,8,5],
[6,0,4]]
try:
 no_diag([[2,3,5],
 [1,5,2]])
 raise Exception("I should have failed!")
except ValueError:
 pass
```

### Exercise - no\_anti\_diag

⊕⊕⊕ Given a ⊕⊕ Given a matrix  $n \times n$  as a list of lists, RETURN a NEW matrix  $n \times n-1$  having the same cells as the original one EXCEPT the cells in the ANTI-diagonal. For examples, see asserts.

- if the matrix is not squared, raises ValueError

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: def no_anti_diag(mat):

 if len(mat) != len(mat[0]):
 raise ValueError("Expected square matrix, found instead: %s x %s" % (len(mat), len(mat[0])))
 ret = []
 for i in range(len(mat)):
 k = len(mat) - i - 1
 nuova = mat[i][:k] + mat[i][k+1:]
 ret.append(nuova)
 return ret

m1 = [[3,4],
 [8,7]]
assert no_anti_diag(m1) == [[3],
 [7]]

assert m1 == [[3,4], # verify the original was not changed
 [8,7]]

m2 = [[9,4,3],
 [8,5,6],
 [0,2,7]]
assert no_anti_diag(m2) == [[9,4],
 [8,6],
 [2,7]]

m3 = [[8,5,3,4],
 [7,2,4,1],
 [9,8,3,5],
 [6,0,4,7]]
assert no_anti_diag(m3) == [[8,5,3],
 [7,2,1],
 [9,3,5],
 [0,4,7]]

try:
```

(continues on next page)

(continued from previous page)

```
no_anti_diag([[2,3,5],
 [1,5,2]])
 raise Exception("I should have failed!")
except ValueError:
 pass
```

&lt;/div&gt;

```
[18]: def no_anti_diag(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [[3,4],
 [8,7]]
assert no_anti_diag(m1) == [[3],
 [7]]

assert m1 == [[3,4], # verify the original was not changed
 [8,7]]

m2 = [[9,4,3],
 [8,5,6],
 [0,2,7]]
assert no_anti_diag(m2) == [[9,4],
 [8,6],
 [2,7]]

m3 = [[8,5,3,4],
 [7,2,4,1],
 [9,8,3,5],
 [6,0,4,7]]
assert no_anti_diag(m3) == [[8,5,3],
 [7,2,1],
 [9,3,5],
 [0,4,7]]

try:
 no_anti_diag([[2,3,5],
 [1,5,2]])
 raise Exception("I should have failed!")
except ValueError:
 pass
```

### Exercise - repcol

⊕⊕ Given a matrix `mat`  $n \times m$  and a `lst` of  $n$  elements, MODIFY `mat` by writing into each cell the corresponding value from `lst`

- if `lst` has a different length from  $n$ , raise `ValueError`
- **DO NOT** create new lists

Example:

```
>>> m = [
 ['z','a','p','p','a'],
 ['c','a','r','t','a'],
 ['p','a','l','l','a']
]
```

(continues on next page)

(continued from previous page)

```
>>> repcol(m, ['E', 'H', '?']) # returns nothing!
>>> m
[['E', 'E', 'E', 'E'],
 ['H', 'H', 'H', 'H'],
 ['?', '?', '?', '?']]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[19]: def repcol(mat, lst):

 for i in range(len(mat)):
 for j in range(len(mat[0])):
 mat[i][j] = lst[i]

m1 = [['a']]
v1 = ['Q']
repcol(m1,v1) # returns nothing
assert m1 == [['Q']]

m2 = [
 ['a', 'b'],
 ['c', 'd'],
 ['e', 'f'],
 ['g', 'h'],
]
saved = m2[0] # we save a pointer to the first original row
v2 = ['P', 'A', 'L', 'A']
repcol(m2,v2) # returns nothing
assert m2 == [[['P', 'P'],
 ['A', 'A'],
 ['L', 'L'],
 ['A', 'A']]]
assert id(saved) == id(m2[0]) # must not create new lists

m3 = [
 ['z', 'a', 'p', 'p', 'a'],
 ['c', 'a', 'r', 't', 'a'],
 ['p', 'a', 'l', 'l', 'a']
]
v3 = ['E', 'H', '?']
repcol(m3,v3) # returns nothing
assert m3 == [[['E', 'E', 'E', 'E'],
 ['H', 'H', 'H', 'H'],
 ['?', '?', '?', '?']]]
```

&lt;/div&gt;

```
[19]: def repcol(mat, lst):
 raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
m1 = [['a']]
v1 = ['Q']
repcol(m1,v1) # returns nothing
assert m1 == [['Q']]

m2 = [
 ['a','b'],
 ['c','d'],
 ['e','f'],
 ['g','h'],
]
saved = m2[0] # we save a pointer to the first original row
v2 = ['P','A','L','A']
repcol(m2,v2) # returns nothing
assert m2 == [['P', 'P'],
 ['A', 'A'],
 ['L', 'L'],
 ['A', 'A']]
assert id(saved) == id(m2[0]) # must not create new lists

m3 = [
 ['z','a','p','p','a'],
 ['c','a','r','t','a'],
 ['p','a','l','l','a']
]
v3 = ['E','H','?']
repcol(m3,v3) # returns nothing
assert m3 == [['E', 'E', 'E', 'E', 'E'],
 ['H', 'H', 'H', 'H', 'H'],
 ['?', '?', '?', '?', '?']]
```

## Exercise - matinc

⊕⊕ Given a matrix of integers RETURN True if all the rows are strictly increasing from left to right, otherwise return False

### Example 1:

```
>>> m = [[1,4,6,7,9],
 [0,1,2,4,8],
 [2,6,8,9,10]]
>>> matinc(m)
True
```

### Example 2:

```
>>> m = [[0,1,3,4],
 [4,6,9,10],
 [3,7,7,15]]
>>> matinc(m)
False
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[20]: def matinc(mat):

 for i in range(len(mat)):
 for j in range(1, len(mat[0])):
 if mat[i][j] <= mat[i][j-1]:
 return False
 return True

TEST
m1 = [[5]]
assert matinc(m1) == True

m2 = [[7],
 [4]]
assert matinc(m2) == True

m3 = [[2, 3],
 [3, 5]]
assert matinc(m3) == True

m4 = [[9, 4]]
assert matinc(m4) == False

m5 = [[5, 5]]
assert matinc(m5) == False

m6 = [[1, 4, 6, 7, 9],
 [0, 1, 2, 4, 8],
 [2, 6, 8, 9, 10]]
assert matinc(m6) == True

m7 = [[0, 1, 3, 4],
 [4, 6, 9, 10],
 [3, 7, 7, 15]]
assert matinc(m7) == False

m8 = [[1, 4, 8, 7, 9],
 [0, 1, 2, 4, 8]]
assert matinc(m8) == False
```

</div>

```
[20]: def matinc(mat):
 raise Exception('TODO IMPLEMENT ME !')

TEST
m1 = [[5]]
assert matinc(m1) == True

m2 = [[7],
 [4]]
assert matinc(m2) == True

m3 = [[2, 3],
```

(continues on next page)

(continued from previous page)

```
[3,5]
assert matinc(m3) == True

m4 = [[9,4]]
assert matinc(m4) == False

m5 = [[5,5]]
assert matinc(m5) == False

m6 = [[1,4,6,7,9],
 [0,1,2,4,8],
 [2,6,8,9,10]]
assert matinc(m6) == True

m7 = [[0,1,3,4],
 [4,6,9,10],
 [3,7,15]]
assert matinc(m7) == False

m8 = [[1,4,8,7,9],
 [0,1,2,4,8]]
assert matinc(m8) == False
```

### Exercise - flip

⊕ Takes a matrix as a list of lists containing zeros and ones, and RETURN a NEW matrix (as list of lists), built first inverting all the rows and then flipping all the rows.

Inverting a list means transform the 0 into 1 and 1 into 0 - i.e.

- [0,1,1] becomes [1,0,0]
- [0,0,1] becomes [1,1,0]

Flipping a list means flipping the elements order - i.e.

- [0,1,1] becomes [1,1,0]
- [0,0,1] becomes [1,0,0]

**Example:** By combining inversion and reversal, if we start from

```
[
 [1,1,0,0],
 [0,1,1,0],
 [0,0,1,0]
]
```

First we invert each element:

```
[
 [0,0,1,1],
 [1,0,0,1],
 [1,1,0,1]
]
```

Then we flip weach row:

```
[
 [1, 1, 0, 0],
 [1, 0, 0, 1],
 [1, 0, 1, 1]
]
```

```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[21]: def flip(mat):

 ret = []
 for row in mat:
 new_row = []
 for elem in row:
 new_row.append(1 - elem)

 new_row.reverse()
 ret.append(new_row)
 return ret

assert flip([]) == []
assert flip([[1]]) == [[0]]
assert flip([[1, 0]]) == [[1, 0]]

m1 = [[1, 0, 0],
 [1, 0, 1]]
r1 = [[1, 1, 0],
 [0, 1, 0]]
assert flip(m1) == r1

m2 = [[1, 1, 0, 0],
 [0, 1, 1, 0],
 [0, 0, 1, 0]]
r2 = [[1, 1, 0, 0],
 [1, 0, 0, 1],
 [1, 0, 1, 1]]

assert flip(m2) == r2

verify the original m was not changed!
assert m2 == [[1, 1, 0, 0],
 [0, 1, 1, 0],
 [0, 0, 1, 0]]
```

</div>

```
[21]: def flip(mat):
 raise Exception('TODO IMPLEMENT ME !!')
```

(continues on next page)

(continued from previous page)

```
assert flip([]) == []
assert flip([[1]]) == [[0]]
assert flip([[1, 0]]) == [[1, 0]]

m1 = [[1, 0, 0],
 [1, 0, 1]]
r1 = [[1, 1, 0],
 [0, 1, 0]]
assert flip(m1) == r1

m2 = [[1, 1, 0, 0],
 [0, 1, 1, 0],
 [0, 0, 1, 0]]

r2 = [[1, 1, 0, 0],
 [1, 0, 0, 1],
 [1, 0, 1, 1]]

assert flip(m2) == r2

verify the original m was not changed!
assert m2 == [[1, 1, 0, 0],
 [0, 1, 1, 0],
 [0, 0, 1, 0]]
```

### Exercise - wall

⊕⊕⊕ Given a list `repe` of repetitions and an  $n \times m$  matrix `mat` as list of lists, RETURN a **completely NEW** matrix by taking the rows of `mat` and replicating them the number of times reported in the corresponding cells of `repe`

- **DO NOT** create structures with pointers to input matrix (nor parts of it)!

Example:

```
>>> wall([3, 4, 1, 2], [['i', 'a', 'a'],
 ['q', 'r', 'f'],
 ['y', 'e', 'v'],
 ['e', 'g', 'h']])
[[['i', 'a', 'a'],
 ['i', 'a', 'a'],
 ['i', 'a', 'a'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['y', 'e', 'v'],
 ['e', 'g', 'h'],
 ['e', 'g', 'h']]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[22]: def wall(ripe, mat):
```

(continues on next page)

(continued from previous page)

```

res = []

i = 0
for i in range(len(mat)):
 row = mat[i]
 n = ripe[i]
 for i in range(n):
 res.append(row[:])
return res

m1 = [['a']]
assert wall([2], m1) == [['a'],
 ['a']]

m2 = [['a','b','c','d'],
 ['e','q','v','r']]
r2 = wall([3,2], m2)
assert r2 == [[['a','b','c','d'],
 ['a','b','c','d'],
 ['a','b','c','d']],
 [['e','q','v','r'],
 ['e','q','v','r']],
 [['e','q','v','r']]]
r2[0][0] = 'z'
assert m2 == [['a','b','c','d'], # we want a NEW matrix
 ['e','q','v','r']]

m3 = [['i','a','a'],
 ['q','r','f'],
 ['y','e','v'],
 ['e','g','h']]
r3 = wall([3,4,1,2], m3)
assert r3 == [[['i', 'a', 'a'],
 ['i', 'a', 'a'],
 ['i', 'a', 'a'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['y', 'e', 'v'],
 ['e', 'g', 'h'],
 ['e', 'g', 'h']]]

```

&lt;/div&gt;

```
[22]: def wall(ripe, mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [['a']]
assert wall([2], m1) == [['a'],
 ['a']]

m2 = [['a','b','c','d'],
 ['e','q','v','r']]
r2 = wall([3,2], m2)
assert r2 == [[['a','b','c','d'],
 ['a','b','c','d']],
 [['e','q','v','r'],
 ['e','q','v','r']]]
```

(continues on next page)

(continued from previous page)

```

 ['a', 'b', 'c', 'd'],
 ['e', 'q', 'v', 'r'],
 ['e', 'q', 'v', 'r']]
r2[0][0] = 'z'
assert m2 == [[['a', 'b', 'c', 'd'], # we want a NEW matrix
 ['e', 'q', 'v', 'r']]]

m3 = [[['i', 'a', 'a'],
 ['q', 'r', 'f'],
 ['y', 'e', 'v'],
 ['e', 'g', 'h']]]
r3 = wall([3, 4, 1, 2], m3)
assert r3 == [[['i', 'a', 'a'],
 ['i', 'a', 'a'],
 ['i', 'a', 'a'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['q', 'r', 'f'],
 ['y', 'e', 'v'],
 ['e', 'g', 'h'],
 ['e', 'g', 'h']]]

```

**Exercise - sortlast**

⊕⊕⊕ Given a matrix as a list of lists of integer numbers, MODIFY the matrix by sorting ONLY the numbers of last column

- All other cells must NOT change

**Example:**

```

>>> m = [[8, 5, 3, 2, 4],
 [7, 2, 4, 1, 1],
 [9, 8, 3, 3, 7],
 [6, 0, 4, 2, 5]]
>>> sortlast(m)
>>> m
[[8, 5, 3, 2, 1],
 [7, 2, 4, 1, 4],
 [9, 8, 3, 3, 5],
 [6, 0, 4, 2, 7]]

```

[Show solution](#)

[23]: **def** sortlast(mat):

```

ordinata = sorted([mat[i][-1] for i in range(len(mat))])
for i in range(len(mat)):
 mat[i][-1] = ordinata[i]

TEST

```

(continues on next page)

(continued from previous page)

```

m1 = [[3]]
sortlast(m1)
assert m1 == [[3]]

m2 = [[9,3,7],
 [8,5,4]]
sortlast(m2)
assert m2 == [[9,3,4],
 [8,5,7]]

m3 = [[8,5,9],
 [7,2,3],
 [9,8,7]]
sortlast(m3)
assert m3 == [[8,5,3],
 [7,2,7],
 [9,8,9]]

m4 = [[8,5,3,2,4],
 [7,2,4,1,1],
 [9,8,3,3,7],
 [6,0,4,2,5]]
sortlast(m4)
assert m4 == [[8, 5, 3, 2, 1],
 [7, 2, 4, 1, 4],
 [9, 8, 3, 3, 5],
 [6, 0, 4, 2, 7]]

assert sortlast([[3]]) == None

```

&lt;/div&gt;

```
[23]: def sortlast(mat):
 raise Exception('TODO IMPLEMENT ME !')

TEST
m1 = [[3]]
sortlast(m1)
assert m1 == [[3]]

m2 = [[9,3,7],
 [8,5,4]]
sortlast(m2)
assert m2 == [[9,3,4],
 [8,5,7]]

m3 = [[8,5,9],
 [7,2,3],
 [9,8,7]]
sortlast(m3)
assert m3 == [[8,5,3],
 [7,2,7],
 [9,8,9]]

m4 = [[8,5,3,2,4],
 [7,2,4,1,1],
 [9,8,3,3,7],
```

(continues on next page)

(continued from previous page)

```
[6, 0, 4, 2, 5]
sortlast(m4)
assert m4 == [[8, 5, 3, 2, 1],
 [7, 2, 4, 1, 4],
 [9, 8, 3, 3, 5],
 [6, 0, 4, 2, 7]]

assert sortlast([[3]]) == None
```

### Exercise - skyscraper

The profile of a city can be represented as a 2D matrix where the 1 represent the buildings. In the example below, the building height is 4 (second column from the right)

```
[[0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0],
 [0, 0, 1, 0, 1, 0],
 [0, 1, 1, 1, 1, 0],
 [1, 1, 1, 1, 1, 1]]
```

Write a function which takes the profile of a 2-D list of 0 and 1 and RETURN the height of the highest skyscraper, for other examples see asserts

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution </a><div class="jupman-sol jupman-sol-code" style="display:none">

[24]:

```
def skyscraper(mat):

 n,m = len(mat), len(mat[0])
 for i in range(n):
 for j in range(m):
 if mat[i][j] == 1:
 return n-i
 return 0

assert skyscraper([
 [0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0],
 [0, 0, 1, 0, 1, 0],
 [0, 1, 1, 1, 1, 0],
 [1, 1, 1, 1, 1, 1]]) == 4

assert skyscraper([
 [0, 0, 0, 0],
 [0, 1, 0, 0],
 [0, 1, 1, 0],
 [1, 1, 1, 1]]) == 3

assert skyscraper([
 [0, 1, 0, 0],
 [0, 1, 0, 0],
 [0, 1, 0, 0],
 [1, 1, 1, 1]]) == 4

assert skyscraper([
 [0, 0, 0, 0],
 [0, 0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
[1, 1, 1, 0],
[1, 1, 1, 1]]) == 2
```

&lt;/div&gt;

[24]:

```
def skyscraper(mat):
 raise Exception('TODO IMPLEMENT ME !')

assert skyscraper([
 [0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0],
 [0, 0, 1, 0, 1, 0],
 [0, 1, 1, 1, 1, 0],
 [1, 1, 1, 1, 1, 1]]) == 4

assert skyscraper([
 [0, 0, 0, 0],
 [0, 1, 0, 0],
 [0, 1, 1, 0],
 [1, 1, 1, 1]]) == 3

assert skyscraper([
 [0, 1, 0, 0],
 [0, 1, 0, 0],
 [0, 1, 1, 0],
 [1, 1, 1, 1]]) == 4

assert skyscraper([
 [0, 0, 0, 0],
 [0, 0, 0, 0],
 [1, 1, 1, 0],
 [1, 1, 1, 1]]) == 2
```

## Exercise - school lab

⊕⊕⊕ If you're a teacher that often see new students, you have this problem: if two students who are friends sit side by side they can start chatting way too much. To keep them quiet, you want to somehow randomize student displacement by following this algorithm:

1. first sort the students alphabetically
2. then sorted students progressively sit at the available chairs one by one, first filling the first row, then the second, till the end.

INPUT:

- students: a list of strings of length  $\leq n*m$
- chairs: an  $n*m$  matrix as list of lists filled with None values (empty chairs)

OUTPUT: MODIFIES BOTH students and chairs inputs, without returning anything

- If students are more than available chairs, raises ValueError

Now implement the algorithm.

Example:

```
ss = ['b', 'd', 'e', 'g', 'c', 'a', 'h', 'f']
mat = [
```

(continues on next page)

(continued from previous page)

```
[None, None, None],
[None, None, None],
[None, None, None],
[None, None, None]
]

lab(ss, mat)

after execution, mat should result changed to this:

assert mat == [
 ['a', 'b', 'c'],
 ['d', 'e', 'f'],
 ['g', 'h', None],
 [None, None, None],
]
after execution, input ss should now be ordered:

assert ss == ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'f']
```

For more examples, see tests

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[25]: def lab(students, chairs):

 n = len(chairs)
 m = len(chairs[0])

 if len(students) > n*m:
 raise ValueError("There are more students than chairs ! Students = %s, chairs = %sx%s" % (len(students), n, m))

 i = 0
 j = 0
 students.sort()
 for s in students:
 chairs[i][j] = s

 if j == m - 1:
 j = 0
 i += 1
 else:
 j += 1

try:
 lab(['a', 'b'], [[None]])
 raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
 "Test passed"

try:
 lab(['a', 'b', 'c'], [[None, None]])
```

(continues on next page)

(continued from previous page)

```

 raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
 "Test passed"

m0 = [[None]]

r0 = lab([], m0)
assert m0 == [[None]]
assert r0 == None # function is not meant to return anything (so returns None by
 # default)

m1 = [[None]]
r1 = lab(['a'], m1)

assert m1 == [['a']]
assert r1 == None # function is not meant to return anything (so returns None by
 # default)

m2 = [[None, None]]
lab(['a'], m2) # 1 student 2 chairs in one row

assert m2 == [['a', None]]

m3 = [[None],
 [None]]
lab(['a'], m3) # 1 student 2 chairs in one column
assert m3 == [['a'],
 [None]]

ss4 = ['b', 'a']
m4 = [[None, None]]
lab(ss4, m4) # 2 students 2 chairs in one row

assert m4 == [['a', 'b']]

assert ss4 == ['a', 'b'] # also modified input list as required by function text

m5 = [[None, None],
 [None, None]]
lab(['b', 'c', 'a'], m5) # 3 students 2x2 chairs

assert m5 == [['a', 'b'],
 ['c', None]]

m6 = [[None, None],
 [None, None]]
lab(['b', 'd', 'c', 'a'], m6) # 4 students 2x2 chairs

assert m6 == [['a', 'b'],
 ['c', 'd']]

m7 = [[None, None, None],
 [None, None, None]]
lab(['b', 'd', 'e', 'c', 'a'], m7) # 5 students 3x2 chairs

```

(continues on next page)

(continued from previous page)

```
assert m7 == [['a', 'b', 'c'],
 ['d', 'e', None]]

ss8 = ['b', 'd', 'e', 'g', 'c', 'a', 'h', 'f']
m8 = [[None, None, None],
 [None, None, None],
 [None, None, None],
 [None, None, None]]
lab(ss8, m8) # 8 students 3x4 chairs

assert m8 == [['a', 'b', 'c'],
 ['d', 'e', 'f'],
 ['g', 'h', None],
 [None, None, None]]

assert ss8 == ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

&lt;/div&gt;

```
[25]: def lab(students, chairs):
 raise Exception('TODO IMPLEMENT ME !')

try:
 lab(['a', 'b'], [[None]])
 raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
 "Test passed"

try:
 lab(['a', 'b', 'c'], [[None, None]])
 raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
 "Test passed"

m0 = [[None]]

r0 = lab([], m0)
assert m0 == [[None]]
assert r0 == None # function is not meant to return anything (so returns None by
 # default)

m1 = [[None]]
r1 = lab(['a'], m1)

assert m1 == [['a']]
assert r1 == None # function is not meant to return anything (so returns None by
 # default)

m2 = [[None, None]]
lab(['a'], m2) # 1 student 2 chairs in one row

assert m2 == [['a', None]]
```

(continues on next page)

(continued from previous page)

```

m3 = [[None],
 [None]]
lab(['a'], m3) # 1 student 2 chairs in one column
assert m3 == [['a'],
 [None]]

ss4 = ['b', 'a']
m4 = [[None, None]]
lab(ss4, m4) # 2 students 2 chairs in one row

assert m4 == [['a', 'b']]

assert ss4 == ['a', 'b'] # also modified input list as required by function text

m5 = [[None, None],
 [None, None]]
lab(['b', 'c', 'a'], m5) # 3 students 2x2 chairs

assert m5 == [['a', 'b'],
 ['c', None]]

m6 = [[None, None],
 [None, None]]
lab(['b', 'd', 'c', 'a'], m6) # 4 students 2x2 chairs

assert m6 == [['a', 'b'],
 ['c', 'd']]

m7 = [[None, None, None],
 [None, None, None]]
lab(['b', 'd', 'e', 'c', 'a'], m7) # 5 students 3x2 chairs

assert m7 == [['a', 'b', 'c'],
 ['d', 'e', None]]

ss8 = ['b', 'd', 'e', 'g', 'c', 'a', 'h', 'f']
m8 = [[None, None, None],
 [None, None, None],
 [None, None, None],
 [None, None, None]]
lab(ss8, m8) # 8 students 3x4 chairs

assert m8 == [['a', 'b', 'c'],
 ['d', 'e', 'f'],
 ['g', 'h', None],
 [None, None, None]]

assert ss8 == ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

```

### Exercise - dump

The multinational ToxiCorp wants to hire you for devising an automated truck driver which will deposit highly contaminated waste in the illegal dumps they own worldwide. You find it ethically questionable, but they pay well, so you accept.

A dump is modelled as a rectangular region of dimensions `nrow` and `ncol`, implemented as a list of lists matrix. Every cell  $i, j$  contains the tons of waste present, and can contain *at most* 7 tons of waste.

The dumpster truck will transport `q` tons of waste, and will try to fill the dump by depositing waste in the first row, filling each cell up to 7 tons. When the first row is filled, it will proceed to the second one *from the left*, then to the third one again *from the left* until there is no waste to dispose of.

Function `dump(m, q)` takes as input the dump mat and the number of tons `q` to dispose of, and RETURN a NEW list representing a plan with the sequence of tons to dispose.

- If waste to dispose exceeds dump capacity, raises `ValueError`.
- **DO NOT** modify the matrix

**Example:**

```
m = [
 [5, 4, 6],
 [4, 7, 1],
 [3, 2, 6],
 [3, 6, 2],
]

dump(m, 22)

[2, 3, 1, 3, 0, 6, 4, 3]
```

For first row we dispose of 2,3,1 tons in three cells, for second row we dispose of 3,0,6 tons in three cells, for third row we only dispose 4, 3 tons in two cells as limit `q=22` is reached.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"< data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[26]: def dump(mat, q):

 rem = q
 ret = []

 for row in mat:
 for j in range(len(row)):
 cellfill = 7 - row[j]
 unload = min(cellfill, rem)
 rem -= unload

 if rem > 0:
 ret.append(unload)
 else:
 if unload > 0:
 ret.append(unload)
 return ret

 if rem > 0:
 raise ValueError("Couldn't fill the dump, %s tons remain!")
```

(continues on next page)

(continued from previous page)

```

m1 = [[5]]

assert dump(m1,0) == [] # nothing to dump

m2 = [[4]]

assert dump(m2,2) == [2]

m3 = [[5,4]]

assert dump(m3,3) == [2, 1]

m3 = [[5,7,3]]

assert dump(m3,3) == [2, 0, 1]

m5 = [[2,5], # 5 2
 [4,3]] # 3 1

assert dump(m5,11) == [5,2,3,1]

 # tons to dump in each cell
m6 = [[5,4,6], # 2 3 1
 [4,7,1], # 3 0 6
 [3,2,6], # 4 3 0
 [3,6,2]] # 0 0 0

assert dump(m6, 22) == [2,3,1,3,0,6,4,3]

try:
 dump ([[5]], 10)
 raise Exception("Should have failed !")
except ValueError:
 pass

```

&lt;/div&gt;

```
[26]: def dump(mat, q):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [[5]]

assert dump(m1,0) == [] # nothing to dump

m2 = [[4]]

assert dump(m2,2) == [2]

m3 = [[5,4]]

assert dump(m3,3) == [2, 1]
```

(continues on next page)

(continued from previous page)

```
m3 = [[5, 7, 3]]

assert dump(m3, 3) == [2, 0, 1]

m5 = [[2, 5], # 5 2
 [4, 3]] # 3 1

assert dump(m5, 11) == [5, 2, 3, 1]

 # tons to dump in each cell
m6 = [[5, 4, 6], # 2 3 1
 [4, 7, 1], # 3 0 6
 [3, 2, 6], # 4 3 0
 [3, 6, 2]] # 0 0 0

assert dump(m6, 22) == [2, 3, 1, 3, 0, 6, 4, 3]

try:
 dump ([[5]], 10)
 raise Exception("Should have failed !")
except ValueError:
 pass
```

## Esercizio - toepliz

⊕⊕⊕ RETURN True if the matrix as list of lists is Toeplitz, otherwise RETURN False.

- A matrix is Toeplitz if and only if every diagonal contains all the same elements
- We assume the matrix always contains at least a row of at least an element

**HINT:** use a couple of `for`, in the first one iterate by rows, in the second one by columns.

Ask yourself:

- from which row should we start scanning? Is the first one useful?
- from which column should we start scanning? Is the first one useful?
- if we scan the rows from the first one toward the last one and we are examining a certain number at a certain row, which condition should that number meet for the matrix to be Toeplitz?

Examples:

```
>>> m1 = [
 [1, 2, 3, 4],
 [5, 1, 2, 3],
 [9, 5, 1, 2]
]

>>> toepliz(m1)
True
```

On every diagonal there are the same numbers so it returns True

```
>>> m2 = [
 [1, 2, 3, 4],
 [4, 5, 6, 7]
```

(continues on next page)

(continued from previous page)

```
[5, 1, 4, 3],
[9, 3, 1, 2]
]

>>> toepliz(m2)
False
```

There is at least one diagonal with different numbers, in this case we have the diagonals 5,3 and 2,4,2 so it returns False

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[27]: def toepliz(matrix):

 for i in range(1, len(matrix)):
 for j in range(1, len(matrix[0])):
 if matrix[i][j] != matrix[i-1][j-1]:
 return False
 return True

TEST START - DO NOT TOUCH !
assert toepliz([[1]]) == True
assert toepliz([[3, 7],
 [5, 3]]) == True
assert toepliz([[3, 7],
 [3, 5]]) == False
assert toepliz([[3, 7],
 [3, 5]]) == False
assert toepliz([[3, 7, 9],
 [5, 3, 7]]) == True
assert toepliz([[3, 7, 9],
 [5, 3, 8]]) == False

assert toepliz([[1, 2, 3, 4],
 [5, 1, 2, 3],
 [9, 5, 1, 2]]) == True

assert toepliz([[1, 2, 3, 4],
 [5, 9, 2, 3],
 [9, 5, 1, 2]]) == False
TEST END
```

</div>

```
[27]: def toepliz(matrix):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH !
assert toepliz([[1]]) == True
assert toepliz([[3, 7],
 [5, 3]]) == True
assert toepliz([[3, 7],
 [3, 5]]) == False
assert toepliz([[3, 7],
 [3, 5]]) == False
assert toepliz([[3, 7, 9],
```

(continues on next page)

(continued from previous page)

```
[5,3,7]]) == True
assert toepliz([[3,7,9],
 [5,3,8]]) == False

assert toepliz([[1,2,3,4],
 [5,1,2,3],
 [9,5,1,2]]) == True

assert toepliz([[1,2,3,4],
 [5,9,2,3],
 [9,5,1,2]]) == False
TEST END
```

### Exercise - matrix multiplication

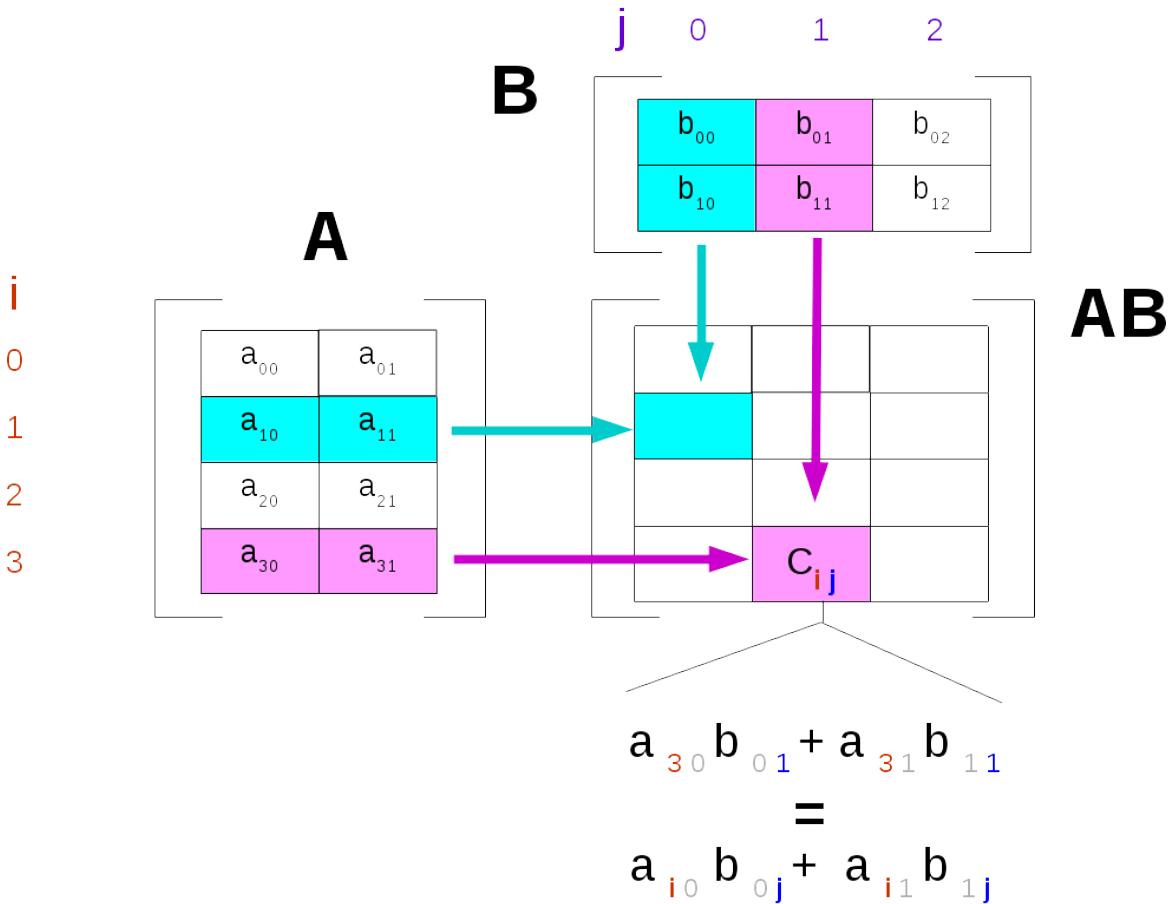
⊕⊕⊕ Have a look at [matrix multiplication definition<sup>288</sup>](#) on Wikipedia and try to implement it in the following function.

Basically, given  $n \times m$  matrix A and  $m \times p$  matrix B you need to output an  $n \times p$  matrix C calculating the entries  $c_{ij}$  with the formula

$$c_{ij} = a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^m a_{ik}b_{kj}$$

You need to fill all the  $n \times p$  cells of C, so sure enough to fill a rectangle you need two `for`s. Do you also need another `for`? Help yourself with the following visualization.

<sup>288</sup> [https://en.wikipedia.org/w/index.php?title=Matrix\\_multiplication&section=2#Definition](https://en.wikipedia.org/w/index.php?title=Matrix_multiplication&section=2#Definition)



Given matrices  $n \times m$  `matA` and  $m \times p$  `matB`, RETURN a NEW  $n \times p$  matrix which is the result of the multiplication of `matA` by `matB`.

- If `matA` has column number different from `matB` row number, raises a `ValueError`.

[Show solution](#)</a><div class="jupman-sol" data-jupman-sol-code" style="display:none">

```
[28]: def mul (matA, matB) :

 n = len (matA)
 m = len (matA[0])
 p = len (matB[0])
 if m != len (matB):
 raise ValueError ("mat1 column number %s must be equal to mat2 row number %s !
→" % (m, len (matB)))
 ret = [[0]*p for i in range (n)]
 for i in range (n):
 for j in range (p):
 ret[i][j] = 0
 for k in range (m):
 ret[i][j] += matA[i][k] * matB[k][j]
 return ret

TEST START - DO NOT TOUCH!
```

(continues on next page)

(continued from previous page)

```
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
→`AssertionError`

let's try wrong matrix dimensions:
try:
 mul([[3,5]], [[7]])
 raise Exception("SHOULD HAVE FAILED!")
except ValueError:
 "passed test"

ma1 = [[3]]
mb1 = [[5]]
r1 = mul(ma1,mb1)
assert r1 == [[15]]

ma2 = [[3],
 [5]]

mb2 = [[2,6]]

r2 = mul(ma2,mb2)

assert r2 == [[3*2, 3*6],
 [5*2, 5*6]]

ma3 = [[3,5]]

mb3 = [[2],
 [6]]

r3 = mul(ma3,mb3)

assert r3 == [[3*2 + 5*6]]

ma4 = [[3,5],
 [7,1],
 [9,4]]

mb4 = [[4,1,5,7],
 [8,5,2,7]]
r4 = mul(ma4,mb4)

assert r4 == [[52, 28, 25, 56],
 [36, 12, 37, 56],
 [68, 29, 53, 91]]
```

&lt;/div&gt;

```
[28]: def mul(mata, matb):
 raise Exception('TODO IMPLEMENT ME !')

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
→`AssertionError`

let's try wrong matrix dimensions:
try:
```

(continues on next page)

(continued from previous page)

```

mul([[3,5]], [[7]])
 raise Exception("SHOULD HAVE FAILED!")
except ValueError:
 "passed test"

ma1 = [[3]]
mb1 = [[5]]
r1 = mul(ma1,mb1)
assert r1 == [[15]]

ma2 = [[3],
 [5]]

mb2 = [[2,6]]

r2 = mul(ma2,mb2)

assert r2 == [[3*2, 3*6],
 [5*2, 5*6]]

ma3 = [[3,5]]

mb3 = [[2],
 [6]]

r3 = mul(ma3,mb3)

assert r3 == [[3*2 + 5*6]]

ma4 = [[3,5],
 [7,1],
 [9,4]]

mb4 = [[4,1,5,7],
 [8,5,2,7]]
r4 = mul(ma4,mb4)

assert r4 == [[52, 28, 25, 56],
 [36, 12, 37, 56],
 [68, 29, 53, 91]]

```

### Exercise - check\_nqueen

⊕⊕⊕⊕ A hard problem for the pros out there.

You have an  $n \times n$  matrix of booleans representing a chessboard where `True` means there is a queen in a cell, and `False` there is nothing.

For the sake of visualization, we can represent a configurations using `o` to mean `False` and letters like '`A`' and '`B`' are queens. Contrary to what we've done so far, for later convenience we show the matrix with the `j` going from bottom to top.

Let's see an example. In this case `A` and `B` can not attack each other, so the algorithm would return `True`:

7	.....B.
6	.....

(continues on next page)

(continued from previous page)

```

5
4
3 A...
2
1
0
i
j 01234567

```

Let's see why by evidencing A attack lines ..

```

7 \....|..B.
6 .\..|.../
5 ..\..|../.
4 ...|\|...
3 ----A---
2 .../|\...\.
1 .../..|\..\.
0 ./...|...\
i
j 01234567

```

... and B attack lines:

```

7 -----B-
6 /|\
5 /..|.
4 .../..|..
3 .../.A.|..
2 ./....|..
1 /.....|..
0 |..
i
j 01234567

```

In this other case the algorithm would return False as A and B can attack each other:

```

7 \./..|...
6 -B--|--/
5 /|\..|../.
4 .|..|/...
3 ----A---
2 .|..|/...\.
1 .|/..|\..\.
0 ./...|...\
i
j 01234567

```

In your algorithm, first you need to scan for queens. When you find one (and for each one of them !), you need to check if it can hit some other queen. Let's see how:

In this 7x7 table we have only one queen A, with at position  $i=1$  and  $j=4$

```

6 |..
5 \....|..
4 .\..|..
3 ..\..|../

```

(continues on next page)

(continued from previous page)

```

2 ... \ | / .
1 ----A--
0 ... / | \ .
i
j 0123456

```

To completely understand the range of the queen and how to calculate the diagonals, it is convenient to visually extend the table like so to have the diagonals hit the vertical axis. Notice we also added letters y and x

**NOTE:** in the algorithm you **do not** need to extend the matrix !

```

y
6|.....
5 \....|..../
4 .\...|.../..
3 ..\..|..../..
2 ... \ | /.....
1 ----A-----
0 ... / | \.....
-1 ... / | ..\...
-2 ... / | ... \..
-3 / ... | \
i
j 01234567 x

```

We see that the top-left to bottom-right diagonal hits the vertical axis at  $y = 5$  and the bottom-left to top-right diagonal hits the axis at  $y = -3$ . You should use this info to calculate the line equations.

Now you should have all the necessary hints to proceed with the implementation.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[29]:

```

def check_nqueen(mat):
 """ Takes an nxn matrix of booleans representing a chessboard where True means
 ↪there is a queen in a cell,
 ↪and False there is nothing. RETURN True if no queen can attack any other one,
 ↪False otherwise

 """

 # bottom-left to top-right line equation
 # y = x - 3
 # -3 = -j + i
 # y = x - j + i

 # top-left to bottom-right line equation
 # y = x + 5
 # 5 = j + i
 # y = x + j + i

 n = len(mat)
 for i in range(n):

```

(continues on next page)

(continued from previous page)

```

for j in range(n):
 if mat[i][j]: # queen is found at i,j
 for y in range(n): # vertical scan
 if y != i and mat[y][j]:
 return False
 for x in range(n): # horizontal scan
 if x != j and mat[i][x]:
 return False
 for x in range(n):
 y = x + j + i # top-left to bottom-right
 if y >= 0 and y < n and y != i and x != j and mat[y][x]:
 return False
 y = x - j + i # bottom-left to top-right
 if y >= 0 and y < n and y != i and x != j and mat[y][x]:
 return False

 return True

assert check_nqueen([[True]])
assert check_nqueen([[True, True],
 [False, False]]) == False

assert check_nqueen([[True, False],
 [False, True]]) == False

assert check_nqueen([[True, False],
 [True, False]]) == False

assert check_nqueen([[True, False, False],
 [False, False, True],
 [False, False, False]]) == True

assert check_nqueen([[True, False, False],
 [False, False, False],
 [False, False, True]]) == False

assert check_nqueen([[False, True, False],
 [False, False, False],
 [False, False, True]]) == True

assert check_nqueen([[False, True, False],
 [False, True, False],
 [False, False, True]]) == False

```

&lt;/div&gt;

[29]:

```

def check_nqueen(mat):
 """ Takes an nxn matrix of booleans representing a chessboard where True means
 ↪there is a queen in a cell,
 and False there is nothing. RETURN True if no queen can attack any other one,
 ↪False otherwise

 """
 raise Exception('TODO IMPLEMENT ME !')

```

(continues on next page)

(continued from previous page)

```

assert check_nqueen([[True]])
assert check_nqueen([[True, True],
 [False, False]]) == False

assert check_nqueen([[True, False],
 [False, True]]) == False

assert check_nqueen([[True, False],
 [True, False]]) == False

assert check_nqueen([[True, False, False],
 [False, False, True],
 [False, False, False]]) == True

assert check_nqueen([[True, False, False],
 [False, False, False],
 [False, False, True]]) == False

assert check_nqueen([[False, True, False],
 [False, False, False],
 [False, False, True]]) == True

assert check_nqueen([[False, True, False],
 [False, True, False],
 [False, False, True]]) == False

```

## Continue

Go on with [matrix challenges](#)<sup>289</sup>

### 7.2.3 Matrices lists 3 - Challenges

#### Download exercises zip

Browse online files<sup>290</sup>

We now propose some exercises without solution, do you accept the challenge?

#### Challenge - Tiles

⊕⊕ You are working in a construction site, and they tell you to fill the floor with tiles. You have a `stack` of tiles to place. On each, there is written at which coordinates it should be placed, and also the decoration it should show. The architect did not tell you exactly the dimensions the tiled floor should have, so you will have to deduce them by looking at the tiles coordinates.

Each time you take one tile from the `stack`, you place it in the appropriate cell on the floor.

Write some code which creates a NEW matrix of lists of lists called `mat`: before placing the tiles you will have to prepare a matrix with the appropriate number of cells to cover (retrieve dimensions by scanning the stack)

<sup>289</sup> <https://en.softpython.org/matrices-lists/matrices-lists3-chal.html#>

<sup>290</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/matrices-lists>

- tiles which remain empty will be marked with '\*' symbol
  - **MODIFY** the given stack, by reducing it one item at a time (suppose the top of the stack is the end of list): at the end of your program it must be empty
  - **DO NOT** create a new stack (so no lines beginning with `stack =`)
  - **DO NOT** use `.remove` nor `.index` methods

Example - given:

```
stack = [(1, 1, 'U'), (1, 3, 'U'), (2, 2, 'H'), (3, 0, 'A'), (3, 1, 'A'),
 (2, 1, '|'), (3, 3, 'A'), (2, 3, '|'), (3, 4, 'A')]
```

Your code should produce:

```
>>> pprint(mat)
[[['*', '*', '*', '*', '*'],
 ['*', 'U', '*', 'U', '*'],
 ['*', '|', 'H', '|', '*'],
 ['A', 'A', '*', 'A', 'A']]
```

```
[1]: from pprint import pprint

stack = [(1, 1, 'U'), (1, 3, 'U'), (2, 2, 'H'), (3, 0, 'A'), (3, 1, 'A'),
 (2, 1, '|'), (3, 3, 'A'), (2, 3, '|'), (3, 4, 'A')]

write some code here

pprint(mat)

assert stack == []
assert mat == [['*', '*', '*', '*', '*'],
 ['*', 'U', '*', 'U', '*'],
 ['*', '|', 'H', '|', '*'],
 ['A', 'A', '*', 'A', 'A']]
```

## Challenge - The Ark

It's year 2050, humans went far too often shopping with their SUVs, and now Earth is without natural resources with sun rays scorching what remains of the atmosphere.

Luckily, a planet which could host life called Aurora has been recently discovered. A billionaire enterpreneur who foresaw the disaster decided to build in secret a spaceship to escape: it was named The Ark. The billionaire is now old and wouldn't survive a long space travel, so he decides to send only animals to Aurora. Humans won't be allowed, as they would probably mess up that planet as they did with Earth. Sophisticated roboservants will then take care of the animals.

Animals are herded in a hurry to a field nearby the secret launch station.

## The Ark 1. Sort the herd

⊗⊗ The herd is a list of animals, in no particular order. The herd needs to stay in a farm while last checks are performed on the spaceship. Before entering the farm, animals are sorted by their weight, which are provided as a dictionary `herd_weights`.

Implement function `sort_herd(animals, weights)` which MODIFIES `animals` so that is sorted according to weight.

Example - given:

```
herd = ['elephant', 'owl', 'elephant', 'lion', 'owl', 'zebra', 'elephant', 'giraffe',
 'giraffe', 'fox', 'zebra', 'fox', 'fox', 'owl', 'zebra', 'owl',
 'lion', 'zebra', 'fox', 'lion', 'lion', 'owl', 'zebra', 'giraffe',
 'owl', 'owl', 'owl', 'owl', 'zebra', 'fox', 'owl',
 'fox', 'lion', 'fox', 'owl', 'owl', 'lion', 'fox', 'fox']
```

```
herd_weights = {'elephant': 4800,
 'fox' : 4,
 'owl' : 3,
 'giraffe' : 800,
 'lion' : 175,
 'zebra' : 384}
```

it should result (NOTE it's one list of strings, here we display it on many lines just for convenience):

```
>>> sort_herd(herd, herd_weights)
>>> print(herd)
['owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl',
 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox',
 'lion', 'lion', 'lion', 'lion', 'lion', 'lion', 'lion',
 'zebra', 'zebra', 'zebra', 'zebra', 'zebra', 'zebra', 'zebra',
 'giraffe', 'giraffe', 'giraffe',
 'elephant', 'elephant', 'elephant']
```

[2]:

```
herd = ['elephant', 'owl', 'elephant', 'lion', 'owl', 'zebra', 'elephant', 'giraffe',
 'giraffe', 'fox', 'zebra', 'fox', 'fox', 'owl', 'zebra', 'owl',
 'lion', 'zebra', 'fox', 'lion', 'lion', 'owl', 'zebra', 'giraffe',
 'owl', 'owl', 'owl', 'owl', 'zebra', 'fox', 'owl',
 'fox', 'lion', 'fox', 'owl', 'owl', 'lion', 'fox', 'fox']
```

```
herd_weights = {'elephant': 4800,
 'fox' : 4,
 'owl' : 3,
 'giraffe' : 800,
 'lion' : 175,
 'zebra' : 384}
```

```
sort them according to the weight
```

```
def sort_herd(animals, weights):
 """ MODIFIES animals so that is sorted in-place according to weight
 """
 raise Exception('TODO IMPLEMENT ME !')
```

```
sort_herd(herd, herd_weights)
```

(continues on next page)

(continued from previous page)

```
assert herd == ['owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl',
 ↪'owl', 'owl',
 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox',
 'lion', 'lion', 'lion', 'lion', 'lion', 'lion', 'lion',
 'zebra', 'zebra', 'zebra', 'zebra', 'zebra', 'zebra',
 'giraffe', 'giraffe', 'giraffe',
 'elephant', 'elephant', 'elephant']

assert sort_herd(herd, herd_weights) == None # should return nothing!
```

## The Ark 2. Enter the Farm

⊕⊕⊕ After getting sorted, animals are ready to enter the farm. A farm is organized in a series of cattle pens: we can model them as a list of lists of different lengths, each holding a different kind of animals. Implement a function `enter_farm(animals)` which RETURN a NEW list of lists.

- NOTE: lists may have different lengths, so we cannot call this a matrix !

Example - given:

```
sorted_herd = ['owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl',
 ↪'owl', 'owl',
 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox',
 'lion', 'lion', 'lion', 'lion', 'lion', 'lion', 'lion',
 'zebra', 'zebra', 'zebra', 'zebra', 'zebra', 'zebra',
 'giraffe', 'giraffe', 'giraffe',
 'elephant', 'elephant', 'elephant']
```

it should result:

```
>> from pprint import pprint
>> farm = enter_farm(sorted_herd)
>> pprint(farm, width=190)

[['owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl'],
 ['fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox'],
 ['lion', 'lion', 'lion', 'lion', 'lion'],
 ['zebra', 'zebra', 'zebra', 'zebra'],
 ['giraffe', 'giraffe'],
 ['elephant', 'elephant']]
```

[3]:

```
def enter_farm(animals):
 """ RETURN a NEW list of lists
 """
 raise Exception('TODO IMPLEMENT ME !')

sorted_herd = ['owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl',
 ↪'owl', 'owl',
 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox',
 'lion', 'lion', 'lion', 'lion', 'lion', 'lion',
 'zebra', 'zebra', 'zebra', 'zebra', 'zebra', 'zebra',
 'giraffe', 'giraffe', 'giraffe',
 'elephant', 'elephant', 'elephant']
```

(continues on next page)

(continued from previous page)

```

assert enter_farm(sorted_herd) == [
 ['owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl'],
 ['fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox'],
 ['lion', 'lion', 'lion', 'lion', 'lion', 'lion', 'lion'],
 ['zebra', 'zebra', 'zebra', 'zebra', 'zebra', 'zebra'],
 ['giraffe', 'giraffe', 'giraffe'],
 ['elephant', 'elephant', 'elephant']
]

assert enter_farm([]) == []
assert enter_farm(['beetle']) == [['beetle']]
assert enter_farm(['sheep', 'sheep']) == [['sheep', 'sheep']]
sorted_herd2 = ['sheep', 'dog']
assert enter_farm(sorted_herd2) == [['sheep'],
 ['dog']]
assert sorted_herd2 == ['sheep', 'dog'] # check it did not change the input

```

### The Ark 3. Enter the Ark

⊗⊗ An Ark spaceship is a huge rocket divided vertically in stages, each divided in exactly three compartments. Laws of physics suggest it's best to stack heavier elements at the bottom, so it is decided to populate the base of the rocket with big animals like elephants, while lighter ones like birds will go to the tip.

We can model The Ark as a list of lists matrix, where each row contains exactly three tuples of animals. Given as input a `farm` with animals already sorted, RETURN a NEW list of lists with the animals grouped in tuples.

- **ASSUME** animals in rows are always exactly divisible by 3
- **WRITE** something general, which handles also weird cases like a floor with mixed animal species

Example - given:

```

farm = [
 ['owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl'],
 ['fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox'],
 ['lion', 'lion', 'lion', 'lion', 'lion', 'lion', 'lion'],
 ['zebra', 'zebra', 'zebra', 'zebra', 'zebra', 'zebra'],
 ['giraffe', 'giraffe', 'giraffe'],
 ['elephant', 'elephant', 'elephant']
]

```

your code should produce (don't care about formatting, here we manually displayed tuples in columns for clarity):

```

>>> enter_ark(farm)
[
 [('owl', 'owl', 'owl', 'owl'), ('owl', 'owl', 'owl', 'owl'), ('owl', 'owl', 'owl', 'owl')],
 [('fox', 'fox', 'fox'), ('fox', 'fox', 'fox'), ('fox', 'fox', 'fox')],
 [('lion', 'lion'), ('lion', 'lion'), ('lion', 'lion')],
 [('zebra', 'zebra'), ('zebra', 'zebra'), ('zebra', 'zebra')],
 [('giraffe',), ('giraffe',), ('giraffe',)],
 [('elephant',), ('elephant',), ('elephant',)]
]

```

[4]:

```
def enter_ark(animals):
 """ RETURN a NEW list of lists
 """
 raise Exception('TODO IMPLEMENT ME !')

farm = [
 ['owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl', 'owl'],
 ['fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox', 'fox'],
 ['lion', 'lion', 'lion', 'lion', 'lion', 'lion'],
 ['zebra', 'zebra', 'zebra', 'zebra', 'zebra', 'zebra'],
 ['giraffe', 'giraffe', 'giraffe'],
 ['elephant', 'elephant', 'elephant']
]

assert enter_ark(farm) == [
 [('owl', 'owl', 'owl', 'owl'), ('owl', 'owl', 'owl', 'owl'), ('owl', 'owl', 'owl', 'owl')],
 [('fox', 'fox', 'fox'), ('fox', 'fox', 'fox'), ('fox', 'fox', 'fox')],
 [('lion', 'lion'), ('lion', 'lion'), ('lion', 'lion')],
 [('zebra', 'zebra'), ('zebra', 'zebra'), ('zebra', 'zebra')],
 [('giraffe',), ('giraffe',), ('giraffe',)],
 [('elephant',), ('elephant',), ('elephant',)]
]

assert enter_ark([]) == []
cat_farm = [['cat', 'cat', 'cat']]
assert enter_ark(cat_farm) == [[('cat',), ('cat',), ('cat',)]]
assert cat_farm == [['cat', 'cat', 'cat']] #check function didn't change the input
assert enter_ark([['cat', 'dog', 'mouse']]) == [[('cat',), ('dog',), ('mouse',)]] #← general case
```

### The Ark 4. Aurora

⊕⊕ Implement function land which takes a spaceship and PRINTS a message about its content.

Example:

```
ark = [
 [('owl', 'owl', 'owl', 'owl'), ('owl', 'owl', 'owl', 'owl'), ('owl', 'owl', 'owl',
 ↪ 'owl')],
 [('fox', 'fox', 'fox'), ('fox', 'fox', 'fox'), ('fox', 'fox', 'fox')],
 [('lion', 'lion'), ('lion', 'lion'), ('lion', 'lion')],
 [('zebra', 'zebra'), ('zebra', 'zebra'), ('zebra', 'zebra')],
 [('giraffe',), ('giraffe',), ('giraffe',)],
 [('elephant',), ('elephant',), ('elephant',)]
]

>>> land(ark)
```

The spaceship reached Aurora and has successfully landed:

```
12 owls
9 foxs
6 lions
6 zebras
```

(continues on next page)

(continued from previous page)

```
3 giraffes
3 elephants

A new chapter begins...
```

[5]:

```
def land(spaceship):
 """ PRINTS a success message
 """
 raise Exception('TODO IMPLEMENT ME !')

ark1 = [
 [('owl', 'owl', 'owl', 'owl'), ('owl', 'owl', 'owl', 'owl'), ('owl', 'owl', 'owl',
 ↵ 'owl')],
 [('fox', 'fox', 'fox'), ('fox', 'fox', 'fox'), ('fox', 'fox', 'fox')],
 [('lion', 'lion'), ('lion', 'lion'), ('lion', 'lion')],
 [('zebra', 'zebra'), ('zebra', 'zebra'), ('zebra', 'zebra')],
 [('giraffe',), ('giraffe',), ('giraffe',)],
 [('elephant',), ('elephant',), ('elephant',)]]
]

land(ark1) # should only print stuff

assert land(ark1) == None # should return nothing!
```

## Challenge - Go camping

⊕⊕⊕ Let's go camping in the wild! But first, you need to check you have everything needed. In particular, you are worried about not having all the items you wrote down in a blocknote. You open the bag with all the camping stuff, and spread the content around on your bed.

Given a matrix bed as a list of lists of strings, and a tuple blocknote of elements to find, write a function go\_camping which RETURN True if ALL the elements of blocknote are present in bed, otherwise RETURN False.

- **DO NOT** use search methods like index, count, remove, ... they're slow!
- **DO NOT** use in operator on lists

Example 1 - given:

```
blocknote = ('bottle','lighter','sunscreen')
bed = [['bottle','glasses','hat','hat'],
 ['hat','lighter','bottle','bottle'],
 ['hat','lighter','sunscreen','bottle']]
```

'bottle', 'lighter' and 'sunscreen' are all present, we expect True:

```
>>> print(go_camping(blocknote, bed))
True
```

Example 2 - given:

```
blocknote = ('bottle','lighter','sunscreen')
bed = [['book','glasses','hat','hat'],
```

(continues on next page)

(continued from previous page)

```
['hat', 'lighter', 'book', 'book'],
['hat', 'lighter', 'sunscreen', 'book']]
```

'lighter' and 'sunscreen' are present, but 'bottle' is not, we expect False:

```
>>> print(go_camping(blocknote, bed))
False
```

[6]:

```
def go_camping(notes, bed):
 raise Exception('TODO IMPLEMENT ME !')

expect True
print(go_camping(('bottle', 'lighter', 'sunscreen'),
 [['bottle', 'glasses', 'hat', 'hat'],
 ['hat', 'lighter', 'bottle', 'bottle'],
 ['hat', 'lighter', 'sunscreen', 'bottle']]))

expect False
print(go_camping(('bottle', 'lighter', 'sunscreen'),
 [['book', 'glasses', 'hat', 'hat'],
 ['hat', 'lighter', 'book', 'book'],
 ['hat', 'lighter', 'sunscreen', 'book']]))

assert go_camping([('a',), [[['a']]]) == True
assert go_camping([('b',), [[['a']]]) == False
assert go_camping([('a', 'b'), [[['a', 'a'],
 ['a', 'b']],]]) == True
assert go_camping([('a', 'c'), [[['a', 'a'],
 ['a', 'b']],]]) == False
assert go_camping([('a', 'c'), [[['a', 'a'],
 ['a', 'b']],]]) == False

assert go_camping([('a', 'c', 'd'), [[['a', 'e', 'b', 'b'],
 ['b', 'c', 'a', 'a'],
 ['b', 'c', 'd', 'a']]]) == True
assert go_camping([('a', 'c', 'd'), [[['f', 'e', 'b', 'b'],
 ['b', 'c', 'f', 'f'],
 ['b', 'c', 'd', 'f']]]) == False

m = [[['f', 'e', 'b', 'b'],
 ['b', 'c', 'f', 'f'],
 ['b', 'c', 'd', 'f']]]
go_camping([('a', 'c', 'd'), m])
shouldn't modify input
assert m == [[['f', 'e', 'b', 'b'],
 ['b', 'c', 'f', 'f'],
 ['b', 'c', 'd', 'f']]]
```

## 7.3 Mixed structures

### 7.3.1 Mixed structures 1

[Download exercises zip](#)

Naviga file online<sup>291</sup>

In this notebook we will see how to manage more complex data structures like lists of dictionaries and dictionaries of lists, examining also the meaning of shallow and deep copy.

#### WARNING

The following exercises contain tests with *asserts*. To understand how to carry them out, read first [Error handling and testing](#)<sup>292</sup>

#### Exercise - Luxury Holding

A luxury holding groups several companies and has a database of managers as a list of dictionaries. Each employee is represented by a dictionary:

```
{
 "name": "Alessandro",
 "surname": "Borgoloso",
 "age": 34,
 "company": {
 "name": "Candied Herrings",
 "sector": "Food"
 }
}
```

The dictionary has several simple attributes like name, surname, age. The attribute company is more complex, because it is represented as another dictionary:

```
"company": {
 "name": "Candied Herrings",
 "sector": "Food"
}
```

```
[1]: managers_db = [
 {
 "name": "Alessandro",
 "surname": "Borgoloso",
 "age": 34,
 "company": {
 "name": "Candied Herrings",
 "sector": "Food"
 }
 },
 {
 "name": "Matilda",
 "surname": "Borgoloso",
 "age": 30,
 "company": {
 "name": "Candied Herrings",
 "sector": "Food"
 }
 }
]
```

(continues on next page)

<sup>291</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/mixed-structures>

<sup>292</sup> <https://en.softpython.org/functions/fun2-errors-and-testing-sol.html>

(continued from previous page)

```

 "surname": "Delle Sòle",
 "age": 25,
 "company": {
 "name": "Pythonic Footwear",
 "sector": "Fashion"
 }
},
{
 "name": "Alfred",
 "surname": "Pennyworth",
 "age": 20,
 "company": {
 "name": "Batworks",
 "sector": "Fashion"
 }
},
{
 "name": "Arianna",
 "surname": "Schei",
 "age": 37,
 "company": {
 "name": "MegaDiamonds Unlimited",
 "sector": "Precious stones"
 }
},
{
 "name": "Antonione",
 "surname": "Cannavacci",
 "age": 25,
 "company": {
 "name": "Pre-chewed Chewing gums",
 "sector": "Food"
 }
},
]

```

**Exercise - extract\_managers**

⊕⊕ RETURN the manager names in a list

&lt;a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide"&gt;Show solution&lt;/a&gt;&lt;div class="jupman-sol jupman-sol-code" style="display:none"&gt;

[2] :

```

def extract_managers(db):

 ret = []
 for d in db:
 ret.append(d["name"])
 return ret

assert extract_managers([]) == []

if it doesn't find managers_db, remember to execute the cell above which defines it

```

(continues on next page)

(continued from previous page)

```
assert extract_managers(managers_db) == ['Alessandro', 'Matilda', 'Alfred', 'Arianna',
 ↵ 'Antonione']
```

&lt;/div&gt;

[2]:

```
def extract_managers(db):
 raise Exception('TODO IMPLEMENT ME !')

assert extract_managers([]) == []

if it doesn't find managers_db, remember to execute the cell above which defines it
↪ !
assert extract_managers(managers_db) == ['Alessandro', 'Matilda', 'Alfred', 'Arianna',
 ↵ 'Antonione']
```

### Exercise - extract\_companies

⊗⊗ RETURN the names of departments in a list.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[3]:

```
def extract_companies(db):

 ret = []
 for d in db:
 ret.append(d["company"]["name"])

 return ret

assert extract_companies([]) == []
if it doesn't find managers_db, remember to execute the cell above which defines it
↪ !
assert extract_companies(managers_db) == ["Candied Herrings", "Pythonic Footwear",
 ↵ "Batworks", "MegaDiamonds Unlimited", "Pre-chewed Chewing gums"]
```

&lt;/div&gt;

[3]:

```
def extract_companies(db):
 raise Exception('TODO IMPLEMENT ME !')

assert extract_companies([]) == []
if it doesn't find managers_db, remember to execute the cell above which defines it
↪ !
assert extract_companies(managers_db) == ["Candied Herrings", "Pythonic Footwear",
 ↵ "Batworks", "MegaDiamonds Unlimited", "Pre-chewed Chewing gums"]
```

### Exercise - avg\_age

⊕⊕ RETURN the average age of managers

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4] :

```
def avg_age(db):

 s = 0
 for d in db:
 s += d["age"]

 return s / len(db)

since the function returns a float we can't compare for exact numbers but
only for close numbers with the function math.isclose
import math
assert math.isclose(avg_age(managers_db), (34 + 25 + 20 + 37 + 25) / 5)
```

</div>

[4] :

```
def avg_age(db):
 raise Exception('TODO IMPLEMENT ME !')

since the function returns a float we can't compare for exact numbers but
only for close numbers with the function math.isclose
import math
assert math.isclose(avg_age(managers_db), (34 + 25 + 20 + 37 + 25) / 5)
```

### Exercise - sectors

⊕⊕ RETURN the company sectors in a list, WITHOUT duplicates and alphabetically sorted!!!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5] : def sectors(db) :

```
ret = []
for d in db:
 sector = d["company"]["sector"]
 if sector not in ret:
 ret.append(sector)

ret.sort()
return ret

assert sectors([]) == []
assert sectors(managers_db) == ["Fashion", "Food", "Precious stones"]
```

```
</div>
```

```
[5]: def sectors(db):
 raise Exception('TODO IMPLEMENT ME !')

assert sectors([]) == []
assert sectors(managers_db) == ["Fashion", "Food", "Precious stones"]
```

### Exercise - averages

⊕⊕ Given a dictionary structured as a tree regarding the grades of a student in class V and VI, RETURN an array containing the average for each subject

Example:

```
>>> averages([
 {'id' : 1, 'subject' : 'math', 'V' : 70, 'VI' : 82},
 {'id' : 1, 'subject' : 'italian', 'V' : 73, 'VI' : 74},
 {'id' : 1, 'subject' : 'german', 'V' : 75, 'VI' : 86}
])
```

returns

```
[76.0 , 73.5, 80.5]
```

which corresponds to

```
[(70+82)/2 , (73+74)/2, (75+86)/2]
```

```
[6]: def averages(lst):
 ret = [0.0, 0.0, 0.0]

 for i in range(len(lst)):
 ret[i] = (lst[i]['V'] + lst[i]['VI']) / 2

 return ret

TEST START - DO NOT TOUCH!
if you wrote the whole code correct, and execute the cell, Python shouldn't raise
`AssertionError`
import math

def is_list_close(lista, listb):
 """ Verifies the float numbers in lista are similar to numbers in listb

 """

 if len(lista) != len(listb):
 return False

 for i in range(len(lista)):
 if not math.isclose(lista[i], listb[i]):
 return False
```

(continues on next page)

(continued from previous page)

```

 return True

assert is_list_close(averages([
 {'id' : 1, 'subject' : 'math', 'V' : 70, 'VI' : 82},
 {'id' : 1, 'subject' : 'italian', 'V' : 73, 'VI' : 74},
 {'id' : 1, 'subject' : 'german', 'V' : 75, 'VI' : 86}
]),
[76.0 , 73.5, 80.5])
TEST END

```

### Exercise - has\_pref

⊕⊕ A big store has a database of clients modelled as a dictionary which associates customer names to their preferences regarding the categories of articles they usually buy:

```
{
 'aldo': ['cinema', 'music', 'sport'],
 'giovanni': ['music'],
 'giacomo': ['cinema', 'videogames']
}
```

Given the dictionary, the customer name and a category, write a function `has_pref` which RETURN `True` if that client has the given preference, `False` otherwise

Example:

```
has_pref({
 'aldo': ['cinema', 'music', 'sport'],
 'giovanni': ['music'],
 'giacomo': ['cinema', 'videogames']
}, 'aldo', 'music')
```

must return `True`, because aldo likes music, while instead:

```
has_pref({
 'aldo': ['cinema', 'music', 'sport'],
 'giovanni': ['music'],
 'giacomo': ['cinema', 'videogames']
}, 'giacomo', 'sport')
```

must return `False` because giacomo doesn't like sport.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[7]:

```

def has_pref(d, name, pref):

 if name in d:
 return pref in d[name]
 else:
 return False

assert has_pref({}, 'a', 'x') == False
assert has_pref({'a':[]}, 'a', 'x') == False

```

(continues on next page)

(continued from previous page)

```

assert has_pref({'a':['x']}, 'a', 'x') == True
assert has_pref({'a':['x']}, 'b', 'x') == False
assert has_pref({'a':['x','y']}, 'a', 'y') == True
assert has_pref({'a':['x','y']},
 'b':['y','x','z']}, 'b', 'y') == True
assert has_pref({'aldo': ['cinema', 'music', 'sport'],
 'giovanni': ['music'],
 'giacomo': ['cinema', 'videogames']
 }, 'aldo', 'music') == True
assert has_pref({'aldo': ['cinema', 'music', 'sport'],
 'giovanni': ['music'],
 'giacomo': ['cinema', 'videogames']
 }, 'giacomo', 'sport') == False

```

&lt;/div&gt;

[7]:

```

def has_pref(d, name, pref):
 raise Exception('TODO IMPLEMENT ME !')

assert has_pref({}, 'a', 'x') == False
assert has_pref({'a':[]}, 'a', 'x') == False
assert has_pref({'a':['x']}, 'a', 'x') == True
assert has_pref({'a':['x']}, 'b', 'x') == False
assert has_pref({'a':['x','y']}, 'a', 'y') == True
assert has_pref({'a':['x','y'],
 'b':['y','x','z']}, 'b', 'y') == True
assert has_pref({'aldo': ['cinema', 'music', 'sport'],
 'giovanni': ['music'],
 'giacomo': ['cinema', 'videogames']
 }, 'aldo', 'music') == True
assert has_pref({'aldo': ['cinema', 'music', 'sport'],
 'giovanni': ['music'],
 'giacomo': ['cinema', 'videogames']
 }, 'giacomo', 'sport') == False

```

### Exercise - festival

⊕⊕⊕ During a country festival in Italy, the local pastry shops decide to donate each a certain amount of pastries. Every shop is represented as a dictionary, which contains pastries names as keys, plus the special key `name` which represents the shop name itself (assume all the shops produce the same types of pastries)

```

shops = [{"babba": 3,
 "bignè": 4,
 "zippole": 2,
 "name": "Da Gigi"},
 {"babba": 5,
 "bignè": 3,
 "zippole": 9,
 "name": "La Delizia"},
 {"babba": 1,
 "bignè": 2,
 "zippole": 6,
 "name": "Gnam gnam"},
 {"babba": 7,

```

(continues on next page)

(continued from previous page)

```
'bignè': 8,
'zippole': 4,
'name': 'Il Dessert'}]
```

Given a list of such dictionaries and a list of pastries pastries, we want to produce as output a NEW list of lists structured like this:

```
>>> festival(pastries, ['bignè', 'zippole', 'babbà'])

[['Name', 'bignè', 'zippole', 'babbà'],
 ['Da Gigi', 4, 2, 3],
 ['La Delizia', 3, 9, 5],
 ['Gnam gnam', 2, 6, 1],
 ['Il Dessert', 8, 4, 7],
 ['Totals', 17, 21, 16]]
```

which has the totals of each pastry type.

Show solutionShow solution

>

[8]:

```
def festival(shops, pastries):

 ret = []
 ret.append(['Name']+ pastries[:]) # we make a copy of pastries to prevent
 ↪modification of the input
 sums = [0]*(len(pastries)+1)
 sums[0] = 'Totals'
 for p in shops:
 j = 1
 row = [p['name']]
 for pastry in pastries:
 row.append(pastry)
 sums[j] += p[pastry]
 j += 1
 ret.append(row)
 ret.append(sums)
 return ret

from pprint import pprint

pastries1 = ['cornetti']
res1 = festival([{name:'La Patisserie',
 'cornetti':2},
 {'cornetti':5,
 'name':'La Casa Del Cioccolato'},
 pastries1])
assert res1 == [['Name', 'cornetti'],
 ['La Patisserie', 2],
 ['La Casa Del Cioccolato', 5],
 ['Totals', 7]]
assert pastries1 == ['cornetti'] # verify the input didn't change

shops2 = [{babbà: 3,
 'bignè': 4,
```

(continues on next page)

(continued from previous page)

```

'zippole':2,
 'name': 'Da Gigi'},
{'babbà': 5,
 'bignè': 3,
 'zippole':9,
 'name': 'La Delizia'},
{'babbà': 1,
 'bignè': 2,
 'zippole':6,
 'name': 'Gnam gnam'},
{'babbà': 7,
 'bignè': 8,
 'zippole':4,
 'name': 'Il Dessert']}]

res2 = festival(shops2, ['bignè', 'zippole', 'babbà'])
#pprint(res2, width=43)

assert res2 == [['Name', 'bignè', 'zippole', 'babbà'],
 ['Da Gigi', 4, 2, 3],
 ['La Delizia', 3, 9, 5],
 ['Gnam gnam', 2, 6, 1],
 ['Il Dessert', 8, 4, 7],
 ['Totals', 17, 21, 16]]

```

&lt;/div&gt;

[8]:

```

def festival(shops, pastries):
 raise Exception('TODO IMPLEMENT ME !')

from pprint import pprint

pastries1 = ['cornetti']
res1 = festival([{'name':'La Patisserie',
 'cornetti':2},
 {'name':'La Casa Del Cioccolato'},
 pastries1])
assert res1 == [[['Name', 'cornetti'],
 ['La Patisserie', 2],
 ['La Casa Del Cioccolato', 5],
 ['Totals', 7]]]
assert pastries1 == ['cornetti'] # verify the input didn't change

shops2 = [{ 'babbà': 3,
 'bignè': 4,
 'zippole':2,
 'name': 'Da Gigi'},
 { 'babbà': 5,
 'bignè': 3,
 'zippole':9,
 'name': 'La Delizia'},
 { 'babbà': 1,
 'bignè': 2,
 'zippole':6,

```

(continues on next page)

(continued from previous page)

```

 'name': 'Gnam gnam'},
 {'babbà': 7,
 'bignè': 8,
 'zippole': 4,
 'name': 'Il Dessert'}]

res2 = festival(shops2, ['bignè', 'zippole', 'babbà'])
#pprint(res2, width=43)

assert res2 == [['Name', 'bignè', 'zippole', 'babbà'],
 ['Da Gigi', 4, 2, 3],
 ['La Delizia', 3, 9, 5],
 ['Gnam gnam', 2, 6, 1],
 ['Il Dessert', 8, 4, 7],
 ['Totals', 17, 21, 16]]

```

**Exercise - actorswap**

⊕⊕⊕ Given a movie list `movies` where each movie is represented as a dictionary, RETURN a NEW list with NEW dictionaries having the male actor names swapped with the female ones.

- **ONLY** swap actor names
- you can't predict actor names
- you only know each dictionary holds exactly three keys, of which these two are known: `title` and `year`.

Example:

```

db = [
 {'title': 'Jerry Maguire',
 'year': 1996,
 'Jerry': 'Dorothy',},
 {'title': 'Superman',
 'Kent': 'Lois',
 'year': 1978},
 {'title': 'The Lord of the Rings',
 'year': 2001,
 'Aragorn': 'Arwen',},
 {'Ron Weasley': 'Hermione',
 'title': 'Harry Potter and the Deathly Hallows, Part 2',
 'year': 2011}
]

>>> actorswap(db)
[{'title': 'Jerry Maguire',
 'year': 1996,
 'Dorothy': 'Jerry',},
 {'title': 'Superman',
 'year': 1978,
 'Lois': 'Kent',},
 {'title': 'The Lord of the Rings',
 'year': 2001,
 'Arwen': 'Aragorn',},
 {'title': 'Harry Potter and the Deathly Hallows, Part 2',

```

(continues on next page)

(continued from previous page)

```
'year': 2011,
'Hermione': 'Ron Weasley',
}]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[9]:

```
def actorswap(movies):

 ret = []
 for diz in movies:
 nuovo = {}
 ret.append(nuovo)
 for k in diz:
 if k == 'title' or k == 'year':
 nuovo[k] = diz[k]
 else:
 nuovo[diz[k]] = k
 return ret

TEST START
l1 = []
assert actorswap(l1) == []

l2 = [{'title': 'Pretty Woman',
 'year': 1990,
 'Edward':'Vivian'},
 {'title': 'Titanic',
 'year': 1997,
 'Jack' : 'Rose'}
]
orig_film = l2[0]
res2 = actorswap(l2)
assert res2 == [{'title': 'Pretty Woman',
 'year': 1990,
 'Vivian':'Edward'},
 {'title': 'Titanic',
 'year': 1997,
 'Rose' : 'Jack'}
]
assert id(l2) != id(res2) # must produce a NEW list
assert id(orig_film) != id(res2[0]) # must produce a NEW dictionary

l3 = [
 {'title':'Jerry Maguire',
 'year':1996,
 'Jerry':'Dorothy',},
 {'title':'Superman',
 'Kent':'Lois',
 'year': 1978},
 {'title':'The Lord of the Rings',
 'year': 2001,
 'Aragorn':'Arwen',},
 {'Ron Weasley':'Hermione',}
```

(continues on next page)

(continued from previous page)

```

 'title': 'Harry Potter and the Deathly Hallows, Part 2',
 'year': 2011}
]

assert actorswap(13) == [{ 'title': 'Jerry Maguire',
 'year': 1996,
 'Dorothy': 'Jerry'},
{ 'title': 'Superman',
 'year': 1978,
 'Lois': 'Kent'},
{ 'title': 'The Lord of the Rings',
 'year': 2001,
 'Arwen': 'Aragorn'},
{ 'title': 'Harry Potter and the Deathly Hallows, Part 2',
 'year': 2011,
 'Hermione': 'Ron Weasley'},
}]

```

&lt;/div&gt;

[9]:

```

def actorswap(movies):
 raise Exception('TODO IMPLEMENT ME !')

TEST START
l1 = []
assert actorswap(l1) == []

l2 = [{ 'title': 'Pretty Woman',
 'year': 1990,
 'Edward':'Vivian'},
{ 'title': 'Titanic',
 'year': 1997,
 'Jack' : 'Rose'}
]
orig_film = l2[0]
res2 = actorswap(l2)
assert res2 == [{ 'title': 'Pretty Woman',
 'year': 1990,
 'Vivian':'Edward'},
{ 'title': 'Titanic',
 'year': 1997,
 'Rose' : 'Jack'}
]
assert id(l2) != id(res2) # must produce a NEW list
assert id(orig_film) != id(res2[0]) # must produce a NEW dictionary

l3 = [
 { 'title': 'Jerry Maguire',
 'year': 1996,
 'Jerry': 'Dorothy'},
{ 'title': 'Superman',
 'Kent': 'Lois',
 'year': 1978},
{ 'title': 'The Lord of the Rings',
 'year': 2001,
 'Aragorn': 'Arwen'},
]

```

(continues on next page)

(continued from previous page)

```

{'Ron Weasley':'Hermione',
 'title': 'Harry Potter and the Deathly Hallows, Part 2',
 'year': 2011}
]

assert actorswap(13) == [{"title": "Jerry Maguire",
 'year': 1996,
 'Dorothy': "Jerry"},
 {"title": "Superman",
 'year': 1978,
 'Lois': "Kent"},
 {"title": "The Lord of the Rings",
 'year': 2001,
 'Arwen': "Aragorn"},
 {"title": "Harry Potter and the Deathly Hallows, Part 2",
 'year': 2011,
 'Hermione': "Ron Weasley",
 }]

```

## Continue

Go on with the challenges<sup>293</sup> ...

[ ]:

### 7.3.2 Mixed structures 2 - Challenges

#### Download exercises zip

Browse online files<sup>294</sup>

We now propose some exercises without solution, do you accept the challenge?

#### Challenge - Guilty!

Recently there has been a brutal execution at a *pizzeria* in Little Italy, and the FBI has a list of suspects as a dictionary list. Each dictionary holds the suspect's name and values for weapon, place and motive, which tell the degree of suspicion for each category. The FBI also has a weights list assigned to each suspicion category, which are used by judges to determine the degree of guiltiness of each suspect. weights is expressed as a list of tuples: each tuple contains a suspicion category and the related weight as float. To calculate the guiltiness of each suspect, each weight is multiplied for the corresponding suspect's suspicion value and an overall sum is performed.

The FBI asks you to produce a NEW table as a list of lists containing the gangster data, plus a column 'guiltiness' calculated as explained above.

- **REMEMBER** the table headers
- **USE** the same items order as found in the weights list

<sup>293</sup> <https://en.softpython.org/mixed-structures/mixed-structures2-chal.html>

<sup>294</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/mixed-structures>

[1]:

```
db = [
 {'name' : 'Cadillac Frank',
 'weapon' : 5,
 'place' : 3,
 'motive' : 7},
 {'name' : 'Lucky Vincent',
 'weapon' : 7,
 'place' : 4,
 'motive' : 8},
 {'name' : 'Three Fingers',
 'weapon' : 1,
 'place' : 7,
 'motive' : 4},
 {'name' : 'Vito The Butcher',
 'weapon' : 3,
 'place' : 6,
 'motive' : 5},
]

def judge(gangsters, weights):
 raise Exception('TODO IMPLEMENT ME !')

res = judge(db, [('weapon', 0.1),
 ('motive', 0.7),
 ('place', 0.2)])
from pprint import pprint
pprint(res, width=80)

note: since the table contains floats it would be better to check for closeness of
values ...
assert res == [[['name', 'weapon', 'motive', 'place', 'guiltiness'],
 ['Cadillac Frank', 5, 7, 3, 6.0],
 ['Lucky Vincent', 7, 8, 4, 7.1],
 ['Three Fingers', 1, 4, 7, 4.3],
 ['Vito The Butcher', 3, 5, 6, 5.0]]]
```

[ ]:

## 7.4 Numpy matrices

### 7.4.1 Matrices: Numpy 1

[Download exercises zip](#)

[Browse files online<sup>295</sup>](#)

<sup>295</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/matrices-numpy>

## Introduction

There are substantially two ways to represent matrices in Python: we've first encountered matrices as lists of lists<sup>296</sup>, in this tutorial we focus on matrices as provided by the widely used **Numpy**<sup>297</sup> library.

Let's see the main differences:

List of lists - see separate notebook<sup>298</sup>

1. native in Python
2. not efficient
3. lists are pervasive in Python, probably you will encounter matrices expressed as list of lists anyway
4. gives an idea of how to build a nested data structure
5. may help in understanding important concepts like pointers to memory and copies

Numpy - this notebook

1. not natively available in Python
2. efficient
3. many libraries for scientific calculations are based on Numpy (scipy, pandas)
4. easier syntax to access elements (slightly different from list of lists)
5. in rare cases might give problems of installation and/or conflicts (implementation is not pure Python)

We will only see main data types and essential commands of **Numpy** library<sup>299</sup>, without going much into the details. In particular, we will review the new data format `ndarray` and compare slow algorithms with Python `for` cycles to faster ones made possible by idiomatic use of Numpy vector operations.

For further references, see [Python Data Science Handbook](#), Numpy part<sup>300</sup>

**WARNING:** Numpy does not work in [Python Tutor](#)<sup>301</sup>

## What to do

- unzip exercises in a folder, you should get something like this:

```
matrices-numpy
 matrices-numpy1.ipynb
 matrices-numpy1-sol.ipynb
 matrices-numpy2.ipynb
 matrices-numpy2-sol.ipynb
 matrices-numpy3-chal.ipynb
 numpy-images.ipynb
 numpy-images-sol.ipynb
 jupman.py
```

<sup>296</sup> <https://en.softpython.org/matrices-lists/matrices-lists-sol.html>

<sup>297</sup> <https://www.numpy.org>

<sup>298</sup> <https://en.softpython.org/matrices-lists/matrices-lists1-sol.html>

<sup>299</sup> <https://www.numpy.org>

<sup>300</sup> <https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html>

<sup>301</sup> <http://www.pythontutor.com/visualize.html#mode=edit>

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `matrices-numpy/matrices-numpy1.ipynb`
- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### np.array

First of all, we import the library, and for convenience we rename it to np

```
[2]: import numpy as np
```

With lists of lists we have often built the matrices one row at a time, adding lists as needed. In Numpy instead we usually create in one shot the whole matrix, filling it with zeroes.

In particular, this command creates an ndarray filled with zeroes:

```
[3]: mat = np.zeros((2,3)) # 2 rows, 3 columns
```

```
[4]: mat
```

```
[4]: array([[0., 0., 0.],
 [0., 0., 0.]])
```

Note like inside `array()` the content seems represented like a list of lists, BUT in reality in physical memory the data is structured in a linear sequence which allows Python to access numbers in a faster way.

We can also create an ndarray from a list of lists:

```
[5]: mat = np.array([[5.0, 8.0, 1.0],
 [4.0, 3.0, 2.0]])
```

```
[6]: mat
```

```
[6]: array([[5., 8., 1.],
 [4., 3., 2.]])
```

```
[7]: type(mat)
```

```
[7]: numpy.ndarray
```

## Creating a matrix filled with ones

```
[8]: np.ones((3,5)) # 3 rows, 5 columns
[8]: array([[1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1.]])
```

## Creating a matrix filled with a number k

```
[9]: np.full((3,5), 7)
[9]: array([[7, 7, 7, 7, 7],
 [7, 7, 7, 7, 7],
 [7, 7, 7, 7, 7]])
```

## Dimensions of a matrix

To obtain the dimension, we write like the following:

**ATTENTION:** after `shape` there are **no** round parenthesis !

`shape` is an attribute, not a function to call

```
[10]: mat = np.array([[5.0,8.0,1.0],
 [4.0,3.0,2.0]])
mat.shape
[10]: (2, 3)
```

If we want to memorize the dimension in separate variables, we can use thi more pythonic mode (note the comma between `num_rows` and `num_cols`):

```
[11]: num_rows, num_cols = mat.shape
```

```
[12]: num_rows
```

```
[12]: 2
```

```
[13]: num_cols
```

```
[13]: 3
```

## Reading and writing

To access data or overwrite square bracket notation is used, with the important difference that in Numpy you can write *both* the indeces *inside* the same brackets, separated by a comma:

**ATTENTION:** notation `mat[i, j]` is only for Numpy, with list of lists **does not** work!

```
[14]: mat = np.array([[5.0, 8.0, 1.0],
 [4.0, 3.0, 2.0]])

Let's put number `9` in cell at row `0` and column `1`

mat[0, 1] = 9
```

```
[15]: mat
[15]: array([[5., 9., 1.],
 [4., 3., 2.]])
```

Let's access cell at row 0 and column 1

```
[16]: mat[0, 1]
[16]: 9.0
```

We put number 7 into cell at row 1 and column 2

```
[17]: mat[1, 2] = 7
```

```
[18]: mat
[18]: array([[5., 9., 1.],
 [4., 3., 7.]])
```

⊗ **EXERCISE:** try to write like the following, what happens?

```
mat[0, 0] = "c"
```

```
[19]: # write here
```

⊗ **EXERCISE:** Try writing like this, what happens?

```
mat[1, 1.0]
```

```
[20]: # write here
```

## Filling the whole matrix

We can MODIFY the matrix by writing inside a number with `fill()`

```
[21]: mat = np.array([[3.0, 5.0, 2.0],
 [6.0, 2.0, 9.0]])

mat.fill(7) # NOTE: returns nothings !!
```

```
[22]: mat
[22]: array([[7., 7., 7.],
 [7., 7., 7.]])
```

## Slices

To extract data from an ndarray we can use slices, with the notation we already used for regular lists. There are important difference, though. Let's see them.

The first difference is that we can extract sub-matrices by specifying two ranges among the same squared brackets:

```
[23]: mat = np.array([[5, 8, 1],
 [4, 3, 2],
 [6, 7, 9],
 [9, 3, 4],
 [8, 2, 7]])
```

```
[24]: mat[0:4, 1:3] # rows from 0 *included* to 4 *excluded*
 # and columns from 1 *included* to 3 *excluded*
[24]: array([[8, 1],
 [3, 2],
 [7, 9],
 [3, 4]])
```

```
[25]: mat[0:1,0:3] # the whole first row
[25]: array([[5, 8, 1]])
```

```
[26]: mat[0:1,:] # another way to extract the whole first row
[26]: array([[5, 8, 1]])
```

```
[27]: mat[0:5, 0:1] # the whole first column
[27]: array([[5],
 [4],
 [6],
 [9],
 [8]])
```

```
[28]: mat[:, 0:1] # another way to extract the whole first column
[28]: array([[5],
 [4],
 [6],
```

(continues on next page)

(continued from previous page)

```
[9],
[8]])
```

**The step:** We can also specify a step as a third parameter after the `:`. For example, to extract only even rows we can add a `2` like this:

```
[29]: mat[0:5:2, :]

[29]: array([[5, 8, 1],
 [6, 7, 9],
 [8, 2, 7]])
```

### WARNING: by modifying the numpy slice you also modify the original matrix!

Differently from slices of lists which always produce new lists, this time of performance reasons with numpy slices we only obtain a *view* on the original data: by writing into the view we will also write on the original matrix:

```
[30]: mat = np.array([[5, 8, 1],
 [4, 3, 2],
 [6, 7, 9],
 [9, 3, 4],
 [8, 2, 7]])
```

```
[31]: sub_mat = mat[0:4, 1:3]
sub_mat

[31]: array([[8, 1],
 [3, 2],
 [7, 9],
 [3, 4]])
```

```
[32]: sub_mat[0,0] = 999
```

```
[33]: mat

[33]: array([[5, 999, 1],
 [4, 3, 2],
 [6, 7, 9],
 [9, 3, 4],
 [8, 2, 7]])
```

### Writing a constant in a slice

We can also write a constant in all the cells of a region by identifying the region with a slice, and assigning a constant to it:

```
[34]: mat = np.array([[5, 8, 1],
 [4, 3, 2],
 [6, 7, 9],
 [9, 3, 4],
 [8, 2, 5]])
```

(continues on next page)

(continued from previous page)

```
mat[0:4, 1:3] = 7

mat

[34]: array([[5, 7, 7],
 [4, 7, 7],
 [6, 7, 7],
 [9, 7, 7],
 [8, 2, 5]])
```

## Writing a matrix into a slice

We can also write into all the cells in a region by identifying the region with a slice, and then assigning to it a matrix from which we want to read the cells.

**WARNING:** To avoid problems, **double check** you're using the same dimensions in both left and right slices!

```
[35]: mat = np.array([[5, 8, 1],
 [4, 3, 2],
 [6, 7, 9],
 [9, 3, 4],
 [8, 2, 5]])

mat[0:4, 1:3] = np.array([
 [10, 50],
 [11, 51],
 [12, 52],
 [13, 53],
])

mat

[35]: array([[5, 10, 50],
 [4, 11, 51],
 [6, 12, 52],
 [9, 13, 53],
 [8, 2, 5]])
```

## Assignment and copy

With Numpy we must take particular care when using the assignment operator `=`: as with regular lists, if we perform an assignment into the new variable, it will only contain a pointer to the original region of memory.

```
[36]: va = np.array([1,2,3])

[37]: va
[37]: array([1, 2, 3])

[38]: vb = va

[39]: vb[0] = 100
```

```
[40]: vb
[40]: array([100, 2, 3])
```

```
[41]: va
[41]: array([100, 2, 3])
```

If we wanted a complete copy of the array, we should use the `.copy()` method:

```
[42]: va = np.array([1, 2, 3])
```

```
[43]: vc = va.copy()
```

```
[44]: vc
[44]: array([1, 2, 3])
```

```
[45]: vc[0] = 100
```

```
[46]: vc
[46]: array([100, 2, 3])
```

```
[47]: va
[47]: array([1, 2, 3])
```

## Calculations

Numpy is extremely flexible, and allows us to perform on arrays almost the same operations from classical vector and matrix algebra:

```
[48]: va = np.array([5, 9, 7])
va
```

```
[48]: array([5, 9, 7])
```

```
[49]: vb = np.array([6, 8, 0])
vb
```

```
[49]: array([6, 8, 0])
```

Whenever we perform an algebraic operation, typically a NEW array is created:

```
[50]: vc = va + vb
vc
```

```
[50]: array([11, 17, 7])
```

Note the sum didn't change the input:

```
[51]: va
[51]: array([5, 9, 7])
```

```
[52]: vb
[52]: array([6, 8, 0])
```

## Scalar multiplication

```
[53]: m = np.array([[5, 9, 7],
 [6, 8, 0]])
```

```
[54]: 3 * m
[54]: array([[15, 27, 21],
 [18, 24, 0]])
```

## Scalar sum

```
[55]: 3 + m
[55]: array([[8, 12, 10],
 [9, 11, 3]])
```

## Multiplication

Be careful about multiplying with `*`: differently from classical matrix multiplication, it multiplies *element by element* and so requires matrices of identical dimensions:

```
[56]: ma = np.array([[1, 2, 3],
 [10, 20, 30]])

mb = np.array([[1, 0, 1],
 [4, 5, 6]])

ma * mb
[56]: array([[1, 0, 3],
 [40, 100, 180]])
```

If we want the matrix multiplication from classical algebra<sup>302</sup>, we must use the `@` operator taking care of having compatible matrix dimensions:

```
[57]: mc = np.array([[1, 2, 3],
 [10, 20, 30]])
md = np.array([[1, 4],
 [0, 5],
 [1, 6]])

mc @ md
[57]: array([[4, 32],
 [40, 320]])
```

<sup>302</sup> [https://en.wikipedia.org/wiki/Matrix\\_multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication)

### Dividing by a scalar

```
[58]: ma = np.array([[1, 2, 0.0],
 [10, 0.0, 30]])

ma / 4

[58]: array([[0.25, 0.5, 0.],
 [2.5, 0., 7.5]])
```

Careful about dividing by 0.0, the program execution will still continue with a warning and we will find a matrix with strange nan and inf which have a bad tendency to create problems later - see the section [NaNs and infinities](#)

```
[59]: print(ma / 0.0)
print("AFTER")

[[inf inf nan]
 [inf nan inf]]
AFTER

/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:
→divide by zero encountered in true_divide
 """Entry point for launching an IPython kernel.
/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:
→invalid value encountered in true_divide
 """Entry point for launching an IPython kernel.
```

### Aggregation

Numpy provides several functions to calculate statistics, we only show some:

```
[60]: m = np.array([[5, 4, 6],
 [3, 7, 1]])
np.sum(m)

[60]: 26
```

```
[61]: np.max(m)
[61]: 7
```

```
[62]: np.min(m)
[62]: 1
```

### Aggregating by row or column

By adding the `axis` parameter we can tell numpy to perform the aggregation on each column (`axis=0`) or row (`axis=1`):

```
[63]: np.max(m, axis=0) # the maximum of each column
[63]: array([5, 7, 6])

[64]: np.sum(m, axis=0) # sum each column
```

```
[64]: array([8, 11, 7])

[65]: np.max(m, axis=1) # the maximum of each row
[65]: array([6, 7])

[66]: np.sum(m, axis=1) # sum each row
[66]: array([15, 11])
```

## Filtering

Numpy offers a mini-language to filter the numbers in an array, by specifying the selection criteria. Let's see an example:

```
[67]: mat = np.array([[5, 2, 6],
 [1, 4, 3]])
mat
[67]: array([[5, 2, 6],
 [1, 4, 3]])
```

Suppose you want to obtain an array with all the numbers from `mat` which are greater than 2.

We can tell numpy the matrix `mat` we want to use, then *inside square brackets* we put a kind of boolean conditions, *reusing* the `mat` variable like so:

```
[68]: mat[mat > 2]
[68]: array([5, 6, 4, 3])
```

Exactly, what is that strange expression we put inside the squared brackets? Let's try executing it alone:

```
[69]: mat > 2
[69]: array([[True, False, True],
 [False, True, True]])
```

We note it gives us a matrix of booleans, which are `True` whenever the corresponding cell in the original matrix satisfies the condition we imposed.

By then placing this expression inside `mat[ ]` we obtain the values from the original matrix which satisfy the expression:

```
[70]: mat[mat > 2]
[70]: array([5, 6, 4, 3])
```

Not only that, we can also build more complex expressions by using

- `&` symbol as the logical conjunction *and*
- `|` (pipe character) as the logical conjunction *or*

```
[71]: mat = np.array([[5, 2, 6],
 [1, 4, 3]])
mat[(mat > 3) & (mat < 6)]
[71]: array([5, 4])
```

```
[72]: mat = np.array([[5, 2, 6],
 [1, 4, 3]])
mat[(mat < 2) | (mat > 4)]
[72]: array([5, 6, 1])
```

**WARNING: REMEMBER THE ROUND PARENTHESIS AMONG THE VARIOUS EXPRESSIONS!**

**EXERCISE:** try to rewrite the expressions above by ‘forgetting’ the round parenthesis in the various components (left/right/both) and see what happens. Do you obtain errors or unexpected results?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[73]: mat = np.array([[5, 2, 6],
 [1, 4, 3]])

write here
print(mat[(mat > 3) & mat < 6])
print(mat[mat > 3 & (mat < 6)])
#print(mat[mat > 3 & mat < 6])
the last one produces:

ValueError Traceback (most recent call last)
<ipython-input-212-33c5a083b265> in <module>
3 print(mat[(mat > 3) & mat < 6])
4 print(mat[mat > 3 & (mat < 6)])
----> 5 print(mat[mat > 3 & mat < 6])

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()

[5 2 6 1 4 3]
[5 2 6 4 3]
```

</div>

```
[73]: mat = np.array([[5, 2, 6],
 [1, 4, 3]])

write here
```

**WARNING: and `and` or `DON'T WORK!`**

**EXERCISE:** try rewriting the expressions above by substituting `&` with `and` and `|` with `or` and see what happens. Do you get errors or unexpected results?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[74]: mat = np.array([[5, 2, 6],
 [1, 4, 3]])

write here
#print(mat[(mat > 3) and (mat < 6)])
#-----
#ValueError Traceback (most recent call last)
#<ipython-input-218-3edf025af7c0> in <module>
4
5 # write here
#--> 6 print(mat[(mat > 3) and (mat < 6)])

#ValueError: The truth value of an array with more than one element is ambiguous. Use
~a.any() or a.all()

#print(mat[(mat > 3) or (mat < 6)])
#-----
#ValueError Traceback (most recent call last)
#<ipython-input-219-192c022d9d87> in <module>
4
5
#--> 6 print(mat[(mat > 3) or (mat < 6)])

#ValueError: The truth value of an array with more than one element is ambiguous. Use
~a.any() or a.all()
```

&lt;/div&gt;

```
[74]: mat = np.array([[5, 2, 6],
 [1, 4, 3]])

write here
```

## Finding indexes with np.where

We've seen how to find the content of cells which satisfy a certain criteria. What if we wanted to find the *indeces* of those cells? In that case we would use the function `np.where`, passing as parameter the condition expressed in the same language used before.

For example, if we wanted to find the *indexes* of cells containing numbers less than 40 or greater than 60 we would write like so:

```
[75]: #0 1 2 3 4 5
v = np.array([30,60,20,70,40,80])

np.where((v < 40) | (v > 60))

[75]: (array([0, 2, 3, 5]),)
```

## Writing into cells which satisfy a criteria

We can use `np.where` to substitute values in the cells which satisfy a criteria with other values which we'll be expressed in two extra matrices `ma` and `mb`. In case the criteria is satisfied, numpy will take the corresponding values from `ma`, otherwise from `mb`.

```
[76]: ma = np.array([
 [1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12]
])

mb = np.array([
 [-1, -2, -3, -4],
 [-5, -6, -7, -8],
 [-9,-10,-11,-12]
])

mat = np.array([
 [40,70,10,80],
 [20,30,60,40],
 [10,60,80,90]
])

np.where(mat < 50, ma, mb)
```

```
[76]: array([[1, -2, 3, -4],
 [5, 6, -7, 8],
 [9, -10, -11, -12]])
```

## arange and linspace sequences

The standard function `range` of Python does not allow for float increments, which we can instead obtain by building sequences of float numbers with `np.arange`, by specifying left limit (**included**), right limit (**excluded**) and the increment:

```
[77]: np.arange(0.0, 1.0, 0.2)

[77]: array([0., 0.2, 0.4, 0.6, 0.8])
```

Alternatively, we can use `np.linspace`, which takes a left limit **included**, a right limit this time **included**, and the **number of repetitions** to subdivide this space:

```
[78]: np.linspace(0, 0.8, 5)

[78]: array([0., 0.2, 0.4, 0.6, 0.8])

[79]: np.linspace(0, 0.8, 10)

[79]: array([0. , 0.08888889, 0.17777778, 0.26666667, 0.35555556,
 0.44444444, 0.53333333, 0.62222222, 0.71111111, 0.8])
```

## NaN<sup>s</sup> and infinities

Float numbers can be numbers and.... not numbers, and infinities. Sometimes during calculations extremal conditions may arise, like when dividing a small number by a huge number. In such cases, you might end up having a float which is a dreaded *Not a Number*, *NaN* for short, or you might get an infinity. This can lead to very awful unexpected behaviours, so you must be well aware of it. Examples:

```
[80]: 10e9999999999999999999999999999999
```

```
[80]: inf
```

```
[81]: 10e9999999999999999999999999999999 / 10e9999999999999999999999999999999
```

```
[81]: nan
```

Following behaviours are dictated by IEEE Standard for Binary Floating-Point for Arithmetic (IEEE 754) which Numpy uses and is implemented in all CPUs, so they actually regard all programming languages.

## NaN<sup>s</sup>

A NaN is *Not a Number*. Which is already a silly name, since a NaN is actually a very special member of floats, with this astonishing property:

### WARNING: NaN IS NOT EQUAL TO ITSELF !!!!

Yes you read it right, NaN is really *not* equal to itself.

Even if your mind wants to refuse it, we are going to confirm it.

To get a NaN, you can use Python module `math` which holds this alien item:

```
[82]: import math
math.nan # notice it prints as 'nan' with lowercase n
```

```
[82]: nan
```

As we said, a NaN is actually considered a float:

```
[83]: type(math.nan)
```

```
[83]: float
```

Still, it behaves very differently from its fellow floats, or any other object in the known universe:

```
[84]: math.nan == math.nan # what the F... else
```

```
[84]: False
```

### Detecting NaN

Given the above, if you want to check if a variable `x` is a NaN, you *cannot* write this:

```
[85]: x = math.nan
if x == math.nan: # WRONG
 print("I'm NaN ")
else:
 print("x is something else ??")
x is something else ??
```

To correctly handle this situation, you need to use `math.isnan` function:

```
[86]: x = math.nan
if math.isnan(x): # CORRECT
 print("x is NaN ")
else:
 print("x is something else ??")
x is NaN
```

Notice `math.isnan` also work with *negative* NaN:

```
[87]: y = -math.nan
if math.isnan(y): # CORRECT
 print("y is NaN ")
else:
 print("y is something else ??")
y is NaN
```

### Sequences with NaNs

Still, not everything is completely crazy. If you compare a sequence holding NaNs to another one, you will get reasonable results:

```
[88]: [math.nan, math.nan] == [math.nan, math.nan]
[88]: True
```

### Exercise NaN: two vars

Given two number variables `x` and `y`, write some code that prints "same" when they are the same, *even* when they are NaN. Otherwise, prints "not the same"

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[89]: # expected output: same
x = math.nan
y = math.nan

expected output: not the same
#x = 3
```

(continues on next page)

(continued from previous page)

```
#y = math.nan

expected output: not the same
#x = math.nan
#y = 5

expected output: not the same
#x = 2
#y = 7

expected output: same
#x = 4
#y = 4

write here
if math.isnan(x) and math.isnan(y):
 print('same')
elif x == y:
 print('same')
else:
 print('not the same')
```

same

&lt;/div&gt;

```
[89]: # expected output: same
x = math.nan
y = math.nan

expected output: not the same
#x = 3
#y = math.nan

expected output: not the same
#x = math.nan
#y = 5

expected output: not the same
#x = 2
#y = 7

expected output: same
#x = 4
#y = 4

write here
```

same

### Operations on NaNs

Any operation on a NaN will generate another NaN:

```
[90]: 5 * math.nan
```

```
[90]: nan
```

```
[91]: math.nan + math.nan
```

```
[91]: nan
```

```
[92]: math.nan / math.nan
```

```
[92]: nan
```

The only thing you cannot do is dividing by zero with an unboxed NaN:

```
math.nan / 0
```

```

ZeroDivisionError Traceback (most recent call last)
<ipython-input-94-1da38377fac4> in <module>
----> 1 math.nan / 0

ZeroDivisionError: float division by zero
```

NaN corresponds to boolean value True:

```
[93]: if math.nan:
 print("That's True")
```

```
That's True
```

### NaN and Numpy

When using Numpy you are quite likely to encounter NaNs, so much so they get redefined inside Numpy, but they are exactly the same as in math module:

```
[94]: np.nan
```

```
[94]: nan
```

```
[95]: math.isnan(np.nan)
```

```
[95]: True
```

```
[96]: np.isnan(math.nan)
```

```
[96]: True
```

In Numpy when you have unknown numbers you might be tempted to put a None. You can actually do it, but look closely at the result:

```
[97]: import numpy as np
np.array([4.9, None, 3.2, 5.1])
```

```
[97]: array([4.9, None, 3.2, 5.1], dtype=object)
```

The resulting array type is *not* an array of float64 which allows fast calculations, instead it is an array containing generic *objects*, as Numpy is assuming the array holds heterogenous data. So what you gain in generality you lose it in performance, which should actually be the whole point of using Numpy.

Despite being weird, NaNs are actually regular float citizen so they can be stored in the array:

```
[98]: np.array([4.9,np.nan,3.2,5.1]) # Notice how the `dtype=object` has disappeared
```

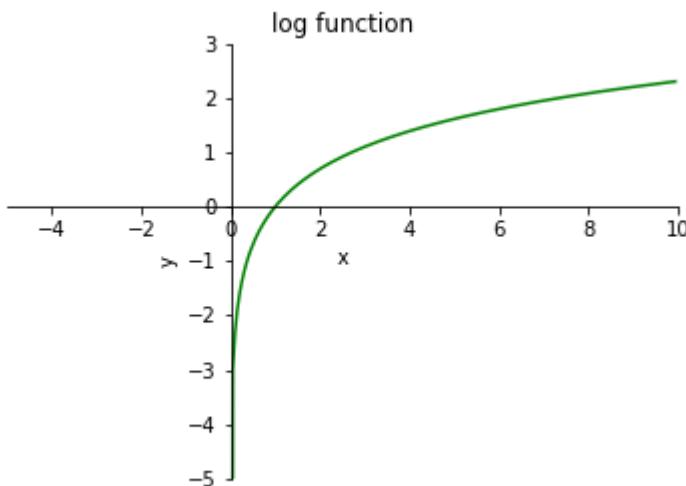
```
[98]: array([4.9, nan, 3.2, 5.1])
```

## Where are the NaNs ?

Let's try to see where we can spot NaNs and other weird things such infinities in the wild

First, let check what happens when we call function `log` of standard module `math`. As we know, `log` function behaves like this:

- $x < 0$ : not defined
- $x = 0$ : tends to minus infinity
- $x > 0$ : defined



So we might wonder what happens when we pass to it a value where it is not defined. Let's first try with the standard `math.log` from Python library:

```
>>> math.log(-1)
```

```
ValueError Traceback (most recent call last)
<ipython-input-38-d6e02ba32da6> in <module>
----> 1 math.log(-1)

ValueError: math domain error
```

In this case `ValueError` is raised and **the execution gets interrupted**.

Let's try the equivalent with Numpy:

```
[99]: np.log(-1)

/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:__
 ↵invalid value encountered in log
 """Entry point for launching an IPython kernel.

[99]: nan
```

In this case we **actually got as a result** `np.nan`, so execution was not interrupted, Jupyter only informed us with an extra print that something dangerous happened.

The default behaviour of Numpy regarding dangerous calculations is to perform them anyway and storing the result in as a NaN or other limit objects. This also works for arrays calculations:

```
[100]: np.log(np.array([3, 7, -1, 9]))

/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:__
 ↵invalid value encountered in log
 """Entry point for launching an IPython kernel.

[100]: array([1.09861229, 1.94591015, nan, 2.19722458])
```

## Infinities

As we said previously, NumPy uses the IEEE Standard for Binary Floating-Point for Arithmetic (IEEE 754). Since somebody at IEEE decided to capture the mysteries of infinity into floating numbers, we have yet another citizen to take into account when performing calculations (for more info see [Numpy documentation on constants<sup>303</sup>](#)):

### Positive infinity `np.inf`

```
[101]: np.array([5]) / 0

/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:__
 ↵divide by zero encountered in true_divide
 """Entry point for launching an IPython kernel.

[101]: array([inf])

[102]: np.array([6, 9, 5, 7]) / np.array([2, 0, 0, 4])

/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:__
 ↵divide by zero encountered in true_divide
 """Entry point for launching an IPython kernel.

[102]: array([3. , inf, inf, 1.75])
```

Be aware that:

- Not a Number is **not** equivalent to infinity
- positive infinity is **not** equivalent to negative infinity
- infinity is equivalent to positive infinity

This time, infinity is equal to infinity:

---

<sup>303</sup> <https://numpy.org/devdocs/reference/constants.html>

```
[103]: np.inf == np.inf
[103]: True
```

so we can safely detect infinity with ==:

```
[104]: x = np.inf

if x == np.inf:
 print("x is infinite")
else:
 print("x is finite")

x is infinite
```

Alternatively, we can use the function np.isinf:

```
[105]: np.isinf(np.inf)
[105]: True
```

## Negative infinity

We can also have negative infinity, which is different from positive infinity:

```
[106]: -np.inf == np.inf
[106]: False
```

Note that isinf detects *both* positive and negative:

```
[107]: np.isinf(-np.inf)
[107]: True
```

To actually check for negative infinity you have to use isneginf:

```
[108]: np.isneginf(-np.inf)
[108]: True
```

```
[109]: np.isneginf(np.inf)
[109]: False
```

Where do they appear? As an example, let's try np.log function:

```
[110]: np.log(0)
/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in log
 """Entry point for launching an IPython kernel.

[110]: -inf
```

## Combining infinities and NaNs

When performing operations involving infinities and NaNs, IEEE arithmetics tries to mimic classical analysis, sometimes including NaN as a result:

```
[111]: np.inf + np.inf
```

```
[111]: inf
```

```
[112]: - np.inf - np.inf
```

```
[112]: -inf
```

```
[113]: np.inf * -np.inf
```

```
[113]: -inf
```

What in classical analysis would be undefined, here becomes NaN:

```
[114]: np.inf - np.inf
```

```
[114]: nan
```

```
[115]: np.inf / np.inf
```

```
[115]: nan
```

As usual, combining with NaN results in NaN:

```
[116]: np.inf + np.nan
```

```
[116]: nan
```

```
[117]: np.inf / np.nan
```

```
[117]: nan
```

## Negative zero

We can even have a *negative* zero - who would have thought?

```
[118]: np.NZERO
```

```
[118]: -0.0
```

Negative zero of course pairs well with the more known and much appreciated *positive* zero:

```
[119]: np.PZERO
```

```
[119]: 0.0
```

**NOTE:** Writing `np.NZERO` or `-0.0` is *exactly* the same thing. Same goes for positive zero.

At this point, you might start wondering with some concern if they are actually *equal*. Let's try:

```
[120]: 0.0 == -0.0
```

```
[120]: True
```

Great! Finally one thing that makes sense.

Given the above, you might think in a formula you can substitute one for the other one and get same results, in harmony with the rules of the universe.

Let's make an attempt of substitution, as an example we first try dividing a number by positive zero (even if math teachers tell us such divisions are forbidden) - what will we ever get??

$$\frac{5.0}{0.0} = ???$$

In Numpy terms, we might write like this to box everything in arrays:

```
[121]: np.array([5.0]) / np.array([0.0])
/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:__
 divide by zero encountered in true_divide
 """Entry point for launching an IPython kernel.

[121]: array([inf])
```

Hmm, we got an array holding an `np.inf`.

If `0.0` and `-0.0` are actually the same, dividing a number by `-0.0` we should get the very same result, shouldn't we?

Let's try:

```
[122]: np.array([5.0]) / np.array([-0.0])
/home/da/.local/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:__
 divide by zero encountered in true_divide
 """Entry point for launching an IPython kernel.

[122]: array([-inf])
```

Oh gosh. This time we got an array holding a *negative* infinity `-np.inf`

If all of this seems odd to you, do not bash at Numpy. This is the way pretty much any CPUs does floating point calculations so you will find it in almost ALL computer languages.

What programming languages can do is add further controls to protect you from paradoxical situations, for example when you directly write `1.0/0.0` Python raises `ZeroDivisionError` (blocking thus execution), and when you operate on arrays Numpy emits a warning (but doesn't block execution).

### Exercise: detect proper numbers

Write some code that PRINTS `equal numbers` if two numbers `x` and `y` passed are equal and actual numbers, and PRINTS `not equal numbers` otherwise.

**NOTE:** `not equal numbers` must be printed if any of the numbers is infinite or `NaN`.

To solve it, feel free to call functions indicated in [Numpy documentation about constants](#)<sup>304</sup>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[123]: # expected: equal numbers
x = 5
y = 5

expected: not equal numbers
```

(continues on next page)

<sup>304</sup> <https://docs.scipy.org/doc/numpy/reference/constants.html>

(continued from previous page)

```
#x = np.inf
#y = 3

expected: not equal numbers
#x = 3
#y = np.inf

expected: not equal numbers
#x = np.inf
#y = np.nan

expected: not equal numbers
#x = np.nan
#y = np.inf

expected: not equal numbers
#x = np.nan
#y = 7

expected: not equal numbers
#x = 9
#y = np.nan

expected: not equal numbers
#x = np.nan
#y = np.nan

write here

SOLUTION 1 - the ugly one
if np.isinf(x) or np.isinf(y) or np.isnan(x) or np.isnan(y):
 print('not equal numbers')
else:
 print('equal numbers')

SOLUTION 2 - the pretty one
if np.isfinite(x) and np.isfinite(y):
 print('equal numbers')
else:
 print('not equal numbers')

equal numbers
equal numbers
```

&lt;/div&gt;

```
[123]: # expected: equal numbers
x = 5
y = 5

expected: not equal numbers
#x = np.inf
#y = 3

expected: not equal numbers
#x = 3
```

(continues on next page)

(continued from previous page)

```
#y = np.inf

expected: not equal numbers
#x = np.inf
#y = np.nan

expected: not equal numbers
#x = np.nan
#y = np.inf

expected: not equal numbers
#x = np.nan
#y = 7

expected: not equal numbers
#x = 9
#y = np.nan

expected: not equal numbers
#x = np.nan
#y = np.nan

write here
```

```
equal numbers
equal numbers
```

### Exercise: guess expressions

For each of the following expressions, try to guess the result

**WARNING: the following may cause severe convulsions and nausea.**

During clinical trials, both mathematically inclined and math-averse patients have experienced illness, for different reasons which are currently being investigated.

```
a. 0.0 * -0.0
b. (-0.0)**3
c. np.log(-7) == math.log(-7)
d. np.log(-7) == np.log(-7)
e. np.isnan(1 / np.log(1))
f. np.sqrt(-1) * np.sqrt(-1) # sqrt = square root
g. 3 ** np.inf
h. 3 ** -np.inf
i. 1/np.sqrt(-3)
j. 1/np.sqrt(-0.0)
m. np.sqrt(np.inf) - np.sqrt(-np.inf)
n. np.sqrt(np.inf) + (1 / np.sqrt(-0.0))
o. np.isneginf(np.log(np.e) / np.sqrt(-0.0))
p. np.isinf(np.log(np.e) / np.sqrt(-0.0))
```

(continues on next page)

(continued from previous page)

```
q. [np.nan, np.inf] == [np.nan, np.inf]
r. [np.nan, -np.inf] == [np.nan, np.inf]
s. [np.nan, np.inf] == [-np.nan, np.inf]
```

## Continue

Go on with [numpy exercises](#)<sup>305</sup>.

### 7.4.2 Matrices: Numpy 2 - Exercises

#### Download exercises zip

Browse files online<sup>306</sup>

#### Introduction

Let's see now some exercises. First ones will be given in two versions: first ones usually adopt for cycles and are thus slow, second ones are denoted 'pro' and avoid loops using all the power offered by Numpy. In particular in many cases you can obtain very efficient and compact programs by using slices in smart ways.

Numpy - this notebook

1. not natively available in Python
2. efficient
3. many libraries for scientific calculations are based on Numpy (scipy, pandas)
4. syntax to access elements is slightly different from list of lists
5. in rare cases might give problems of installation and/or conflicts (implementation is not pure Python)

#### ATTENTION

Following exercises contain tests with *asserts*. To understand how to carry them out, read first [Error handling and testing](#)<sup>307</sup>

#### frame

⊕⊕⊕ RETURN a NEW Numpy matrix of n rows and n columns, in which all the values are zero except those on borders, which must be equal to a given k

For example, `frame(4, 7.0)` must give:

```
array([[7.0, 7.0, 7.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 7.0, 7.0, 7.0]])
```

<sup>305</sup> <https://en.softpython.org/matrices-numpy/matrices-numpy2-sol.html>

<sup>306</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/matrices-numpy>

<sup>307</sup> <https://en.softpython.org/functions/fun2-errors-and-testing-sol.html>

Ingredients:

- create a matrix filled with zeros. ATTENTION: which dimensions does it have? Do you need n or k ? Read WELL the text.

For this first version, try filling the rows and columns using `for` in `range` and writing directly in the single cells

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]: `import numpy as np`

```
def frame(n, k):

 #SLOW SOLUTION
 mat = np.zeros((n,n))
 for i in range(n):
 mat[0, i] = k
 mat[i, 0] = k
 mat[i, n-1] = k
 mat[n-1, i] = k
 return mat

expected_mat = np.array([[7.0, 7.0, 7.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 7.0, 7., 7.0]])
all_close return True if all the values in the first matrix are close enough
(that is, within a given tolerance) to corresponding values in the second
assert np.allclose(frame(4, 7.0), expected_mat)

expected_mat = np.array([[7.0]
])
assert np.allclose(frame(1, 7.0), expected_mat)

expected_mat = np.array([[7.0, 7.0],
 [7.0, 7.0]
])
assert np.allclose(frame(2, 7.0), expected_mat)
```

</div>

[2]: `import numpy as np`

```
def frame(n, k):
 raise Exception('TODO IMPLEMENT ME !')

expected_mat = np.array([[7.0, 7.0, 7.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 7.0, 7., 7.0]])
all_close return True if all the values in the first matrix are close enough
(that is, within a given tolerance) to corresponding values in the second
assert np.allclose(frame(4, 7.0), expected_mat)

expected_mat = np.array([[7.0]
```

(continues on next page)

(continued from previous page)

```

])
assert np.allclose(frame(1, 7.0), expected_mat)

expected_mat = np.array([[7.0, 7.0],
 [7.0, 7.0]
])
assert np.allclose(frame(2, 7.0), expected_mat)

```

**Exercise - frameslices**⊕⊕⊕ Solve the precious exercise, this time **using 4 slices**

- **DO NOT** use for nor while loops

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[3]:

```

def frameslices(n, k):

 mat = np.zeros((n,n))

 mat[0, :] = k
 mat[:, 0] = k
 mat[:, n-1] = k
 mat[n-1, :] = k

 return mat

r1 = np.array([[7.0, 7.0, 7.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 7.0, 7., 7.0]])

all_close return True if all the values in the first matrix are close enough
(that is, within a given tolerance) to corresponding values in the second
assert np.allclose(frameslices(4, 7.0), r1)

r2 = np.array([[7.0]])
assert np.allclose(frameslices(1, 7.0), r2)

r3 = np.array([[7.0, 7.0],
 [7.0, 7.0]])
assert np.allclose(frameslices(2, 7.0), r3)

```

&lt;/div&gt;

[3]:

```

def frameslices(n, k):
 raise Exception('TODO IMPLEMENT ME !')

r1 = np.array([[7.0, 7.0, 7.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],

```

(continues on next page)

(continued from previous page)

```
[7.0, 0.0, 0.0, 7.0],
[7.0, 7.0, 7., 7.0]])

all_close return True if all the values in the first matrix are close enough
(that is, within a given tolerance) to corresponding values in the second
assert np.allclose(frameslices(4, 7.0), r1)

r2 = np.array([[7.0]])
assert np.allclose(frameslices(1, 7.0), r2)

r3 = np.array([[7.0, 7.0],
 [7.0, 7.0]])
assert np.allclose(frameslices(2, 7.0), r3)
```

### Exercise - framefill

⊕⊕⊕ Solve the precious exercise, this using `np.full` function and with **only one slice**

- **DO NOT** use `for` nor `while` loops

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4]:

```
def framefill(n, k):
 mat = np.full((n,n), k)
 mat[1:n-1, 1:n-1] = 0.0
 return mat

r1 = np.array([[7.0, 7.0, 7.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 7.0, 7., 7.0]])

all_close return True if all the values in the first matrix are close enough
(that is, within a given tolerance) to corresponding values in the second
assert np.allclose(framefill(4, 7.0), r1)

r2 = np.array([[7.0]])
assert np.allclose(framefill(1, 7.0), r2)

r3 = np.array([[7.0, 7.0],
 [7.0, 7.0]])
assert np.allclose(framefill(2, 7.0), r3)
```

&lt;/div&gt;

[4]:

```
def framefill(n, k):
 raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
r1 = np.array([[7.0, 7.0, 7.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 0.0, 0.0, 7.0],
 [7.0, 7.0, 7., 7.0]])

all_close return True if all the values in the first matrix are close enough
(that is, within a given tolerance) to corresponding values in the second
assert np.allclose(framefill(4, 7.0), r1)

r2 = np.array([[7.0]])
assert np.allclose(framefill(1, 7.0), r2)

r3 = np.array([[7.0, 7.0],
 [7.0, 7.0]])]
assert np.allclose(framefill(2, 7.0), r3)
```

**Exercise - avg\_rows**

$\oplus\oplus\oplus$  Takes a numpy matrix  $n \times m$  and RETURN a NEW numpy matrix consisting in a single column in which the values are the average of the values in corresponding rows of input matrix

Example:

Input: 5x4 matrix

3	2	1	4
6	2	3	5
4	3	6	2
4	6	5	4
7	2	9	3

Output: 5x1 matrix

(3+2+1+4) / 4
(6+2+3+5) / 4
(4+3+6+2) / 4
(4+6+5+4) / 4
(7+2+9+3) / 4

Basic version ingredients (slow)

- create a matrix  $n \times 1$  to return, filling it with zeros
- visit all cells of original matrix with two nested fors
- during visit, accumulate in the matrix to return the sum of elements takes from each row of original matrix
- once completed the sum of a row, you can divide it by the dimension of columns of original matrix
- return the matrix

Pro version (fast):

- try using `axis` parameter and `reshape`<sup>308</sup>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

<sup>308</sup> [https://www.tutorialspoint.com/numpy/numpy\\_reshape.htm](https://www.tutorialspoint.com/numpy/numpy_reshape.htm)

```
[5]: def avg_rows(mat):

 #SLOW SOLUTION
 nrows, ncols = mat.shape

 ret = np.zeros((nrows,1))

 for i in range(nrows):

 for j in range(ncols):
 ret[i] += mat[i,j]

 ret[i] = ret[i] / ncols
 # for brevity we could also write
 # ret[i] /= ncols

 return ret

m1 = np.array([[5.0]])
r1 = np.array([[5.0]])
assert np.allclose(avg_rows(m1), r1)

m2 = np.array([[5.0, 3.0]])
r2 = np.array([[4.0]])
assert np.allclose(avg_rows(m2), r2)

m3 = np.array([[3,2,1,4],
 [6,2,3,5],
 [4,3,6,2],
 [4,6,5,4],
 [7,2,9,3]])

r3 = np.array([[(3+2+1+4)/4],
 [(6+2+3+5)/4],
 [(4+3+6+2)/4],
 [(4+6+5+4)/4],
 [(7+2+9+3)/4]])

assert np.allclose(avg_rows(m3), r3)
```

&lt;/div&gt;

```
[5]: def avg_rows(mat):
 raise Exception('TODO IMPLEMENT ME !')
 return ret

m1 = np.array([[5.0]])
r1 = np.array([[5.0]])
assert np.allclose(avg_rows(m1), r1)

m2 = np.array([[5.0, 3.0]])
r2 = np.array([[4.0]])
assert np.allclose(avg_rows(m2), r2)
```

(continues on next page)

(continued from previous page)

```
m3 = np.array([[3,2,1,4],
 [6,2,3,5],
 [4,3,6,2],
 [4,6,5,4],
 [7,2,9,3]])

r3 = np.array([[(3+2+1+4)/4],
 [(6+2+3+5)/4],
 [(4+3+6+2)/4],
 [(4+6+5+4)/4],
 [(7+2+9+3)/4]])

assert np.allclose(avg_rows(m3), r3)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"  
data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[6]: #EFFICIENT SOLUTION avg_rows_pro

def avg_rows_pro(mat):
 rows, cols = mat.shape # obtain number of rows and columns
 media = np.mean(mat, axis=1) # average for rows
 media.shape = (rows, 1) # transform into a matrix with one col and n rows
 return media

m1 = np.array([[5.0]])
r1 = np.array([[5.0]])
assert np.allclose(avg_rows_pro(m1), r1)

m2 = np.array([[5.0, 3.0]])
r2 = np.array([[4.0]])
assert np.allclose(avg_rows_pro(m2), r2)

m3 = np.array(
 [[3,2,1,4],
 [6,2,3,5],
 [4,3,6,2],
 [4,6,5,4],
 [7,2,9,3]])

r3 = np.array([[(3+2+1+4)/4],
 [(6+2+3+5)/4],
 [(4+3+6+2)/4],
 [(4+6+5+4)/4],
 [(7+2+9+3)/4]])

assert np.allclose(avg_rows_pro(m3), r3)
```

</div>

```
[6]: #EFFICIENT SOLUTION avg_rows_pro
```

## Exercise - matrot

⊗⊗⊗ RETURN a NEW Numpy matrix which has the numbers of input matrix rotated by a column.

With rotation we mean that:

- if a number of input matrix is found in column  $j$ , in the output matrix it will be in the column  $j+1$  in the same row.
- if a number is found in the last column, in the output matrix it will be in the zeroth column

Example:

If we have as input:

```
np.array([
 [0, 1, 0],
 [1, 1, 0],
 [0, 0, 0],
 [0, 1, 1]
])
```

We expect as output:

```
np.array([
 [0, 0, 1],
 [0, 1, 1],
 [0, 0, 0],
 [1, 0, 1]
])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[7]: import numpy as np

def matrot(mat):

 #SLOW SOLUTION
 ret = np.zeros(mat.shape)

 for i in range(mat.shape[0]):
 ret[i, 0] = mat[i, -1]
 for j in range(1, mat.shape[1]):
 ret[i, j] = mat[i, j-1]
 return ret

m1 = np.array([[1]])
r1 = np.array([[1]])

assert np.allclose(matrot(m1), r1)

m2 = np.array([[0, 1]])
r2 = np.array([[1, 0]])
assert np.allclose(matrot(m2), r2)

m3 = np.array([[0, 1, 0]])
r3 = np.array([[0, 0, 1]])
```

(continues on next page)

(continued from previous page)

```
assert np.allclose(matrot(m3), r3)

m4 = np.array([
 [0, 1, 0],
 [1, 1, 0]
])
r4 = np.array([
 [0, 0, 1],
 [0, 1, 1]
])
assert np.allclose(matrot(m4), r4)

m5 = np.array([
 [0, 1, 0],
 [1, 1, 0],
 [0, 0, 0],
 [0, 1, 1]
])
r5 = np.array([
 [0, 0, 1],
 [0, 1, 1],
 [0, 0, 0],
 [1, 0, 1]
])
assert np.allclose(matrot(m5), r5)
```

&lt;/div&gt;

```
[7]: import numpy as np

def matrot(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[1]])
r1 = np.array([[1]])

assert np.allclose(matrot(m1), r1)

m2 = np.array([[0, 1]])
r2 = np.array([[1, 0]])
assert np.allclose(matrot(m2), r2)

m3 = np.array([[0, 1, 0]])
r3 = np.array([[0, 0, 1]])

assert np.allclose(matrot(m3), r3)

m4 = np.array([
 [0, 1, 0],
 [1, 1, 0]
])
r4 = np.array([
 [0, 0, 1],
 [0, 1, 1]
])
```

(continues on next page)

(continued from previous page)

```

assert np.allclose(matrot(m4), r4)

m5 = np.array([
 [0, 1, 0],
 [1, 1, 0],
 [0, 0, 0],
 [0, 1, 1]
])
r5 = np.array([
 [0, 0, 1],
 [0, 1, 1],
 [0, 0, 0],
 [1, 0, 1]
])
assert np.allclose(matrot(m5), r5)

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8]: #EFFICIENT SOLUTION

```

def matrot_pro(mat):
 m = mat.shape[0]
 n = mat.shape[1]

 ret = np.zeros((m, n))

 ret[:, 0] = mat[:, -1]
 ret[:, 1:] = mat[:, :-1]
 return ret

m1 = np.array([[1]])
r1 = np.array([[1]])
assert np.allclose(matrot_pro(m1), r1)

m2 = np.array([[0, 1]])
r2 = np.array([[1, 0]])
assert np.allclose(matrot_pro(m2), r2)

m3 = np.array([[0, 1, 0]])
r3 = np.array([[0, 0, 1]])
assert np.allclose(matrot_pro(m3), r3)

m4 = np.array([[0, 1, 0],
 [1, 1, 0]])
r4 = np.array([[0, 0, 1],
 [0, 1, 1]])
assert np.allclose(matrot_pro(m4), r4)

m5 = np.array([[0, 1, 0],
 [1, 1, 0],
 [0, 0, 0],
 [0, 1, 1]])
r5 = np.array([[0, 0, 1],
 [0, 1, 1],

```

(continues on next page)

(continued from previous page)

```
[0,0,0],
[1,0,1])
assert np.allclose(matrot_pro(m5), r5)
```

&lt;/div&gt;

[8]: #EFFICIENT SOLUTION

### Exercise - odd

⊕⊕⊕ Takes a Numpy matrix `mat` of dimension `nrows` by `ncols` containing integer numbers and RETURN a NEW Numpy matrix of dimension `nrows` by `ncols` which is like the original, ma in the cells which contained even numbers now there will be odd numbers obtained by summing 1 to the existing even number.

Example:

```
odd(np.array([
 [2,5,6,3],
 [8,4,3,5],
 [6,1,7,9]
]))
```

Must give as output

```
array([[3., 5., 7., 3.],
 [9., 5., 3., 5.],
 [7., 1., 7., 9.]])
```

Basic versions hints (slow):

- Since you need to return a matrix, start with creating an empty one
- go through the whole input matrix with indeces `i` and `j`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[9]: import numpy as np

```
def odd(mat):

 #SLOW SOLUTION
 nrows, ncols = mat.shape
 ret = np.zeros((nrows, ncols))

 for i in range(nrows):
 for j in range(ncols):
 if mat[i,j] % 2 == 0:
 ret[i,j] = mat[i,j] + 1
 else:
 ret[i,j] = mat[i,j]
 return ret
```

(continues on next page)

(continued from previous page)

```
m1 = np.array([[2]])
m2 = np.array([[3]])
assert np.allclose(odd(m1), m2)
assert m1[0][0] == 2 # checks we are not modifying original matrix

m3 = np.array([[2,5,6,3],
 [8,4,3,5],
 [6,1,7,9]])
m4 = np.array([[3,5,7,3],
 [9,5,3,5],
 [7,1,7,9]])
assert np.allclose(odd(m3), m4)
```

&lt;/div&gt;

```
[9]: import numpy as np

def odd(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[2]])
m2 = np.array([[3]])
assert np.allclose(odd(m1), m2)
assert m1[0][0] == 2 # checks we are not modifying original matrix

m3 = np.array([[2,5,6,3],
 [8,4,3,5],
 [6,1,7,9]])
m4 = np.array([[3,5,7,3],
 [9,5,3,5],
 [7,1,7,9]])
assert np.allclose(odd(m3), m4)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[10]: #EFFICIENT SOLUTION 1 with np.where

def odd_pro1(mat):
 ret = np.array(np.where(mat % 2 == 0, mat+1, mat))
 return ret

m1 = np.array([[2]])
m2 = np.array([[3]])
assert np.allclose(odd_pro1(m1), m2)
assert m1[0][0] == 2 # checks we are not modifying original matrix
```

(continues on next page)

(continued from previous page)

```
m3 = np.array([[2,5,6,3],
 [8,4,3,5],
 [6,1,7,9]])
m4 = np.array([[3,5,7,3],
 [9,5,3,5],
 [7,1,7,9]])
assert np.allclose(odd_pro1(m3), m4)
```

&lt;/div&gt;

[10]: #EFFICIENT SOLUTION 1 with np.where

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[11]: #EFFICIENT SOLUTION 2 without np.where

```
def odd_pro2(mat):
 ret = mat.copy()
 ret[ret % 2 == 0] += 1
 return ret

m1 = np.array([[2]])
m2 = np.array([[3]])
assert np.allclose(odd_pro2(m1), m2)
assert m1[0][0] == 2 # checks we are not modifying original matrix

m3 = np.array([[2,5,6,3],
 [8,4,3,5],
 [6,1,7,9]])
m4 = np.array([[3,5,7,3],
 [9,5,3,5],
 [7,1,7,9]])
assert np.allclose(odd_pro2(m3), m4)
```

&lt;/div&gt;

[11]: #EFFICIENT SOLUTION 2 without np.where

### Exercise - doublealt

⊕⊕⊕ Takes a Numpy matrix mat of dimensions nrows x ncols containing integer numbers and RETURN a NEW Numpy matrix of dimension nrows x ncols having at rows of even **index** the numbers of original matrix multiplied by two, and at rows of odd **index** the same numbers as the original matrix.

Example:

```
m = np.array([
 [2, 5, 6, 3], # index
 [8, 4, 3, 5], # 0 even
 [7, 1, 6, 9], # 1 odd
 [5, 2, 4, 1], # 2 even
 [6, 3, 4, 3] # 3 odd
 [6, 3, 4, 3] # 4 even
])
```

A call to

```
doublealt(m)
```

will return the Numpy matrix:

```
array([[4, 10, 12, 6],
 [8, 4, 3, 5],
 [14, 2, 12, 18],
 [5, 2, 4, 1],
 [12, 6, 8, 6]])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[12]: import numpy as np

def doublealt(mat):

 #SLOW SOLUTION
 nrows, ncols = mat.shape
 ret = np.zeros((nrows, ncols))

 for i in range(nrows):
 for j in range(ncols):
 if i % 2 == 0:
 ret[i,j] = mat[i,j] * 2
 else:
 ret[i,j] = mat[i,j]
 return ret

m1 = np.array([[2]])
m2 = np.array([[4]])
assert np.allclose(doublealt(m1), m2)
assert m1[0][0] == 2 # checks we are not modifying original matrix

m3 = np.array([[2, 5, 6],
 [8, 4, 3]])
m4 = np.array([[4,10,12],
```

(continues on next page)

(continued from previous page)

```

 [8, 4, 3]])
assert np.allclose(doublealt(m3), m4)

m5 = np.array([[2, 5, 6, 3],
 [8, 4, 3, 5],
 [7, 1, 6, 9],
 [5, 2, 4, 1],
 [6, 3, 4, 3]])
m6 = np.array([[4, 10, 12, 6],
 [8, 4, 3, 5],
 [14, 2, 12, 18],
 [5, 2, 4, 1],
 [12, 6, 8, 6]])
assert np.allclose(doublealt(m5), m6)

```

&lt;/div&gt;

```
[12]: import numpy as np

def doublealt(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[2]])
m2 = np.array([[4]])
assert np.allclose(doublealt(m1), m2)
assert m1[0][0] == 2 # checks we are not modifying original matrix

m3 = np.array([[2, 5, 6],
 [8, 4, 3]])
m4 = np.array([[4, 10, 12],
 [8, 4, 3]])
assert np.allclose(doublealt(m3), m4)

m5 = np.array([[2, 5, 6, 3],
 [8, 4, 3, 5],
 [7, 1, 6, 9],
 [5, 2, 4, 1],
 [6, 3, 4, 3]])
m6 = np.array([[4, 10, 12, 6],
 [8, 4, 3, 5],
 [14, 2, 12, 18],
 [5, 2, 4, 1],
 [12, 6, 8, 6]])
assert np.allclose(doublealt(m5), m6)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[13]: # EFFICIENT SOLUTION
```

```
def radalt_pro(mat):
 ret = mat.copy()
 ret[:,::2] *= 2
```

(continues on next page)

(continued from previous page)

```

return ret

m1 = np.array([[2]])
m2 = np.array([[4]])
assert np.allclose(doublealt(m1), m2)
assert m1[0][0] == 2 # checks we are not modifying original matrix

m3 = np.array([[2, 5, 6],
 [8, 4, 3]])
m4 = np.array([[4,10,12],
 [8, 4, 3]])
assert np.allclose(doublealt(m3), m4)

m5 = np.array([[2, 5, 6, 3],
 [8, 4, 3, 5],
 [7, 1, 6, 9],
 [5, 2, 4, 1],
 [6, 3, 4, 3]])
m6 = np.array([[4,10,12, 6],
 [8, 4, 3, 5],
 [14, 2,12,18],
 [5, 2, 4, 1],
 [12, 6, 8, 6]])
assert np.allclose(doublealt(m5), m6)

```

&lt;/div&gt;

[13]: # EFFICIENT SOLUTION

### Exercise - chessboard

⊗⊗⊗ RETURN a NEW Numpy matrix of n rows and n columns, in which all cells alternate zeros and ones.

For example, chessboard(4) must give:

```
array([[1.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 1.0],
 [1.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 1.0]])
```

Basic version ingredients (slow):

- to alternate, you can use `range` in the form in which takes 3 parameters, for example `range(0, n, 2)` starts from 0, arrives to n excluded by jumping one item at a time, generating 0,2,4,6,8, ....
- `range(1, n, 2)` would instead generate 1,3,5,7, ...

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[14]: def chessboard(n):

 #SLOW SOLUTION
 mat = np.zeros((n,n))

 for i in range(0,n, 2):
 for j in range(0,n, 2):
 mat[i, j] = 1

 for i in range(1,n, 2):
 for j in range(1,n, 2):
 mat[i, j] = 1

 return mat

r1 = np.array([[1.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 1.0],
 [1.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 1.0]])
assert np.allclose(chessboard(4), r1)

r2 = np.array([[1.0]])
assert np.allclose(chessboard(1), r2)

r3 = np.array([[1.0, 0.0],
 [0.0, 1.0]])
assert np.allclose(chessboard(2), r3)
```

&lt;/div&gt;

```
[14]: def chessboard(n):
 raise Exception('TODO IMPLEMENT ME !')

r1 = np.array([[1.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 1.0],
 [1.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 1.0]])
assert np.allclose(chessboard(4), r1)

r2 = np.array([[1.0]])
assert np.allclose(chessboard(1), r2)

r3 = np.array([[1.0, 0.0],
 [0.0, 1.0]])
assert np.allclose(chessboard(2), r3)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[15]: #FAST SOLUTION
```

```
def chessboard_pro(n):
 ret = np.zeros((n, n))
```

(continues on next page)

(continued from previous page)

```

ret[::2, ::2] = 1
ret[1::2, 1::2] = 1
return ret

r1 = np.array([[1.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 1.0],
 [1.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 1.0]])
assert np.allclose(chessboard_pro(4), r1)

r2 = np.array([[1.0]])
assert np.allclose(chessboard_pro(1), r2)

r3 = np.array([[1.0, 0.0],
 [0.0, 1.0]])
assert np.allclose(chessboard_pro(2), r3)

```

&lt;/div&gt;

[15]: #FAST SOLUTION

### Exercise - altsum

⊕⊕⊕ MODIFY the input Numpy matrix ( $n \times n$ ), by summing to all the odd rows the even rows. For example

```
m = [[1.0, 3.0, 2.0, 5.0],
 [2.0, 8.0, 5.0, 9.0],
 [6.0, 9.0, 7.0, 2.0],
 [4.0, 7.0, 2.0, 4.0]]
altsum(m)
```

after the call to altsum m should be:

```
m = [[1.0, 3.0, 2.0, 5.0],
 [3.0, 11.0, 7.0, 14.0],
 [6.0, 9.0, 7.0, 2.0],
 [10.0, 16.0, 9.0, 6.0]]
```

Basic version ingredients (slow):

- to alternate, you can use `range` in the form in which takes 3 parameters, for example `range(0, n, 2)` starts from 0, arrives to n excluded by jumping one item at a time, generating 0,2,4,6,8, ....
- instead `range(1, n, 2)` would generate 1,3,5,7, ..

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[16]: def altsum(mat):

```

#SLOW SOLUTION
nrows, ncols = mat.shape
for i in range(1,nrows, 2):
```

(continues on next page)

(continued from previous page)

```
for j in range(0,ncols):
 mat[i, j] = mat[i,j] + mat[i-1, j]

m1 = np.array([[1.0, 3.0, 2.0, 5.0],
 [2.0, 8.0, 5.0, 9.0],
 [6.0, 9.0, 7.0, 2.0],
 [4.0, 7.0, 2.0, 4.0]])

r1 = np.array([[1.0, 3.0, 2.0, 5.0],
 [3.0, 11.0, 7.0, 14.0],
 [6.0, 9.0, 7.0, 2.0],
 [10.0, 16.0, 9.0, 6.0]])

altsum(m1)
assert np.allclose(m1, r1) # checks we MODIFIED the original matrix

m2 = np.array([[5.0]])
r2 = np.array([[5.0]])
altsum(m1)
assert np.allclose(m2, r2)

m3 = np.array([[6.0, 1.0],
 [3.0, 2.0]])
r3 = np.array([[6.0, 1.0],
 [9.0, 3.0]])
altsum(m3)
assert np.allclose(m3, r3)
```

&lt;/div&gt;

```
[16]: def altsum(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[1.0, 3.0, 2.0, 5.0],
 [2.0, 8.0, 5.0, 9.0],
 [6.0, 9.0, 7.0, 2.0],
 [4.0, 7.0, 2.0, 4.0]])

r1 = np.array([[1.0, 3.0, 2.0, 5.0],
 [3.0, 11.0, 7.0, 14.0],
 [6.0, 9.0, 7.0, 2.0],
 [10.0, 16.0, 9.0, 6.0]])

altsum(m1)
assert np.allclose(m1, r1) # checks we MODIFIED the original matrix

m2 = np.array([[5.0]])
r2 = np.array([[5.0]])
altsum(m1)
assert np.allclose(m2, r2)

m3 = np.array([[6.0, 1.0],
 [3.0, 2.0]])
r3 = np.array([[6.0, 1.0],
```

(continues on next page)

(continued from previous page)

```
[9.0, 3.0]])
altsum(m3)
assert np.allclose(m3, r3)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[17]: #EFFICIENT SOLUTION

```
def altsum_pro(mat):
 mat[1::2] += mat[::2]
 return mat

m1 = np.array([[1.0, 3.0, 2.0, 5.0],
 [2.0, 8.0, 5.0, 9.0],
 [6.0, 9.0, 7.0, 2.0],
 [4.0, 7.0, 2.0, 4.0]])

r1 = np.array([[1.0, 3.0, 2.0, 5.0],
 [3.0, 11.0, 7.0, 14.0],
 [6.0, 9.0, 7.0, 2.0],
 [10.0, 16.0, 9.0, 6.0]])

altsum_pro(m1)
assert np.allclose(m1, r1) # checks we MODIFIED the original matrix

m2 = np.array([[5.0]])
r2 = np.array([[5.0]])
altsum_pro(m1)
assert np.allclose(m2, r2)

m3 = np.array([[6.0, 1.0],
 [3.0, 2.0]])
r3 = np.array([[6.0, 1.0],
 [9.0, 3.0]])
altsum_pro(m3)
assert np.allclose(m3, r3)
```

</div>

[17]: #EFFICIENT SOLUTION

### Exercise - avg\_half

⊕⊕⊕ Takes as input a Numpy matrix with an even number of columns, and RETURN as output a Numpy matrix 1x2, in which the first element will be the average of the left half of the matrix, and the second element will be the average of the right half.

Ingredients:

- to obtain the number of columns divided by two as integer number, use // operator

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: def avg_half(mat):

 nrows, ncols = mat.shape
 half_cols = ncols // 2

 avg_sx = 0.0
 avg_dx = 0.0

 for i in range(nrows):
 for j in range(half_cols):
 avg_sx += mat[i,j]
 for j in range(half_cols, ncols):
 avg_dx += mat[i,j]

 half_elements = nrows * half_cols
 avg_sx /= half_elements
 avg_dx /= half_elements
 return np.array([avg_sx, avg_dx])

m1 = np.array([[7,9]])
r1 = np.array([(7)/1, (9)/1])
assert np.allclose(avg_half(m1), r1)

m2 = np.array([
 [3,4],
 [6,3],
 [5,2]])
r2 = np.array([(3+6+5)/3, (4+3+2)/3])
assert np.allclose(avg_half(m2), r2)

m3 = np.array([
 [3,2,1,4],
 [6,2,3,5],
 [4,3,6,2],
 [4,6,5,4],
 [7,2,9,3]])
r3 = np.array([(3+2+6+2+4+3+4+6+7+2)/10, (1+4+3+5+6+2+5+4+9+3)/10])
assert np.allclose(avg_half(m3), r3)
```

</div>

```
[18]: def avg_half(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[7,9]])

r1 = np.array([(7)/1, (9)/1])
assert np.allclose(avg_half(m1), r1)

m2 = np.array([[3,4],
 [6,3],
 [5,2]])

r2 = np.array([(3+6+5)/3, (4+3+2)/3])
assert np.allclose(avg_half(m2), r2)

m3 = np.array([[3,2,1,4],
 [6,2,3,5],
 [4,3,6,2],
 [4,6,5,4],
 [7,2,9,3]])

r3 = np.array([(3+2+6+2+4+3+4+6+7+2)/10, (1+4+3+5+6+2+5+4+9+3)/10])

assert np.allclose(avg_half(m3), r3)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[19]: #EFFICIENT SOLUTION

def avg_half_pro(mat):
 n,m = mat.shape

 m2 = m // 2
 half_els = n * m2

 avg=np.zeros((1,2))

 avg[0,0]= np.sum(mat[:, :m2])/half_els
 avg[0,1]= np.sum(mat[:, m2:])/half_els

 return avg

m1 = np.array([[7,9]])
r1 = np.array([(7)/1, (9)/1])
assert np.allclose(avg_half_pro(m1), r1)

m2 = np.array([[3,4],
 [6,3],
 [5,2]])
r2 = np.array([(3+6+5)/3, (4+3+2)/3])
assert np.allclose(avg_half_pro(m2), r2)

m3 = np.array([[3,2,1,4],
 [6,2,3,5],
```

(continues on next page)

(continued from previous page)

```
[4,3,6,2],
[4,6,5,4],
[7,2,9,3]])
r3 = np.array([(3+2+6+2+4+3+4+6+7+2)/10, (1+4+3+5+6+2+5+4+9+3)/10])
assert np.allclose(avg_half_pro(m3), r3)
```

&lt;/div&gt;

[19]: #EFFICIENT SOLUTION

### Exercise - matxarr

⊕⊕ Takes a Numpy matrix  $n \times m$  and an ndarray of  $m$  elements, and RETURN a NEW Numpy matrix in which the values of each column of input matrix are multiplied by the corresponding value in the  $n$  elements array.

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[20]:

```
def matxarr(mat, arr):

 #SLOW SOLUTION

 ret = np.zeros(mat.shape)

 for i in range(mat.shape[0]):
 for j in range(mat.shape[1]):
 ret[i,j] = mat[i,j] * arr[j]

 return ret

m1 = np.array([[3,2,1],
 [6,2,3],
 [4,3,6],
 [4,6,5]])
a1 = [5, 2, 6]
r1 = [[3*5, 2*2, 1*6],
 [6*5, 2*2, 3*6],
 [4*5, 3*2, 6*6],
 [4*5, 6*2, 5*6]]

assert np.allclose(matxarr(m1,a1), r1)
```

&lt;/div&gt;

[20]:

```
def matxarr(mat, arr):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[3,2,1],
 [6,2,3],
```

(continues on next page)

(continued from previous page)

```
[4,3,6],
[4,6,5])
a1 = [5, 2, 6]
r1 = [[3*5, 2*2, 1*6],
 [6*5, 2*2, 3*6],
 [4*5, 3*2, 6*6],
 [4*5, 6*2, 5*6]]

assert np.allclose(matxarr(m1,a1), r1)
```

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol" jupman-sol-code" style="display:none">

[21]: #EFFICIENT SOLUTION

```
def matxarr_pro(mat, arr):
 return np.array(arr) * mat

m1 = np.array([[3,2,1],
 [6,2,3],
 [4,3,6],
 [4,6,5]])
a1 = [5, 2, 6]
r1 = [[3*5, 2*2, 1*6],
 [6*5, 2*2, 3*6],
 [4*5, 3*2, 6*6],
 [4*5, 6*2, 5*6]]

assert np.allclose(matxarr_pro(m1,a1), r1)
```

</div>

[21]: #EFFICIENT SOLUTION

## Exercise - colgap

⊗⊗ Given a numpy matrix of  $n$  rows and  $m$  columns, RETURN a numpy vector of  $m$  elements consisting in the difference between the maximum and minimum values of each column.

Example:

```
m = np.array([[5,4,2],
 [8,5,1],
 [6,7,9],
 [3,6,4],
 [4,3,7]])
>>> colgap(m)
array([5, 4, 8])
```

because:

```
5 = 8 - 3
4 = 7 - 3
8 = 9 - 1
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[22]: import numpy as np

def colgap(mat):

 #SLOW SOLUTION
 mx = mat[0].copy()
 mn = mat[0].copy()
 for i in range(mat.shape[0]):
 for j in range(mat.shape[1]):
 if mat[i,j] > mx[j]:
 mx[j] = mat[i,j]
 if mat[i,j] < mn[j]:
 mn[j] = mat[i,j]
 return mx - mn

TEST
m1 = np.array([[6]])
assert np.allclose(colgap(m1), np.array([0]))
ret = colgap(m1)
assert type(ret) == np.ndarray

m2 = np.array([[6,8]])
assert np.allclose(colgap(m2), np.array([0,0]))
m3 = np.array([[2],
 [5]])

assert np.allclose(colgap(m3), np.array([3]))
m4 = np.array([[5,7],
 [2,9]])
assert np.allclose(colgap(m4), np.array([3,2]))
m5 = np.array([[4,7],
 [4,9]])
assert np.allclose(colgap(m5), np.array([0,2]))
m6 = np.array([[5,2],
 [3,7],
 [9,0]])
assert np.allclose(colgap(m6), np.array([6,7]))
m7 = np.array([[5,4,2],
 [8,5,1],
 [6,7,9],
 [3,6,4],
 [4,3,7]])
assert np.allclose(colgap(m7), np.array([5,4,8]))
```

</div>

```
[22]: import numpy as np

def colgap(mat):
 raise Exception('TODO IMPLEMENT ME !')

TEST
m1 = np.array([[6]])
assert np.allclose(colgap(m1), np.array([0]))
```

(continues on next page)

(continued from previous page)

```

ret = colgap(m1)
assert type(ret) == np.ndarray

m2 = np.array([[6,8]])
assert np.allclose(colgap(m2), np.array([0,0]))
m3 = np.array([[2],
 [5]])

assert np.allclose(colgap(m3), np.array([3]))
m4 = np.array([[5,7],
 [2,9]])
assert np.allclose(colgap(m4), np.array([3,2]))
m5 = np.array([[4,7],
 [4,9]])
assert np.allclose(colgap(m5), np.array([0,2]))
m6 = np.array([[5,2],
 [3,7],
 [9,0]])
assert np.allclose(colgap(m6), np.array([6,7]))
m7 = np.array([[5,4,2],
 [8,5,1],
 [6,7,9],
 [3,6,4],
 [4,3,7]])
assert np.allclose(colgap(m7), np.array([5,4,8]))

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[23]: #EFFICIENT SOLUTION

```

def colgap_pro(mat):
 mx = np.max(mat, axis=0)
 mn = np.min(mat, axis=0)
 return mx - mn

TEST
m1 = np.array([[6]])
assert np.allclose(colgap_pro(m1), np.array([0]))
ret = colgap_pro(m1)
assert type(ret) == np.ndarray

m2 = np.array([[6,8]])
assert np.allclose(colgap_pro(m2), np.array([0,0]))
m3 = np.array([[2],
 [5]])
assert np.allclose(colgap_pro(m3), np.array([3]))
m4 = np.array([[5,7],
 [2,9]])
assert np.allclose(colgap_pro(m4), np.array([3,2]))
m5 = np.array([[4,7],
 [4,9]])
assert np.allclose(colgap_pro(m5), np.array([0,2]))
m6 = np.array([[5,2],
 [3,7],
 [9,0]])

```

(continues on next page)

(continued from previous page)

```
assert np.allclose(colgap_pro(m6), np.array([6,7]))
m7 = np.array([[5,4,2],
 [8,5,1],
 [6,7,9],
 [3,6,4],
 [4,3,7]])
assert np.allclose(colgap_pro(m7), np.array([5,4,8]))
```

&lt;/div&gt;

[23]: #EFFICIENT SOLUTION

### Exercise - substmax

⊕⊕ Given an  $n \times m$  numpy matrix mat, MODIFY the matrix substituting each cell with the maximum value found in the corresponding column.

Example:

```
>>> m = np.array([[5,4,2],
 [8,5,1],
 [6,7,9],
 [3,6,4],
 [4,3,7]])
>>> substmax(m) # returns nothing!
>>> m
np.array([[8, 7, 9],
 [8, 7, 9],
 [8, 7, 9],
 [8, 7, 9],
 [8, 7, 9]])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[24]:

```
import numpy as np

def substmax(mat):

 #SLOW SOLUTION
 mx = mat[0].copy()
 for i in range(mat.shape[0]):
 for j in range(mat.shape[1]):
 if mat[i,j] > mx[j]:
 mx[j] = mat[i,j]
 for i in range(mat.shape[0]):
 for j in range(mat.shape[1]):
 mat[i,j] = mx[j]

 # TEST
 m1 = np.array([[6]])
```

(continues on next page)

(continued from previous page)

```

substmax(m1)
assert np.allclose(m1, np.array([6]))
ret = substmax(m1)
assert ret == None # returns nothing!

m2 = np.array([[6,8]])
substmax(m2)
assert np.allclose(m2, np.array([6,8]))

m3 = np.array([[2],
 [5]])
substmax(m3)
assert np.allclose(m3, np.array([[5],
 [5]]))

m4 = np.array([[5,7],
 [2,9]])
substmax(m4)

assert np.allclose(m4, np.array([[5,9],
 [5,9]]))

m5 = np.array([[4,7],
 [4,9]])
substmax(m5)
assert np.allclose(m5, np.array([[4,9],
 [4,9]]))

m6 = np.array([[5,2],
 [3,7],
 [9,0]])
substmax(m6)
assert np.allclose(m6, np.array([[9,7],
 [9,7],
 [9,7]]))

m7 = np.array([[5,4,2],
 [8,5,1],
 [6,7,9],
 [3,6,4],
 [4,3,7]])
substmax(m7)
assert np.allclose(m7, np.array([[8, 7, 9],
 [8, 7, 9],
 [8, 7, 9],
 [8, 7, 9],
 [8, 7, 9]]))

```

&lt;/div&gt;

```
[24]:

import numpy as np

def substmax(mat):

 raise Exception('TODO IMPLEMENT ME !')

TEST
```

(continues on next page)

(continued from previous page)

```
m1 = np.array([[6]])
substmax(m1)
assert np.allclose(m1, np.array([6]))
ret = substmax(m1)
assert ret == None # returns nothing!

m2 = np.array([[6,8]])
substmax(m2)
assert np.allclose(m2, np.array([6,8]))

m3 = np.array([[2],
 [5]])
substmax(m3)
assert np.allclose(m3, np.array([[5],
 [5]]))

m4 = np.array([[5,7],
 [2,9]])
substmax(m4)

assert np.allclose(m4, np.array([[5,9],
 [5,9]]))

m5 = np.array([[4,7],
 [4,9]])
substmax(m5)
assert np.allclose(m5, np.array([[4,9],
 [4,9]]))

m6 = np.array([[5,2],
 [3,7],
 [9,0]])
substmax(m6)
assert np.allclose(m6, np.array([[9,7],
 [9,7],
 [9,7]]))

m7 = np.array([[5,4,2],
 [8,5,1],
 [6,7,9],
 [3,6,4],
 [4,3,7]])
substmax(m7)
assert np.allclose(m7, np.array([[8, 7, 9],
 [8, 7, 9],
 [8, 7, 9],
 [8, 7, 9],
 [8, 7, 9]]))
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[25] : #EFFICIENT SOLUTION

```
def substmax_pro(mat):
```

(continues on next page)

(continued from previous page)

```

mat[:, :] = np.max(mat, axis=0)

TEST
m1 = np.array([[6]])
substmax_pro(m1)
assert np.allclose(m1, np.array([6]))
ret = substmax_pro(m1)
assert ret == None # non ritorna nulla!

m2 = np.array([[6, 8]])
substmax_pro(m2)
assert np.allclose(m2, np.array([6, 8]))

m3 = np.array([[2],
 [5]])
substmax_pro(m3)
assert np.allclose(m3, np.array([[5],
 [5]]))

m4 = np.array([[5, 7],
 [2, 9]])
substmax_pro(m4)

assert np.allclose(m4, np.array([[5, 9],
 [5, 9]]))

m5 = np.array([[4, 7],
 [4, 9]])
substmax_pro(m5)
assert np.allclose(m5, np.array([[4, 9],
 [4, 9]]))

m6 = np.array([[5, 2],
 [3, 7],
 [9, 0]])
substmax_pro(m6)
assert np.allclose(m6, np.array([[9, 7],
 [9, 7],
 [9, 7]]))

m7 = np.array([[5, 4, 2],
 [8, 5, 1],
 [6, 7, 9],
 [3, 6, 4],
 [4, 3, 7]])
substmax_pro(m7)
assert np.allclose(m7, np.array([[8, 7, 9],
 [8, 7, 9],
 [8, 7, 9],
 [8, 7, 9],
 [8, 7, 9]])))

```

&lt;/div&gt;

[25]: #EFFICIENT SOLUTION

### Exercise - quadrants

⊕⊕⊕ Given a matrix  $2n \times 2n$ , divide the matrix in 4 equal square parts (see example) and RETURN a NEW matrix  $2 \times 2$  containing the average of each quadrant.

We assume the matrix is always of even dimensions

HINT: to divide by two and obtain an integer number, use `//` operator

Example:

```
1, 2 , 5 , 7
4, 1 , 8 , 0
2, 0 , 5 , 1
0, 2 , 1 , 1
```

can be divided in

```
1, 2 | 5 , 7
4, 1 | 8 , 0

2, 0 | 5 , 1
0, 2 | 1 , 1
```

and returns

$(1+2+4+1) / 4$	$ $	$(5+7+8+0) / 4$	$=>$	2.0 , 5.0
<hr/>				1.0 , 2.0
$(2+0+0+2) / 4$	$ $	$(5+1+1+1) / 4$		

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[26]:

```
import numpy as np

def quadrants(mat):

 #SLOW SOLUTION
 ret = np.zeros((2,2))

 dim = mat.shape[0]
 n = dim // 2
 elements_per_quad = n * n

 for i in range(n):
 for j in range(n):
 ret[0,0] += mat[i,j]
 ret[0,0] /= elements_per_quad

 for i in range(n,dim):
 for j in range(n):
 ret[1,0] += mat[i,j]
 ret[1,0] /= elements_per_quad

 for i in range(n,dim):
 for j in range(n,dim):
 ret[1,1] += mat[i,j]
 ret[1,1] /= elements_per_quad
```

(continues on next page)

(continued from previous page)

```

 ret[1,1] += mat[i,j]
 ret[1,1] /= elements_per_quad

 for i in range(n):
 for j in range(n,dim):
 ret[0,1] += mat[i,j]
 ret[0,1] /= elements_per_quad

 return ret

m1 = np.array([[3.0, 5.0],
 [4.0, 9.0]])
r1 = np.array([[3.0, 5.0],
 [4.0, 9.0],
])
assert np.allclose(quadrants(m1),r1)

m2 = np.array([[1.0, 2.0 , 5.0 , 7.0],
 [4.0, 1.0 , 8.0 , 0.0],
 [2.0, 0.0 , 5.0 , 1.0],
 [0.0, 2.0 , 1.0 , 1.0]])
r2 = np.array([[2.0, 5.0],
 [1.0, 2.0]])
assert np.allclose(quadrants(m2),r2)

```

&lt;/div&gt;

[26]:

```

import numpy as np

def quadrants(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[3.0, 5.0],
 [4.0, 9.0]])
r1 = np.array([[3.0, 5.0],
 [4.0, 9.0],
])
assert np.allclose(quadrants(m1),r1)

m2 = np.array([[1.0, 2.0 , 5.0 , 7.0],
 [4.0, 1.0 , 8.0 , 0.0],
 [2.0, 0.0 , 5.0 , 1.0],
 [0.0, 2.0 , 1.0 , 1.0]])
r2 = np.array([[2.0, 5.0],
 [1.0, 2.0]])
assert np.allclose(quadrants(m2),r2)

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[27]: #EFFICIENT SOLUTION

(continues on next page)

(continued from previous page)

```

def quadrants_pro(matrice):
 m = matrice.shape[0]

 ret = np.zeros((2, 2))

 n = m // 2

 qarea = n * n

 ret[0, 0] = np.sum(matrice[:n, :n]) / qarea
 ret[0, 1] = np.sum(matrice[:n, n:]) / qarea
 ret[1, 0] = np.sum(matrice[n:, :n]) / qarea
 ret[1, 1] = np.sum(matrice[n:, n:]) / qarea

 return ret

m1 = np.array([[3.0, 5.0],
 [4.0, 9.0]])
r1 = np.array([[3.0, 5.0],
 [4.0, 9.0],
 []])
assert np.allclose(quadrants_pro(m1), r1)

m2 = np.array([[1.0, 2.0, 5.0, 7.0],
 [4.0, 1.0, 8.0, 0.0],
 [2.0, 0.0, 5.0, 1.0],
 [0.0, 2.0, 1.0, 1.0]])
r2 = np.array([[2.0, 5.0],
 [1.0, 2.0]])
assert np.allclose(quadrants_pro(m2), r2)

```

&lt;/div&gt;

[27]: #EFFICIENT SOLUTION

### Exercise - downup

⊗⊗⊗ Write a function which given the dimensions of n rows and m columns, RETURN a NEW n x m numpy matrix with sequences which go down and up in alternating rows as in the examples.

- if m is odd, raises ValueError

```

>>> downup(6,10)
array([[0., 0., 0., 0., 4., 3., 2., 1., 0.],
 [0., 1., 2., 3., 4., 0., 0., 0., 0.],
 [0., 0., 0., 0., 4., 3., 2., 1., 0.],
 [0., 1., 2., 3., 4., 0., 0., 0., 0.],
 [0., 0., 0., 0., 4., 3., 2., 1., 0.],
 [0., 1., 2., 3., 4., 0., 0., 0., 0.]])

```

Show solution  
Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[28]: import numpy as np

def downup(n,m):

 #SLOW SOLUTION

 if m%2 == 1:
 raise ValueError("m deve essere pari, trovato %s" % m)
 mat = np.zeros((n,m))
 for i in range(0,n,2):
 for j in range(m//2):
 mat[i,j+m//2] = m//2 - j - 1
 for i in range(1,n,2):
 for j in range(m//2):
 mat[i,j] = j
 return mat

assert np.allclose(downup(2,2), np.array([[0., 0.], [0., 0.]]))
assert type(downup(2,2)) == np.ndarray

assert np.allclose(downup(2,6), np.array([[0., 0., 0., 2., 1., 0.], [0., 1., 2., 0., 0., 0.]]))

assert np.allclose(downup(6,10), np.array([[0., 0., 0., 0., 0., 4., 3., 2., 1., 0.], [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 4., 3., 2., 1., 0.], [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 4., 3., 2., 1., 0.], [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.]]))
try:
 downup(2,3)
 raise Exception("I should have failed!")
except ValueError:
 pass

</div>

[28]: import numpy as np

def downup(n,m):
 raise Exception('TODO IMPLEMENT ME !')

assert np.allclose(downup(2,2), np.array([[0., 0.], [0., 0.]]))
assert type(downup(2,2)) == np.ndarray

assert np.allclose(downup(2,6), np.array([[0., 0., 0., 2., 1., 0.], [0., 1., 2., 0., 0., 0.]]))

assert np.allclose(downup(6,10), np.array([[0., 0., 0., 0., 0., 4., 3., 2., 1., 0.], [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 4., 3., 2., 1., 0.], [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.], [0., 0., 0., 0., 0., 4., 3., 2., 1., 0.], [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.]]))

```

(continues on next page)

(continued from previous page)

```
[0., 0., 0., 0., 0., 4., 3., 2., 1., 0.],
[0., 1., 2., 3., 4., 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 4., 3., 2., 1., 0.],
[0., 1., 2., 3., 4., 0., 0., 0., 0., 0.] ↵
) ↵
try:
 downup(2,3)
 raise Exception("I should have failed!")
except ValueError:
 pass
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"; data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[29]: #EFFICIENT SOLUTION (HINT: use np.tile)

```
import numpy as np

def downup_pro(n,m):

 if m%2 == 1:
 raise ValueError("m must be even, found %s" % m)

 ret = np.zeros((n,m))

 left = np.tile(np.arange(0.0,m/2,1.0),(n//2,1))
 right = np.tile(np.arange(m/2 - 1,-0.5,-1.0),(n//2,1))
 ret[1::2,:m//2] = left
 ret[0::2,m//2:] = right
 return ret

assert np.allclose(downup_pro(2,2), np.array([
 [0., 0.],
 [0., 0.]]))
assert type(downup_pro(2,2)) == np.ndarray

assert np.allclose(downup_pro(2,6), np.array([
 [0., 0., 0., 2., 1., 0.],
 [0., 1., 2., 0., 0., 0.]]))

assert np.allclose(downup_pro(6,10), np.array([
 [0., 0., 0., 0., 0., 4., 3., 2., 1., 0.],
 [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 4., 3., 2., 1., 0.],
 [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0., 4., 3., 2., 1., 0.],
 [0., 1., 2., 3., 4., 0., 0., 0., 0., 0.]]))

try:
 downup_pro(2,3)
 raise Exception("I should have failed!")
```

(continues on next page)

(continued from previous page)

```
except ValueError:
 pass
```

&lt;/div&gt;

[29]: #EFFICIENT SOLUTION (HINT: use np.tile)

### Exercise - stairsteps

⊕⊕⊕ Given a numpy square matrix `mat` of dimension `n`, RETURN a NEW numpy array containing the values retrieved from the matrix in the followin order:

```
1,2,*,*,*
,3,4,,*
,,5,6,*
,,*,7,8
,,*,*,9
```

- if the matrix is not square, raises `ValueError`
- **DO NOT** use python lists!
- **HINT:** how many elements must the array to return have?

Example:

```
>>> stairsteps(np.array([
 [6,3,5,2,5],
 [3,4,2,3,4],
 [6,5,4,5,1],
 [4,3,2,3,9],
 [2,5,1,6,7]]))
array([6., 3., 4., 2., 4., 5., 3., 9., 7.])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[30]:

```
import numpy as np

def stairsteps(mat):

 #SLOW SOLUTION

 n,m = mat.shape
 if n != m:
 raise ValueError("Required a square matrix, found instead: %s x %s" % (n,m))

 res = np.zeros(n + n - 1)

 for i in range(n):
 res[2*i] = mat[i,i]

 for i in range(n-1):
 res[2*i+1] = mat[i,i+1]
```

(continues on next page)

(continued from previous page)

```

 return res

m1 = np.array([[7]])
assert np.allclose(stairsteps(m1), np.array([7]))
assert type(m1) == np.ndarray

m2 = np.array([
 [6,8],
 [9,3]])
assert np.allclose(stairsteps(m2), np.array([6,8,3]))
assert type(m1) == np.ndarray

m3 = np.array([
 [6,3,5,2,5],
 [3,4,2,3,4],
 [6,5,4,5,1],
 [4,3,2,3,9],
 [2,5,1,6,7]])

assert np.allclose(stairsteps(m3), np.array([6,3,4,2,4,5,3,9,7]))

try:
 stairsteps(np.array([[1,2,3],
 [4,5,6]]))
 raise Exception("I should have failed!")
except ValueError:
 pass

```

&lt;/div&gt;

```
[30]: import numpy as np

def stairsteps(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[7]])
assert np.allclose(stairsteps(m1), np.array([7]))
assert type(m1) == np.ndarray

m2 = np.array([
 [6,8],
 [9,3]])
assert np.allclose(stairsteps(m2), np.array([6,8,3]))
assert type(m1) == np.ndarray

m3 = np.array([
 [6,3,5,2,5],
 [3,4,2,3,4],
 [6,5,4,5,1],
 [4,3,2,3,9],
 [2,5,1,6,7]])

assert np.allclose(stairsteps(m3), np.array([6,3,4,2,4,5,3,9,7]))

try:
 stairsteps(np.array([[1,2,3],
 [4,5,6]]))
 raise Exception("I should have failed!")

```

(continues on next page)

(continued from previous page)

```
except ValueError:
 pass
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[31]: #EFFICIENT SOLUTION

```
import numpy as np
def gradini_pro(mat):

 n,m = mat.shape
 if n != m:
 raise ValueError("Richiesta una n x n, trovata invece una %s x %s" % (n,m))
 a = np.diag(mat)
 b = np.diag(mat, 1)
 ret = np.zeros((1, a.shape[0] + b.shape[0]))
 ret[:, ::2] = a
 ret[:, 1::2] = b
 return ret

m1 = np.array([[7]])
assert np.allclose(gradini_pro(m1), np.array([7]))
assert type(m1) == np.ndarray

m2 = np.array([[6,8],
 [9,3]])
assert np.allclose(gradini_pro(m2), np.array([6,8,3]))

m3 = np.array([[6,3,5,2,5],
 [3,4,2,3,4],
 [6,5,4,5,1],
 [4,3,2,3,9],
 [2,5,1,6,7]])

assert np.allclose(gradini_pro(m3), np.array([6,3,4,2,4,5,3,9,7]))
```

</div>

[31]: #EFFICIENT SOLUTION

## Exercise - vertstairs

⊕⊕⊕ Given a numbers of rows  $n$  and of columns  $m$ , RETURN a NEW  $n \times m$  numpy matrix having the numbers in even columns progressively increasing from 1 to  $n$ , and numbers in odd columns progressively decreasing from  $n$  to 1.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[32]: import numpy as np

def vertstairs(n,m):

(continues on next page)

(continued from previous page)

```
#SLOW SOLUTION
ret = np.zeros((n,m))
for i in range(n):
 for j in range(m):
 if j % 2 == 0:
 ret[i,j] = i + 1
 else:
 ret[i,j] = n - i
return ret

assert np.allclose(vertstairs(1,1), np.array([[1]]))
assert np.allclose(vertstairs(1,2), np.array([[1,1]]))
assert np.allclose(vertstairs(2,1), np.array([
 [1],
 [2]]))
assert np.allclose(vertstairs(2,2), np.array([
 [1,2],
 [2,1]]))
assert type(vertstairs(2,2)) == np.ndarray
assert np.allclose(vertstairs(4,5), np.array([
 [1,4,1,4,1],
 [2,3,2,3,2],
 [3,2,3,2,3],
 [4,1,4,1,4]]))
```

&lt;/div&gt;

[32]:

```
import numpy as np

def vertstairs(n,m):
 raise Exception('TODO IMPLEMENT ME !')

assert np.allclose(vertstairs(1,1), np.array([[1]]))
assert np.allclose(vertstairs(1,2), np.array([[1,1]]))
assert np.allclose(vertstairs(2,1), np.array([
 [1],
 [2]]))
assert np.allclose(vertstairs(2,2), np.array([
 [1,2],
 [2,1]]))
assert type(vertstairs(2,2)) == np.ndarray
assert np.allclose(vertstairs(4,5), np.array([
 [1,4,1,4,1],
 [2,3,2,3,2],
 [3,2,3,2,3],
 [4,1,4,1,4]]))
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[33]: #EFFICIENT SOLUTION (HINT: use np.tile)

```
def vertstairs_pro(n,m):
 ret = np.zeros((n,m))
 ret[:,0::2] = np.tile(np.transpose([np.arange(1,n+1,1)]), (m+1) // 2)
 ret[:,1::2] = np.tile(np.transpose([np.arange(n,0,-1)]), m // 2)
```

(continues on next page)

(continued from previous page)

```

 return ret

assert np.allclose(vertstairs_pro(1,1), np.array([[1]]))
assert np.allclose(vertstairs_pro(1,2), np.array([[1,1]]))
assert np.allclose(vertstairs_pro(2,1), np.array([[1],
 [2]]))
assert np.allclose(vertstairs_pro(2,2), np.array([[1,2],
 [2,1]]))

assert type(vertstairs_pro(2,2)) == np.ndarray
assert np.allclose(vertstairs_pro(4,5), np.array([[1,4,1,4,1],
 [2,3,2,3,2],
 [3,2,3,2,3],
 [4,1,4,1,4]]))

```

&lt;/div&gt;

[33]: #EFFICIENT SOLUTION (HINT: use np.tile)

## Exercise - comprescol

⊕⊕⊕ Given an  $n \times 2m$  matrix mat with an even number of columns, RETURN a NEW  $n \times m$  matrix in which the columns are given by the sum of corresponding column pairs from mat

- if mat doesn't have an even number of columns, raise ValueError

Example:

```

>>> m = np.array([[5,4,2,6,4,2],
 [7,5,1,0,6,1],
 [6,7,9,2,3,7],
 [5,2,4,6,1,3],
 [7,2,3,4,2,5]])

>>> comprescol(m)
np.array([[9, 8, 6],
 [12, 1, 7],
 [13, 11, 10],
 [7, 10, 4],
 [9, 7, 7]])

```

because

```

9 = 5 + 4 8 = 2 + 6 6 = 4 + 2
12= 7 + 5 1 = 1 + 0 7 = 6 + 1
. . .

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[34]: import numpy as np

```

def comprescol(mat):

```

(continues on next page)

(continued from previous page)

```
#EFFICIENT SOLUTION
if mat.shape[1] % 2 != 0:
 raise ValueError("Expected matrix with an even number of columns, got instead
→%s" % mat.shape[1])
n,m = mat.shape[0], mat.shape[1] // 2
ret = mat[:,::2].copy()
ret += mat[:,1::2]
return ret

m1 = [[7,9]]
res = comprescol(np.array(m1))
assert type(res) == np.ndarray
assert np.allclose(res, np.array([[16]]))

m2 = np.array([[5,8],
 [7,2]])
assert np.allclose(comprescol(m2), np.array([[13],
 [9]]))
assert np.allclose(m2, np.array([[5,8],
 [7,2]])) # check doesn't MODIFY original matrix

m3 = np.array([[5,4,2,6,4,2],
 [7,5,1,0,6,1],
 [6,7,9,2,3,7],
 [5,2,4,6,1,3],
 [7,2,3,4,2,5]])

assert np.allclose(comprescol(m3), np.array([[9, 8, 6],
 [12, 1, 7],
 [13,11,10],
 [7,10, 4],
 [9, 7, 7]]))

try:
 comprescol(np.array([[7,1,6],
 [5,2,4]]))
 raise Exception("I should have failed!")
except ValueError:
 pass
```

&lt;/div&gt;

```
[34]: import numpy as np

def comprescol(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [[7,9]]
res = comprescol(np.array(m1))
assert type(res) == np.ndarray
assert np.allclose(res, np.array([[16]]))

m2 = np.array([[5,8],
 [7,2]])
```

(continues on next page)

(continued from previous page)

```

assert np.allclose(comprescol(m2), np.array([[13], [9]]))
assert np.allclose(m2, np.array([[5,8], [7,2]])) # check doesn't MODIFY original matrix

m3 = np.array([[5,4,2,6,4,2],
 [7,5,1,0,6,1],
 [6,7,9,2,3,7],
 [5,2,4,6,1,3],
 [7,2,3,4,2,5]])

assert np.allclose(comprescol(m3), np.array([[9, 8, 6],
 [12, 1, 7],
 [13,11,10],
 [7,10, 4],
 [9, 7, 7]]))

try:
 comprescol(np.array([[7,1,6],
 [5,2,4]]))
 raise Exception("I should have failed!")
except ValueError:
 pass

```

## Exercise - revtriang

⊕⊕⊕ Given a square numpy matrix, RETURN a NEW numpy matrix having the same dimensions as the original one, and the numbers in the lower triangular part (excluding the diagonal) in reverse.

- if the matrix is not square, raise ValueError

Example:

```

m = np.array([[5, 4, 2, 6, 4],
 [3, 5, 1, 0, 6],
 [6, 4, 9, 2, 3],
 [5, 2, 8, 6, 1],
 [7, 9, 3, 2, 2]])

>>> revtriang(m)
np.array([[5, 4, 2, 6, 4],
 [3, 5, 1, 0, 6], # 3 -> 3
 [4, 6, 9, 2, 3], # 6,4 -> 4,6
 [8, 2, 5, 6, 1], # 5,2,8 -> 8,2,5
 [2, 3, 9, 7, 2]]) # 7,9,3,2 -> 2,3,9,7

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"  
data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[35]: import numpy as np

def revtriang(mat):
```

```
 #EFFICIENT SOLUTION
```

(continues on next page)

(continued from previous page)

```

n,m = mat.shape
if n != m:
 raise ValueError("Expected square matrix, got instead n=%s, m=%s" % (n,m))

ret = mat.copy()

for i in range(1,n):
 ret[i,:i] = np.flip(mat[i,:i])
return ret

m1 = np.array([[8]])
assert np.allclose(revtriang(m1), np.array([[8]]))

m3 = np.array([[1,5],
 [9,6]])
assert np.allclose(revtriang(m3), np.array([[1,5],
 [9,6]]))

m4 = np.array([[1,5,8],
 [9,6,2],
 [3,2,5]])
assert np.allclose(revtriang(m4), np.array([[1,5,8],
 [9,6,2],
 [2,3,5]]))

assert np.allclose(m4, np.array([[1,5,8],
 [9,6,2],
 [3,2,5]])) # shouldn't change the original

m5 = np.array([[5,4,2,6,4],
 [3,5,1,0,6],
 [6,4,9,2,3],
 [5,2,8,6,1],
 [7,9,3,2,2]])
assert np.allclose(revtriang(m5), np.array([[5, 4, 2, 6, 4],
 [3, 5, 1, 0, 6],
 [4, 6, 9, 2, 3],
 [8, 2, 5, 6, 1],
 [2, 3, 9, 7, 2]]))

try:
 revtriang(np.array([[7,1,6],
 [5,2,4]]))
 raise Exception("I should have failed!")
except ValueError:
 pass

```

&lt;/div&gt;

```
[35]: import numpy as np

def revtriang(mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = np.array([[8]])
assert np.allclose(revtriang(m1), np.array([[8]]))
```

(continues on next page)

(continued from previous page)

```
m3 = np.array([[1,5],
 [9,6]])
assert np.allclose(revtriang(m3), np.array([[1,5],
 [9,6]]))

m4 = np.array([[1,5,8],
 [9,6,2],
 [3,2,5]])
assert np.allclose(revtriang(m4), np.array([[1,5,8],
 [9,6,2],
 [2,3,5]]))

assert np.allclose(m4, np.array([[1,5,8],
 [9,6,2],
 [3,2,5]])) # shouldn't change the original

m5 = np.array([[5,4,2,6,4],
 [3,5,1,0,6],
 [6,4,9,2,3],
 [5,2,8,6,1],
 [7,9,3,2,2]])
assert np.allclose(revtriang(m5), np.array([[5, 4, 2, 6, 4],
 [3, 5, 1, 0, 6],
 [4, 6, 9, 2, 3],
 [8, 2, 5, 6, 1],
 [2, 3, 9, 7, 2]]))

try:
 revtriang(np.array([[7,1,6],
 [5,2,4]]))
 raise Exception("I should have failed!")
except ValueError:
 pass
```

## Exercise - walkas

⊕⊕⊕ Given a numpy matrix  $n \times m$  with odd  $m$ , RETURN a numpy array containing all the numbers found along the path of an S, from bottom to top.

**HINT:** can you determine the array dimension right away?

Example:

```
m = np.array([[5,8,2,4,6,5,7],
 [7,9,5,8,3,2,2],
 [6,1,8,3,6,6,1],
 [1,5,3,7,9,4,7],
 [1,5,3,2,9,5,4],
 [4,3,8,5,6,1,5]])
```

it must walk, **from bottom to top**:

```
m = np.array([[5,8,2,>,>,>,>],
 [7,9,5,^,3,2,2],
 [6,1,8,^,6,6,1],
 [1,5,3,^,9,4,7],
```

(continues on next page)

(continued from previous page)

```
[1,5,3,^,9,5,4],
[>,>,>,^,6,1,5]])
```

To obtain:

```
>>> walkas(m)
array([4., 3., 8., 5., 2., 7., 3., 8., 4., 6., 5., 7.])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"  
data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[36]: import numpy as np
```

```
def walkas(mat):

 #EFFICIENT SOLUTION
 n,m = mat.shape
 ret = np.zeros(n + m-1)
 ret[:m//2] = mat[-1,:m//2]
 ret[m//2:m//2+n] = mat[::-1,m//2]
 ret[-m//2:] = mat[0,m//2:]
 return ret

TEST
m1 = np.array([[7]])
assert np.allclose(walkas(m1), np.array([7]))

m2 = np.array([[7,5,2]])
assert np.allclose(walkas(m2), np.array([7,5,2]))

m3 = np.array([[9,3,5,6,0]])
assert np.allclose(walkas(m3), np.array([9,3,5,6,0]))

m4 = np.array([[7,5,2],
 [9,3,4]])
assert np.allclose(walkas(m4), np.array([9,3,5,2]))

m5 = np.array([[7,4,6],
 [8,2,1],
 [0,5,3]])
assert np.allclose(walkas(m5), np.array([0,5,2,4,6]))

m6 = np.array([[5,8,2,4,6,5,7],
 [7,9,5,8,3,2,2],
 [6,1,8,3,6,6,1],
 [1,5,3,7,9,4,7],
 [1,5,3,2,9,5,4],
 [4,3,8,5,6,1,5]])
assert np.allclose(walkas(m6), np.array([4,3,8,5,2,7,3,8,4,6,5,7]))
```

```
</div>
```

```
[36]: import numpy as np

def walkas(mat):
```

(continues on next page)

(continued from previous page)

```

raise Exception('TODO IMPLEMENT ME !')

TEST
m1 = np.array([[7]])
assert np.allclose(walkas(m1), np.array([7]))

m2 = np.array([[7,5,2]])
assert np.allclose(walkas(m2), np.array([7,5,2]))

m3 = np.array([[9,3,5,6,0]])
assert np.allclose(walkas(m3), np.array([9,3,5,6,0]))

m4 = np.array([[7,5,2],
 [9,3,4]])
assert np.allclose(walkas(m4), np.array([9,3,5,2]))

m5 = np.array([[7,4,6],
 [8,2,1],
 [0,5,3]])
assert np.allclose(walkas(m5), np.array([0,5,2,4,6]))

m6 = np.array([[5,8,2,4,6,5,7],
 [7,9,5,8,3,2,2],
 [6,1,8,3,6,6,1],
 [1,5,3,7,9,4,7],
 [1,5,3,2,9,5,4],
 [4,3,8,5,6,1,5]])
assert np.allclose(walkas(m6), np.array([4,3,8,5,2,7,3,8,4,6,5,7]))

```

### Exercise - walkaz

⊕⊕⊕ Given a numpy matrix  $n \times m$  with odd  $m$ , RETURN a numpy array containing all the numbers found along the path of an Z, from bottom to top.

**HINT:** can you determine the array dimension right away?

Example:

```

m = np.array([[5,8,2,4,6,5,7],
 [7,9,5,8,3,2,2],
 [6,1,8,3,6,6,1],
 [1,5,3,7,9,4,7],
 [1,5,3,2,9,5,4],
 [4,3,8,5,6,1,5]])

```

it must walk, **from bottom to top**:

```

m = np.array([[<,<,<,<,^,6,5,7],
 [7,9,5,^,3,2,2],
 [6,1,8,^,6,6,1],
 [1,5,3,^,9,4,7],
 [1,5,3,^,9,5,4],
 [4,3,8,^,<,<,<]])
```

To obtain:

```
>>> walkaz(m)
array([5., 1., 6., 5., 2., 7., 3., 8., 4., 2., 8., 5.])
```

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol" jupman-sol-code" style="display:none">

[37]: import numpy as np

```
def walkaz(mat):

 #EFFICIENT SOLUTION
 n,m = mat.shape
 ret = np.zeros(n + m-1)
 ret[:m//2] = mat[-1,-1:m//2:-1]
 ret[m//2:m//2+n] = mat[::-1,m//2]
 ret[-m//2:] = mat[0,m//2::-1]
 return ret

TEST
m1 = np.array([[7]])
assert np.allclose(walkaz(m1), np.array([7]))

m2 = np.array([[7,5,2]])
assert np.allclose(walkaz(m2), np.array([2,5,7]))

m3 = np.array([[9,3,5,6,0]])
assert np.allclose(walkaz(m3), np.array([0,6,5,3,9]))

m4 = np.array([[7,5,2],
 [9,3,4]])
assert np.allclose(walkaz(m4), np.array([4,3,5,7]))

m5 = np.array([[7,4,6],
 [8,2,1],
 [0,5,3]])
assert np.allclose(walkaz(m5), np.array([3,5,2,4,7]))

m6 = np.array([[5,8,2,4,6,5,7],
 [7,9,5,8,3,2,2],
 [6,1,8,3,6,6,1],
 [1,5,3,7,9,4,7],
 [1,5,3,2,9,5,4],
 [4,3,8,5,6,1,5]])
assert np.allclose(walkaz(m6), np.array([5,1,6,5,2,7,3,8,4,2,8,5]))
```

</div>

[37]: import numpy as np

```
def walkaz(mat):
 raise Exception('TODO IMPLEMENT ME !')

TEST
m1 = np.array([[7]])
assert np.allclose(walkaz(m1), np.array([7]))
```

(continues on next page)

(continued from previous page)

```
m2 = np.array([[7,5,2]])
assert np.allclose(walkaz(m2), np.array([2,5,7]))

m3 = np.array([[9,3,5,6,0]])
assert np.allclose(walkaz(m3), np.array([0,6,5,3,9]))

m4 = np.array([[7,5,2],
 [9,3,4]])
assert np.allclose(walkaz(m4), np.array([4,3,5,7]))

m5 = np.array([[7,4,6],
 [8,2,1],
 [0,5,3]])
assert np.allclose(walkaz(m5), np.array([3,5,2,4,7]))

m6 = np.array([[5,8,2,4,6,5,7],
 [7,9,5,8,3,2,2],
 [6,1,8,3,6,6,1],
 [1,5,3,7,9,4,7],
 [1,5,3,2,9,5,4],
 [4,3,8,5,6,1,5]])
assert np.allclose(walkaz(m6), np.array([5,1,6,5,2,7,3,8,4,2,8,5]))
```

## Continue

- Try doing exercises from [lists of lists<sup>309</sup>](#) using Numpy instead - try making the exercises performant by using Numpy features and functions (i.e. `2 * arr` multiplies all numbers in `arr` without the need of a slow Python `for`)
- For some nice application, follow [Numpy images tutorial<sup>310</sup>](#)

## References

- You can find much more details on [Python Data Science Handbook, Numpy part<sup>311</sup>](#)
- [machinelearningplus<sup>312</sup>](#) has Numpy exercises - (difficulty L1, L2, you can also try L3)

[ ]:

<sup>309</sup> <https://en.softpython.org/matrizes-lists/matrizes-lists2-sol.html>

<sup>310</sup> <https://en.softpython.org/matrizes-numpy/numpy-images-sol.html>

<sup>311</sup> <https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html>

<sup>312</sup> <https://www.machinelearningplus.com/python/101-numpy-exercises-python/>



## B - DATA ANALYSIS

### 8.1 Data formats

#### 8.1.1 Data formats 1 - introduction

[Download exercises zip](#)

[Browse files online<sup>313</sup>](#)

#### Introduction

In these tutorials we will see how to load and write tabular data such as CSV, and we will mention tree-like data such as JSON files. We will also spend a couple of words about opendata catalogs and licenses (creative commons).

In these tutorials we will review main data formats:

Textual formats

- Line files
- CSV (tabular data)
- JSON (tree-like data, just mention)

Binary formats (just mention)

- fogli Excel

We will also mention open data catalogs and licenses (Creative Commons)

#### What to do

1. unzip exercises in a folder, you should get something like this:

```
formats
formats1-lines.ipynb
formats1-lines-sol.ipynb
formats2-csv.ipynb
formats2-csv-sol.ipynb
formats3-json.ipynb
formats3-json-sol.ipynb
```

(continues on next page)

<sup>313</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/formats>

(continued from previous page)

```
formats4-chal.ipynb
jupman.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `formats/format1-lines.ipynb`
3. Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### Line files

Line files are typically text files which contain information grouped by lines. An example using historical characters might be like the following:

```
Leonardo
da Vinci
Sandro
Botticelli
Niccolò
Macchiavelli
```

We can immediately see a regularity: first two lines contain data of Leonardo da Vinci, second one the name and then the surname. Successive lines instead have data of Sandro Botticelli, with again first the name and then the surname and so on.

We might want to do a program that reads the lines and prints on the terminal names and surnames like the following:

```
Leonardo da Vinci
Sandro Botticelli
Niccolò Macchiavelli
```

To start having an approximation of the final result, we can open the file, read only the first line and print it:

```
[1]: with open('people-simple.txt', encoding='utf-8') as f:
 line=f.readline()
 print(line)
```

```
Leonardo
```

What happened? Let's examine first rows:

## open command

The command

```
open('people-simple.txt', encoding='utf-8')
```

allows us to open the text file by telling PYthon the file path 'people-simple.txt' and the encoding in which it was written (encoding='utf-8').

## The encoding

The encoding depends on the operating system and on the editor used to write the file. When we open a file, Python is not capable to divine the encoding, and if we do not specify anything Python might open the file assuming an encoding different from the original - in other words, if we omit the encoding (or we put a wrong one) we might end up seeing weird characters (like little squares instead of accented letters).

In general, when you open a file, try first to specify the encoding utf-8 which is the most common one. If it doesn't work try others, for example for files written in south Europe with Windows you might check encoding='latin-1'. If you open a file written elsewhere, you might need other encodings. For more in-depth information, you can read [Dive into Python - Chapter 4 - Strings<sup>314</sup>](#), and [Dive into Python - Chapter 11 - File<sup>315</sup>](#), **both of which are extremely recommended readings.**

## with block

The with defines a block with instructions inside:

```
with open('people-simple.txt', encoding='utf-8') as f:
 line=f.readline()
 print(line)
```

We used the with to tell PYthon that in any case, even if errors occur, we want that after having used the file, that is after having executed the instructions inside the internal block (the line=f.readline() and print(line)) Python must automatically close the file. Properly closing a file avoids to waste memory resources and creating hard to find paranormal errors. If you want to avoid hunting for never closed zombie files, always remember to open all files in with blocks! Furthermore, at the end of the row in the part as f: we assigned the file to a variable hereby called f, but we could have used any other name we liked.

**WARNING:** To indent the code, ALWAYS use sequences of four white spaces. Sequences of 2 spaces. Sequences of only 2 spaces even if allowed are not recommended.

**WARNING:** Depending on the editor you use, by pressing TAB you might get a sequence o f white spaces like it happens in Jupyter (4 spaces which is the recommended length), or a special tabulation character (to avoid)! As much as this annoying this distinction might appear, remember it because it might generate very hard to find errors.

**WARNING:** In the commands to create blocks such as with, always remember to put the character of colon : at the end of the line !

<sup>314</sup> <https://diveintopython3.problemsolving.io/strings.html>

<sup>315</sup> <https://diveintopython3.problemsolving.io/files.html>

The command

```
line=f.readline()
```

puts in the variable `line` the entire line, like a string. Warning: the string will contain at the end the special character of line return !

You might wonder where that `readline` comes from. Like everything in Python, our variable `f` which represents the file we just opened is an object, and like any object, depending on its type, it has particular methods we can use on it. In this case the method is `readline`.

The following command prints the string content:

```
print(line)
```

⊗ **1.1 EXERCISE:** Try to rewrite here the block we've just seen, and execute the cell by pressing Control+Enter. Rewrite the code with the fingers, not with copy-paste ! Pay attention to correct indentation with spaces in the block.

```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[2]: # write here

with open('people-simple.txt', encoding='utf-8') as f:
 line=f.readline()
 print(line)
```

Leonardo

</div>

```
[2]: # write here
```

Leonardo

⊗ **1.2 EXERCISE:** you might wondering what exactly is that `f`, and what exatly the method `readlines` should be doing. When you find yourself in these situations, you might help yourself with functions `type` and `help`. This time, directly copy paste the same code here, but insert inside `with` block the commands:

- `print(type(f))`
- `help(f)`
- `help(f.readline) # Attention: remember the f. before the readline !!`

Every time you add something, try to execute with Control+Enter and see what happens

```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[3]: # write here the code (copy and paste)
with open('people-simple.txt', encoding='utf-8') as f:
 line=f.readline()
 print(line)
 print(type(f))
 help(f.readline)
 help(f)
```

Leonardo

```
<class '_io.TextIOWrapper'>
Help on built-in function readline:

readline(size=-1, /) method of _io.TextIOWrapper instance
 Read until newline or EOF.

 Returns an empty string if EOF is hit immediately.

Help on TextIOWrapper object:

class TextIOWrapper(_TextIOBase)
| TextIOWrapper(buffer, encoding=None, errors=None, newline=None, line_
| buffering=False, write_through=False)
|
| Character and line based layer over a BufferedIOBase object, buffer.
|
| encoding gives the name of the encoding that the stream will be
| decoded or encoded with. It defaults to locale.getpreferredencoding(False).
|
| errors determines the strictness of encoding and decoding (see
| help(codecs.Codec) or the documentation for codecs.register) and
| defaults to "strict".
|
| newline controls how line endings are handled. It can be None, '',
| '\n', '\r', and '\r\n'. It works as follows:
|
| * On input, if newline is None, universal newlines mode is
| enabled. Lines in the input can end in '\n', '\r', or '\r\n', and
| these are translated into '\n' before being returned to the
| caller. If it is '', universal newline mode is enabled, but line
| endings are returned to the caller untranslated. If it has any of
| the other legal values, input lines are only terminated by the given
| string, and the line ending is returned to the caller untranslated.
|
| * On output, if newline is None, any '\n' characters written are
| translated to the system default line separator, os.linesep. If
| newline is '' or '\n', no translation takes place. If newline is any
| of the other legal values, any '\n' characters written are translated
| to the given string.
|
| If line_buffering is True, a call to flush is implied when a call to
| write contains a newline character.
|
| Method resolution order:
| TextIOWrapper
| _TextIOBase
| _IOBase
| builtins.object
|
| Methods defined here:
|
| __getstate__(...)
|
| __init__(self, /, *args, **kwargs)
| Initialize self. See help(type(self)) for accurate signature.
```

(continues on next page)

(continued from previous page)

```
| __next__(self, /)
| Implement next(self).
|
| __repr__(self, /)
| Return repr(self).
|
| close(self, /)
| Flush and close the IO object.
|
| This method has no effect if the file is already closed.
|
| detach(self, /)
| Separate the underlying buffer from the TextIOBase and return it.
|
| After the underlying buffer has been detached, the TextIO is in an
| unusable state.
|
| fileno(self, /)
| Returns underlying file descriptor if one exists.
|
| OSError is raised if the IO object does not use a file descriptor.
|
| flush(self, /)
| Flush write buffers, if applicable.
|
| This is not implemented for read-only and non-blocking streams.
|
| isatty(self, /)
| Return whether this is an 'interactive' stream.
|
| Return False if it can't be determined.
|
| read(self, size=-1, /)
| Read at most n characters from stream.
|
| Read from underlying buffer until we have n characters or we hit EOF.
| If n is negative or omitted, read until EOF.
|
| readable(self, /)
| Return whether object was opened for reading.
|
| If False, read() will raise OSError.
|
| readline(self, size=-1, /)
| Read until newline or EOF.
|
| Returns an empty string if EOF is hit immediately.
|
| reconfigure(self, /, *, encoding=None, errors=None, newline=None, line_
| buffering=None, write_through=None)
| Reconfigure the text stream with new parameters.
|
| This also does an implicit stream flush.
|
| seek(self, cookie, whence=0, /)
| Change stream position.
```

(continues on next page)

(continued from previous page)

```

| Change the stream position to the given byte offset. The offset is
| interpreted relative to the position indicated by whence. Values
| for whence are:
|
| * 0 -- start of stream (the default); offset should be zero or positive
| * 1 -- current stream position; offset may be negative
| * 2 -- end of stream; offset is usually negative
|
| Return the new absolute position.
|
| seekable(self, /)
| Return whether object supports random access.
|
| If False, seek(), tell() and truncate() will raise OSError.
| This method may need to do a test seek().
|
| tell(self, /)
| Return current stream position.
|
| truncate(self, pos=None, /)
| Truncate file to size bytes.
|
| File pointer is left unchanged. Size defaults to the current IO
| position as reported by tell(). Returns the new size.
|
| writable(self, /)
| Return whether object was opened for writing.
|
| If False, write() will raise OSError.
|
| write(self, text, /)
| Write string to stream.
| Returns the number of characters written (which is always equal to
| the length of the string).
|
| -----
| Static methods defined here:
|
| __new__(*args, **kwargs) from builtins.type
| Create and return a new object. See help(type) for accurate signature.
|
| -----
| Data descriptors defined here:
|
| buffer
|
| closed
|
| encoding
| Encoding of the text stream.
|
| Subclasses should override.
|
| errors
| The error setting of the decoder or encoder.
|

```

(continues on next page)

(continued from previous page)

```
| Subclasses should override.
|
| line_buffering
|
| name
|
| newlines
| Line endings translated so far.
|
| Only line endings translated during reading are considered.
|
| Subclasses should override.
|
| write_through
|
| -----
| Methods inherited from _IOBase:
|
| __del__(...)
|
| __enter__(...)
|
| __exit__(...)
|
| __iter__(self, /)
| Implement iter(self).
|
| readlines(self, hint=-1, /)
| Return a list of lines from the stream.
|
| hint can be specified to control the number of lines read: no more
| lines will be read if the total size (in bytes/characters) of all
| lines so far exceeds hint.
|
| writelines(self, lines, /)
| Write a list of lines to stream.
|
| Line separators are not added, so it is usual for each of the
| lines provided to have a line separator at the end.
|
| -----
| Data descriptors inherited from _IOBase:
|
| __dict__
```

&lt;/div&gt;

[3]: # write here the code (copy and paste)

Leonardo

<class '\_io.TextIOWrapper'>  
Help on built-in function readline:

(continues on next page)

(continued from previous page)

```
readline(size=-1, /) method of _io.TextIOWrapper instance
 Read until newline or EOF.
```

Returns an empty string if EOF is hit immediately.

Help on TextIOWrapper object:

```
class TextIOWrapper(_TextIOWrapper)
| TextIOWrapper(buffer, encoding=None, errors=None, newline=None, line_
| buffering=False, write_through=False)
|
| Character and line based layer over a BufferedIOBase object, buffer.
|
| encoding gives the name of the encoding that the stream will be
| decoded or encoded with. It defaults to locale.getpreferredencoding(False).
|
| errors determines the strictness of encoding and decoding (see
| help(codecs.Codec) or the documentation for codecs.register) and
| defaults to "strict".
|
| newline controls how line endings are handled. It can be None, '',
| '\n', '\r', and '\r\n'. It works as follows:
|
| * On input, if newline is None, universal newlines mode is
| enabled. Lines in the input can end in '\n', '\r', or '\r\n', and
| these are translated into '\n' before being returned to the
| caller. If it is '', universal newline mode is enabled, but line
| endings are returned to the caller untranslated. If it has any of
| the other legal values, input lines are only terminated by the given
| string, and the line ending is returned to the caller untranslated.
|
| * On output, if newline is None, any '\n' characters written are
| translated to the system default line separator, os.linesep. If
| newline is '' or '\n', no translation takes place. If newline is any
| of the other legal values, any '\n' characters written are translated
| to the given string.
|
| If line_buffering is True, a call to flush is implied when a call to
| write contains a newline character.
|
| Method resolution order:
| TextIOWrapper
| _TextIOWrapper
| _IOBase
| builtins.object
|
| Methods defined here:
|
| __getstate__(...)
|
| __init__(self, /, *args, **kwargs)
| Initialize self. See help(type(self)) for accurate signature.
|
| __next__(self, /)
| Implement next(self).
|
| __repr__(self, /)
```

(continues on next page)

(continued from previous page)

```
| Return repr(self).
|
| close(self, /)
| Flush and close the IO object.
|
| This method has no effect if the file is already closed.
|
| detach(self, /)
| Separate the underlying buffer from the TextIOBase and return it.
|
| After the underlying buffer has been detached, the TextIO is in an
| unusable state.
|
| fileno(self, /)
| Returns underlying file descriptor if one exists.
|
| OSError is raised if the IO object does not use a file descriptor.
|
| flush(self, /)
| Flush write buffers, if applicable.
|
| This is not implemented for read-only and non-blocking streams.
|
| isatty(self, /)
| Return whether this is an 'interactive' stream.
|
| Return False if it can't be determined.
|
| read(self, size=-1, /)
| Read at most n characters from stream.
|
| Read from underlying buffer until we have n characters or we hit EOF.
| If n is negative or omitted, read until EOF.
|
| readable(self, /)
| Return whether object was opened for reading.
|
| If False, read() will raise OSError.
|
| readline(self, size=-1, /)
| Read until newline or EOF.
|
| Returns an empty string if EOF is hit immediately.
|
| reconfigure(self, /, *, encoding=None, errors=None, newline=None, line_
| buffering=None, write_through=None)
| Reconfigure the text stream with new parameters.
|
| This also does an implicit stream flush.
|
| seek(self, cookie, whence=0, /)
| Change stream position.
|
| Change the stream position to the given byte offset. The offset is
| interpreted relative to the position indicated by whence. Values
| for whence are:
```

(continues on next page)

(continued from previous page)

```

| * 0 -- start of stream (the default); offset should be zero or positive
| * 1 -- current stream position; offset may be negative
| * 2 -- end of stream; offset is usually negative
|
| Return the new absolute position.
|
| seekable(self, /)
| Return whether object supports random access.
|
| If False, seek(), tell() and truncate() will raise OSError.
| This method may need to do a test seek().
|
| tell(self, /)
| Return current stream position.
|
| truncate(self, pos=None, /)
| Truncate file to size bytes.
|
| File pointer is left unchanged. Size defaults to the current IO
| position as reported by tell(). Returns the new size.
|
| writable(self, /)
| Return whether object was opened for writing.
|
| If False, write() will raise OSError.
|
| write(self, text, /)
| Write string to stream.
| Returns the number of characters written (which is always equal to
| the length of the string).
|
| -----
| Static methods defined here:
|
| __new__(*args, **kwargs) from builtins.type
| Create and return a new object. See help(type) for accurate signature.
|
| -----
| Data descriptors defined here:
|
| buffer
|
| closed
|
| encoding
| Encoding of the text stream.
|
| Subclasses should override.
|
| errors
| The error setting of the decoder or encoder.
|
| Subclasses should override.
|
| line_buffering
|
| name

```

(continues on next page)

(continued from previous page)

```

| newlines
| Line endings translated so far.
|
| Only line endings translated during reading are considered.
|
| Subclasses should override.
|
| write_through
|
| -----
| Methods inherited from _IOBase:
|
| __del__(...)
|
| __enter__(...)
|
| __exit__(...)
|
| __iter__(self, /)
| Implement iter(self).
|
| readlines(self, hint=-1, /)
| Return a list of lines from the stream.
|
| hint can be specified to control the number of lines read: no more
| lines will be read if the total size (in bytes/characters) of all
| lines so far exceeds hint.
|
| writelines(self, lines, /)
| Write a list of lines to stream.
|
| Line separators are not added, so it is usual for each of the
| lines provided to have a line separator at the end.
|
| -----
| Data descriptors inherited from _IOBase:
|
| __dict__

```

First we put the content of the first line into the variable `name`, now we might put it in a variable with a more meaningful name, like `name`. Also, we can directly read the next row into the variable `surname` and then print the concatenation of both:

```
[4]: with open('people-simple.txt', encoding='utf-8') as f:
 name=f.readline()
 surname=f.readline()
 print(name + ' ' + surname)
```

```
Leonardo
da Vinci
```

**PROBLEM !** The printing puts a weird carriage return. Why is that? If you remember, first we said that `readline` reads the line content in a string adding to the end also the special newline character. To eliminate it, you can use the

```
command rstrip():
```

```
[5]: with open('people-simple.txt', encoding='utf-8') as f:
 name=f.readline().rstrip()
 surname=f.readline().rstrip()
 print(name + ' ' + surname)
```

Leonardo da Vinci

⊗ **1.3 EXERCISE:** Again, rewrite the block above in the cell below, ed execute the cell with Control+Enter. Question: what happens if you use `strip()` instead of `rstrip()`? What about `lstrip()`? Can you deduce the meaning of `r` and `l`? If you can't manage it, try to use python command `help` by calling `help(string.rstrip)`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[6]: # write here

with open('people-simple.txt', encoding='utf-8') as f:
 name=f.readline().rstrip()
 surname=f.readline().rstrip()
 print(name + ' ' + surname)
```

Leonardo da Vinci

</div>

```
[6]: # write here
```

Leonardo da Vinci

Very good, we have the first line ! Now we can read all the lines in sequence. To this end, we can use a `while` cycle:

```
[7]: with open('people-simple.txt', encoding='utf-8') as f:
 line=f.readline()
 while line != "":
 name = line.rstrip()
 surname=f.readline().rstrip()
 print(name + ' ' + surname)
 line=f.readline()
```

Leonardo da Vinci

Sandro Botticelli

Niccolò Macchiavelli

---

**NOTE:** In Python there are [shorter ways<sup>316</sup>](#) to read a text file line by line, we used this approach to make explicit all passages.

What did we do? First, we added a `while` cycle in a new block

**WARNING:** In new block, since it is already within the external `with`, the instructions are indented of 8 spaces and not 4! If you use the wrong spaces, bad things happen !

<sup>316</sup> <https://thispointer.com/5-different-ways-to-read-a-file-line-by-line-in-python/>

We first read a line, and two cases are possible:

- a. we are at the end of the file (or file is empty) : in this case `readline()` call returns an empty string
- b. we are not at the end of the file: the first line is put as a string inside the variable `line`. Since Python internally uses a pointer to keep track at which position we are when reading inside the file, after the read such pointer is moved at the beginning of the next line. This way the next call to `readline()` will read a line from the new position.

In `while` block we tell Python to continue the cycle as long as `line` is *not* empty. If this is the case, inside the `while` block we parse the name from the line and put it in variable `name` (removing extra newline character with `rstrip()` as we did before), then we proceed reading the next line and parse the result inside the `surname` variable. Finally, we read again a line into the `line` variable so it will be ready for the next round of name extraction. If `line` is empty the cycle will terminate:

```
while line != "":
 name = line.rstrip() # enter cycle if line contains characters
 surname=f.readline().rstrip() # parses the name
 print(name + ' ' + surname) # reads next line and parses surname
 line=f.readline() # read next line
```

⊕ **1.4 EXERCISE:** As before, rewrite in the cell below the code with the `while`, paying attention to the indentation (for the external `with` line use copy-and-paste):

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8]: # write here the code of internal while

```
with open('people-simple.txt', encoding='utf-8') as f:
 line=f.readline()
 while line != "":
 name = line.rstrip()
 surname=f.readline().rstrip()
 print(name + ' ' + surname)
 line=f.readline()
```

Leonardo da Vinci  
Sandro Botticelli  
Niccolò Macchiavelli

</div>

[8]: # write here the code of internal while

Leonardo da Vinci  
Sandro Botticelli  
Niccolò Macchiavelli

## people-complex line file

Look at the file `people-complex.txt`:

```
name: Leonardo
surname: da Vinci
birthdate: 1452-04-15
name: Sandro
surname: Botticelli
birthdate: 1445-03-01
name: Niccolò
surname: Macchiavelli
birthdate: 1469-05-03
```

Supposing to read the file to print this output, how would you do it?

```
Leonardo da Vinci, 1452-04-15
Sandro Botticelli, 1445-03-01
Niccolò Macchiavelli, 1469-05-03
```

**Hint 1:** to obtain the string '`abcde`', the substring '`cde`', which starts at index 2, you can use the operator square brackets, using the index followed by colon :

```
[9]: x = 'abcde'
x[2:]
```

```
[9]: 'cde'
```

```
[10]: x[3:]
```

```
[10]: 'de'
```

**Hint 2:** To know the length of a string, use the function `len`:

```
[11]: len('abcde')
```

```
[11]: 5
```

⊗ **1.5 EXERCISE:** Write here the solution of the exercise ‘People complex’:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[12]: # write here
```

```
with open('people-complex.txt', encoding='utf-8') as f:
 line=f.readline()
 while line != "":
 name = line.rstrip()[len("name: "):]
 surname= f.readline().rstrip()[len("surname: "):]
 born = f.readline().rstrip()[len("birthdate: "):]
 print(name + ' ' + surname + ', ' + born)
 line=f.readline()
```

```
Leonardo da Vinci, 1452-04-15
Sandro Botticelli, 1445-03-01
Niccolò Macchiavelli, 1469-05-03
```

</div>

```
[12]: # write here
```

```
Leonardo da Vinci, 1452-04-15
Sandro Botticelli, 1445-03-01
Niccolò Macchiavelli, 1469-05-03
```

### Exercise - line file immersione-in-python-toc

⊕⊕⊕ This exercise is more challenging, if you are a beginner you might skip it and go on to CSVs

The book Dive into Python is nice and for the italian version there is a PDF, which has a problem though: if you try to print it, you will discover that the index is missing. Without despairing, we found a program to extract titles in a file as follows, but you will discover it is not exactly nice to see. Since we are Python ninjas, we decided to transform raw titles in a [real table of contents](#)<sup>317</sup>. Sure enough there are smarter ways to do this, like loading the pdf in Python with an appropriate module for pdfs, still this makes for an interesting exercise.

You are given the file `immersione-in-python-toc.txt`:

```
BookmarkBegin
BookmarkTitle: Il vostro primo programma Python
BookmarkLevel: 1
BookmarkPageNumber: 38
BookmarkBegin
BookmarkTitle: Immersione!
BookmarkLevel: 2
BookmarkPageNumber: 38
BookmarkBegin
BookmarkTitle: Dichiarare funzioni
BookmarkLevel: 2
BookmarkPageNumber: 41
BookmarkBeginint
BookmarkTitle: Argomenti opzionali e con nome
BookmarkLevel: 3
BookmarkPageNumber: 42
BookmarkBegin
BookmarkTitle: Scrivere codice leggibile
BookmarkLevel: 2
BookmarkPageNumber: 44
BookmarkBegin
BookmarkTitle: Stringhe di documentazione
BookmarkLevel: 3
BookmarkPageNumber: 44
BookmarkBegin
BookmarkTitle: Il percorso di ricerca di import
BookmarkLevel: 2
BookmarkPageNumber: 46
BookmarkBegin
BookmarkTitle: Ogni cosa è un oggetto
BookmarkLevel: 2
BookmarkPageNumber: 47
```

Write a python program to print the following output:

---

<sup>317</sup> [http://softpython.readthedocs.io/it/latest/\\_static/toc-immersione-in-python-3.txt](http://softpython.readthedocs.io/it/latest/_static/toc-immersione-in-python-3.txt)

```

Il vostro primo programma Python 38
Immersione! 38
Dichiarare funzioni 41
 Argomenti opzionali e con nome 42
Scrivere codice leggibile 44
 Stringhe di documentazione 44
Il percorso di ricerca di import 46
Ogni cosa è un oggetto 47

```

For this exercise, you will need to insert in the output artificial spaces, in a quantity determined by the rows `BookmarkLevel`

**QUESTION:** what's that weird value `&#232;` at the end of the original file? Should we report it in the output?

**HINT 1:** To convert a string into an integer number, use the function `int`:

```
[13]: x = '5'
```

```
[14]: x
```

```
[14]: '5'
```

```
[15]: int(x)
```

```
[15]: 5
```

**Warning:** `int(x)` returns a value, and never modifies the argument `x`!

**HINT 2:** To substitute a substring in a string, you can use the method `.replace`:

```
[16]: x = 'abcde'
x.replace('cd', 'HELLO')
```

```
[16]: 'abHELLOe'
```

**HINT 3:** while there is only one sequence to substitute, `replace` is fine, but if we had a milion of horrible sequences like `&gt;`, `&#62;`, `&x3e;`, what should we do? As good data cleaners, we recognize these are [HTML escape sequences](#)<sup>318</sup>, so we could use methods specific to sequences like `html.escape`<sup>319</sup>. Try it instead of `replace` and check if it works!

NOTE: Before using `html.unescape`, import the module `html` with the command:

```
import html
```

**HINT 4:** To write  $n$  copies of a character, use `*` like this:

```
[17]: "b" * 3
```

```
[17]: 'bbb'
```

```
[18]: "b" * 7
```

```
[18]: 'bbbbbbb'
```

**IMPLEMENTATION:** Write here the solution for the line file `immersione-in-python-toc.txt`, and try execute it by pressing Control + Enter:

<sup>318</sup> <https://corsidia.com/materia/web-design/caratterispecialihtml>

<sup>319</sup> <https://docs.python.org/3/library/html.html#html.unescape>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[19]: # write here

import html

with open("immersione-in-python-toc.txt", encoding='utf-8') as f:

 line=f.readline()
 while line != "":
 line = f.readline().strip()
 title = html.unescape(line[len("BookmarkTitle: "):])
 line=f.readline().strip()
 level = int(line[len("BookmarkLevel: "):])
 line=f.readline().strip()
 page = line[len("BookmarkPageNumber: "):]
 print((" " * level) + title + " " + page)
 line=f.readline()

 Il vostro primo programma Python 38
 Immersione! 38
 Dichiarare funzioni 41
 Argomenti opzionali e con nome 42
 Scrivere codice leggibile 44
 Stringhe di documentazione 44
 Il percorso di ricerca di import 46
 Ogni cosa è un oggetto 47
```

</div>

```
[19]: # write here

 Il vostro primo programma Python 38
 Immersione! 38
 Dichiarare funzioni 41
 Argomenti opzionali e con nome 42
 Scrivere codice leggibile 44
 Stringhe di documentazione 44
 Il percorso di ricerca di import 46
 Ogni cosa è un oggetto 47
```

### Continue

Go on with CSV tabular files<sup>320</sup>

```
[]:
```

<sup>320</sup> <https://en.softpython.org/formats/format2-csv-sol.html>

## 8.1.2 Data formats 2 - CSV files

### Download exercises zip

Browse files online<sup>321</sup>

There can be various formats for tabular data, among which you surely know Excel (.xls or .xlsx). Unfortunately, if you want to programmatically process data, you should better avoid them and prefer if possible the CSV format, literally 'Comma Separated Value'. Why? Excel format is very complex and may hide several things which have nothing to do with the raw data:

- formatting (bold fonts, colors ...)
- merged cells
- formulas
- multiple tabs
- macros

Correctly parsing complex files may become a nightmare. Instead, CSVs are far simpler, so much so you can even open them with a simple text editor.

We will try to open some CSV, taking into consideration the possible problems we might get. CSVs are not necessarily the perfect solution for everything, but they offer more control over reading and typically if there are conversion problems it is because we made a mistake, and not because the reader module decided on its own to exchange days with months in dates.

### Why parsing a CSV ?

To load and process CSVs there exist many powerful and intuitive modules such as Pandas in Python or R dataframes. Yet, in this notebook we will load CSVs using the most simple method possible, that is reading row by row, mimicking the method already seen in the previous part of the tutorial. Don't think this method is primitive or stupid, according to the situation it may save the day. How? Some files may potentially occupy huge amounts of memory, and in modern laptops as of 2019 we only have 4 gigabytes of RAM, the memory where Python stores variables. Given this, Python base functions to read files try their best to avoid loading everything in RAM. Typically a file is read sequentially one piece at a time, putting in RAM only one row at a time.

**QUESTION 2.1:** if we want to know if a given file of 1000 terabytes contains only 3 million rows in which the word 'ciao' is present, are we obliged to put in RAM *all* of the rows ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); " data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** no, it is sufficient to keep in memory one row at a time, and hold the count in another variable

</div>

**QUESTION 2.2:** What if we wanted to take a 100 terabyte file and create another one by appending to each row of the first one the word 'ciao'? Should we put in RAM at the same time all the rows of the first file ? What about the rows of second one?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); " data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** No, it is enough to keep in RAM one row at a time, which is first read from the first file and then written right away in the second file.

---

<sup>321</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/formats>

```
</div>
```

## Reading a CSV

We will start with artifical example CSV. Let's look at `example-1.csv` which you can find in the same folder as this Jupyter notebook. It contains animals with their expected lifespan:

```
animal, lifespan
dog, 12
cat, 14
pelican, 30
squirrel, 6
eagle, 25
```

We notice right away that the CSV is more structured than files we've seen in the previous section

- in the first line there are column names, separated with commas: `animal, lifespan`
- fields in successive rows are also separated by commas, : `dog, 12`

Let's try now to import this file in Python:

```
[1]: import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:

 # we create an object 'my_reader' which will take rows from the file
 my_reader = csv.reader(f, delimiter=',')

 # 'my_reader' is an object considered 'iterable', that is,
 # if used in a 'for' will produce a sequence of rows from csv
 # NOTE: here every file row is converted into a list of Python strings !

 for row in my_reader:
 print('We just read a row !')
 print(row) # prints variable 'row', which is a list of strings
 print('') # prints an empty string, to separate in vertical
```

We just read a row !  
['animal', 'lifespan']

We just read a row !  
['dog', '12']

We just read a row !  
['cat', '14']

We just read a row !  
['pelican', '30']

We just read a row !  
['squirrel', '6']

We just read a row !  
['eagle', '25']

We immediatly notice from output that example file is being printed, but there are square parrenthesis ( [ ] ). What do they mean? Those we printed are *lists of strings*

Let's analyze what we did:

```
import csv
```

Python natively has a module to deal with csv files, which has the intuitive `csv` name. With this instruction, we just loaded the module.

What happens next? As already did for files with lines before, we open the file in a `with` block:

```
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 for row in my_reader:
 print(row)
```

For now ignore the `newline=''` and notice how first we specified the encoding

Once the file is open, in the row

```
my_reader = csv.reader(f, delimiter=',')
```

we ask to `csv` module to create a reader object called `my_reader` for our file, telling Python that comma is the delimiter for fields.

**NOTE:** `my_reader` is the name of the variable we are creating, it could be any name.

This reader object can be exploited as a sort of generator of rows by using a `for` cycle:

```
for row in my_reader:
 print(row)
```

In `for` cycle we employ `letto` to iterate in the reading of the file, producing at each iteration a row we call `row` (but it could be any name we like). At each iteration, the variable `row` gets printed.

If you look closely the prints of first lists, you will see that each time to each row is assigned only one Python list. The list contains as many elements as the number of fields in the CSV.

⊕ **EXERCISE 2.3:** Rewrite in the cell below the instructions to read and print the CSV, paying attention to indentation:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]:

```
write here

import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:

 # we create an object 'my_reader' which will take rows from the file
 my_reader = csv.reader(f, delimiter=',')

 # 'my_reader' is an object considered 'iterable', that is,
 # if used in a 'for' will produce a sequence of rows from csv
 # NOTE: here every file row is converted into a list of Python strings !

 for row in my_reader:
 print("We just read a row !")
 print(row) # prints variable 'row', which is a list of strings
 print('') # prints an empty string, to separate in vertical
```

```
We just read a row !
['animal', ' lifespan']
```

```
We just read a row !
['dog', '12']
```

```
We just read a row !
['cat', '14']
```

```
We just read a row !
['pelican', '30']
```

```
We just read a row !
['squirrel', '6']
```

```
We just read a row !
['eagle', '25']
```

```
</div>
```

[2]:

```
write here
```

⊗⊗ **EXERCISE 2.4:** try to put into `big_list` a list containing all the rows extracted from the file, which will be a list of lists like so:

```
[['animal', 'lifespan'],
 ['dog', '12'],
 ['cat', '14'],
 ['pelican', '30'],
 ['squirrel', '6'],
 ['eagle', '25']]
```

**HINT:** Try creating an empty list and then adding elements with `.append` method

[Show solution](#)

```
we create an object 'my_reader' which will take rows from the file
my_reader = csv.reader(f, delimiter=',')

'my_reader' is an object considered 'iterable', that is,
if used in a 'for' will produce a sequence of rows from csv
NOTE: here every file row is converted into a list of Python strings !

big_list = []
for row in my_reader:
 big_list.append(row)
```

[3]:

```
write here

import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:

 # we create an object 'my_reader' which will take rows from the file
 my_reader = csv.reader(f, delimiter=',')

 # 'my_reader' is an object considered 'iterable', that is,
 # if used in a 'for' will produce a sequence of rows from csv
 # NOTE: here every file row is converted into a list of Python strings !

 big_list = []
 for row in my_reader:
 big_list.append(row)
```

(continues on next page)

(continued from previous page)

```

print(big_list)

[['animal', ' lifespan'], ['dog', '12'], ['cat', '14'], ['pelican', '30'], ['squirrel
→', '6'], ['eagle', '25']]
```

</div>

[3]:

```
write here
```

**⊗⊗ EXERCISE 2.5:** You may have noticed that numbers in lists are represented as strings like '12' (note apeces), instead that like Python integer numbers (represented without apeces), 12:

```
We just read a row!
['dog', '12']
```

So, by reading the file and using normal for cycles, try to create a new variable `big_list` like this, which

- has only data, the row with the header is not present
- numbers are represented as proper integers

```
[['dog', 12],
 ['cat', 14],
 ['pelican', 30],
 ['squirrel', 6],
 ['eagle', 25]]
```

**HINT 1:** to jump a row you can use the instruction `next(my_reader)`

**HINT 2:** to convert a string into an integer, you can use for example. `int('25')`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4]:

```
write here

import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 big_list = []
 next(my_reader)
 for row in my_reader:
 big_list.append([row[0], int(row[1])])
 print(big_list)

[['dog', 12], ['cat', 14], ['pelican', 30], ['squirrel', 6], ['eagle', 25]]
```

</div>

[4]:

```
write here
```

### What's a reader ?

We said that `my_reader` generates a sequence of rows, and it is *iterable*. In `for` cycle, at every cycle we ask to read a new line, which is put into variable `row`. We might then ask ourselves, what happens if we directly print `my_reader`, without any `for`? Will we see a nice list or something else? Let's try:

```
[5]: import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 print(my_reader)

<_csv.reader object at 0x7f7e701b7050>
```

This result is quite disappointing

⊕ **EXERCISE 2.6:** you probably found yourself in the same situation when trying to print a sequence generated by a call to `range(5)`: instead of the actual sequence you get a `range` object. If you want to convert the generator to a list, what should you do?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution </a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[6]: # write here

import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 print(list(my_reader))

[['animal', ' lifespan'], ['dog', '12'], ['cat', '14'], ['pelican', '30'], ['squirrel
→', '6'], ['eagle', '25']]
```

</div>

```
[6]: # write here

[[{"animal": " lifespan"}, {"dog": "12"}, {"cat": "14"}, {"pelican": "30"}, {"squirrel
→", "6"}, {"eagle": "25"}]]
```

### Consuming a file

Not all sequences are the same. From what you've seen so far, going through a file in Python looks a lot like iterating a list. Which is very handy, but you need to pay attention to some things. Given that files potentially might occupy terabytes, basic Python functions to load them avoid loading everything into memory and typically a file is read one piece at a time. But if the whole file is loaded into Python environment in one shot, what happens if we try to go through it twice inside the same `with`? What happens if we try using it outside `with`? To find out look at next exercises.

⊕ **EXERCISE 2.7:** taking the solution to previous exercise, try to call `print(list(my_reader))` twice, in sequence. Do you get the same output in both occasions?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution </a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[7]: # write here
```

(continues on next page)

(continued from previous page)

```

import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 print(list(my_reader))
 print(list(my_reader))

[['animal', 'lifespan'], ['dog', '12'], ['cat', '14'], ['pelican', '30'], ['squirrel',
→, '6'], ['eagle', '25']]
[]

</div>

```

[7]: # write here

⊕ **EXERCISE 2.8:** Taking the solution from previous exercise (using only one print), try down here to move the print to the left (removing any spaces). Does it still work ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8]: # write here

```

import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
print(list(my_reader)) # COMMENTED, AS IT WOULD RAISE ON ERROR OF CLOSED FILE
 # We can't use commands which read the file outside the
→ with !

```

</div>

[8]: # write here

⊕⊕ **EXERCISE 2.9:** Now that we understood which kind of beast `my_reader` is, try to produce this result as done before, but using a *list comprehension* instead of the `for`:

```

[['dog', 12],
 ['cat', 14],
 ['pelican', 30],
 ['squirrel', 6],
 ['eagle', 25]]

```

- If you can, try also to write the whole transformation to create `big_list` in one row, usinf the function `itertools.islice`<sup>322</sup> to jump the header (for example `itertools.islice(['A', 'B', 'C', 'D', 'E'], 2, None)` first two elements and produces the sequence C D E F G - in our case the elements produced by `my_reader` would be rows)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

<sup>322</sup> <https://docs.python.org/3/library/itertools.html#itertools.islice>

[9]:

```
import csv
import itertools
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 # write here
 big_list = [[row[0], int(row[1])] for row in itertools.islice(my_reader, 1, None)]
print(big_list)

[['dog', 12], ['cat', 14], ['pelican', 30], ['squirrel', 6], ['eagle', 25]]
```

</div>

[9]:

```
import csv
import itertools
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 # write here
```

⊕ **EXERCISE 2.10:** Create a file `my-example.csv` in the same folder where this Jupyter notebook is, and copy inside the content of the file `example-1.csv`. Then add a column `description`, remembering to separate the column name from the preceding one with a comma. As column values, put into successive rows strings like `dogs walk`, `pelicans fly`, etc according to the animal, remembering to separate them from lifespan using a comma, like this:

`dog,12,dogs walk`

After this, copy and paste down here the Python code to load the file, putting the file name `my-example.csv`, and try to load everything, just to check everything is working:

[10]: # write here

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
data-jupman-show="Show answer"
data-jupman-hide="Hide">Show answer<div class="jupman-sol jupman-sol-question" style="display:none">
```

**ANSWER:**

```
animal,lifespan,description
dog,12,dogs walk
cat,14,cats walk
pelican,30,pelicans fly
squirrel,6,squirrels fly
eagle,25,eagles fly
```

</div>

⊕ **EXERCISE 2.11:** Not every CSV is structured in the same way, sometimes when we write csvs or import them some tweak is necessary. Let's see which problems may arise:

- In the file, try to put one or two spaces before numbers, for example write down here and look what happens

```
dog, 12,dogs fly
```

**QUESTION 2.11.1:** Does the space get imported?

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol" jupman-sol-question" style="display:none">

**ANSWER:** yes

</div>

**QUESTION 2.11.2:** if we convert to integer, is the space a problem?

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol" jupman-sol-question" style="display:none">

**ANSWER:** no

</div>

**QUESTION 2.11.3** Modify only dogs description from dogs walk to dogs walk, but don't fly and try to reexecute the cell which opens the file. What happens?

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol" jupman-sol-question" style="display:none">

**ANSWER:** Python reads one element more in the list

</div>

**QUESTION 2.11.4:** To overcome previous problem, a solution you can adopt in CSVs is to round strings containing commas with double quotes, like this: "dogs walk, but don't fly". Does it work ?

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol" jupman-sol-question" style="display:none">

**ANSWER:** yes

</div>

## Reading as dictionaries

To read a CSV, instead of getting lists, you may more conveniently get dictionaries in the form of `OrderedDicts`

See [Python documentation](#)<sup>323</sup>

**NOTE:** different Python versions give different dictionaries:

- < 3.6: `dict`
- 3.6, 3.7: `OrderedDict`
- ≥ 3.8: `dict`

Python 3.8 returned to old `dict` because in the implementation of its dictionaries the key order is guaranteed, so it will be consistent with the one of CSV headers

```
[11]: import csv
with open('example-1.csv', encoding='utf-8', newline='') as f:
 my_reader = csv.DictReader(f, delimiter=',') # Notice we now used DictReader
 for d in my_reader:
 print(d)
```

<sup>323</sup> <https://docs.python.org/3/library/csv.html#csv.DictReader>

```
OrderedDict([('animal', 'dog'), ('lifespan', '12')])
OrderedDict([('animal', 'cat'), ('lifespan', '14')])
OrderedDict([('animal', 'pelican'), ('lifespan', '30')])
OrderedDict([('animal', 'squirrel'), ('lifespan', '6')])
OrderedDict([('animal', 'eagle'), ('lifespan', '25')])
```

### Writing a CSV

You can easily create a CSV by instantiating a `writer` object:

#### ATTENTION: BE SURE TO WRITE IN THE CORRECT FILE!

If you don't pay attention to file names, **you risk deleting data !**

```
[12]: import csv

To write, REMEMBER to specify the `w` option.
WARNING: 'w' *completely* replaces existing files !!
with open('written-file.csv', 'w', encoding='utf-8', newline='') as csvfile_out:

 my_writer = csv.writer(csvfile_out, delimiter=',')

 my_writer.writerow(['This', 'is', 'a header'])
 my_writer.writerow(['some', 'example', 'data'])
 my_writer.writerow(['some', 'other', 'example data'])
```

### Reading and writing a CSV

To create a copy of an existing CSV, you may nest a `with` for writing inside another for reading:

#### ATTENTION: CAREFUL NOT TO SWAP FILE NAMES!

When we read and write it's easy to make mistakes and accidentally overwrite our precious data.

#### To avoid issues:

- use explicit names both for output files (es: `example-1-enriched.csv`) and handles (i.e. `csvfile_out`)
- backup data to read
- always check before carelessly executing code you just wrote !

```
[13]: import csv

WARNING: handle here is called *csvfile_in*
with open('example-1.csv', encoding='utf-8', newline='') as csvfile_in:
 my_reader = csv.reader(csvfile_in, delimiter=',')

 # Notice this 'with' is *inside* the outer one
 # To write, REMEMBER to specify the `w` option.
 # WARNING 1: handle here is called *csvfile_out*
```

(continues on next page)

(continued from previous page)

```
WARNING 2: 'w' *completely* replaces existing files !!
with open('example-1-enriched.csv', 'w', encoding='utf-8', newline='') as csvfile_
↪out:

 my_writer = csv.writer(csvfile_out, delimiter=',')

 for row in my_reader:
 row.append('something else')
 my_writer.writerow(row)
```

Let's see the new file was actually created by reading it:

```
[14]: with open('example-1-enriched.csv', encoding='utf-8', newline='') as csvfile_in:
 my_reader = csv.reader(csvfile_in, delimiter=',')

 for row in my_reader:
 print(row)

['animal', ' lifespan', 'something else']
['dog', '12', 'something else']
['cat', '14', 'something else']
['pelican', '30', 'something else']
['squirrel', '6', 'something else']
['eagle', '25', 'something else']
```

## CSV Botteghe storiche

Usually in open data catalogs like the popular CKAN platform (for example [dati.trentino.it](http://dati.trentino.it)<sup>324</sup>, [data.gov.uk](https://data.gov.uk)<sup>325</sup>, European data portal<sup>326</sup> run instances of CKAN) files are organized in *datasets*, which are collections of *resources*: each resource directly contains a file inside the catalog (typically CSV, JSON or XML) or a link to the real file located in a server belonging to the organization which created the data.

The first dataset we will look at will be 'Botteghe storiche del Trentino':

<https://dati.trentino.it/dataset/botteghe-storiche-del-trentino>

Here you will find some generic information about the dataset, of importance note the data provider: Provincia Autonoma di Trento and the license [Creative Commons Attribution v4.0](#)<sup>327</sup>, which basically allows any reuse provided you cite the author.

Inside the dataset page, there is a resource called 'Botteghe storiche'

<https://dati.trentino.it/dataset/botteghe-storiche-del-trentino/resource/43fc327e-99b4-4fb8-833c-1807b5ef1d90>

At the resource page, we find a link to the CSV file (you can also find it by clicking on the blue button 'Go to the resource'):

[http://www.commercio.provincia.tn.it/binary/pat\\_commercio/valorizzazione\\_luoghi\\_storici/Albo\\_botteghe\\_storiche\\_in\\_ordine\\_iscrizione\\_](http://www.commercio.provincia.tn.it/binary/pat_commercio/valorizzazione_luoghi_storici/Albo_botteghe_storiche_in_ordine_iscrizione_)

Accordingly to the browser and operating system you have, by clicking on the link above you might get different results. In our case, on browser Firefox and operating system Linux we get (here we only show first 10 rows):

<sup>324</sup> <http://dati.trentino.it/>

<sup>325</sup> <https://data.gov.uk/>

<sup>326</sup> <https://www.europeandataportal.eu/>

<sup>327</sup> <https://creativecommons.org/licenses/by/4.0/deed.en>

```
Numeri,Insegna,Indirizzo,Civico,Comune,Cap,Frazione/LocalitÃ ,Note
1,BAZZANELLA RENATA,Via del Lagorai,30,Sover,38068,Piscine di Sover,"generi misti, ↵
 ↵bar - ristorante"
2,CONFEZIONI MONTIBELLER S.R.L.,Corso Ausugum,48,Borgo Valsugana,38051,,esercizio ↵
 ↵commerciale
3,FOTOGRAFICA TRINTINAGLIA UMBERTO S.N.C.,Largo Dordi,8,Borgo Valsugana,38051,, ↵
 ↵"esercizio commerciale, attivitÃ artigianale"
4,BAR SERAFINI DI MINATI RENZO,,24,Grigno,38055,Serafini,esercizio commerciale
6,SEMBENINI GINO & FIGLI S.R.L.,Via S. Francesco,35,Riva del Garda,38066,,,
7,HOTEL RISTORANTE PIZZERIA ªALLA NAVEª, Via Nazionale,29,Lavis,38015,Nave San ↵
 ↵Felice,
8,OBRELLI GIOIELLERIA DAL 1929 S.R.L.,Via Roma,33,Lavis,38015,,,
9,MACELLERIE TROIER S.A.S. DI TROIER DARIO E C.,Via Roma,13,Lavis,38015,,,
10,NARDELLI TIZIANO,Piazza Manci,5,Lavis,38015,,esercizio commerciale
```

As expected, values are separated with commas.

### Problem: wrong characters ??

You can suddenly discover a problem in the first row of headers, in the column Frazione/LocalitÃ . It seems last character is wrong, in italian it should show accented like à. Is it truly a problem of the file ? Not really. Probably, the server is not telling Firefox which encoding is the correct one for the file. Firefox is not magical, and tries its best to show the CSV on the base of the info it has, which may be limited and / or even wrong. World is never like we would like it to be ...

⊗ **2.12 EXERCISE:** download the CSV, and try opening it in Excel and / or LibreOffice Calc. Do you see a correct accented character? If not, try to set the encoding to 'Unicode (UTF-8)' (in Calc is called 'Character set').

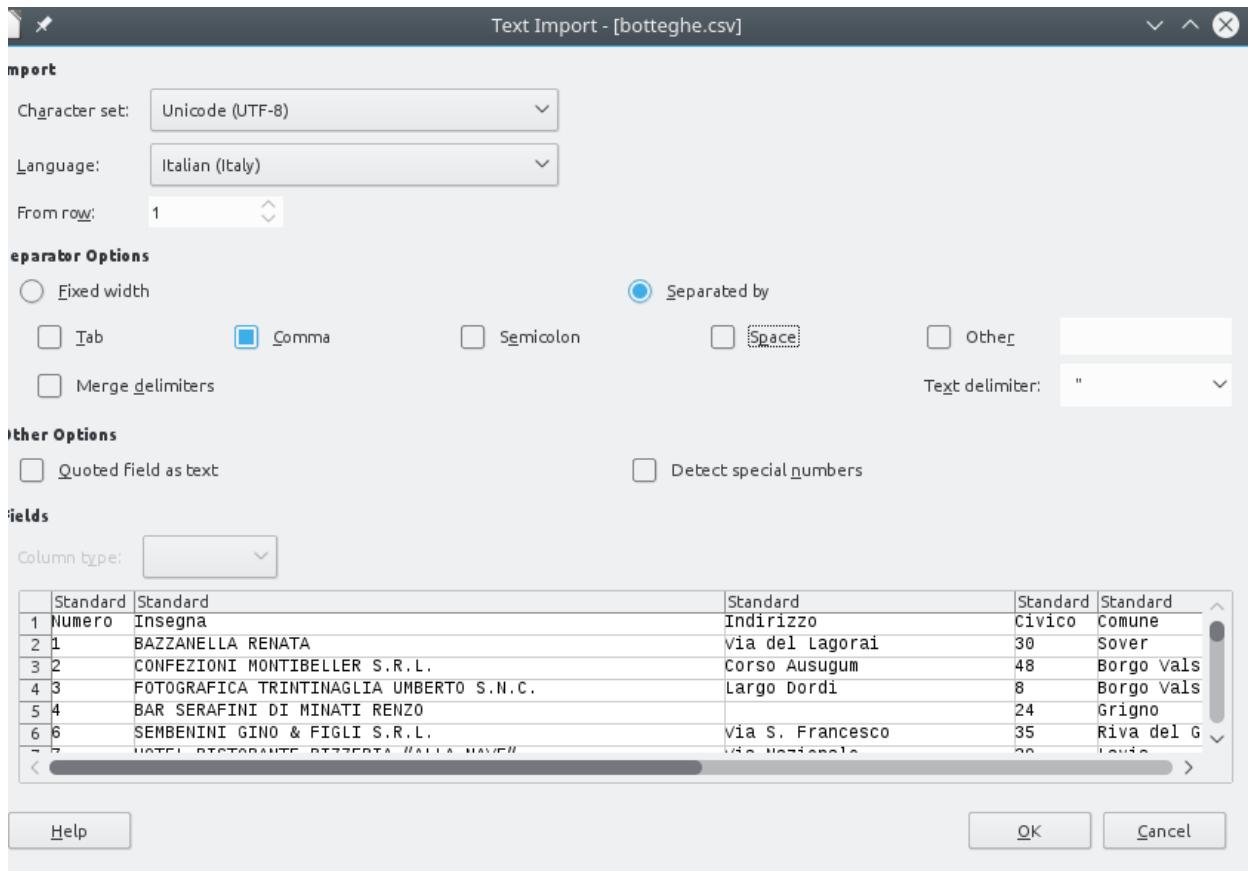
### WARNING: CAREFUL IF YOU USE Excel!

By clicking directly on File->Open in Excel, probably Excel will try to guess on its own how to put the CSV in a table, and will make the mistake to place everything in a column. To avoid the problem, we have to tell Excel to show a panel to ask us how we want to open the CSV, by doing like so:

- In old Excels, find File-> Import
- In recent Excels, click on tab Data and then select From text. For further information, see copytrans guide<sup>328</sup>

- **NOTE:** If the file is not available, in the folder where this notebook is you will find the same file renamed to botteghe-storiche.csv

<sup>328</sup> <https://www.copytrans.net/support/how-to-open-a-csv-file-in-excel/>



We should get a table like this. Notice how the *Frazione/Località* header displays with the right accent because we selected Character set: Unicode (UTF-8) which is the appropriate one for this dataset:

	A	B	C	D	E	F	G	H
1	Numero	Insegna	Indirizzo	Civico	Comune	Cap	Frazione/Località	Note
2	1	BAZZANELLA RENATA	Via del Lagorai	30	Sover	38068	Piscine di Sover	generi misti, bar - ristorante
3	2	CONFEZIONI MONTIBELLER S.R.L.	Corso Ausugum	48	Borgo Vals	38051		esercizio commerciale
4	3	FOTOGRAFICA TRINTINAGLIA UMBERTO S.N.C.	Largo Dordi	8	Borgo Vals	38051		esercizio commerciale, attività artigianale
5	4	BAR SERAFINI DI MINATI RENZO		24	Grigno	38055	Serafini	esercizio commerciale
6	6	SEMBENINI GINO & FIGLI S.R.L.	via S. Francesco	35	Riva del Garda	38066		
7	7	HOTEL RISTORANTE PIZZERIA ALLA NAVE"	Via Nazionale	29	Lavis	38015	Nave San Felice	
8	8	OBRELLI GIOIELLERIA DAL 1929 S.R.L.	Via Roma	33	Lavis	38015		
9	9	MACELLERIE TROIER S.A.S. DI TROIER DARIO E C.	Via Roma	13	Lavis	38015		
10	10	NARDELLI TIZIANO	Piazza Manci	5	Lavis	38015		esercizio commerciale

## Botteghe storiche in Python

Now that we understood a couple of things about encoding, let's try to import the file in Python.

If we load in Python the first 5 entries with a csv DictReader and print them we should see something like this:

```
OrderedDict([('Numero', '1'),
 ('Insegna', 'BAZZANELLA RENATA'),
 ('Indirizzo', 'Via del Lagorai'),
 ('Civico', '30'),
 ('Comune', 'Sover'),
 ('Cap', '38068'),
 ('Frazione/Località', 'Piscine di Sover'),
 ('Note', 'generi misti, bar - ristorante'))],
OrderedDict([('Numero', '2'),
 ('Insegna', 'CONFEZIONI MONTIBELLER S.R.L.')],
```

(continues on next page)

(continued from previous page)

```

 ('Indirizzo', 'Corso Ausugum'),
 ('Civico', '48'),
 ('Comune', 'Borgo Valsugana'),
 ('Cap', '38051'),
 ('Frazione/Località', ''),
 ('Note', 'esercizio commerciale'))),
OrderedDict([('Numero', '3'),
 ('Insegna', 'FOTOGRAFICA TRINTINAGLIA UMBERTO S.N.C.'),
 ('Indirizzo', 'Largo Dordi'),
 ('Civico', '8'),
 ('Comune', 'Borgo Valsugana'),
 ('Cap', '38051'),
 ('Frazione/Località', ''),
 ('Note', 'esercizio commerciale, attività artigianale'))),
OrderedDict([('Numero', '4'),
 ('Insegna', 'BAR SERAFINI DI MINATI RENZO'),
 ('Indirizzo', ''),
 ('Civico', '24'),
 ('Comune', 'Grigno'),
 ('Cap', '38055'),
 ('Frazione/Località', 'Serafini'),
 ('Note', 'esercizio commerciale'))),
OrderedDict([('Numero', '6'),
 ('Insegna', 'SEMBENINI GINO & FIGLI S.R.L.'),
 ('Indirizzo', 'Via S. Francesco'),
 ('Civico', '35'),
 ('Comune', 'Riva del Garda'),
 ('Cap', '38066'),
 ('Frazione/Località', ''),
 ('Note', '')])

```

We would like to know which different categories of *bottega* there are, and count them. Unfortunately, there is no specific field for *Categoria*, so we will need to extract this information from other fields such as *Insegna* and *Note*. For example, this *Insegna* contains the category *BAR*, while the *Note* (*commercial enterprise*) is a bit too generic to be useful:

```
'Insegna': 'BAR SERAFINI DI MINATI RENZO',
'Note': 'esercizio commerciale',
```

while this other *Insegna* contains just the owner name and *Note* holds both the categories *bar* and *ristorante*:

```
'Insegna': 'BAZZANELLA RENATA',
'Note': 'generi misti, bar - ristorante',
```

As you see, data is non uniform:

- sometimes the category is in the *Insegna*
- sometimes is in the *Note*
- sometimes is in both
- sometimes is lowercase
- sometimes is uppercase
- sometimes is single
- sometimes is multiple (*bar* – *ristorante*)

First we want to extract all categories we can find, and rank them according their frequency, from most frequent to least frequent.

To do so, you need to

- count all words you can find in both `Insegna` and `Note` fields, and sort them. Note you need to normalize the uppercase.
- consider a category relevant if it is present at least 11 times in the dataset.
- filter non relevant words: some words like prepositions, type of company ('S.N.C', S.R.L., ..), etc will appear a lot, and will need to be ignored. To detect them, you are given a list called `stopwords`.

**NOTE:** the rules above do not actually extract all the categories, for the sake of the exercise we only keep the most frequent ones.

To know how to proceed, read the following.

### Botteghe storiche - rank\_categories

Load the file with `csv.DictReader` and while you are loading it, calculate the words as described above. Afterwards, return a list of words with their frequencies.

Do **not** load the whole file into memory, just process one dictionary at a time and update statistics accordingly.

Expected output:

```
[('BAR', 191),
('RISTORANTE', 150),
('HOTEL', 67),
('ALBERGO', 64),
('MACELLERIA', 27),
('PANIFICIO', 22),
('CALZATURE', 21),
('FARMACIA', 21),
('ALIMENTARI', 20),
('PIZZERIA', 16),
('SPORT', 16),
('TABACCHI', 12),
('FERRAMENTA', 12),
('BAZAR', 11)]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"< data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[15]: def rank_categories(stopwords):
 ret = {}
 import csv
 with open('botteghe.csv', newline='', encoding='utf-8') as csvfile:
 reader = csv.DictReader(csvfile, delimiter=',')
 for d in reader:
 words = d['Insegna'].split(' ') + d['Note'].upper().split(' ')
 for word in words:
 if word in ret and not word in stopwords:
 ret[word] += 1
 else:
 ret[word] = 1
 return sorted([(key, val) for key, val in ret.items() if val > 10], key=lambda c:c[1], reverse=True)
```

(continues on next page)

(continued from previous page)

```

stopwords = [
 'S.N.C.', 'SNC', 'S.A.S.', 'S.R.L.', 'S.C.A.R.L.', 'SCARL', 'S.A.S',
 ↪'COMMERCIALE', 'FAMIGLIA', 'COOPERATIVA',
 '!', '&', 'C.', 'ESERCIZIO',
 'IL', 'DE', 'DI', 'A', 'DA', 'E', 'LA', 'AL', 'DEL', 'ALLA',]
categories = rank_categories(stopwords)

categories

```

```
[15]: [('BAR', 191),
 ('RISTORANTE', 150),
 ('HOTEL', 67),
 ('ALBERGO', 64),
 ('MACELLERIA', 27),
 ('PANIFICIO', 22),
 ('CALZATURE', 21),
 ('FARMACIA', 21),
 ('ALIMENTARI', 20),
 ('PIZZERIA', 16),
 ('SPORT', 16),
 ('TABACCHI', 12),
 ('FERRAMENTA', 12),
 ('BAZAR', 11)]
```

&lt;/div&gt;

```

[15]: def rank_categories(stopwords):
 raise Exception('TODO IMPLEMENT ME !')

stopwords = [
 'S.N.C.', 'SNC', 'S.A.S.', 'S.R.L.', 'S.C.A.R.L.', 'SCARL', 'S.A.S',
 ↪'COMMERCIALE', 'FAMIGLIA', 'COOPERATIVA',
 '!', '&', 'C.', 'ESERCIZIO',
 'IL', 'DE', 'DI', 'A', 'DA', 'E', 'LA', 'AL', 'DEL', 'ALLA',]
categories = rank_categories(stopwords)

categories

```

```
[15]: [('BAR', 191),
 ('RISTORANTE', 150),
 ('HOTEL', 67),
 ('ALBERGO', 64),
 ('MACELLERIA', 27),
 ('PANIFICIO', 22),
 ('CALZATURE', 21),
 ('FARMACIA', 21),
 ('ALIMENTARI', 20),
 ('PIZZERIA', 16),
 ('SPORT', 16),
 ('TABACCHI', 12),
 ('FERRAMENTA', 12),
 ('BAZAR', 11)]
```

## Botteghe storiche - enrich

Once you found the categories, implement function `enrich`, which takes the db and previously computed categories, and WRITES a NEW file `botteghe-enriched.csv` where the rows are enriched with a new field `Categorie`, which holds a list of the categories a particular `bottega` belongs to.

- Write the new file with a `DictWriter`, see documentation<sup>329</sup>

The new file should contain rows like this (showing only first 5):

```
OrderedDict([
 ('Numero', '1'),
 ('Insegna', 'BAZZANELLA RENATA'),
 ('Indirizzo', 'Via del Lagorai'),
 ('Civico', '30'),
 ('Comune', 'Sover'),
 ('Cap', '38068'),
 ('Frazione/Località', 'Piscine di Sover'),
 ('Note', 'generi misti, bar - ristorante'),
 ('Categorie', "['BAR', 'RISTORANTE'])])
OrderedDict([
 ('Numero', '2'),
 ('Insegna', 'CONFEZIONI MONTIBELLER S.R.L.'),
 ('Indirizzo', 'Corso Ausugum'),
 ('Civico', '48'),
 ('Comune', 'Borgo Valsugana'),
 ('Cap', '38051'),
 ('Frazione/Località', ''),
 ('Note', 'esercizio commerciale'),
 ('Categorie', '[]')])
OrderedDict([
 ('Numero', '3'),
 ('Insegna', 'FOTOGRAFICA TRINTINAGLIA UMBERTO S.N.C. '),
 ('Indirizzo', 'Largo Dordi'),
 ('Civico', '8'),
 ('Comune', 'Borgo Valsugana'),
 ('Cap', '38051'),
 ('Frazione/Località', ''),
 ('Note', 'esercizio commerciale, attività artigianale'),
 ('Categorie', '[]')])
OrderedDict([
 ('Numero', '4'),
 ('Insegna', 'BAR SERAFINI DI MINATI RENZO'),
 ('Indirizzo', ''),
 ('Civico', '24'),
 ('Comune', 'Grigno'),
 ('Cap', '38055'),
 ('Frazione/Località', 'Serafini'),
 ('Note', 'esercizio commerciale'),
 ('Categorie', "['BAR'])")]
OrderedDict([
 ('Numero', '6'),
 ('Insegna', 'SEMBENINI GINO & FIGLI S.R.L. '),
 ('Indirizzo', 'Via S. Francesco'),
 ('Civico', '35'),
 ('Comune', 'Riva del Garda'),
 ('Cap', '38066'),
 ('Frazione/Località', ''),
 ('Note', ''),
 ('Categorie', '[]')])
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution"

<sup>329</sup> <https://docs.python.org/3/library/csv.html#csv.DictWriter>

data-jupman-hide="Hide">>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[16]: def enrich(categories):

 ret = []

 fieldnames = []
 # read headers
 with open('botteghe.csv', newline='', encoding='utf-8') as csvfile_in:
 reader = csv.DictReader(csvfile_in, delimiter=',')
 d1 = next(reader)
 fieldnames = list(d1.keys()) # otherwise we cannot append

 fieldnames.append('Categorie')

 with open('botteghe.csv', newline='', encoding='utf-8',) as csvfile_in:
 reader = csv.DictReader(csvfile_in, delimiter=',')

 with open('botteghe-enriched-solution.csv', 'w', newline='', encoding='utf-8') as csvfile_out:
 writer = csv.DictWriter(csvfile_out, fieldnames=fieldnames)
 writer.writeheader()

 for d in reader:

 new_d = {key:val for key, val in d.items()}
 new_d['Categorie'] = []
 for cat in categories:
 if cat[0] in d['Insegna'].upper() or cat[0] in d['Note'].upper():
 new_d['Categorie'].append(cat[0])
 writer.writerow(new_d)

enrich(rank_categories(stopwords))
```

</div>

```
[16]: def enrich(categories):
 raise Exception('TODO IMPLEMENT ME !')

enrich(rank_categories(stopwords))
```

```
[17]: # let's see if we created the file we wanted
(using botteghe-enriched-solution.csv to avoid polluting your file)

with open('botteghe-enriched-solution.csv', newline='', encoding='utf-8',) as csvfile_in:
 reader = csv.DictReader(csvfile_in, delimiter=',')
 # better to pretty print the OrderedDicts, otherwise we get unreadable output
 # for documentation see https://docs.python.org/3/library/pprint.html
 import pprint
```

(continues on next page)

(continued from previous page)

```

pp = pprint.PrettyPrinter(indent=4)
for i in range(5):
 d = next(reader)
 pp.pprint(d)

OrderedDict([
 ('Numero', '1'),
 ('Insegna', 'BAZZANELLA RENATA'),
 ('Indirizzo', 'Via del Lagorai'),
 ('Civico', '30'),
 ('Comune', 'Sover'),
 ('Cap', '38068'),
 ('Frazione/Località', 'Piscine di Sover'),
 ('Note', 'generi misti, bar - ristorante'),
 ('Categorie', "['BAR', 'RISTORANTE'])])
OrderedDict([
 ('Numero', '2'),
 ('Insegna', 'CONFEZIONI MONTIBELLER S.R.L.'),
 ('Indirizzo', 'Corso Ausugum'),
 ('Civico', '48'),
 ('Comune', 'Borgo Valsugana'),
 ('Cap', '38051'),
 ('Frazione/Località', ''),
 ('Note', 'esercizio commerciale'),
 ('Categorie', '[]')])
OrderedDict([
 ('Numero', '3'),
 ('Insegna', 'FOTOGRAFICA TRINTINAGLIA UMBERTO S.N.C.'),
 ('Indirizzo', 'Largo Dordi'),
 ('Civico', '8'),
 ('Comune', 'Borgo Valsugana'),
 ('Cap', '38051'),
 ('Frazione/Località', ''),
 ('Note', 'esercizio commerciale, attività artigianale'),
 ('Categorie', '[]')])
OrderedDict([
 ('Numero', '4'),
 ('Insegna', 'BAR SERAFINI DI MINATI RENZO'),
 ('Indirizzo', ''),
 ('Civico', '24'),
 ('Comune', 'Grigno'),
 ('Cap', '38055'),
 ('Frazione/Località', 'Serafini'),
 ('Note', 'esercizio commerciale'),
 ('Categorie', "['BAR'])")]
OrderedDict([
 ('Numero', '6'),
 ('Insegna', 'SEMBENINI GINO & FIGLI S.R.L.'),
 ('Indirizzo', 'Via S. Francesco'),
 ('Civico', '35'),
 ('Comune', 'Riva del Garda'),
 ('Cap', '38066'),
 ('Frazione/Località', ''),
 ('Note', ''),
 ('Categorie', '[]')])
```

### CSV Air quality

You will now analyze air\_quality in Trentino. You are given a dataset which records various pollutants ('Inquinante') at various stations ('Stazione') in Trentino. Pollutants values can be 'PM10', 'Birossido Zolfo', and a few others. Each station records some set of pollutants. For each pollutant values are recorded ('Valore') 24 times per day.

Data provider: [dati.trentino.it](https://dati.trentino.it/)<sup>330</sup>

### load\_air\_quality

Implement a function to load the dataset air-quality.csv

- USE encoding latin-1

Expected output:

```
>>> load_air_quality('air-quality.csv')
[
 OrderedDict([('Stazione', 'Parco S. Chiara'),
 ('Inquinante', 'PM10'),
 ('Data', '2019-05-04'),
 ('Ora', '1'),
 ('Valore', '17'),
 ('Unità di misura', 'µg/mc')]),
 OrderedDict([('Stazione', 'Parco S. Chiara'),
 ('Inquinante', 'PM10'),
 ('Data', '2019-05-04'),
 ('Ora', '2'),
 ('Valore', '19'),
 ('Unità di misura', 'µg/mc')]),
 .
 .
 .
]
```

**IMPORTANT 1:** look at the dataset by yourself !

Here we show only first rows, but to get a clear picture of the dataset you need to study it a bit by yourself

**IMPORTANT 2:** EVERY field is a STRING, including 'Valore' !

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[18] :

```
import csv

def load_air_quality(filename):
 """Loads file data and RETURN a list of dictionaries
 """

```

(continues on next page)

<sup>330</sup> <https://dati.trentino.it/dataset/qualita-dell-aria-rilevazioni-delle-stazioni-monitoraggio>

(continued from previous page)

```

with open(filename, newline='', encoding='latin-1') as csvfile:
 reader = csv.DictReader(csvfile)
 lst = []
 for d in reader:
 lst.append(d)
return lst

air_quality = load_air_quality('air-quality.csv')

[18]: [OrderedDict([('Stazione', 'Parco S. Chiara'),
 ('Inquinante', 'PM10'),
 ('Data', '2019-05-04'),
 ('Ora', '1'),
 ('Valore', '17'),
 ('Unità di misura', 'µg/mc')]),
 OrderedDict([('Stazione', 'Parco S. Chiara'),
 ('Inquinante', 'PM10'),
 ('Data', '2019-05-04'),
 ('Ora', '2'),
 ('Valore', '19'),
 ('Unità di misura', 'µg/mc')]),
 OrderedDict([('Stazione', 'Parco S. Chiara'),
 ('Inquinante', 'PM10'),
 ('Data', '2019-05-04'),
 ('Ora', '3'),
 ('Valore', '17'),
 ('Unità di misura', 'µg/mc')]),
 OrderedDict([('Stazione', 'Parco S. Chiara'),
 ('Inquinante', 'PM10'),
 ('Data', '2019-05-04'),
 ('Ora', '4'),
 ('Valore', '15'),
 ('Unità di misura', 'µg/mc')]),
 OrderedDict([('Stazione', 'Parco S. Chiara'),
 ('Inquinante', 'PM10'),
 ('Data', '2019-05-04'),
 ('Ora', '5'),
 ('Valore', '13'),
 ('Unità di misura', 'µg/mc'))]

```

&lt;/div&gt;

```

[18]: import csv

def load_air_quality(filename):
 """Loads file data and RETURN a list of dictionaries
 """
 raise Exception('TODO IMPLEMENT ME !')

air_quality = load_air_quality('air-quality.csv')

```

### calc\_avg\_pollution

Implement a function to RETURN a dictionary containing two elements tuples as keys:

- first tuple element is the station ('Stazione'),
- second tuple element is the name of a pollutant ('Inquinante')

To each tuple key, you must associate as value the average for that station *and* pollutant over all days.

Expected output:

```
>>> calc_avg_pollution(air_quality)
{('Parco S. Chiara', 'PM10'): 11.385752688172044,
 ('Parco S. Chiara', 'PM2.5'): 7.9471544715447155,
 ('Parco S. Chiara', 'Biessido di Azoto'): 20.828146143437078,
 ('Parco S. Chiara', 'Ozono'): 66.69541778975741,
 ('Parco S. Chiara', 'Biessido Zolfo'): 1.2918918918918918,
 ('Via Bolzano', 'PM10'): 12.526881720430108,
 ('Via Bolzano', 'Biessido di Azoto'): 29.28493894165536,
 ('Via Bolzano', 'Ossido di Carbonio'): 0.5964769647696474,
 ('Piana Rotaliana', 'PM10'): 9.728744939271255,
 ('Piana Rotaliana', 'Biessido di Azoto'): 15.170068027210885,
 ('Piana Rotaliana', 'Ozono'): 67.03633916554509,
 ('Rovereto', 'PM10'): 9.475806451612904,
 ('Rovereto', 'PM2.5'): 7.764784946236559,
 ('Rovereto', 'Biessido di Azoto'): 16.284167794316645,
 ('Rovereto', 'Ozono'): 70.54655870445345,
 ('Borgo Valsugana', 'PM10'): 11.819407008086253,
 ('Borgo Valsugana', 'PM2.5'): 7.413746630727763,
 ('Borgo Valsugana', 'Biessido di Azoto'): 15.73806275579809,
 ('Borgo Valsugana', 'Ozono'): 58.599730458221025,
 ('Riva del Garda', 'PM10'): 9.912398921832883,
 ('Riva del Garda', 'Biessido di Azoto'): 17.125845737483086,
 ('Riva del Garda', 'Ozono'): 68.38159675236807,
 ('A22 (Avio)', 'PM10'): 9.651821862348179,
 ('A22 (Avio)', 'Biessido di Azoto'): 33.0650406504065,
 ('A22 (Avio)', 'Ossido di Carbonio'): 0.4228848821081822,
 ('Monte Gaza', 'PM10'): 7.794520547945205,
 ('Monte Gaza', 'Biessido di Azoto'): 4.34412955465587,
 ('Monte Gaza', 'Ozono'): 99.0858310626703}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[19]:

```
def calc_avg_pollution(db):

 ret = {}
 counts = {}
 for diz in db:
 t = (diz['Stazione'], diz['Inquinante'])
 if t in ret:
 ret[t] += float(diz['Valore'])
 counts[t] += 1
 else:
 ret[t] = float(diz['Valore'])
 counts[t] = 1
```

(continues on next page)

(continued from previous page)

```

for t in ret:
 ret[t] /= counts[t]
return ret

calc_avg_pollution(air_quality)
[19]: {('Parco S. Chiara', 'PM10'): 11.385752688172044,
 ('Parco S. Chiara', 'PM2.5'): 7.9471544715447155,
 ('Parco S. Chiara', 'Biessido di Azoto'): 20.828146143437078,
 ('Parco S. Chiara', 'Ozono'): 66.69541778975741,
 ('Parco S. Chiara', 'Biessido Zolfo'): 1.2918918918918918,
 ('Via Bolzano', 'PM10'): 12.526881720430108,
 ('Via Bolzano', 'Biessido di Azoto'): 29.28493894165536,
 ('Via Bolzano', 'Ossido di Carbonio'): 0.5964769647696474,
 ('Piana Rotaliana', 'PM10'): 9.728744939271255,
 ('Piana Rotaliana', 'Biessido di Azoto'): 15.170068027210885,
 ('Piana Rotaliana', 'Ozono'): 67.03633916554509,
 ('Rovereto', 'PM10'): 9.475806451612904,
 ('Rovereto', 'PM2.5'): 7.764784946236559,
 ('Rovereto', 'Biessido di Azoto'): 16.284167794316645,
 ('Rovereto', 'Ozono'): 70.54655870445345,
 ('Borgo Valsugana', 'PM10'): 11.819407008086253,
 ('Borgo Valsugana', 'PM2.5'): 7.413746630727763,
 ('Borgo Valsugana', 'Biessido di Azoto'): 15.73806275579809,
 ('Borgo Valsugana', 'Ozono'): 58.599730458221025,
 ('Riva del Garda', 'PM10'): 9.912398921832883,
 ('Riva del Garda', 'Biessido di Azoto'): 17.125845737483086,
 ('Riva del Garda', 'Ozono'): 68.38159675236807,
 ('A22 (Avio)', 'PM10'): 9.651821862348179,
 ('A22 (Avio)', 'Biessido di Azoto'): 33.0650406504065,
 ('A22 (Avio)', 'Ossido di Carbonio'): 0.4228848821081822,
 ('Monte Gaza', 'PM10'): 7.794520547945205,
 ('Monte Gaza', 'Biessido di Azoto'): 4.34412955465587,
 ('Monte Gaza', 'Ozono'): 99.0858310626703}

```

&lt;/div&gt;

[19]:

```

def calc_avg_pollution(db):
 raise Exception('TODO IMPLEMENT ME !')

calc_avg_pollution(air_quality)

```

**Continue**

Go on with JSON format<sup>331</sup>

### 8.1.3 Data formats 3 - JSON

#### Download exercises zip

Browse files online<sup>332</sup>

JSON is a more elaborated format, widely used in the world of web applications.

A json is simply a text file, structured as *a tree*. Let's see an example, extracted from the data Bike sharing stations of Lavis municipality as found on dati.trentino :

- Data source: dati.trentino.it<sup>333</sup> - Trasport Service of the Autonomous Province of Trento
- License: CC-BY 4.0<sup>334</sup>

File bike-sharing-lavis.json:

```
[
 {
 "name": "Grazioli",
 "address": "Piazza Grazioli - Lavis",
 "id": "Grazioli - Lavis",
 "bikes": 3,
 "slots": 7,
 "totalslots": 10,
 "position": [
 46.139732902099794,
 11.111516155225331
]
 },
 {
 "name": "Pressano",
 "address": "Piazza della Croce - Pressano",
 "id": "Pressano - Lavis",
 "bikes": 2,
 "slots": 5,
 "totalslots": 7,
 "position": [
 46.15368174037716,
 11.106601229430453
]
 },
 {
 "name": "Stazione RFI",
 "address": "Via Stazione - Lavis",
 "id": "Stazione RFI - Lavis",
 "bikes": 4,
 "slots": 6,
 "totalslots": 10,
```

(continues on next page)

<sup>331</sup> <https://en.softpython.org/formats/format3-json-sol.html>

<sup>332</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/formats>

<sup>333</sup> <https://dati.trentino.it/dataset/stazioni-bike-sharing-emotion-trentino>

<sup>334</sup> <http://creativecommons.org/licenses/by/4.0/deed.it>

(continued from previous page)

```

"position": [
 46.148180371138814,
 11.096753997622727
]
}
]
```

As you can see, the json format is very similar to data structures we already have in Python, such as strings, integer numbers, floats, lists and dictionaries. The only difference are the json null fields which become None in Python. So the conversion to Python is almost always easy and painless, to perform it you can use the native Python module called `json` by calling the function `json.load`, which interprets the json text file and converts it to a Python data structure:

```
[1]: import json

with open('bike-sharing-lavis.json', encoding='utf-8') as f:
 python_content = json.load(f)

print(python_content)

[{'name': 'Grazioli', 'address': 'Piazza Grazioli - Lavis', 'id': 'Grazioli - Lavis',
 'bikes': 3, 'slots': 7, 'totalSlots': 10, 'position': [46.139732902099794, 11.
 111516155225331]}, {'name': 'Pressano', 'address': 'Piazza della Croce - Pressano',
 'id': 'Pressano - Lavis', 'bikes': 2, 'slots': 5, 'totalSlots': 7, 'position': [46.
 15368174037716, 11.106601229430453]}, {'name': 'Stazione RFI', 'address': 'Via
 Stazione - Lavis', 'id': 'Stazione RFI - Lavis', 'bikes': 4, 'slots': 6, 'totalSlots
 ': 10, 'position': [46.148180371138814, 11.096753997622727]}]
```

Notice that what we've just read with the function `json.load` is not simple text anymore, but Python objects. For this json, the most external object is a list (note the square brackets at the file beginning and end). We can check using `type` on `python_content`:

```
[2]: type(python_content)
[2]: list
```

By looking at the JSON closely, you will see it is a list of dictionaries. Thus, to access the first dictionary (that is, the one at zero-th index), we can write

```
[3]: python_content[0]
[3]: {'name': 'Grazioli',
 'address': 'Piazza Grazioli - Lavis',
 'id': 'Grazioli - Lavis',
 'bikes': 3,
 'slots': 7,
 'totalSlots': 10,
 'position': [46.139732902099794, 11.111516155225331]}
```

We see it's the station in Piazza Grazioli. To get the exact name, we will access the `'address'` key in the first dictionary:

```
[4]: python_content[0]['address']
[4]: 'Piazza Grazioli - Lavis'
```

To access the position, we will use the corresponding key:

```
[5]: python_content[0]['position']
```

```
[5]: [46.139732902099794, 11.111516155225331]
```

Note how the position is a list itself. In JSON we can have arbitrarily branched trees, without necessarily a regular structure (althouth when we're generating a json it certainly helps maintaining a regular data scheme).

### JSONL

There is a particular JSON file type which is called **JSONL**<sup>335</sup> (note the *L* at the end), which is a text file containing a sequence of lines, each representing a valid json object.

Let's have a look at the file `employees.jsonl`:

```
{ "name": "Mario", "surname": "Rossi" }
{ "name": "Paolo", "surname": "Bianchi" }
{ "name": "Luca", "surname": "Verdi" }
```

To read it, we can open the file, separating the text lines and then interpret each of them as a single JSON object:

```
[6]: import json

with open('./employees.jsonl', encoding='utf-8',) as f:
 json_texts_list = list(f) # converts file text lines into a Python list

in this case we will have a python content for each row of the original file

i = 0
for json_text in json_texts_list:
 python_content = json.loads(json_text) # converts json text to a python object
 print('Object ', i)
 print(python_content)
 i = i + 1

Object 0
{'name': 'Mario', 'surname': 'Rossi'}
Object 1
{'name': 'Paolo', 'surname': 'Bianchi'}
Object 2
{'name': 'Luca', 'surname': 'Verdi'}
```

#### WARNING: this notebook is IN-PROGRESS

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
</div>
```

---

<sup>335</sup> <http://jsonlines.org/>

**Continue**

Go on with [graph formats<sup>336</sup>](#)

[ ] :

## 8.1.4 Formats 4 - Challenges

### [Download exercises zip](#)

Browse online [files<sup>337</sup>](#)

#### Parsing challenge - Spam killer

Roughly half of all emails sent in the world are spam.

Enraged by the number of pointless messages arriving each day, you decide to develop the definitive spam filter.

#### Spam killer 1. mail reader

A mail is a text file formatted as specified by standard [RFC 822<sup>338</sup>](#) (you don't need to read the specs, but keep in mind RFC are typically specs!)

A mail contains a certain number of fields, an empty line, and then the mail body:

```
Received: from forwarder@mailforeverybody.net
Message-Id: <v121c0404ad6a23934739@>
Mime-Version: 1.0
Content-Type: text/plain; charset="us-ascii"
Date: Thursday, 4 Jun 2020 09:43:14 -0800
To: noreply@softpython.org
From: Harvey The Salesman <harvey@thegreatvacuum.com>
Subject: DISCOUNTED Vacuum Cleaners
Precedence: bulk

Hi!
Find the best offers on our website: thegreatvacuum.com !!

Cheers,
Harvey
```

Each field name is separated from the value by a colon :

For example, in:

```
From: Harvey The Salesman <harvey@thegreatvacuum.com>
```

From is the field name, and Harvey The Salesman <harvey@thegreatvacuum.com> is the field value.

Implement a function `read_mail(filename)` which parses a `mailn.txt` file ([download files](#)) and RETURN a dictionary holding all the field names

<sup>336</sup> <https://en.softpython.org/formats/format4-graph-sol.html>

<sup>337</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/formats>

<sup>338</sup> <https://tools.ietf.org/html/rfc822>

- the body has no field name in the file: in the dictionary you can use Body as field name
- **REMEMBER** to remove newlines from field values
- **DO NOT** remove newlines from the body
- **HINT:** Getting the body text right might be tricky, so first try just parsing the fields

Example:

```
>>> pprint(read_mail('mail1.txt'))

{'Received': 'from forwarder@mailforeverybody.net',
 'Message-ID': '<v121c0404ad6a23934739@>',
 'Mime-Version': '1.0',
 'Content-Type': 'text/plain; charset="us-ascii"',
 'Date': 'Thursday, 4 Jun 2020 09:43:14 -0800',
 'To': 'noreply@softpython.org',
 'From': 'Harvey The Salesman <harvey@thegreatvacuum.com>',
 'Subject': 'DISCOUNTED Vacuum Cleaners',
 'Precedence': 'bulk',
 'Body': 'Hi!\nFind the best offers on our website: thegreatvacuum.com !!!\nCheers,\n\u2192Harvey'}
```

```
[1]:
from pprint import pprint

def read_mail(filename):
 """RETURN a NEW dictionary
 """
 raise Exception('TODO IMPLEMENT ME !')

pprint(read_mail('mail1.txt'))

assert read_mail('mail1.txt') == {
 'Body': 'Hi!\n'
 'Find the best offers on our website: thegreatvacuum.com !!!\n'
 'Cheers, \n'
 'Harvey',
 'Content-Type': 'text/plain; charset="us-ascii"',
 'Date': 'Thursday, 4 Jun 2020 09:43:14 -0800',
 'From': 'Harvey The Salesman <harvey@thegreatvacuum.com>',
 'Message-ID': '<v121c0404ad6a23934739@>',
 'Mime-Version': '1.0',
 'Precedence': 'bulk',
 'Received': 'from forwarder@mailforeverybody.net',
 'Subject': 'DISCOUNTED Vacuum Cleaners',
 'To': 'noreply@softpython.org'
}

read_mail('mail2.txt') == {'Received': 'from mailman@networked-solutions.net',
 'Message-ID': '<v47gc04e7ad6a249f4539@>',
 'Mime-Version': '1.0',
 'Content-Type': 'text/plain; charset="us-ascii"',
 'Date': 'Tuesday, 7 Jul 2020 16:25:14 -0800',
 'To': 'info@softpython.org',
 'From': 'Mr Boss <head@overpaid-data-scientists.com>',
 'Subject': '20K/month Job offer',
 'Precedence': 'bulk',
```

(continues on next page)

(continued from previous page)

```
'Body': "Congratulations! You've been crunching so many matrices \nduring the job_
↪interview you deserve 20.000€ salary/month + benefits.\nWe will install in your_
↪office three pinball machines \nand a dispenser of M&Ms - which colors do you_
↪prefer?\nBest,\nYour Next Boss\n"
}
```

## Spam killer 2. running filters

You defined various filters you want to run on the mails. Each filter is defined as a tuple containing a field name and a string to search for. If the field value contains the string, the mail is marked as spam.

Write a function `run_filter` which takes filters as list of tuples and a list of mail files, and RETURN a report as a list of lists. It must have:

- a header
- rows
- columns Subject, From
- column SPAM? as a boolean: True if any of the filters detected the mail as spam, False otherwise

Example:

```
>>> report = run_filters([('From', 'secret-encounters-at-night.com'),
 ('Body', 'offer')],
 ['mail1.txt', 'mail2.txt', 'mail3.txt', 'mail4.txt'])
>>> pprint(report, width=90)
[['Subject', 'From', 'SPAM?'],
 ['DISCOUNTED Vacuum Cleaners', 'Harvey The Salesman <harvey@thegreatvacuum.com>', ↪
True],
 ['20K/month Job offer', 'Mr Boss <head@overpaid-data-scientists.com>', False],
 ['I noticed you ...', 'That lady <lady@secret-encounters-at-night.com>', True],
 ['Some help with your thesis', 'John <john@yourfriends.net>', False]]
```

[2]:

```
from pprint import pprint

def run_filters(filters, filenames):
 """RETURN a NEW list of lists
 """
 raise Exception('TODO IMPLEMENT ME !')

report1 = run_filters([('From', 'secret-encounters-at-night.com'), ('Body', 'offer')],
 ['mail1.txt', 'mail2.txt', 'mail3.txt', 'mail4.txt'])
assert report1 == [
 ['Subject', 'From', 'SPAM?'],
 ['DISCOUNTED Vacuum Cleaners', 'Harvey The Salesman <harvey@thegreatvacuum.com>', ↪
True],
 ['20K/month Job offer', 'Mr Boss <head@overpaid-data-scientists.com>', False],
 ['I noticed you ...', 'That lady <lady@secret-encounters-at-night.com>', True],
 ['Some help with your thesis', 'John <john@yourfriends.net>', False]
]

report2 = run_filters([('From', 'vacuum'), ('From', 'Guru')],
```

(continues on next page)

(continued from previous page)

```
 ['mail4.txt', 'mail1.txt', 'mail5.txt'])
assert report2 == [
 ['Subject', 'From', 'SPAM?'],
 ['Some help with your thesis', 'John <john@yourfriends.net>', False],
 ['DISCOUNTED Vacuum Cleaners', 'Harvey The Salesman <harvey@thegreatvacuum.com>'],
 True],
 ['Is somebody stealing your domain?', 'Internet Guru <service@cndomaintrouble.
 org>', True],
]
```

## Parsing challenge - Markdown

Markdown is a language for writing documents, which allows writing plain text with additional syntax to express the way it should be formatted. Many editors support Markdown (Jupyter and Github included). For example, some Markdown text like this:

```
My Heading

some paragraph, so much interesting

another paragraph, with a some bla bla

Another big heading

There is **something notable** and then regular words.
```

would be displayed in Jupyter like this:

# My Heading

some paragraph, so much interesting

another paragraph, with a some bla bla

# Another big heading

There is **something notable** and then regular words.

Try writing some Python code which reads a text file with a subset of markdown syntax and translates it into suitable Python data structures. See [Markdown basic syntax](#)<sup>339</sup>

- **DO NOT** use special purpose libraries!
- **IMPORTANT:** markdown supports arbitrary depth of subparagraphs: to keep things simple start supporting one level, then two. Doing more would require some kind of level tracking, which could be cumbersome to implement.

<sup>339</sup> <https://www.markdownguide.org/basic-syntax/>

Example - a possible model for the above text could be this one:

```
[3]: parsed = [
 {'type': 'header',
 'level': 1,
 'text': 'My Heading',
 'subelements': [{ 'type': 'paragraph',
 'level': 2,
 'text': 'some paragraph, so much interesting',
 },
 { 'type': 'paragraph',
 'level': 2,
 'rich_text': [('normal', 'another paragraph, with a\u202a'),
 ('normal', 'some bla bla')]
 }
],
 {'type': 'header',
 'level': 1,
 'text': 'Another big heading',
 'subelements': [{ 'type': 'paragraph',
 'level': 2,
 'rich_text': [('normal', 'There is'),
 ('bold', 'something notable'),
 ('normal', 'and then regular words.')]
 }
]
}
```

## Parsing challenge - Other languages

Try developing simple parsers for other languages, like:

- JSON: [syntax<sup>340</sup>](https://restfulapi.net/json-syntax/) (it's very similar to Python)
- HTML web pages: [Basic syntax<sup>341</sup>](https://marksheet.io/html-syntax.html)
- YAML: [Wikipedia<sup>342</sup>](https://en.wikipedia.org/wiki/YAML)
- See others [lightweight markup languages<sup>343</sup>](https://en.wikipedia.org/wiki/Lightweight_markup_language)

**DO NOT** use special purpose libraries!

**IMPORTANT:** Many of these languages support arbitrary depth of subparagraphs: to keep things simple start supporting one level, then two. Doing more would require some kind of level tracking, which could be cumbersome to implement.

---

<sup>340</sup> <https://restfulapi.net/json-syntax/>

<sup>341</sup> <https://marksheet.io/html-syntax.html>

<sup>342</sup> <https://en.wikipedia.org/wiki/YAML>

<sup>343</sup> [https://en.wikipedia.org/wiki/Lightweight\\_markup\\_language](https://en.wikipedia.org/wiki/Lightweight_markup_language)

## CSV Challenge - Over the top

With your friends, you're opening a start-up for tourists who like mountain hiking.

You decide to focus on the north-east region of Italy and develop an app: one of the first tasks is to collect in a table all the mountain peaks with the names in italian, german, latitude, longitude and elevation.

You take some data from OpenStreetMap ( [openstreetmap.org<sup>344</sup>](https://openstreetmap.org) ) the free world map made by volunteers (*OSM* for short). As data format, you choose an CSV export generated by [SLIPO Project<sup>345</sup>](http://slipo.eu/).

### Over the top 1. reading OpenStreetMap data

Have a look at `osm.csv` file, try also to open it with [LibreOffice<sup>346</sup>](#) or Microsoft Office

Then implement function `read_osm` which reads a given CSV file with a `csv.DictReader` and just PRINTS ONLY the peaks (with `pprint`).

- At this stage you can just PRINT the whole retrieved dictionary, we will extract stuff later.

You should see something like this:

- **NOTE 1:** here we show only some printed rows:
- **NOTE 2:** according to the python version you have, you might see instead regular dictionaries instead of `OrderedDict`

```
OrderedDict([("ID", "node/26862480"),
 ("NAME", "Alpe di Succiso"),
 ("CATEGORY", "TOURISM"),
 ("SUBCATEGORY", "PEAK"),
 ("LON", "10.1955113"),
 ("LAT", "44.3327854"),
 ("SRID", "4326"),
 ("WKT", "POINT (10.19551130000000 44.33278540000000)"),
 ("INTERNATIONAL_NAME", ""),
 ("STREET", ""),
 ("WIKIPEDIA", "it:Alpe di Succiso"),
 ("PHONE", ""),
 ("CITY", ""),
 ("EMAIL", ""),
 ("ALTERNATIVE_NAME", ""),
 ("OPENING_HOURS", ""),
 ("DESCRIPTION", ""),
 ("WEBSITE", ""),
 ("LAST_UPDATE", ""),
 ("OPERATOR", ""),
 ("POSTCODE", ""),
 ("COUNTRY", ""),
 ("FAX", ""),
 ("IMAGE", ""),
 ("HOUSENUMBER", ""),
 ("OTHER_TAGS",
 {"PDOP": "1.87", "natural": "peak", "importance": "regional", "name": "Alpe di Succiso", "source": "survey", "wikidata": "Q1810954", "ele": "2016"})])
```

(continues on next page)

<sup>344</sup> <https://openstreetmap.org>

<sup>345</sup> <http://slipo.eu/>

<sup>346</sup> <https://www.libreoffice.org/>

(continued from previous page)

```
OrderedDict([('ID', 'node/26862538'),
 ('NAME', 'Becco di Filadonna'),
 ('CATEGORY', 'TOURISM'),
 ('SUBCATEGORY', 'PEAK'),
 ('LON', '11.1934654'),
 ('LAT', '45.9636324'),
 ('SRID', '4326'),
 .
 .
 .])
```

```
[4]:
import csv

def read_osm(in_filename):
 raise Exception('TODO IMPLEMENT ME !')

read_osm('osm.csv')
```

## Over the top 2. extract peak

Implement function `extract_peak` which given a peak as a raw dictionary, RETURN the list of relevant values in this order: italian name, german name, latitude, longitude, elevation

Note elevation, italian and german names are inside the field `other_tags` as `name:it`, `name:de`, `ele`

- **WARNING 1:** `name:it` is not always present! In such cases use `NAME` field from the main dictionary
- **WARNING 2:** `name:de` is not always present! In such cases put an empty string
- **HINT:** the field `other_tags` looks very much like an embedded JSON. To parse it quickly, use the function `json.loads`<sup>347</sup>, which takes a string as input and outputs a Python object, in this case you will obtain a dictionary. NOTE THE `s` at the end of `json.loads!!`

Example - given:

```
d = OrderedDict([('ID', 'node/26862713'),
 ('NAME', 'Cima Undici'),
 ('CATEGORY', 'TOURISM'),
 ('SUBCATEGORY', 'PEAK'),
 ('LON', '12.3783333'),
 ('LAT', '46.6363889'),
 ('SRID', '4326'),
 ('WKT', 'POINT (12.37833330000001 46.6363889)'),
 ('INTERNATIONAL_NAME', ''),
 ('STREET', ''),
 ('WIKIPEDIA', 'it:Cima Undici'),
 ('PHONE', ''),
 ('CITY', ''),
 ('EMAIL', ''),
 ('ALTERNATIVE_NAME', ''),
 ('OPENING_HOURS', ''),
 ('DESCRIPTION', ''),
 ('WEBSITE', '')])
```

(continues on next page)

<sup>347</sup> <https://docs.python.org/3/library/json.html>

(continued from previous page)

```
('LAST_UPDATE', ''),
('OPERATOR', ''),
('POSTCODE', ''),
('COUNTRY', ''),
('FAX', ''),
('IMAGE', ''),
('HOUSENUMBER', ''),
('OTHER_TAGS',
'{"name:de":"Elferkofel","natural":"peak","name":"Cima ' +
'Undici","name:it":"Cima ' +
'Undici","wikidata":"Q628931","ele":"3090"}'))
```

You should obtain:

```
>>> extract_peak(d)
['Cima Undici', 'Elferkofel', 46.6363889, 12.3783333, 3090.0]
```

**NOTE:** numbers should be numbers, not strings!

[5]:

```
import json

def extract_peak(rawd):
 """Takes a dictionary and RETURN a list
 """
 raise Exception('TODO IMPLEMENT ME !')

from collections import OrderedDict
d = OrderedDict([
 ('ID', 'node/26862713'),
 ('NAME', 'Cima Undici'),
 ('CATEGORY', 'TOURISM'),
 ('SUBCATEGORY', 'PEAK'),
 ('LON', '12.3783333'),
 ('LAT', '46.6363889'),
 ('SRID', '4326'),
 ('WKT', 'POINT (12.378333300000001 46.6363889)'),
 ('INTERNATIONAL_NAME', ''),
 ('STREET', ''),
 ('WIKIPEDIA', 'it:Cima Undici'),
 ('PHONE', ''),
 ('CITY', ''),
 ('EMAIL', ''),
 ('ALTERNATIVE_NAME', ''),
 ('OPENING_HOURS', ''),
 ('DESCRIPTION', ''),
 ('WEBSITE', ''),
 ('LAST_UPDATE', ''),
 ('OPERATOR', ''),
 ('POSTCODE', ''),
 ('COUNTRY', ''),
 ('FAX', ''),
 ('IMAGE', ''),
 ('HOUSENUMBER', ''),
 ('OTHER_TAGS',
'{"name:de":"Elferkofel","natural":"peak","name":"Cima ' +
'Undici","name:it":"Cima ' +
'Undici","wikidata":"Q628931","ele":"3090"}'))
```

(continues on next page)

(continued from previous page)

```
extract_peak(d)
```

### Over the top 3. write file

Implement function `write_peaks` so it calls `extract_peak` and writes the obtained lists into `peaks.csv` with a `csv.writer` (so this time we write lists, not dictionaries!)

**REMEMBER** to put also the header

First lines should be like (for complete expected file see [expected-peaks.csv](#))

```
name_it,name_de,latitude,longitude,elevation
Alpe di Succiso,,44.3327854,10.1955113,2016.0
Becco di Filadonna,,45.9636324,11.1934654,2150.0
Bechel di Sopra,,46.6077439,12.0444775,2794.0
Catinaccio d'Antermoia,Kesselkogel,46.4740893,11.6438283,3004.0
Cima Ambrizzola,,46.4791667,12.0980556,2715.0
Cima Bastioni,,46.4851159,12.2678531,2926.0
Cima Brenta,,46.1797021,10.900036,3151.0
Cima Cadin di San Lucano,,46.5776149,12.2882724,2839.0
Cima d'Asta,,46.1766183,11.6052937,2847.0
Cima dei Preti,,46.3423245,12.4210592,2707.0
Cima della Vezzana,,46.2899137,11.8297409,3192.0
Cima Dodici,,45.9976856,11.4680336,2337.0
Cima Mora,,46.240557,12.3431523,1940.0
Cima Palon,,45.7922301,11.1765372,2232.0
Cima Pape,,46.3343734,11.9283766,2503.0
Punta di mezzodì,,45.731185,11.1380772,1858.0
Cima Presanella,Cima Presanella,46.2199321,10.6641189,3556.0
Cima Rolle,Rollspitze,46.9463889,11.5077778,2800.0
Cima Tosa,,46.1565222,10.8711276,3136.0
Cima Undici,Elferkofel,46.6363889,12.3783333,3090.0
.
.
```

[7]:

```
import csv

def write_peaks(in_filename):
 raise Exception('TODO IMPLEMENT ME !')

write_peaks('osm.csv')
```

## 8.2 Visualization

### 8.2.1 Visualization 1

#### Download exercises zip

Browse files online<sup>348</sup>

#### Introduction

We will review the famous library Matplotlib which allows to display a variety of charts, and it is the base of many other visualization libraries.

#### What to do

- unzip exercises in a folder, you should get something like this:

```
visualization
 visualization1.ipynb
 visualization1-sol.ipynb
 visualization2-chal.ipynb
 visualization-images.ipynb
 visualization-images-sol.ipynb
 jupman.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `visualization/visualization1.ipynb`

**WARNING 2:** DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

<sup>348</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/visualization>

## First example

Let's start with a very simple plot:

```
[2]: # this is *not* a python command, it is a Jupyter-specific magic command,
to tell jupyter we want the graphs displayed in the cell outputs
%matplotlib inline

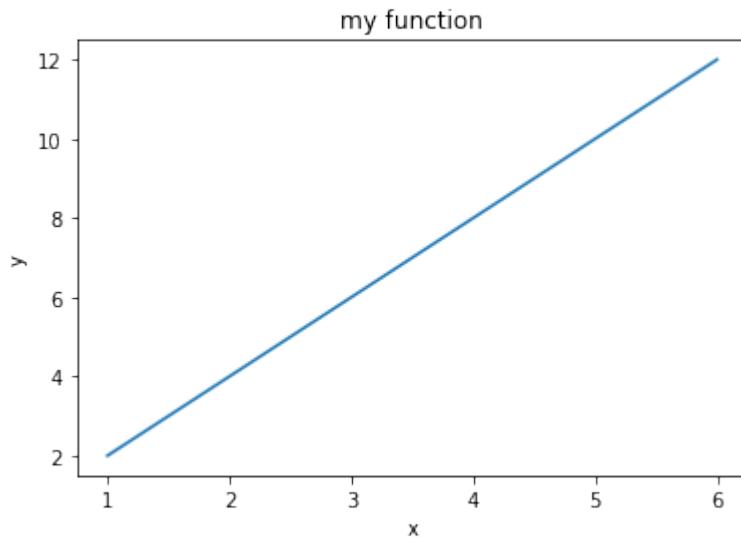
imports matplotlib
import matplotlib.pyplot as plt

we can give coordinates as simple numberlists
this are couples for the function y = 2 * x
xs = [1, 2, 3, 4, 5, 6]
ys = [2, 4, 6, 8, 10, 12]

plt.plot(xs, ys)

we can add this after plot call, it doesn't matter
plt.title("my function")
plt.xlabel('x')
plt.ylabel('y')

prevents showing '<matplotlib.text.Text at 0x7fbcf3c4ff28>' in Jupyter
plt.show()
```



## Plot style

To change the way the line is displayed, you can set dot styles with another string parameter. For example, to display red dots, you would add the string `r o`, where `r` stands for red and `o` stands for dot.

```
[3]: %matplotlib inline
import matplotlib.pyplot as plt

xs = [1, 2, 3, 4, 5, 6]
ys = [2, 4, 6, 8, 10, 12]
```

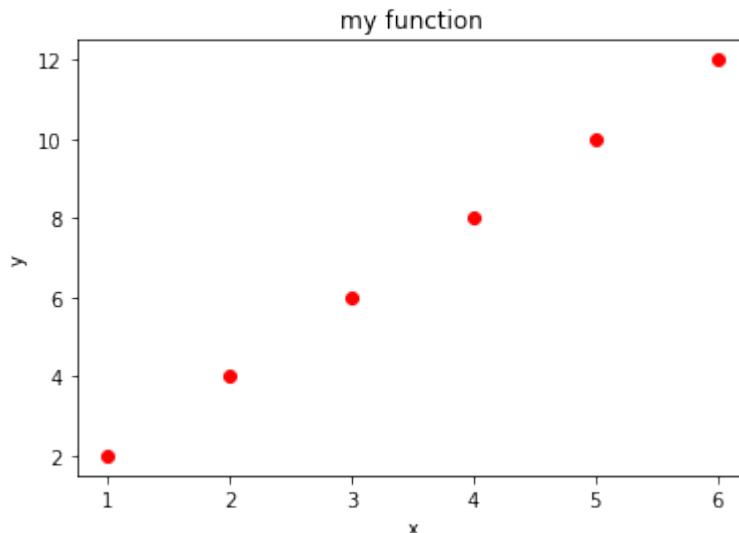
(continues on next page)

(continued from previous page)

```
plt.plot(xs, ys, 'ro') # NOW USING RED DOTS

plt.title("my function")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



## x power 2 exercise

Try to display the function  $y = x^{**}2$  (x power 2) using green dots and for integer xs going from -10 to 10

```
[4]: # write here the solution
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[5]: # SOLUTION
```

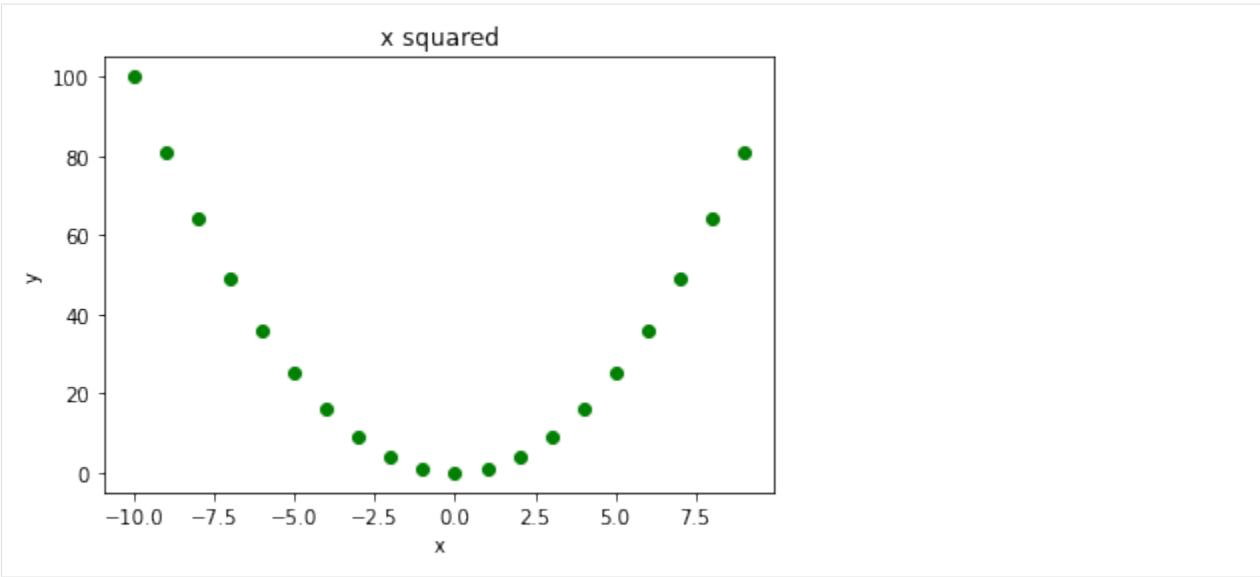
```
%matplotlib inline
import matplotlib.pyplot as plt

xs = range(-10, 10)
ys = [x**2 for x in xs]

plt.plot(xs, ys, 'go')

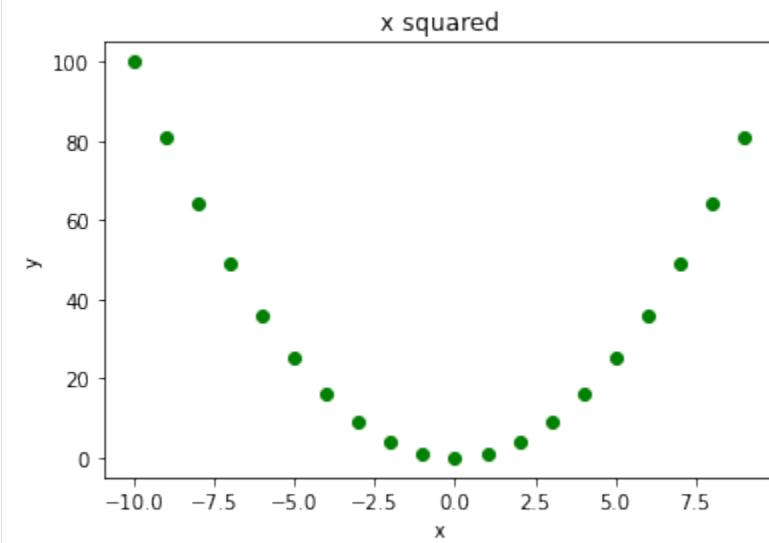
plt.title("x squared")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



&lt;/div&gt;

[5] :



## Axis limits

If you want to change the x axis, you can use plt.xlim:

```
[6]: %matplotlib inline
import matplotlib.pyplot as plt

xs = [1, 2, 3, 4, 5, 6]
ys = [2, 4, 6, 8, 10, 12]

plt.plot(xs, ys, 'ro')
```

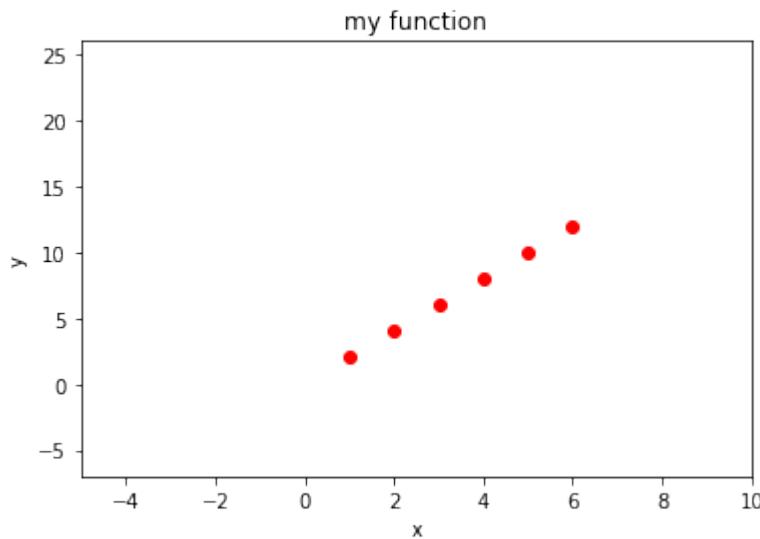
(continues on next page)

(continued from previous page)

```
plt.title("my function")
plt.xlabel('x')
plt.ylabel('y')

plt.xlim(-5, 10) # SETS LOWER X DISPLAY TO -5 AND UPPER TO 10
plt.ylim(-7, 26) # SETS LOWER Y DISPLAY TO -7 AND UPPER TO 26

plt.show()
```



## Axis size

```
[7]: %matplotlib inline
import matplotlib.pyplot as plt

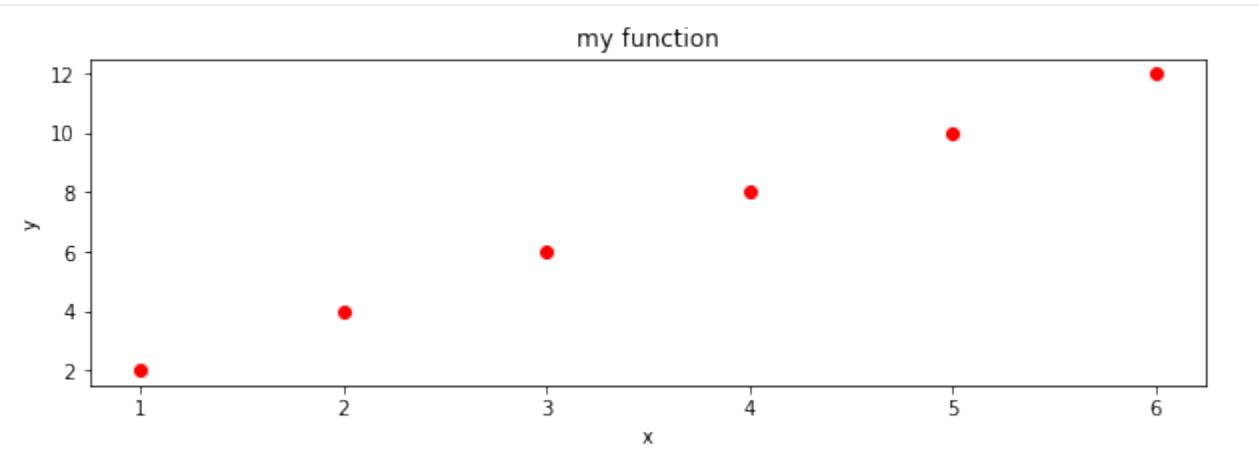
xs = [1, 2, 3, 4, 5, 6]
ys = [2, 4, 6, 8, 10, 12]

fig = plt.figure(figsize=(10,3)) # width: 10 inches, height 3 inches

plt.plot(xs, ys, 'ro')

plt.title("my function")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



### Changing tick labels

You can also change labels displayed on ticks on axis with `plt.xticks` and `plt.yticks` functions:

**Note:** instead of `xticks` you might directly use [categorical variables<sup>349</sup>](#) IF you have matplotlib >= 2.1.0

Here we use `xticks` as sometimes you might need to fiddle with them anyway

```
[8]: %matplotlib inline
import matplotlib.pyplot as plt

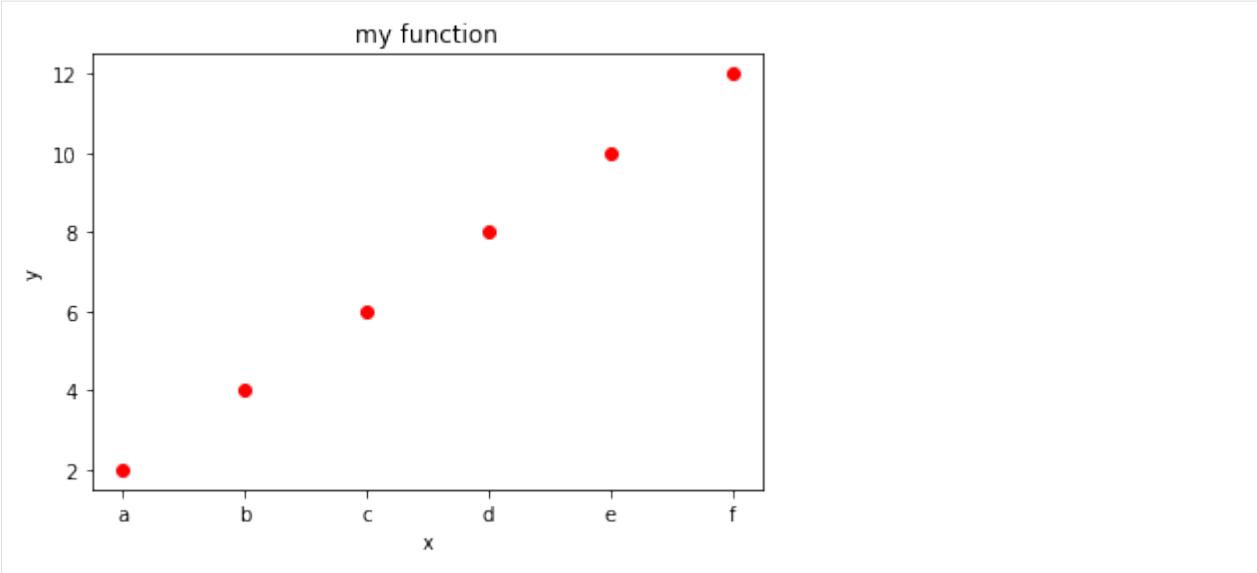
xs = [1, 2, 3, 4, 5, 6]
ys = [2, 4, 6, 8, 10, 12]

plt.plot(xs, ys, 'ro')

plt.title("my function")
plt.xlabel('x')
plt.ylabel('y')

FIRST NEEDS A SEQUENCE WITH THE POSITIONS, THEN A SEQUENCE OF SAME LENGTH WITH_
←LABELS
plt.xticks(xs, ['a', 'b', 'c', 'd', 'e', 'f'])
plt.show()
```

<sup>349</sup> [https://matplotlib.org/gallery/lines\\_bars\\_and\\_markers/categorical\\_variables.html](https://matplotlib.org/gallery/lines_bars_and_markers/categorical_variables.html)



## Multiple lines

To overlay multiple lines, you just need to perform several calls to `plt.plot`. Matplotlib will automatically use a different color for each line.

**REMEMBER:** you should call `plt.show` only ONCE at the very end!!

[9]:

```
%matplotlib inline
import matplotlib.pyplot as plt

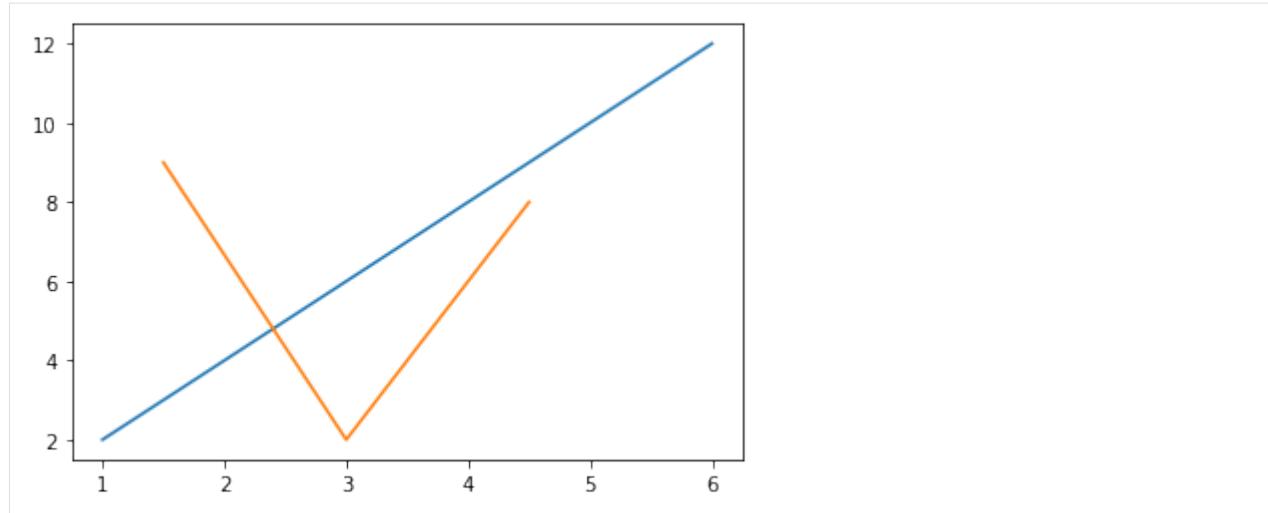
we can give coordinates as simple numberlists
this are couples for the function y = 2 * x
xsa = [1, 2, 3, 4, 5, 6]
ysa = [2, 4, 6, 8, 10, 12]

plt.plot(xsa, ysa)

xsb = [1.5, 3.0, 4.5] # note this other series can have a different number of
 # points at different places
ysb = [9, 2, 8]

plt.plot(xsb, ysb)

plt.show()
```



## Numpy

For functions involving reals, vanilla python starts showing its limits and it's better to switch to numpy library. Matplotlib can easily handle both vanilla python sequences like lists and numpy array. Let's see an example without numpy and one with it.

### Example without numpy

If we only use *vanilla* Python (that is, Python without extra libraries like numpy), to display the function  $y = 2x + 1$  we can come up with a solution like this

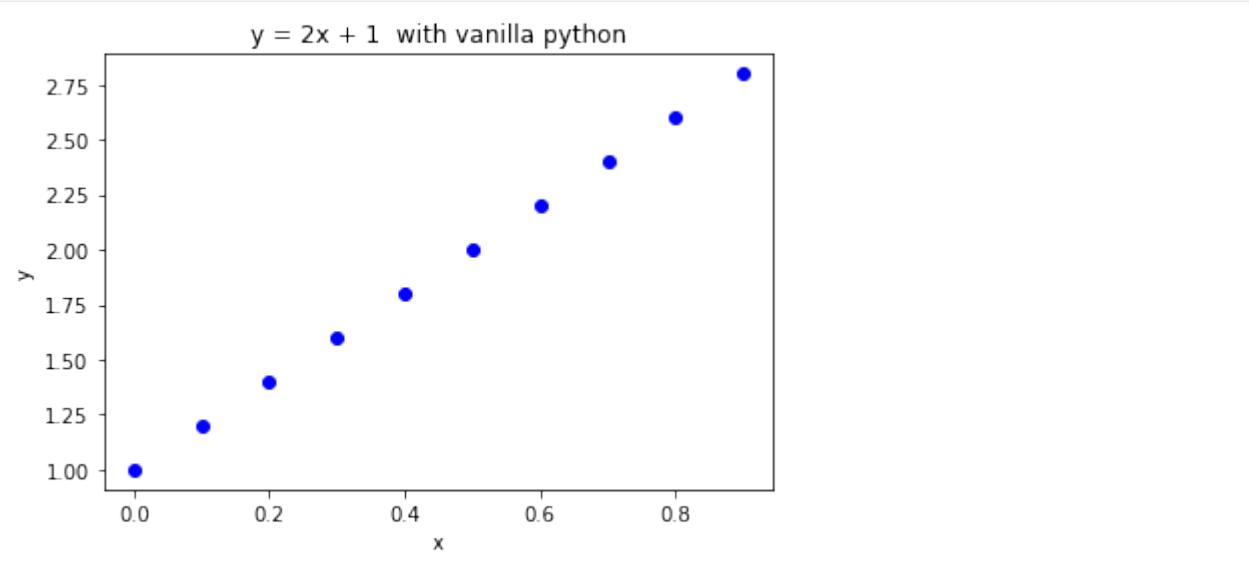
```
[10]: %matplotlib inline
import matplotlib.pyplot as plt

xs = [x*0.1 for x in range(10)] # notice we can't do a range with float increments
(and it would also introduce rounding errors)
ys = [(x * 2) + 1 for x in xs]

plt.plot(xs, ys, 'bo')

plt.title("y = 2x + 1 with vanilla python")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



### Example with numpy

With numpy, we have at our disposal several new methods for dealing with arrays.

First we can generate an interval of values<sup>350</sup> with one of these methods.

Sine Python range does not allow float increments, we can use np.arange:

```
[11]: import numpy as np

xs = np.arange(0,1.0,0.1)
xs

[11]: array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

Equivalently, we could use np.linspace:

```
[12]: xs = np.linspace(0,0.9,10)

xs

[12]: array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

Numpy allows us to easily write functions on arrays in a natural manner. For example, to calculate ys we can now do like this:

```
[13]: ys = 2*xs + 1

ys

[13]: array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8])
```

Let's put everything together:

<sup>350</sup> <https://en.softpython.org/matrices-numpy/matrices-numpy1-sol.html#arange-and-linspace-sequences>

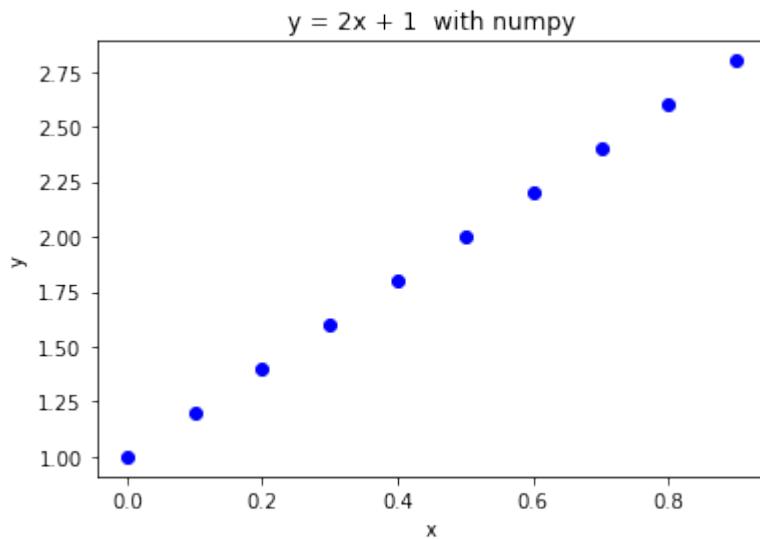
```
[14]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

xs = np.linspace(0,0.9,10) # left end: 0 *included* right end: 0.9 *included*
number of values: 10
ys = 2*xs + 1

plt.plot(xs, ys, 'bo')

plt.title("y = 2x + 1 with numpy")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



### Exercise - $\sin(x) + 3$

⊕⊕⊕ Try to display the function  $y = \sin(x) + 3$  for  $x$  at  $\pi/4$  intervals, starting from 0. Use exactly 8 ticks.

**NOTE:** 8 is the *number of x ticks* (telecom people would use the term ‘samples’), **NOT** the  $x$  of the last tick !!

- try to solve it *without* using numpy. For  $\pi$ , use constant `math.pi` (first you need to import `math` module)
  - try to solve it *with* numpy. For  $\pi$ , use constant `np.pi` (which is exactly the same as `math.pi`)
- b.1) solve it with `np.arange`
- b.2) solve it with `np.linspace`
- For each tick, use the label sequence " $0\pi/4$ ", " $1\pi/4$ ", " $2\pi/4$ ", " $3\pi/4$ ", " $4\pi/4$ ", " $5\pi/4$ ", .... Obviously writing them by hand is easy, try instead to devise a method that works for any number of ticks. What is changing in the sequence? What is constant? What is the type of the part changes ? What is final type of the labels you want to obtain ?
  - If you are in the mood, try to display them better like  $0, \pi/4, \pi/2, \pi, 3\pi/4, \pi, 5\pi/4$  possibly using Latex (requires some search, [this example<sup>351</sup>](#) might be a starting point)

<sup>351</sup> <https://stackoverflow.com/a/40642200>

**NOTE:** Latex often involves the usage of the \ bar, like in  $\frac{2}{3}$ . If we use it directly, Python will interpret \f as a special character and will not send to the Latex processor the string we meant:

```
[15]: '\frac{2,3}'
[15]: '\x0crac{2,3}'
```

One solution would be to double the slashes, like this:

```
[16]: '\\"frac{2,3}'
[16]: '\\\\frac{2,3}'
```

An even better one is to prepend the string with the r character, which allows to write slashes only once:

```
[17]: r'\\frac{2,3}'
[17]: '\\\\frac{2,3}'
```

```
[18]: # write here solution for a) y = sin(x) + 3 with vanilla python
```

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[19]: # SOLUTION a) y = sin(x) + 3 with vanilla python

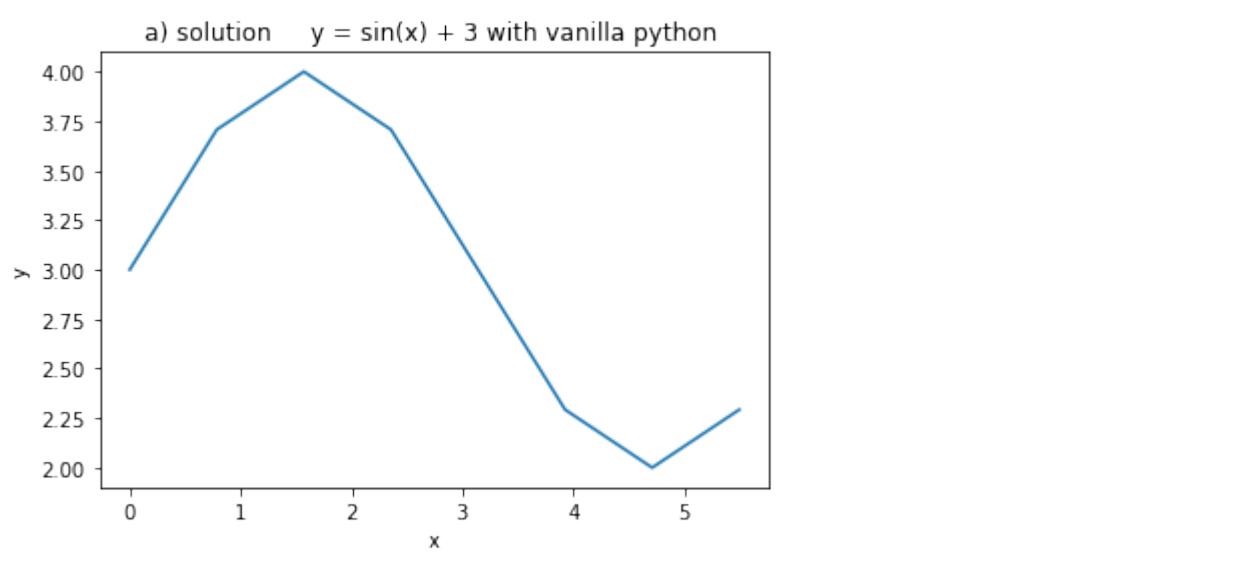
%matplotlib inline
import matplotlib.pyplot as plt
import math

xs = [x * (math.pi)/4 for x in range(8)]
ys = [math.sin(x) + 3 for x in xs]

plt.plot(xs, ys)

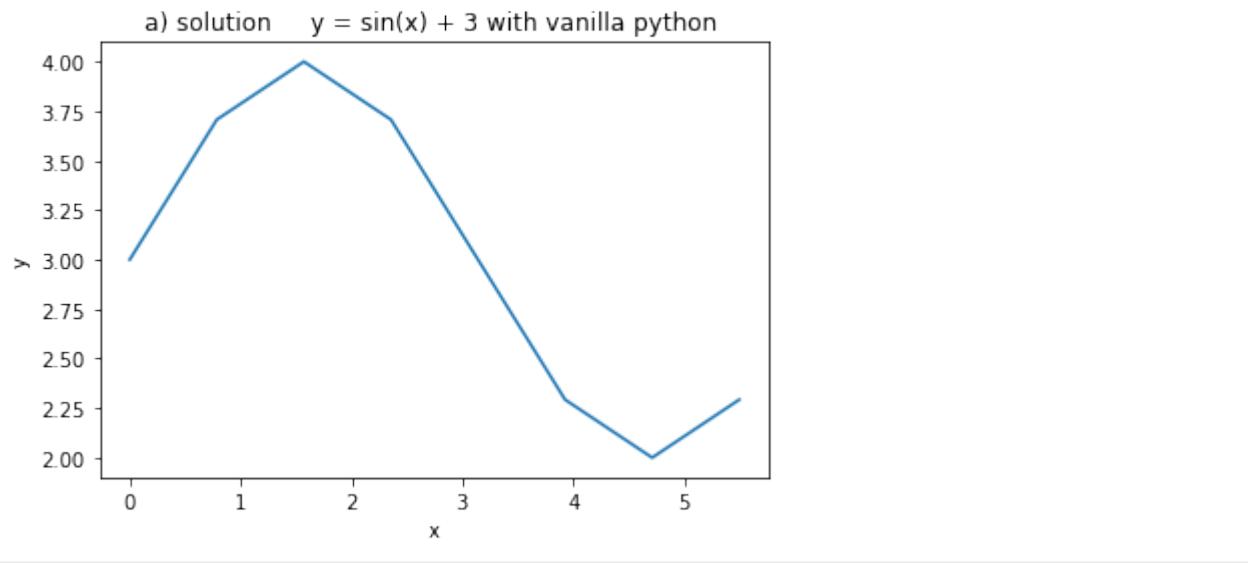
plt.title("a) solution y = sin(x) + 3 with vanilla python ")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



&lt;/div&gt;

[19]:



[20]: # write here solution b.1) y = sin(x) + 3 with numpy, arange

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[21]: # SOLUTION b.1) y = sin(x) + 3 with numpy, linspace

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

left end = 0 right end = 7/4 pi 8 points
notice numpy.pi is exactly the same as vanilla math.pi
```

(continues on next page)

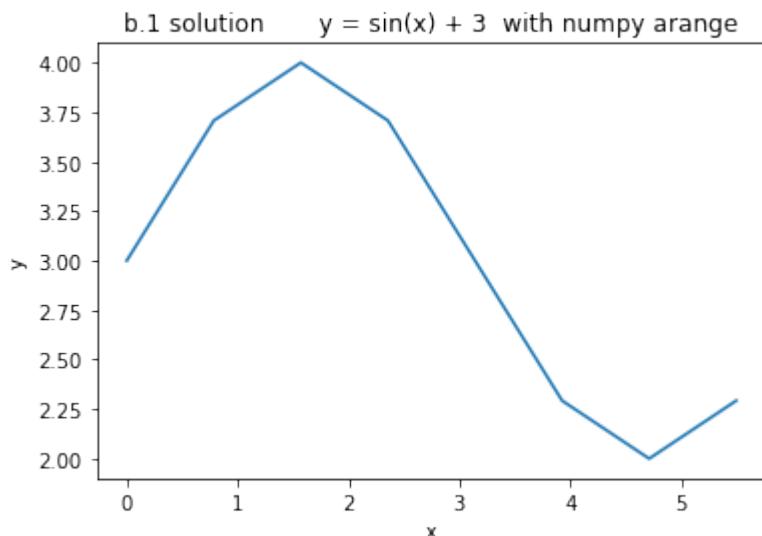
(continued from previous page)

```
xs = np.arange(0, # included
 8 * np.pi/4, # *not* included (we put 8, as we actually want 7 to be_
 ↪ included)
 np.pi/4)
ys = np.sin(xs) + 3 # notice we know operate on arrays. All numpy functions can_
 ↪ operate on them

plt.plot(xs, ys)

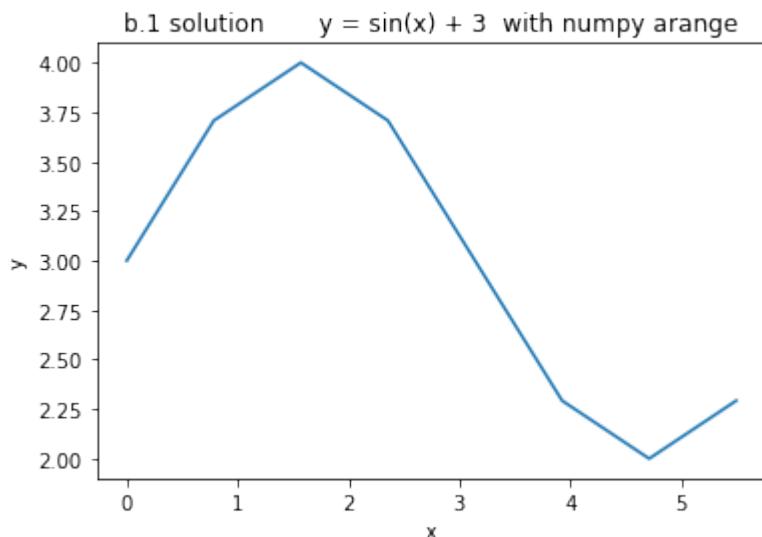
plt.title("b.1 solution y = sin(x) + 3 with numpy arange")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



&lt;/div&gt;

[21]:



```
[22]: # write here solution b.2) y = sin(x) + 3 with numpy, linspace
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[23]: # SOLUTION b.2) y = sin(x) + 3 with numpy, linspace
```

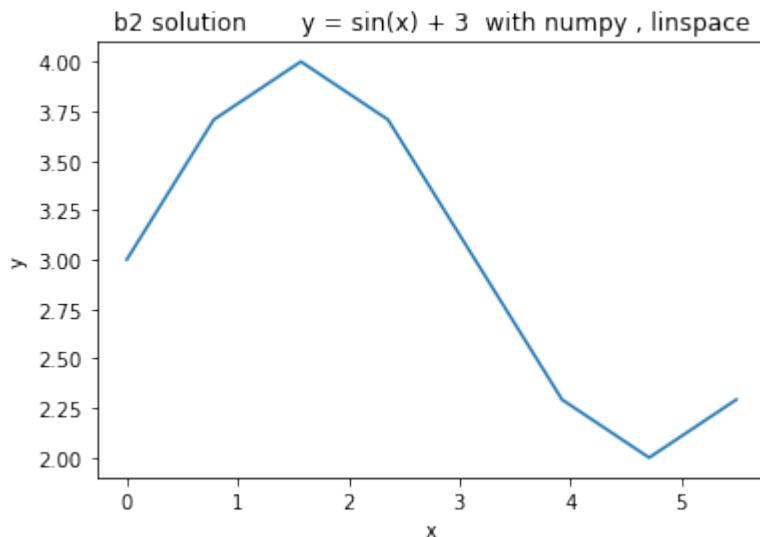
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

left end = 0 right end = 7/4 pi 8 points
notice numpy.pi is exactly the same as vanilla math.pi
xs = np.linspace(0, (np.pi/4) * 7, 8)
ys = np.sin(xs) + 3 # notice we know operate on arrays. All numpy functions can
 # operate on them

plt.plot(xs, ys)

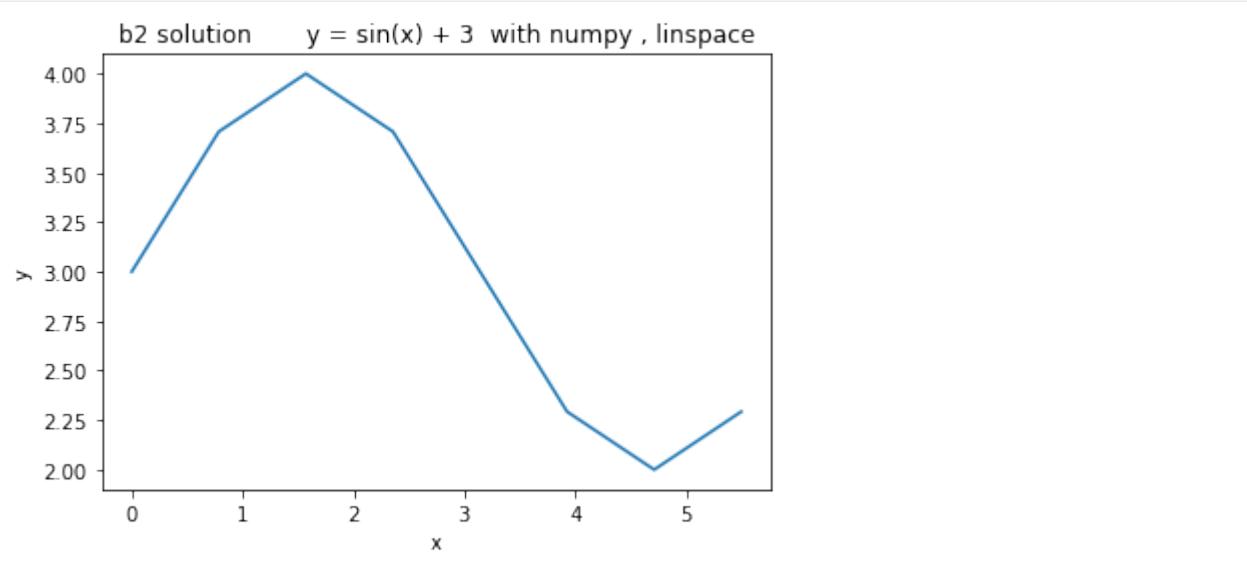
plt.title("b2 solution y = sin(x) + 3 with numpy , linspace")
plt.xlabel('x')
plt.ylabel('y')

plt.show()
```



</div>

```
[23]:
```



```
[24]: # write here solution c) y = sin(x) + 3 with numpy and pi xlabel
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[25]: # SOLUTION c) y = sin(x) + 3 with numpy and pi xlabel

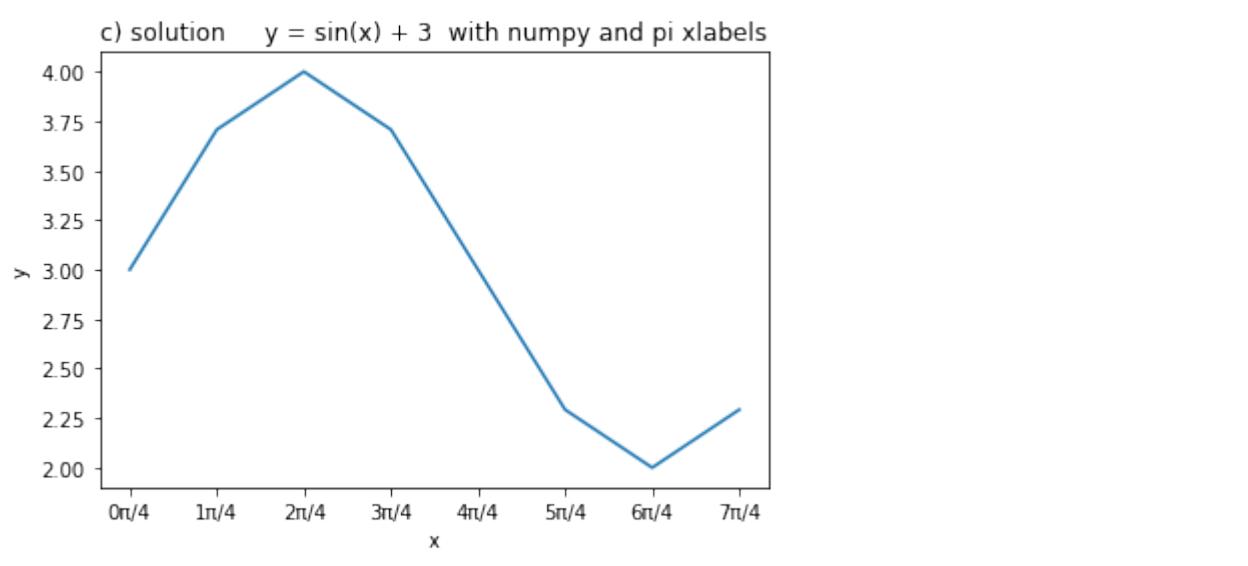
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

xs = np.linspace(0, (np.pi/4) * 7 , 8) # left end = 0 right end = 7/4 pi 8 points
ys = np.sin(xs) + 3 # notice we know operate on arrays. All numpy functions can_
operate on them

plt.plot(xs, ys)

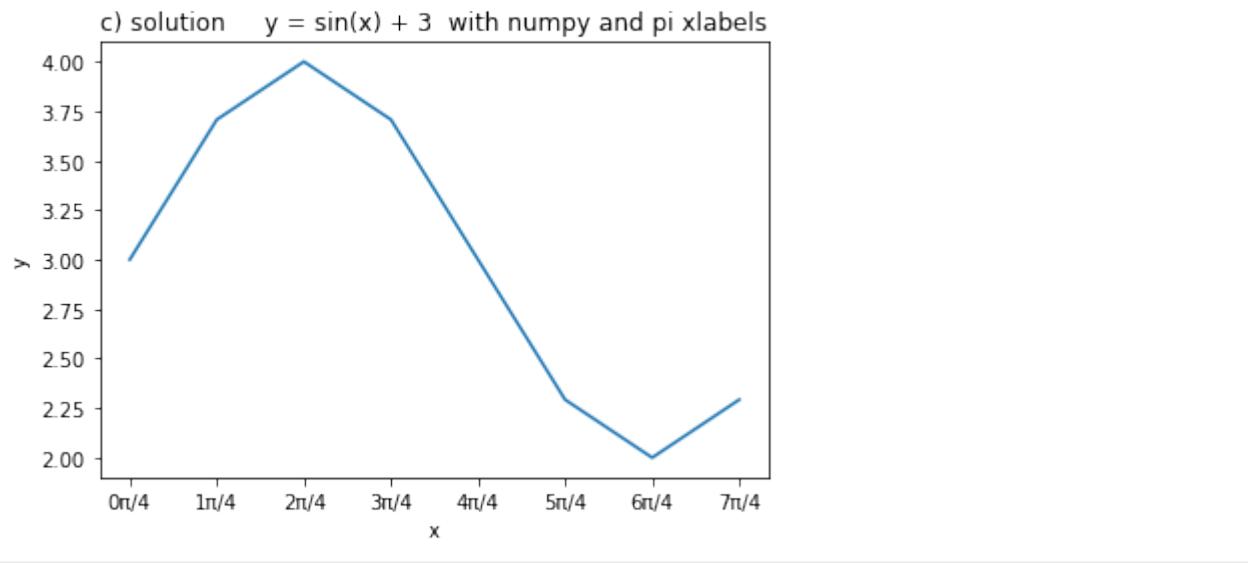
plt.title("c) solution y = sin(x) + 3 with numpy and pi xlabel")
plt.xlabel('x')
plt.ylabel('y')

FIRST NEEDS A SEQUENCE WITH THE POSITIONS, THEN A SEQUENCE OF SAME LENGTH WITH_
LABELS
plt.xticks(xs, ["%sπ/4" % x for x in range(8)])
plt.show()
```



&lt;/div&gt;

[25] :



## Bar plots

First look at this example, then proceed with the next exercises

```
[26]: import numpy as np
import matplotlib.pyplot as plt

xs = [1, 2, 3, 4]
ys = [7, 5, 8, 2]

plt.bar(xs, ys,
 0.5, # the width of the bars
```

(continues on next page)

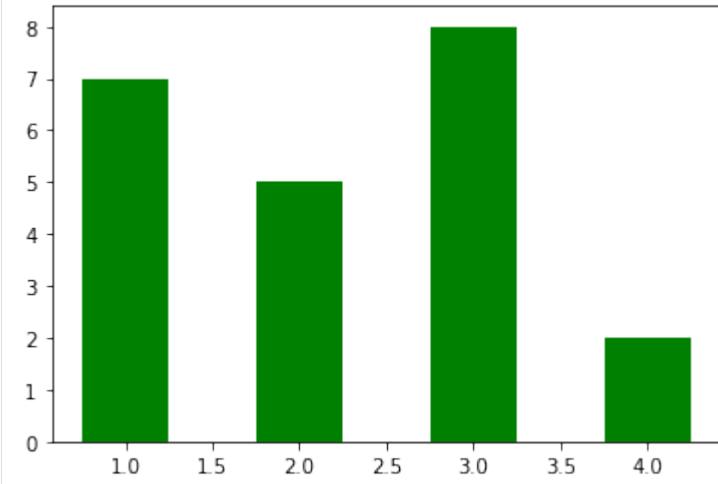
(continued from previous page)

```

color='green', # someone suggested the default blue color is depressing, so
→let's put green
 align='center') # bars are centered on the xtick

plt.show()

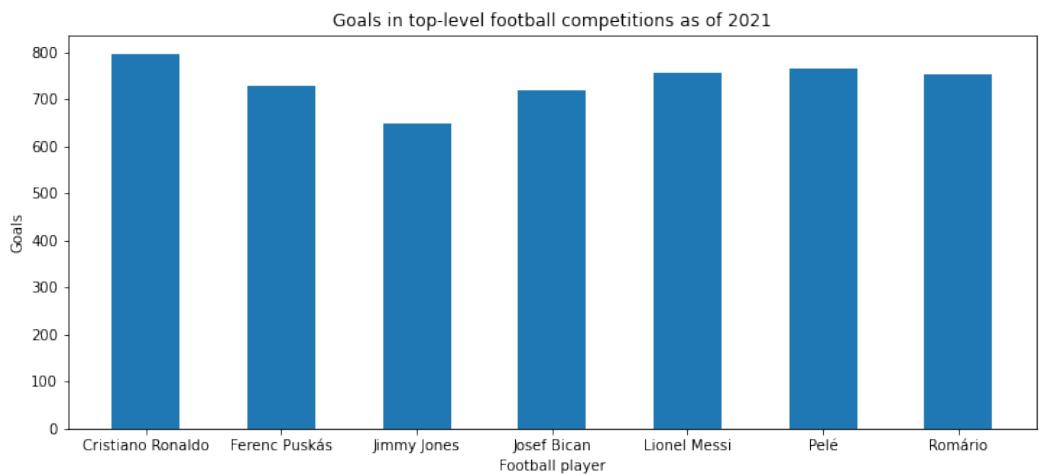
```



### Exercise - goleadors

⊕⊕ Display a bar plot of football players and their total goals in top-level football competitions (as of 2021), with their names sorted alphabetically

**REMEMBER** title and axis labels, make sure all texts are clearly visible



EXPECTED OUTPUT:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[27]:

```

import numpy as np
import matplotlib.pyplot as plt

```

(continues on next page)

(continued from previous page)

```

players = {
 "Cristiano Ronaldo" : 795,
 "Pelé" : 765,
 "Lionel Messi" : 755,
 "Romário" : 753,
 "Ferenc Puskás" : 729,
 "Josef Bican": 720,
 "Jimmy Jones": 647
}

#players={"Zlatan Ibrahimović": 566, "Alfredo Di Stéfano": 530}

write here

fig = plt.figure(figsize=(12,5))

xs = np.arange(len(players))

xs_labels = sorted(players.keys())

ys = [players[n] for n in xs_labels]

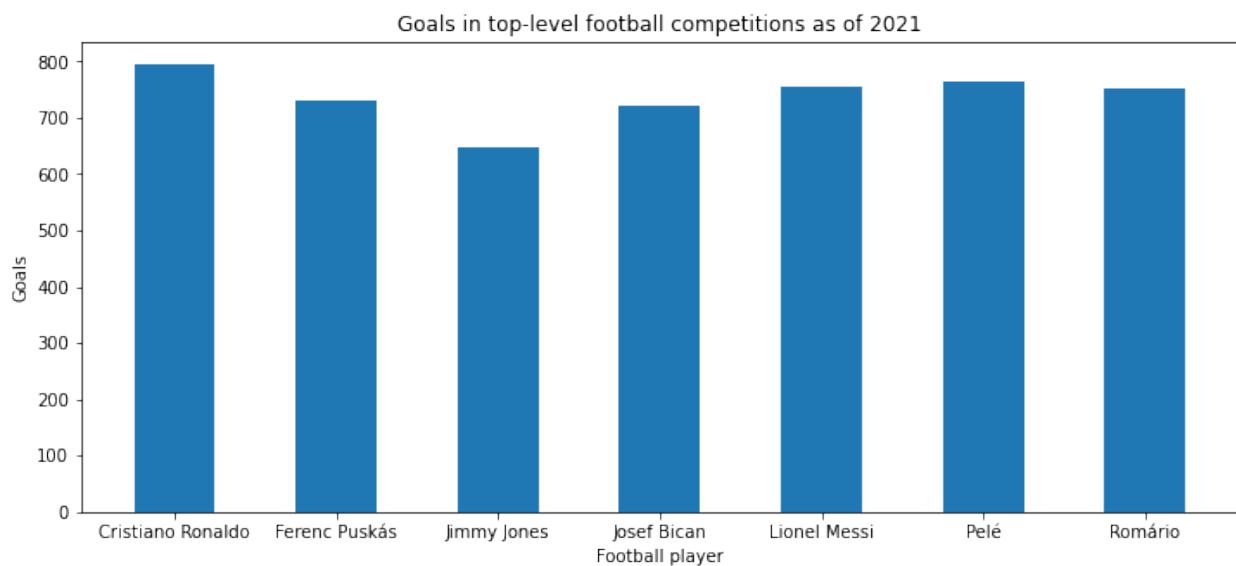
plt.bar(xs, ys, 0.5, align='center')

plt.title("Goals in top-level football competitions as of 2021")
plt.xticks(xs, xs_labels)

plt.xlabel('Football player')
plt.ylabel('Goals')

plt.show()

```



&lt;/div&gt;

[27]:

```
import numpy as np
import matplotlib.pyplot as plt

players = {
 "Cristiano Ronaldo" : 795,
 "Pelé" : 765,
 "Lionel Messi" : 755,
 "Romário" : 753,
 "Ferenc Puskás" : 729,
 "Josef Bican": 720,
 "Jimmy Jones": 647
}

#players={"Zlatan Ibrahimović": 566, "Alfredo Di Stéfano": 530}

write here
```

### Exercise - chemical elements

⊕⊕⊕ Given multiple lists representig data about chemical elements, show a bar plot where elements are sorted alphabetically according to their name.

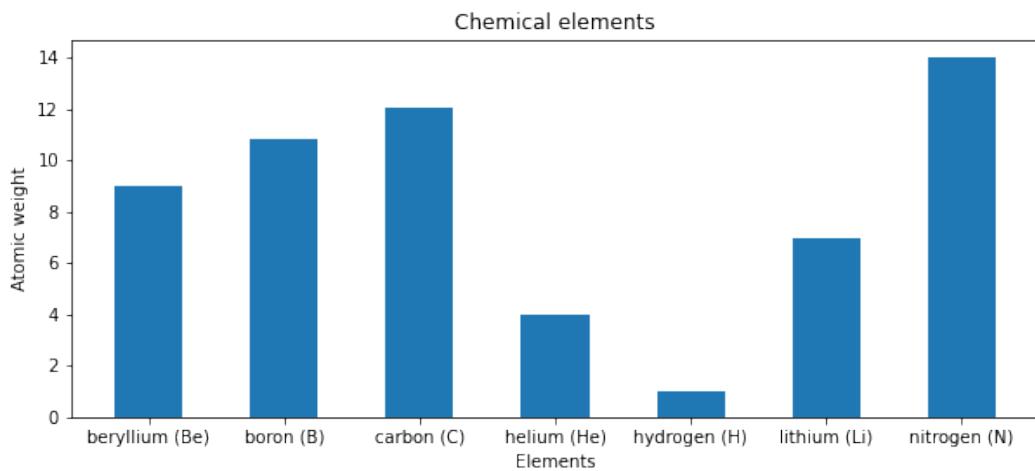
- show elements as *name (symbol)*

**REMEMBER** title and axis labels, make sure all texts are clearly visible

**HINT:** This is more challenging, you need some sort trick - First read the [Python documentation](#)<sup>352</sup> and then:

1. create a list of couples (list of tuples) where each tuple is the node identifier and the corresponding weight
2. sort the list by using the second value of the tuples as a key.

**EXPECTED OUTPUT:**



<sup>352</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[28]:

```
import numpy as np
import matplotlib.pyplot as plt

symbols = ['H', 'He', 'Li', 'Be', 'B', 'C', 'N']
names = ['hydrogen', 'helium', 'lithium', 'beryllium', 'boron', 'carbon',
 ↪'nitrogen']
atomic_weight = [1.008, 4.0026, 6.94, 9.0122, 10.81, 12.011, 14.007]

write here

fig = plt.figure(figsize=(10,4))

xs = np.arange(len(symbols))

coords = [(names[i] + " " + "(" + symbols[i] + ") ", atomic_weight[i]) for i in
 ↪range(len(symbols))]

coords.sort(key=lambda c: c[0])

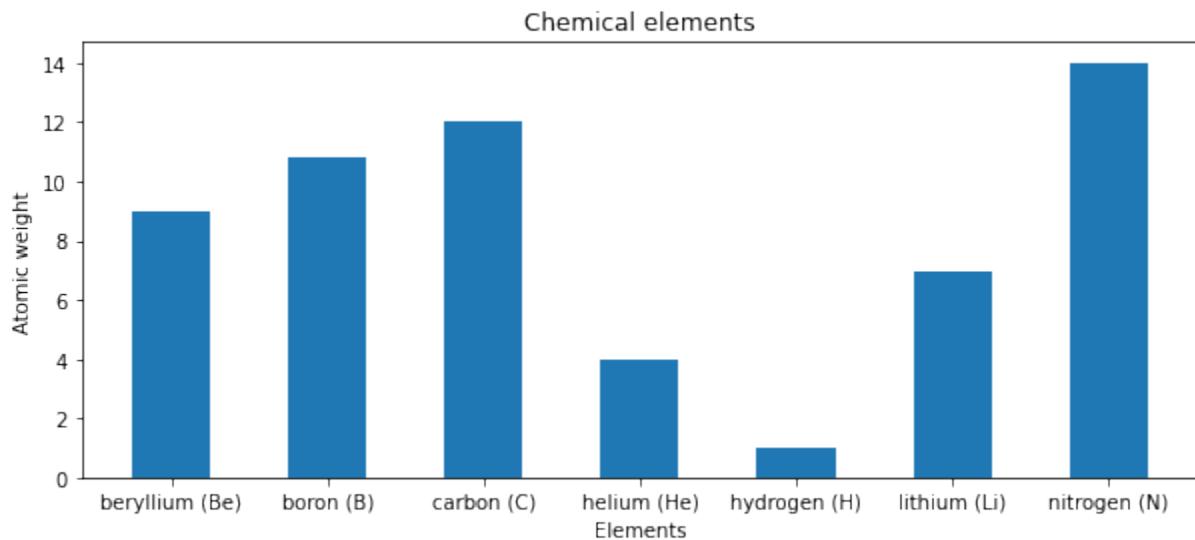
ys = [c[1] for c in coords]

plt.bar(xs, ys, 0.5, align='center')

plt.title("Chemical elements")
plt.xticks(xs, [c[0] for c in coords])

plt.xlabel('Elements')
plt.ylabel('Atomic weight')

plt.show()
```



</div>

[28]:

```
import numpy as np
import matplotlib.pyplot as plt

symbols = ['H', 'He', 'Li', 'Be', 'B', 'C', 'N']
names = ['hydrogen', 'helium', 'lithium', 'beryllium', 'boron', 'carbon',
←'nitrogen']
atomic_weight = [1.008, 4.0026, 6.94, 9.0122, 10.81, 12.011, 14.007]

write here
```

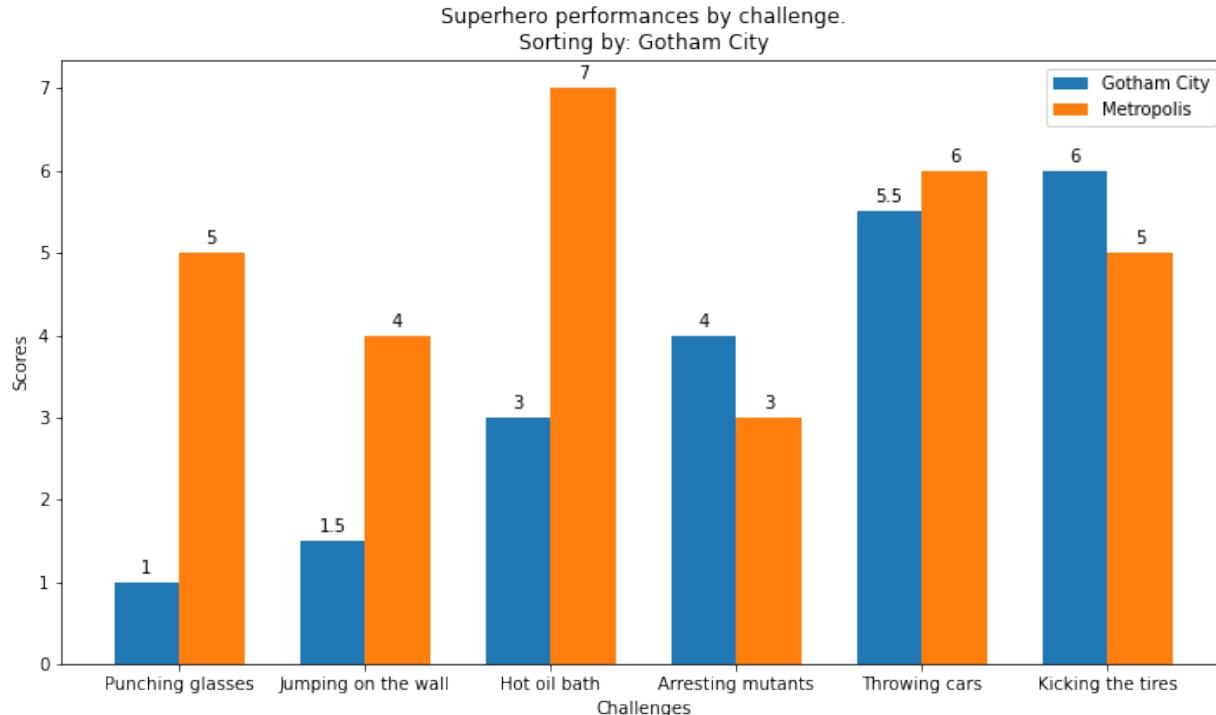
## Exercise - superheroes

⊕⊕⊕⊕ Each year a contest between the super-heroes of two crime-ridden cities is held. The superheroes perform several challenges and each city receives a score. At the end, the mayor of each city wants to see how its city compared against the other city. The mayor wants you to show the performances in **sorted** order with respect to the mayor's city, while showing also the performance of the other city for comparison.

Look at [this example<sup>353</sup>](#), and make a double bar chart

- specify the city in the title
- remember x and y axis labels

EXPECTED OUTPUT (here the performances of Gotham City are shown in sorted order):



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

<sup>353</sup> [https://matplotlib.org/gallery/lines\\_bars\\_and\\_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py](https://matplotlib.org/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py)

[29]:

```
import matplotlib.pyplot as plt
import numpy as np

d = {
 'Punching glasses' : (1, 5),
 'Kicking the tires' : (6, 5),
 'Throwing cars' : (5.5, 6),
 'Hot oil bath' : (3, 7),
 'Jumping on the wall': (1.5, 4),
 'Arresting mutants' : (4, 3),
}

city = 'Gotham City'
cities = ['Gotham City', 'Metropolis']
#city= 'Sin City'
#cities = ['District X', 'Sin City']

write here

ind = cities.index(city)

labels = sorted(d.keys(), key=lambda k: d[k][ind])
perf1 = [d[k][0] for k in labels]
perf2 = [d[k][1] for k in labels]

x = np.arange(len(labels)) # the label locations
width = 0.35 # the width of the bars

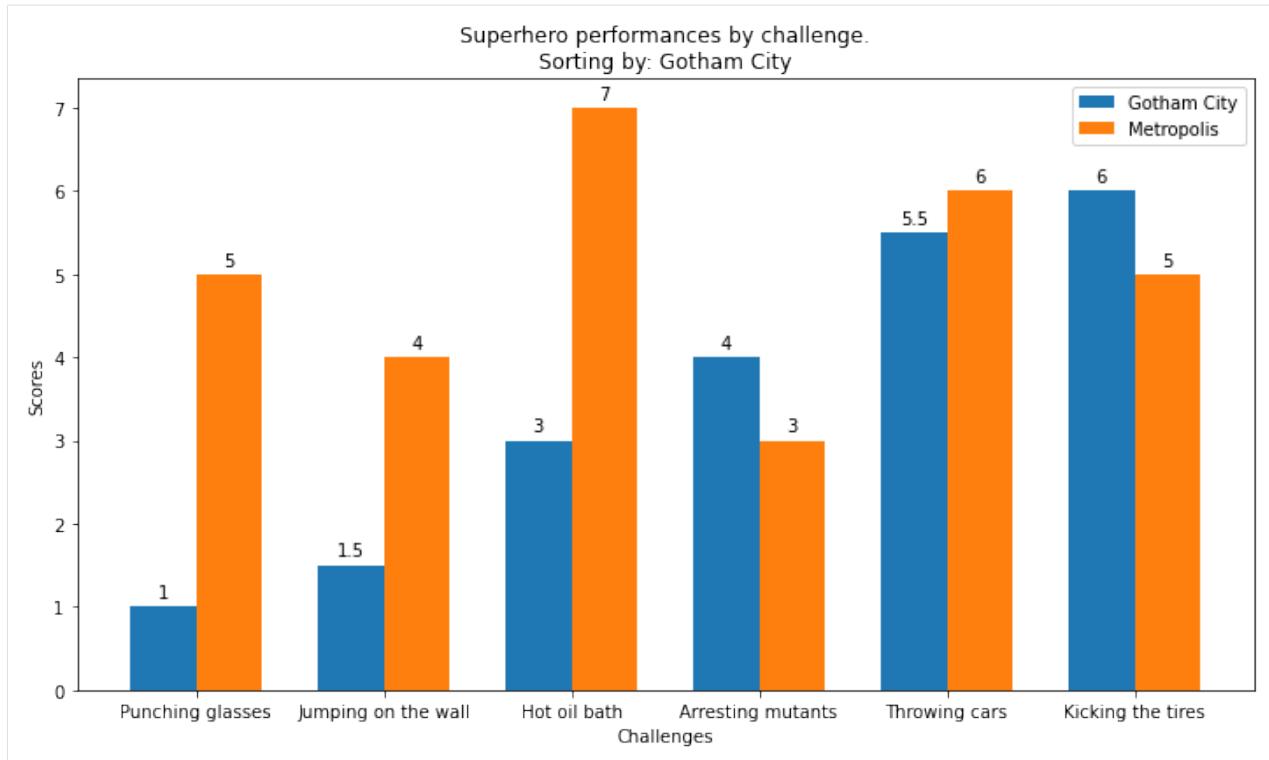
fig, ax = plt.subplots(figsize=(10,6))
rects1 = ax.bar(x - width/2, perf1, width, label=cities[0])
rects2 = ax.bar(x + width/2, perf2, width, label=cities[1])

Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_xlabel('Challenges')
ax.set_title('Superhero performances by challenge.\nSorting by: ' + city)
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

ax.bar_label(rects1, padding=3)
ax.bar_label(rects2, padding=3)

fig.tight_layout()

plt.show()
```



&lt;/div&gt;

[29]:

```

import matplotlib.pyplot as plt
import numpy as np

d = {
 'Punching glasses' : (1, 5),
 'Kicking the tires' : (6, 5),
 'Throwing cars' : (5.5, 6),
 'Hot oil bath' : (3, 7),
 'Jumping on the wall': (1.5, 4),
 'Arresting mutants' : (4, 3),
}

city = 'Gotham City'
cities = ['Gotham City', 'Metropolis']
#city= 'Sin City'
#cities = ['District X', 'Sin City']

write here

```

## Showing plots side by side

You can display plots on a grid. Each cell in the grid is identified by only one number. For example, for a grid of two rows and three columns, you would have cells indexed like this:

```
1 2 3
4 5 6
```

**REMEMBER:** plt.figure and plt.show should be called only **ONCE** in the whole program !

```
[30]: %matplotlib inline
import matplotlib.pyplot as plt
import math

xs = [1,2,3,4,5,6]

cells:
1 2 3
4 5 6

plt.subplot(2, # 2 rows
 3, # 3 columns
 1) # plotting in first cell
ys1 = [x**3 for x in xs]
plt.plot(xs, ys1)
plt.title('first cell')

plt.subplot(2, # 2 rows
 3, # 3 columns
 2) # plotting in second cell

ys2 = [2*x + 1 for x in xs]
plt.plot(xs, ys2)
plt.title('2nd cell')

plt.subplot(2, # 2 rows
 3, # 3 columns
 3) # plotting in third cell

ys3 = [-2*x + 1 for x in xs]
plt.plot(xs, ys3)
plt.title('3rd cell')

plt.subplot(2, # 2 rows
 3, # 3 columns
 4) # plotting in fourth cell

ys4 = [-2*x**2 for x in xs]
plt.plot(xs, ys4)
plt.title('4th cell')

plt.subplot(2, # 2 rows
 3, # 3 columns
 5) # plotting in fifth cell
```

(continues on next page)

(continued from previous page)

```

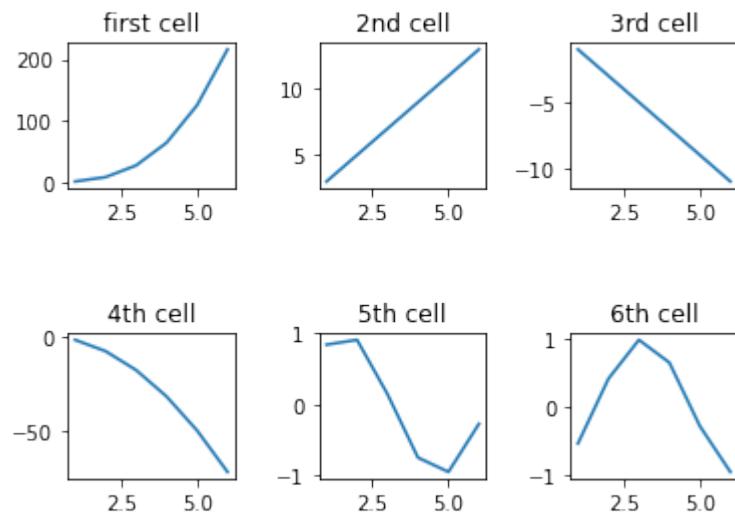
ys5 = [math.sin(x) for x in xs]
plt.plot(xs,ys5)
plt.title('5th cell')

plt.subplot(2, 3, 6) # 2 rows
3 columns
plotting in sixth cell

ys6 = [-math.cos(x) for x in xs]
plt.plot(xs,ys6)
plt.title('6th cell')

plt.subplots_adjust(wspace = 0.5, hspace = 1) # to avoid text overlapping
plt.show() # ONLY call this ONCE at the very end

```



### Exercise - $\sin(kx)$

Given a list `ks` containing  $n$  floats, show  $n$  plots stacked vertically of function  $\sin(kx)$  with limits left to right subdivided in 50 intervals.

- display the `k` values as titles
- define a function `plot_sin` to be called  $n$  times
- put adequate vertical space
- **HINT:** use numpy vector operations
- **REMEMBER** `plt.figure` and `plt.show` must be called only **ONCE** in the whole program !

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[31]:

```

import matplotlib.pyplot as plt
import math

```

(continues on next page)

(continued from previous page)

```

ks = [1, 2, 3]

write here

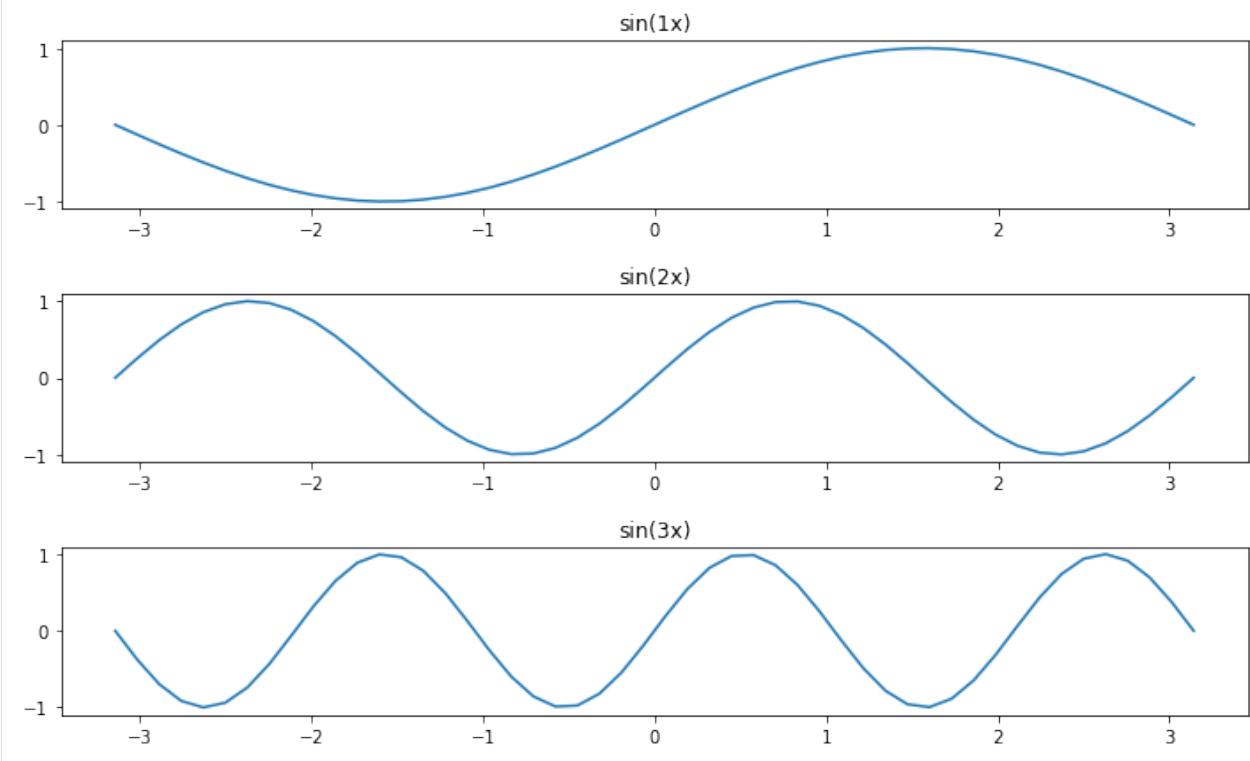
def plot_sin(left, right, k, i):
 xs = np.linspace(left, right, 50)
 plt.subplot(len(ks), # rows
 1, # columns
 i+1) # cell number
 ys = np.sin(k * xs)
 plt.plot(xs, ys)
 plt.title("sin(%sx)" % k)

fig = plt.figure(figsize=(12, 7))

for i in range(len(ks)):
 plot_sin(-np.pi, np.pi, ks[i], i)

plt.subplots_adjust(wspace = 0.5, hspace = 0.5) # to avoid text overlapping
plt.show()

```



&lt;/div&gt;

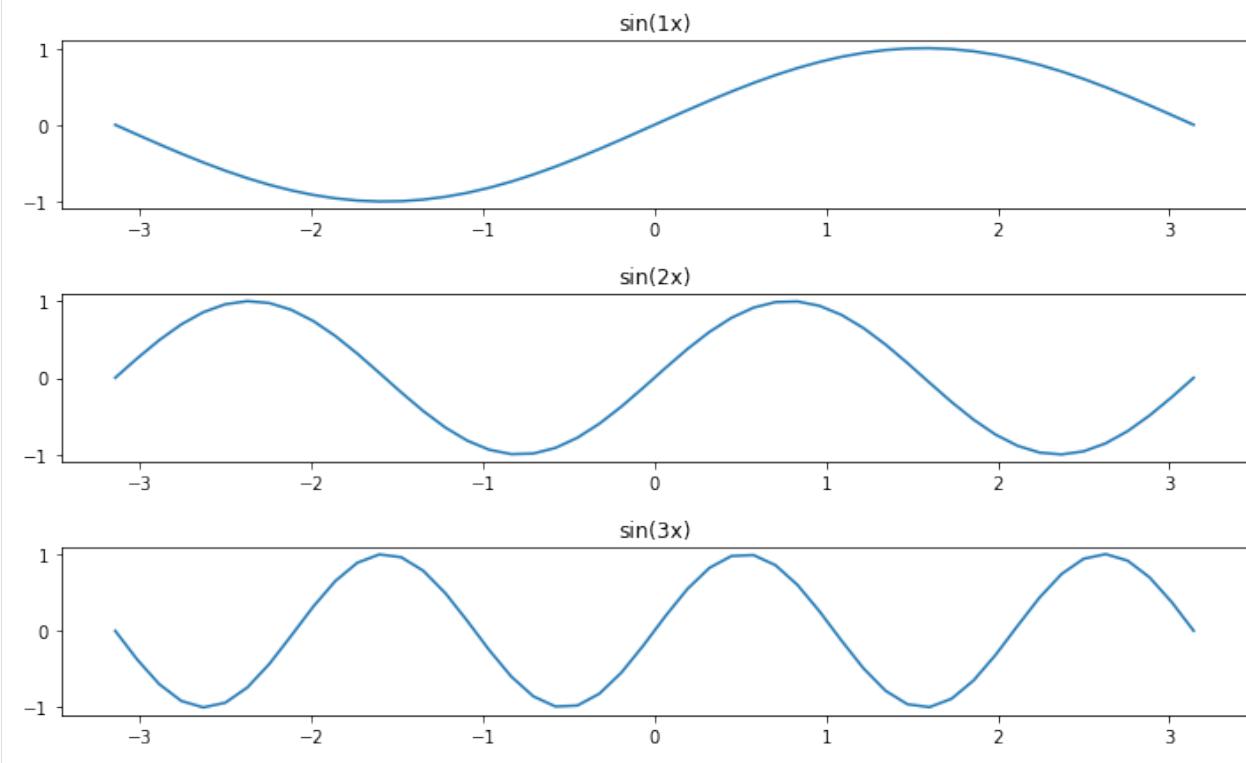
```
[31]: import matplotlib.pyplot as plt
import math

ks = [1, 2, 3]
```

(continues on next page)

(continued from previous page)

```
write here
```



## Other plots

Matplotlib allows to display pretty much anything, here we collect some we use in the book, for others, see the extensive Matplotlib documentation<sup>354</sup>

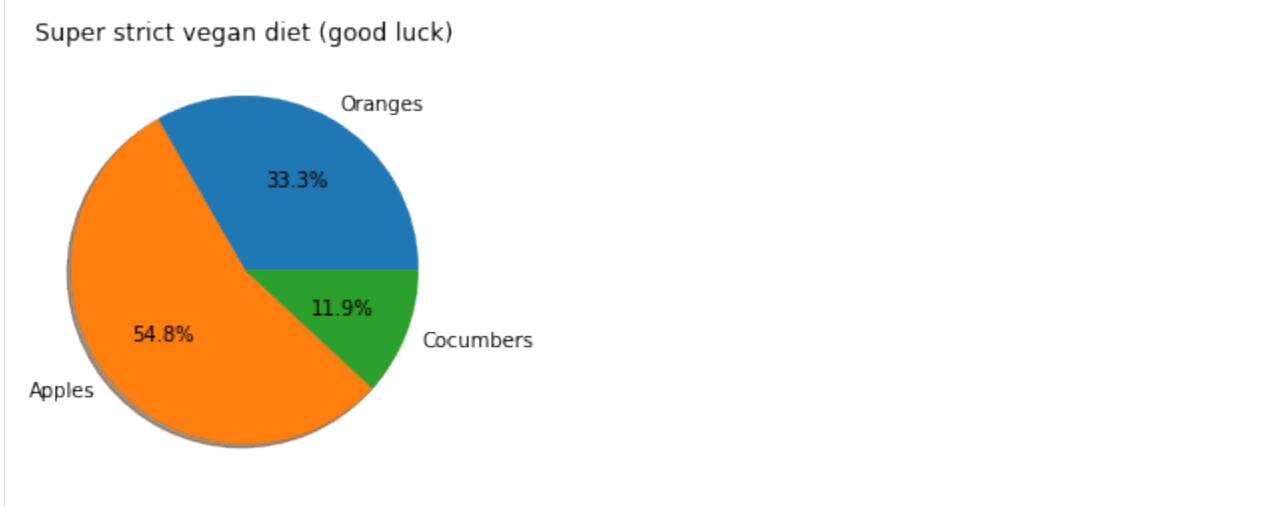
### Pie chart

```
[32]: %matplotlib inline
import matplotlib.pyplot as plt

labels = ['Oranges', 'Apples', 'Cocumbers']
fracs = [14, 23, 5] # how much for each sector, note doesn't need to add up to 100

plt.pie(fracs, labels=labels, autopct='%.1f%%', shadow=True)
plt.title("Super strict vegan diet (good luck)")
plt.show()
```

<sup>354</sup> <https://matplotlib.org/gallery/index.html>

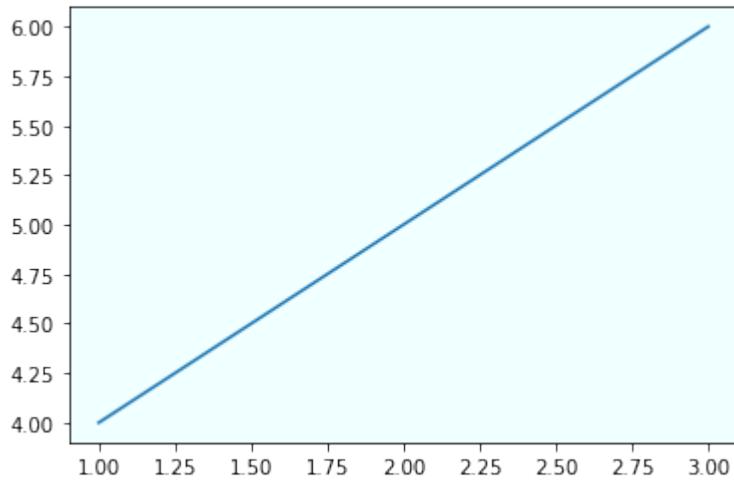


## Fancy plots

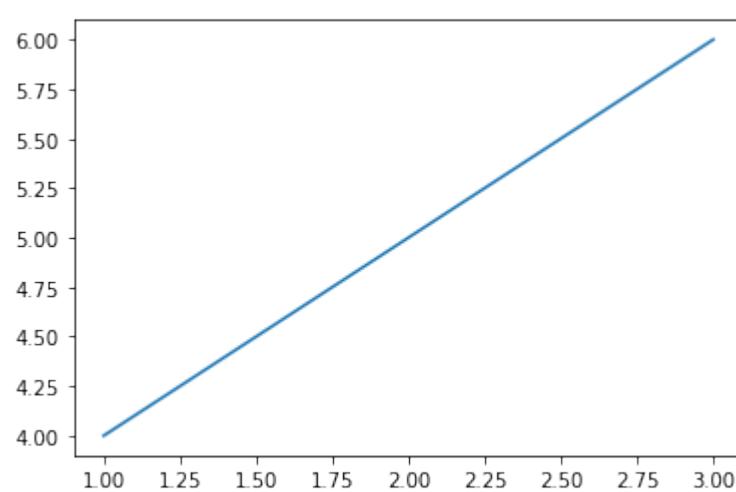
You can enhance your plots with some eyecandy, we put some example.

### Background color

```
[33]: # CHANGES THE BACKGROUND COLOR FOR *ALL* SUBSEQUENT PLOTS
plt.rcParams['axes.facecolor'] = 'azure'
plt.plot([1,2,3],[4,5,6])
plt.show()
```



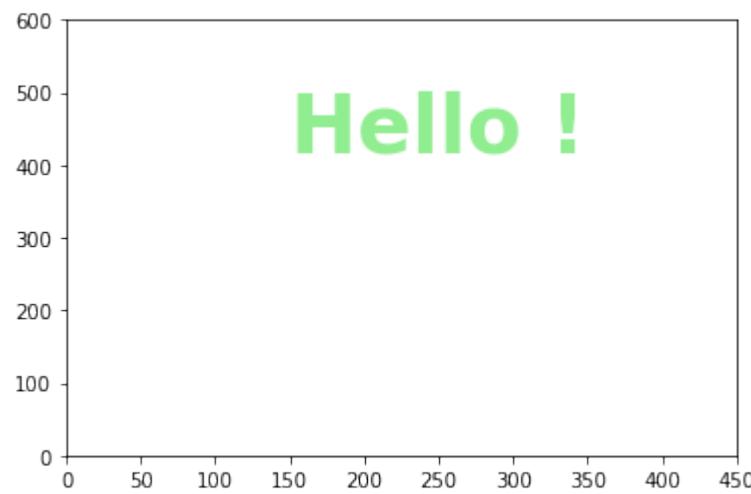
```
[34]: plt.rcParams['axes.facecolor'] = 'white' # restores the white for all following plots
plt.plot([1,2,3],[4,5,6])
plt.show()
```



## Text

```
[35]: plt.xlim(0,450) # important to set when you add text
plt.ylim(0,600) # as matplotlib doesn't automatically resize to show them

plt.text(250,
 450,
 "Hello !",
 fontsize=40,
 fontweight='bold',
 color="lightgreen",
 ha='center', # centers text horizontally
 va='center') # centers text vertically
plt.show()
```



## Images

Let's try adding the image clef.png

```
[36]: %matplotlib inline
import matplotlib.pyplot as plt

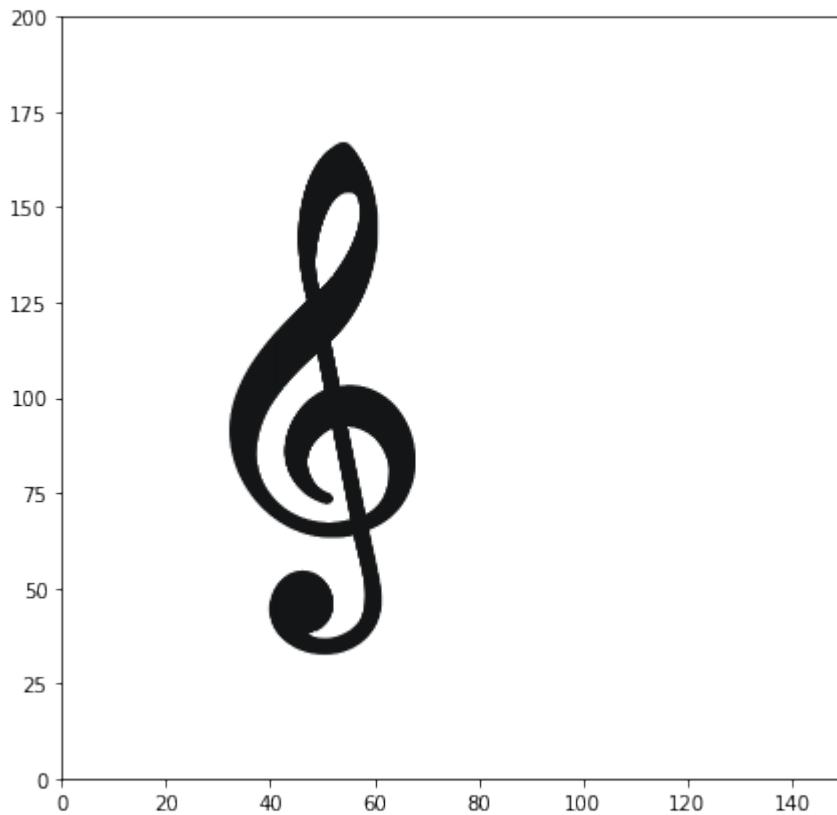
fig = plt.figure(figsize=(7,7))

NOTE: if you don't see anything, check position and/or zoom factor

from matplotlib.offsetbox import OffsetImage, AnnotationBbox

plt.xlim(0,150) # important to set when you add images
plt.ylim(0,200) # as matplotlib doesn't automatically resize to show them
ax=fig.gca()
img = plt.imread('clef.png')
ax.add_artist(AnnotationBbox(OffsetImage(img, zoom=0.5),
 (50, 100),
 frameon=False))

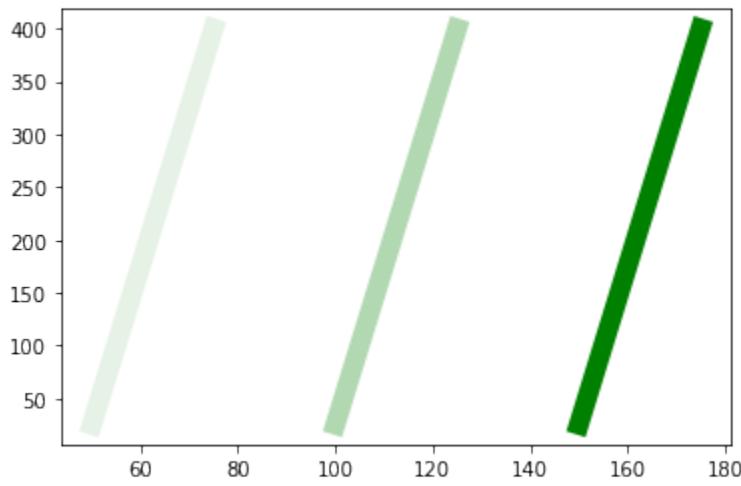
plt.show()
```



## Color intensity

To tweak the color intensity we can use the `alpha` parameter, which varies from `0.0` to `1.0`

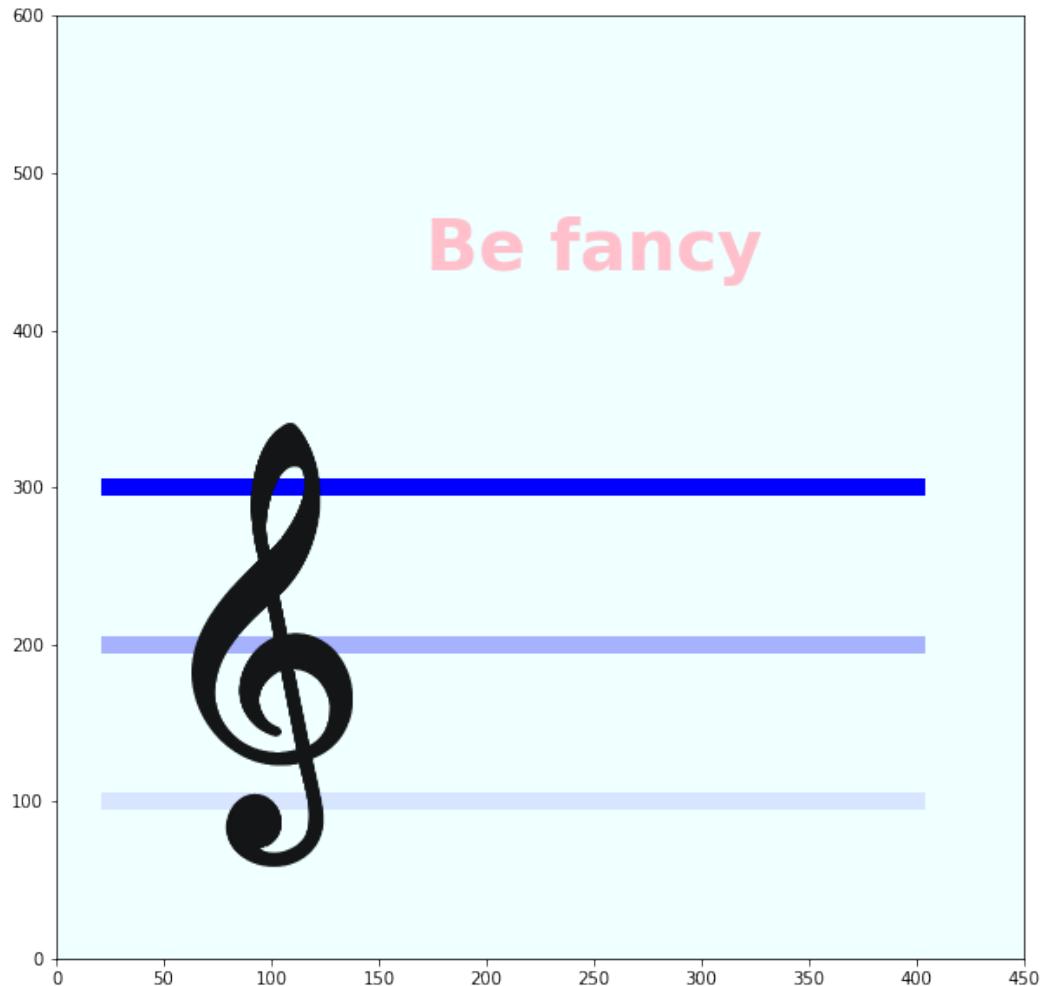
```
[37]: plt.plot([150,175], [25,400],
 color='green',
 alpha=1.0, # full color
 linewidth=10)
plt.plot([100,125],[25,400],
 color='green',
 alpha=0.3, # lighter
 linewidth=10)
plt.plot([50,75], [25,400],
 color='green',
 alpha=0.1, # almost invisible
 linewidth=10)
plt.show()
```



## Exercise - Be fancy

Try writing some code to visualize the image down here

EXPECTED OUTPUT



[ ]:

```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[38]:

```
%matplotlib inline
import matplotlib.pyplot as plt

write here

fig = plt.figure(figsize=(10,10))

CHANGES BACKGROUND COLOR
```

(continues on next page)

(continued from previous page)

```
plt.rcParams['axes.facecolor'] = 'azure'

SHOWS TEXT
plt.text(250,
 450,
 "Be fancy",
 fontsize=40,
 fontweight='bold',
 color="pink",
 ha='center',
 va='center')

CHANGES COLOR INTENSITY WITH alpha

plt.plot([25,400], [300,300],
 color='blue',
 alpha=1.0, # full color
 linewidth=10)
plt.plot([25,400], [200,200],
 color='blue',
 alpha=0.3, # softer
 linewidth=10)
plt.plot([25,400], [100,100],
 color='blue',
 alpha=0.1, # almost invisible
 linewidth=10)

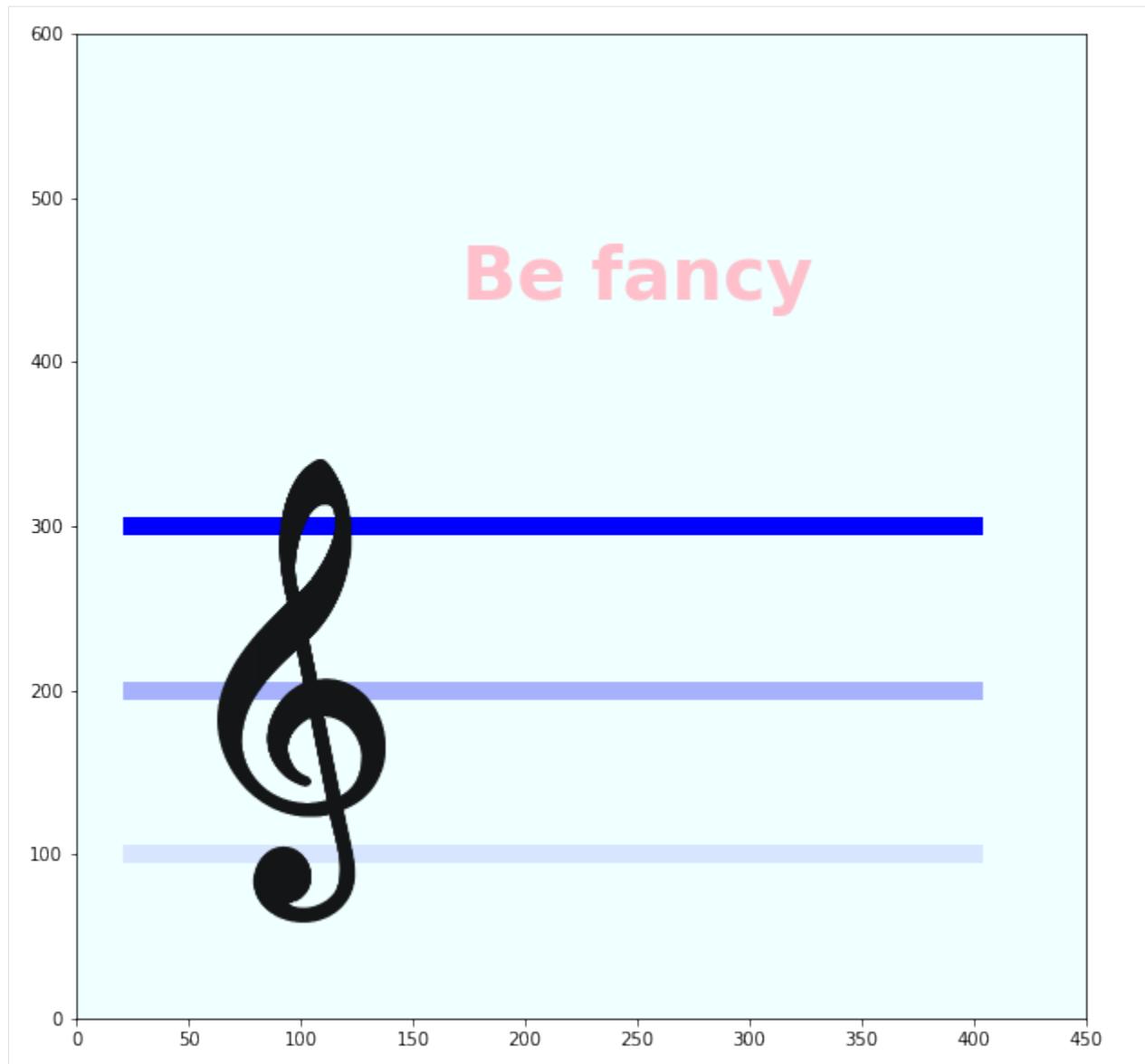
NOTE: if you don't see anything, check position and/or zoom factor

from matplotlib.offsetbox import OffsetImage, AnnotationBbox

plt.xlim(0,450) # important to set when you add images
plt.ylim(0,600) # as matplotlib doesn't automatically resize to show them

ax=fig.gca()
img = plt.imread('clef.png')
ax.add_artist(AnnotationBbox(OffsetImage(img, zoom=0.5),
 (100, 200),
 frameon=False))

plt.show()
```



</div>

[38]:

```
%matplotlib inline
import matplotlib.pyplot as plt

write here
```

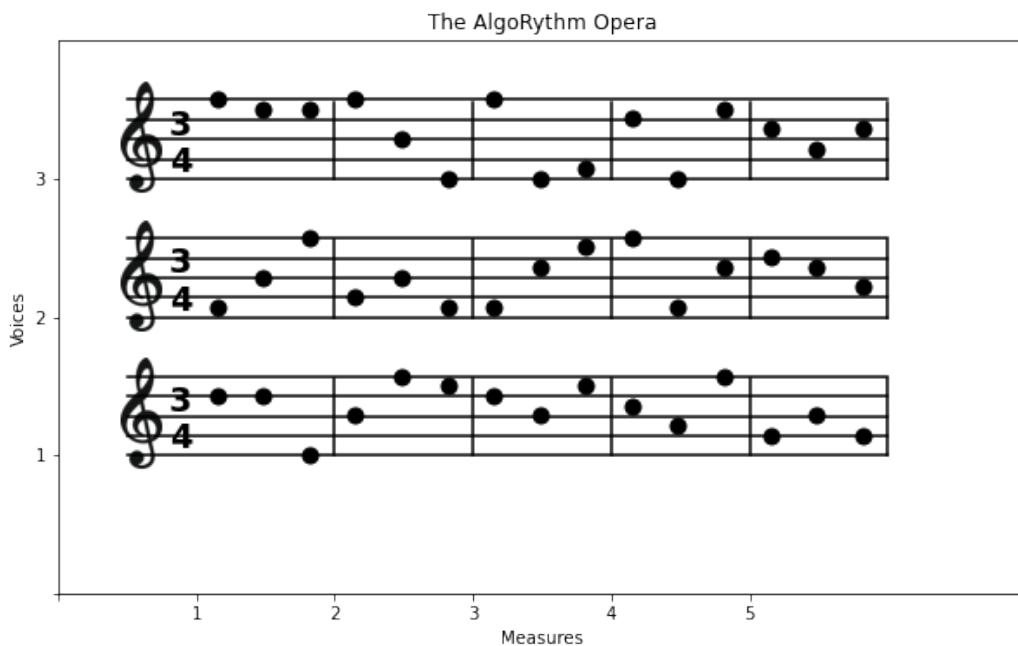
## Continue

Go on with [the AlgoRhythm challenge<sup>355</sup>](#) or the [numpy images tutorial<sup>356</sup>](#)

### 8.2.2 The AlgoRhythm Opera Challenge

#### Download exercises zip

Browse files online<sup>357</sup>



Some people say music is not important, pupils should do math instead. Let's show them music *is* math.

A musical sheet is divided vertically in `voices`. Each voice has a pentagram divided in `measures` (or `bars`, or *battute* in italian), each having a number of beats indicated by its *time signature*, like  $\frac{3}{4}$

Simple time signatures consist of two numerals, one stacked above the other:

- The upper numeral `time_sig_num` (3) indicates how many such beats constitute a bar
- The lower numeral `time_sig_denom` (4) indicates the note value that represents one beat (the beat unit)
- **NOTE:** the lower numeral is not important for our purposes, you will just print it on the pentagrams

For the purposes of this exercise, we assume each measure contains exactly `time_sig_num` notes.

---

<sup>355</sup> <https://en.softpython.org/visualization/visualization2-chal.html>

<sup>356</sup> <https://en.softpython.org/visualization/visualization-images-sol.html>

<sup>357</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/visualization>

On the chart, note the x axis measures start **at 1** and each measure has **chart length 1**. Notice the pentagrams start a bit before the 1 position because there is some info like the time signature and the clef.

- **NOTE:** horizontal tick 0 is not shown

Vertically, each voice pentagram begins at an integer position, **starting from 1**. Each line of the pentagram occupies a vertical space we can imagine subdivided in `divs=7` divisions, in the chart you see 5 divisions for the pentagram lines and 2 invisible ones to separate from voice above.

- **NOTE:** vertical tick 0 is not shown

## The variables

**DO NOT put unnecessary constants in your code !**

For example, instead of writing 5 you should use the variable `measures` defined down here

```
[1]: # this is *not* a python command, it is a Jupyter-specific magic command,
to tell jupyter we want the graphs displayed in the cell outputs
%matplotlib inline

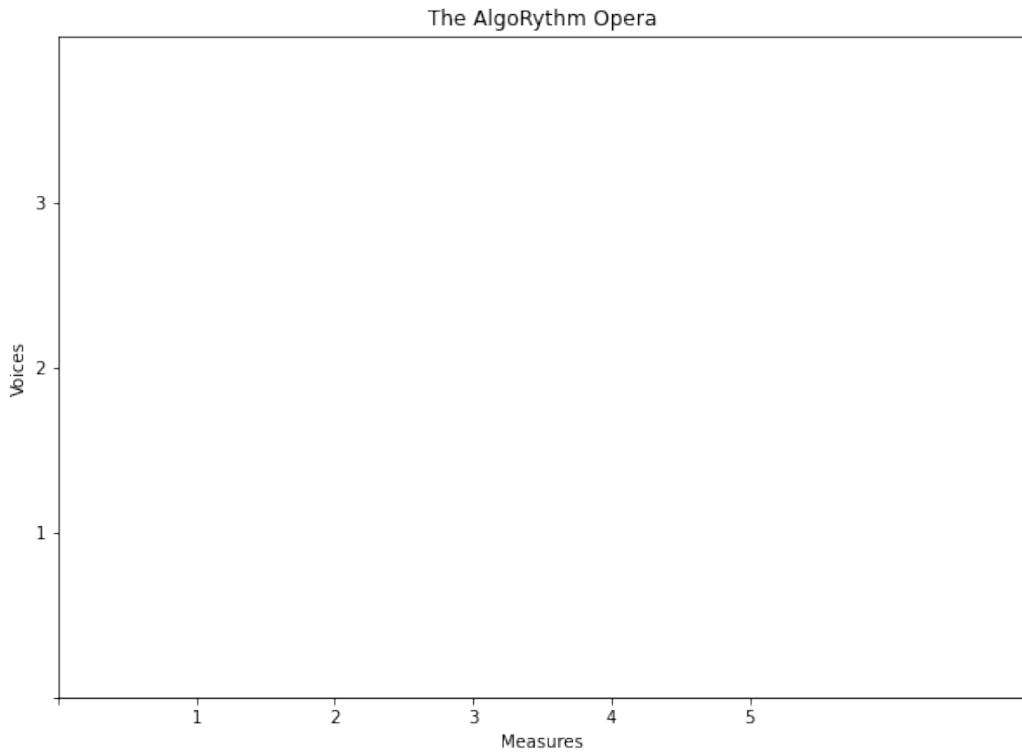
imports matplotlib
import matplotlib.pyplot as plt
from pprint import pprint
import numpy as np

USE THESE VARIABLES !!
measures = 5 # also called bars
voices = 3 # number of pentagrams
time_sig_num = 3
time_sig_denom = 4
divs = 7 # number of vertical divisions for each voice (5 lines for each pentagram
 # + 2 imaginary lines)
```

### 1. plot\_sheet

Implement `plot_sheet`, which draws sheet info like title, axes, xticks, yticks ...

- **DO NOT** draw the pentagrams
- **NOTE:** tick 0 is not shown



**WARNING 1:** you need only this ONE call to `plt.figure`

**WARNING 2:** beware of `plt.show()`

If you execute this outside of Jupyter, you will need to call `plt.show()` ONLY ONCE, at the very end of all plotting stuff (outside the functions!)

[2]:

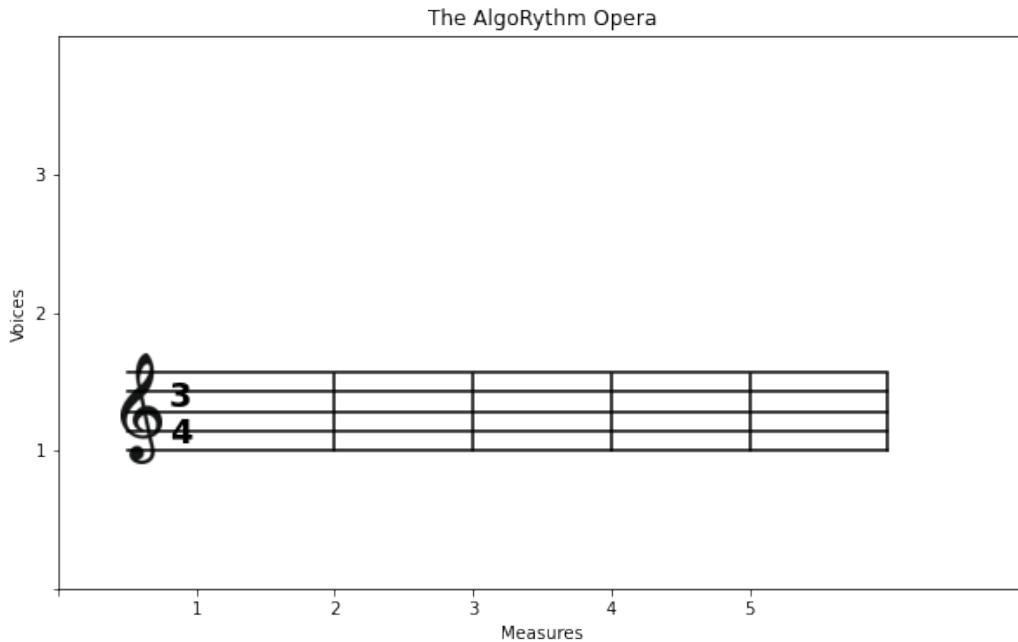
```
def plot_sheet():
 fig = plt.figure(figsize=(10,7)) # 10 inches large by 7 high
 raise Exception('TODO IMPLEMENT ME !')

plot_sheet()
```

## 2. plot\_pentagram

Given a voice integer **from 1 (NOT ZERO!!!)** to voices included, draws its pentagram.

- **DO NOT** draw the notes



Try drawing stuff in this sequence:

1 - draw horizontal lines, starting from measure 1. Leave some space before 1 to put later the clef. To obtain the y coordinates, use `np.linspace`

- **REMEMBER** to check you have a measure drawn *after* the 5 tick !
- **HINT:** Since you will have to plot several detached lines, for each you will need a separate call to `plt.plot`.

2 - draw vertical bars between measures - don't put a bar at beginning of first measure. To obtain the x coordinates, use `np.linspace`

3 - draw time signature text. To draw a string s at position x, y, you need to call this:

```
plt.text(x, y, s, fontsize=19, fontweight='bold')
```

4 TODO - draw clef: put provided image `clef.png`. To draw the image, you need to call some code like this, by using the appropriate numerical coordinates in place of `xleft`, `xright`, `ybottom`, `ytop` which delimit the place where the image is put.

```
clef = plt.imread('clef.png')
plt.imshow(clef, extent=[xleft, xright, ybottom, ytop])
```

- **NOTE 1:** If you see nothing, it maybe be you are drawing outside of the area or the given frame is too small.

- **NOTE 2:** `xright` and `ytop` are absolute coordinates, **not** width and height!

[3]:

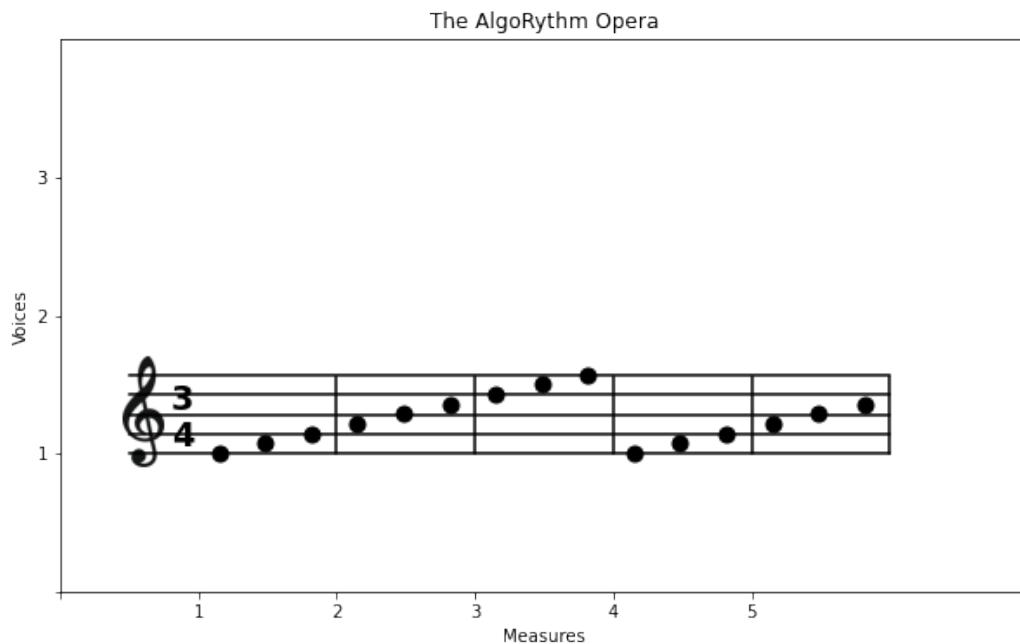
```
def plot_pentagram(voice):
 raise Exception('TODO IMPLEMENT ME !')

NOTE: putting *all* commands in a cell
plot_sheet()
plot_pentagram(1)
```

### 3. plot\_notes

Implement function `plot_notes` which takes a voice integer **from 1** to n and a database of notes as a list of lists and draws the notes of that voice

- notes are integers from `min_note=0` to `max_note=8`
- assume we can only have notes that can be positioned inside the pentagram, so bottom note starts at E4 (middle height of bottom pentagram line) and highest note is F5 (middle height of top pentagram line).
- to set dots size, use `markersize=9` parameter
- to set dots color, use `color='black'` parameter



[4]:

```
these are just labels, but you don't need to put them anywhere
```

(continues on next page)

(continued from previous page)

```

http://newt.phys.unsw.edu.au/jw/notes.html
0 1 2 3 4 5 6 7 8
notes_scale=['E4','F4','G4','A4','B4','C5','D5','E5','F5']
min_note = 0
max_note = len(notes_scale) - 1

This is provided, DO NOT TOUCH IT!
def random_notes(voices, measures, time_sig_num, seed):
 """ Generates a random list of lists of notes. Generated notes depend on seed ↴numerical value.
 """
 import random
 random.seed(seed)
 ret = []
 for i in range(voices):
 ret.append([random.randint(min_note,max_note) for i in range(measures*(time_ ↴sig_num))])
 return ret

This is provided, DO NOT TOUCH IT!
def musical_scale(voices, measures, time_sig_num):
 """ Generates a scale of notes
 """
 ret = []
 for i in range(voices):
 j = min_note
 ret.append((list(range(max_note+1))*100) [:measures*time_sig_num])
 return ret

def plot_notes(voice, notes):
 raise Exception('TODO IMPLEMENT ME !')

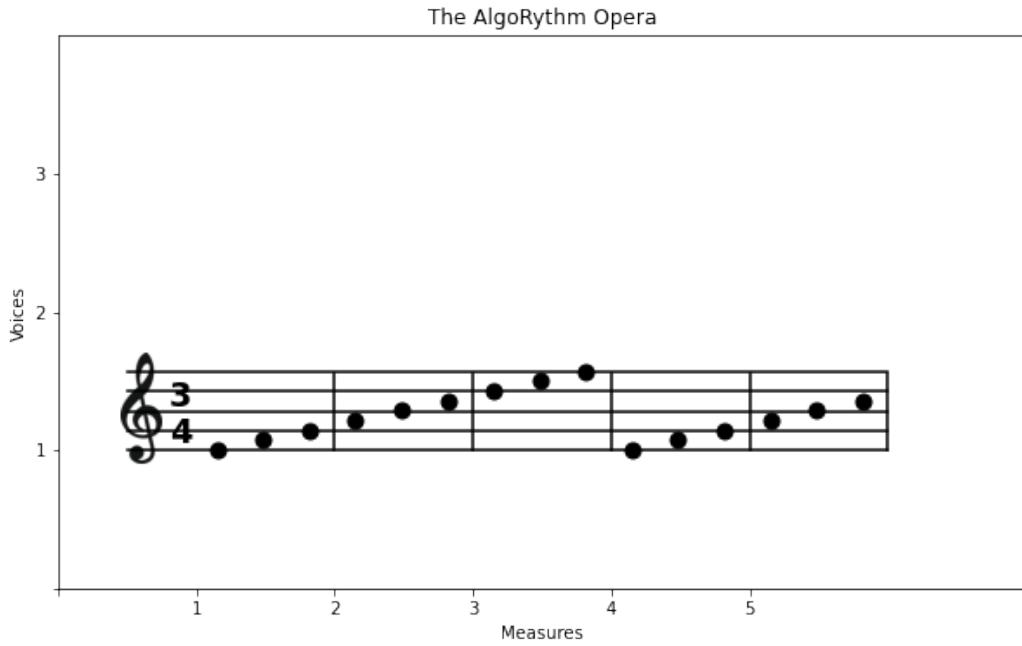
```

[5]:

```

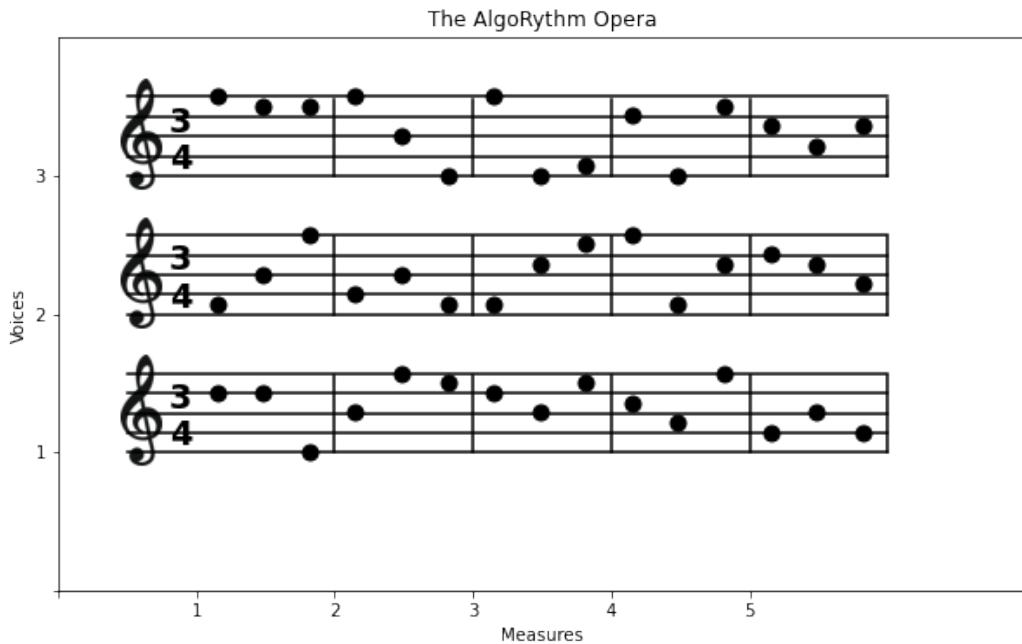
notes = musical_scale(voices, measures, time_sig_num)
#notes = random_notes(voices, measures, time_sig_num, 0)
from pprint import pprint
print('notes:')
pprint(notes)
plot_sheet()
plot_pentagram(1)
plot_notes(1, notes)

```



**Final result 1:**

Putting all together, and using random notes, you should get this:



[6]:

```

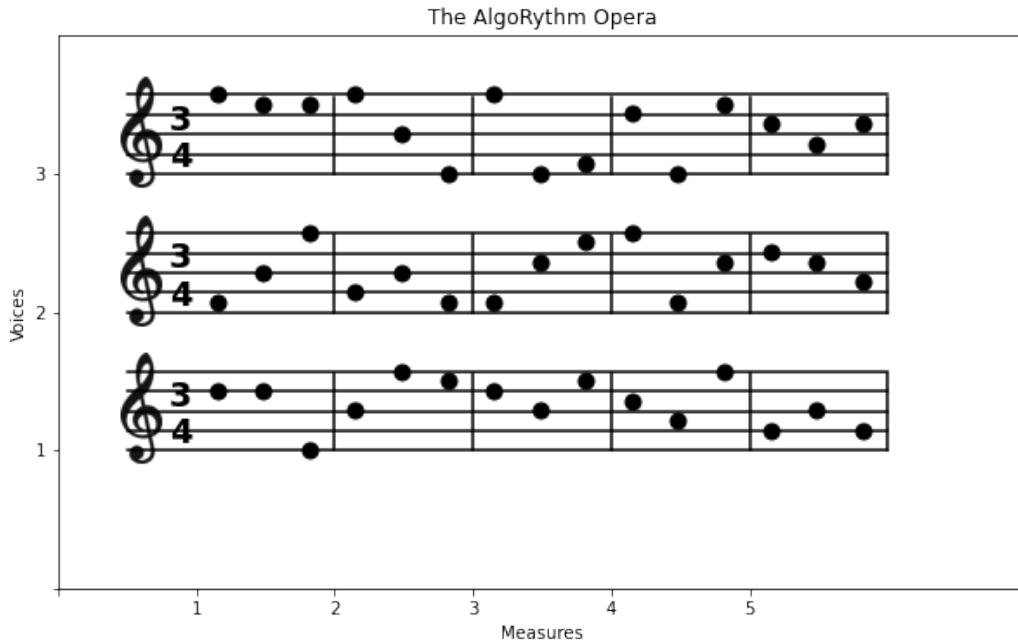
measures = 5 # also called bars
voices = 3 # number of pentagrams
time_sig_num = 3
time_sig_denom = 4
divs = 7

plot_sheet()
for i in range(1,voices+1):
 plot_pentagram(i)

#notes = musical_scale(voices, measures, time_sig_num)
notes = random_notes(voices, measures, time_sig_num, 0)

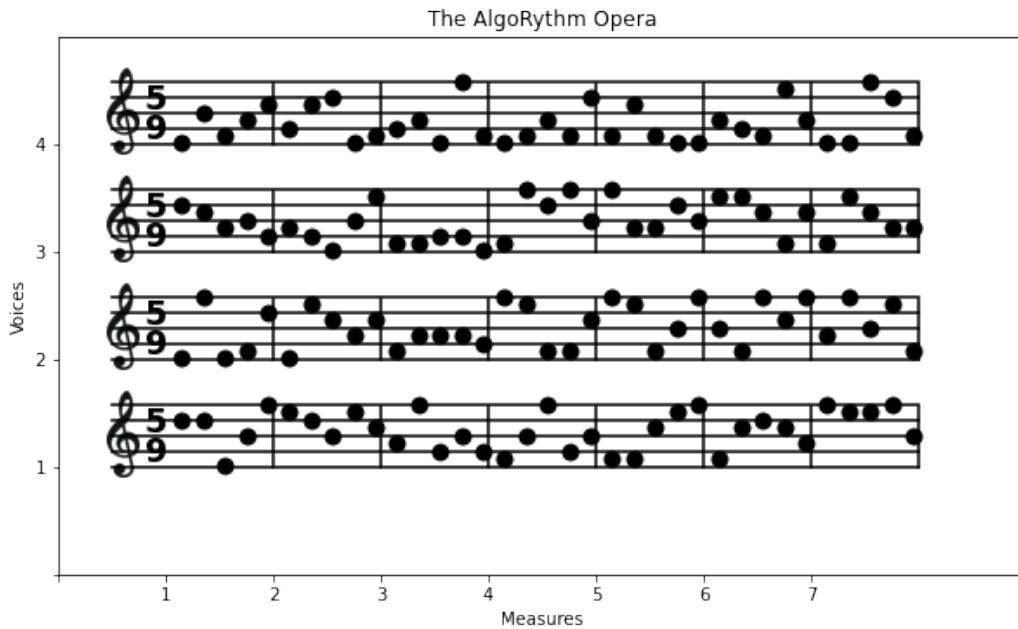
for i in range(1,voices+1):
 plot_notes(i, notes)

```



## Final result 2

Quite probably, you used too many constants in your code instead of the variables at the beginning of the notebook, so let's see if your code is general enough to work with sheets with a different number of voices, measures. You should get something like this:



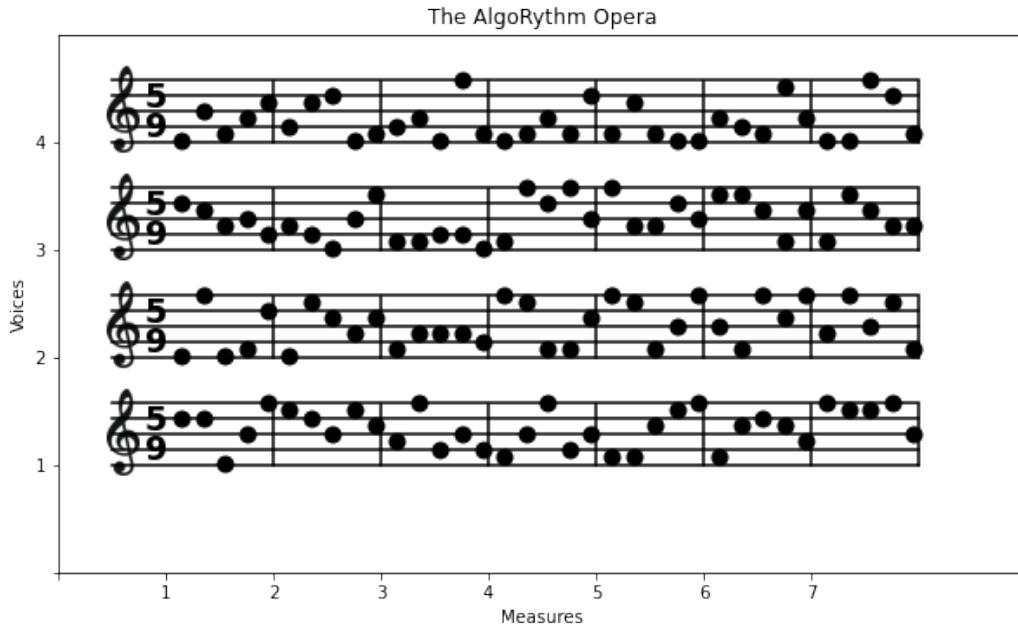
[7]:

```
WARNING: VARIABLES WERE CHANGED !!!!
measures = 7
voices = 4
time_sig_num = 5
time_sig_denom = 9
divs = 7

plot_sheet()
for i in range(1,voices+1):
 plot_pentagram(i)

#notes = musical_scale(voices, measures, time_sig_num)
notes = random_notes(voices, measures, time_sig_num, 0)

for i in range(1,voices+1):
 plot_notes(i, notes)
```



### 8.2.3 Visualization - Numpy images

[Download exercises zip](#)

[Browse files online<sup>358</sup>](#)

Images are a direct application of matrices, and show we can nicely translate a numpy matrix cell into a pixel on the screen.

Typically, images are divided into color channels: a common scheme is the RGB model, which stands for Red Green and Blue. In this tutorial we will load an image where each pixel is made of three integer values ranging from 0 to 255 included. Each integer indicates how much of a color component is present in the pixel, with zero meaning absence and 255 bright colors.

<sup>358</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/visualization>

## What to do

- unzip exercises in a folder, you should get something like this:

```
visualization
 visualization1.ipynb
 visualization1-sol.ipynb
 visualization2-chal.ipynb
 visualization-images.ipynb
 visualization-images-sol.ipynb
 jupman.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `visualization-images.ipynb`
- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

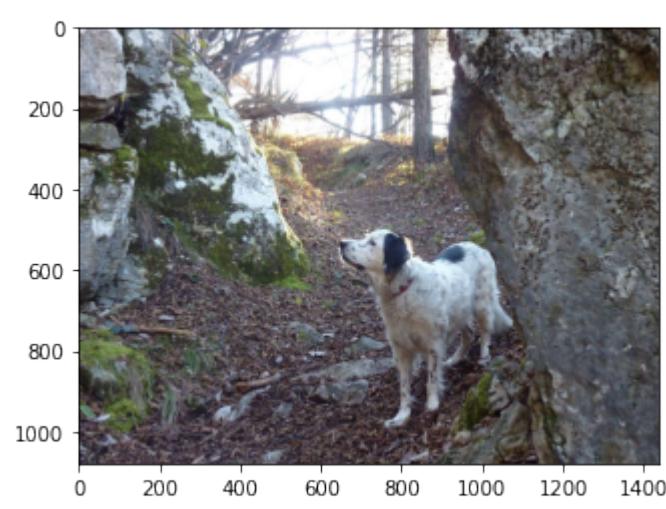
## Introduction

Let's load the image:

```
[1]: # this is *not* a python command, it is a Jupyter-specific magic command,
to tell jupyter we want the graphs displayed in the cell outputs
%matplotlib inline
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np

img = mpimg.imread('lulu.jpg')
#img = mpimg.imread('il-piccolo-principe.jpg')
#img = mpimg.imread('rifugio-7-selle.jpg')
#img = mpimg.imread('alright.jpg')
```

```
[2]: plt.imshow(img)
[2]: <matplotlib.image.AxesImage at 0x7ff5875da550>
```



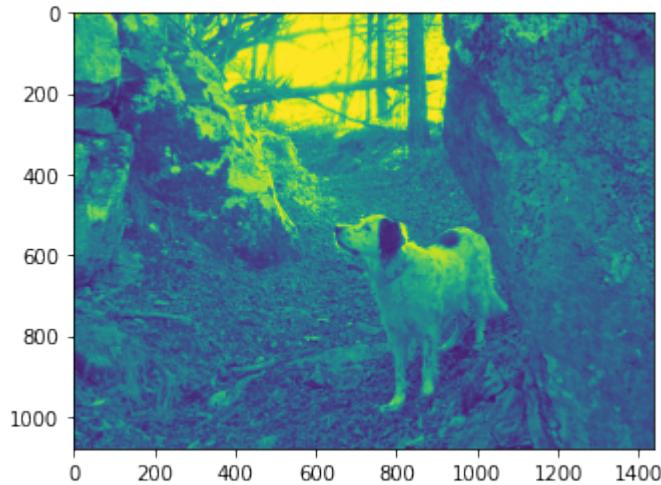
## Monochrome

For an easy start, we first get a monochromatic view of the image we call `gimg`:

```
[3]: gimg = img[:, :, 0] # this trick selects only one channel (the red one)

plt.imshow(gimg)

[3]: <matplotlib.image.AxesImage at 0x7ff586c08fd0>
```



If we have taken the RED, why is it shown GREEN?? For Matplotlib, the picture is only a square matrix of integer numbers, for now it has no notion of the best color scheme we would like to see:

```
[4]: print(gimg)

[[209 209 210 ... 117 118 117]
 [214 214 215 ... 112 116 117]
 [217 217 217 ... 105 110 114]
 ...
 [36 33 30 ... 72 67 64]]
```

(continues on next page)

(continued from previous page)

```
[42 36 31 ... 70 65 61]
[37 31 24 ... 68 63 60]]
```

[5]: `type(gimg)`

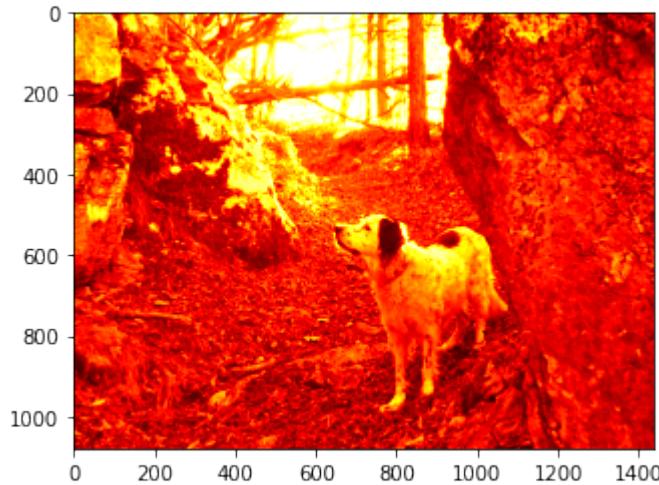
[5]: `numpy.ndarray`

By default matplotlib shows the intensity of light using with a greenish colormap.

Luckily, many color maps<sup>359</sup> are available, for example the 'hot' one:

[6]: `plt.imshow(gimg, cmap='hot')`

[6]: <matplotlib.image.AxesImage at 0x7ff586ba6d50>



To avoid confusion, we will pick a proper gray colormap:

[7]: `plt.imshow(gimg, cmap='gray')`

[7]: <matplotlib.image.AxesImage at 0x7ff586b2ea50>

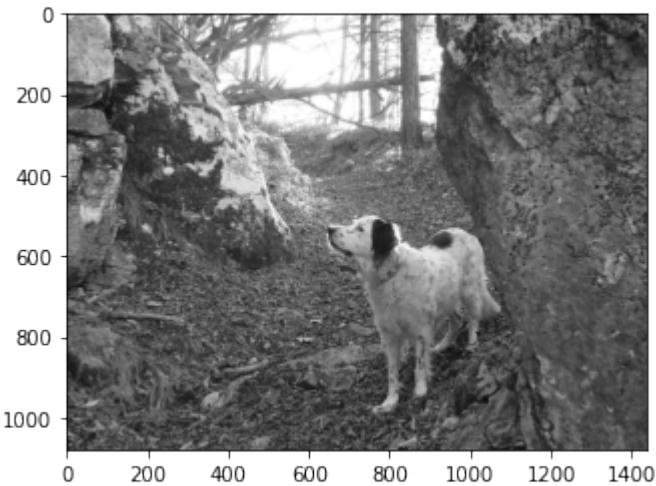


Let's define this shorthand function to type a little less:

<sup>359</sup> <https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html>

```
[8]: def gs(some_img):
 # vmin and vmax prevent normalization that occurs only with monochromatic images
 plt.imshow(some_img, cmap='gray', vmin=0, vmax=255)
```

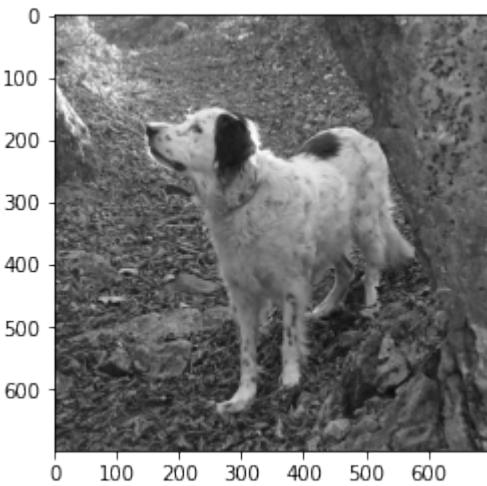
```
[9]: gs(gimg)
```



### Focus

Let's try some simple transformation. As with regular Python lists, we can do slicing:

```
[10]: gs(gimg[350:1050, 500:1200])
```



**NOTE 1:** differently from regular lists of lists, in Numpy we can write slices for different dimensions **within the same square brackets**

**NOTE 2:** We are still talking about matrices, so pictures also follow the very same conventions of regular algebra we've also seen with lists of lists: the first index is for rows and starts from 0 in the left upper corner, and second index is for columns.

**NOTE 3:** the indeces shown on the extracted picture are *not* the indeces of the original matrix!

## Exercise - Head focus

Try selecting the head:

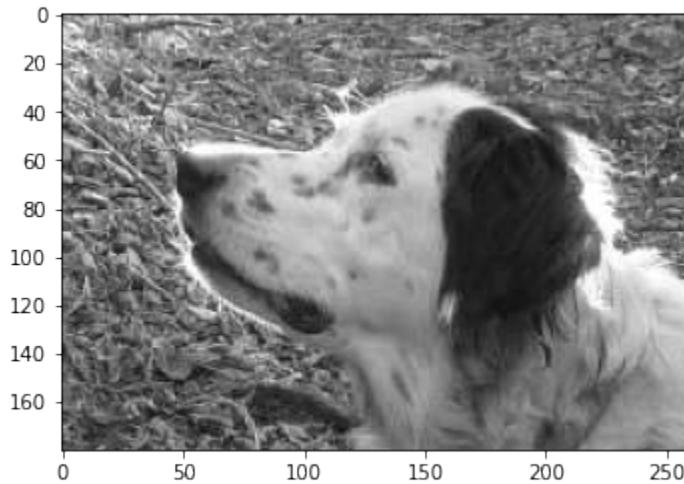
```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
```

```
data-jupman-show="Show solution"
```

```
data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

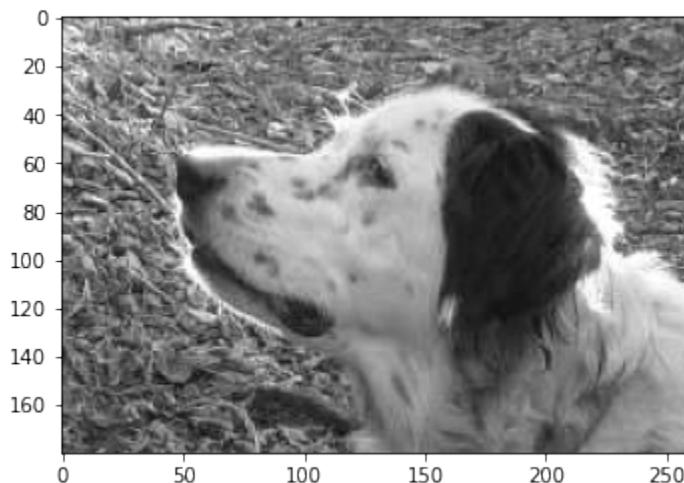
```
[11]: # write here
```

```
gs(gimg[470:650, 600:860])
```



```
</div>
```

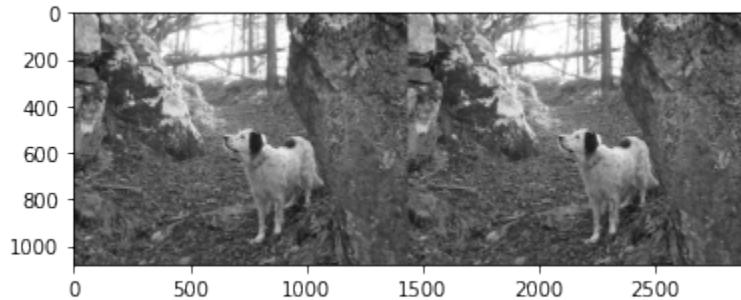
```
[11]: # write here
```



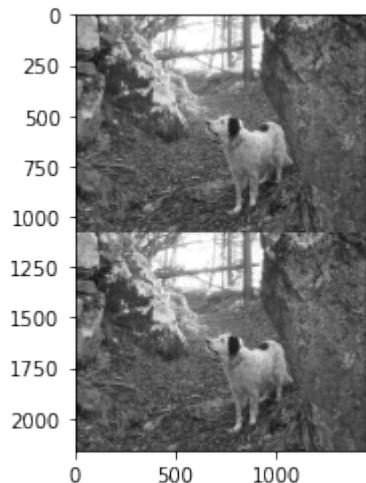
### hstack and vstack

We can stitch together pictures with `hstack` and `vstack`. Note they produce a NEW matrix:

```
[12]: gs(np.hstack((gimg, gimg)))
```



```
[13]: gs(np.vstack((gimg, gimg)))
```



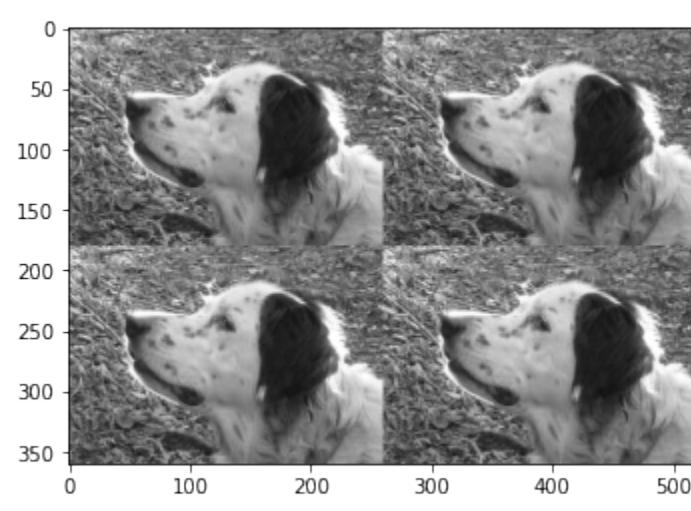
### Exercise - Passport

Try to replicate somehow the head

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

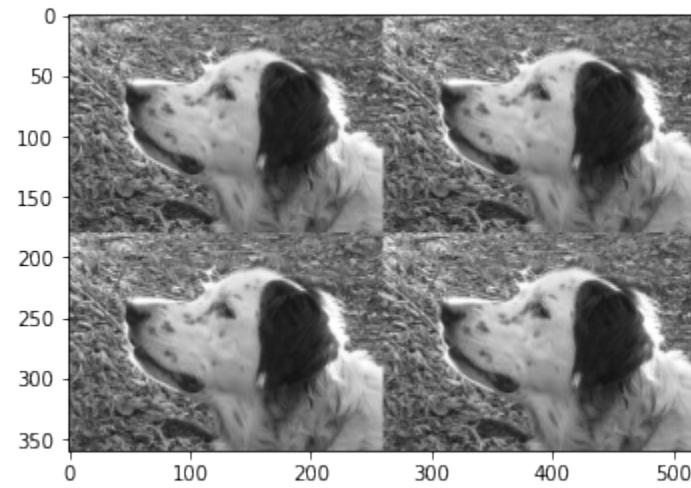
Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[14]: # write here
head = gimg[470:650, 600:860]
col = np.vstack((head, head))
gs(np.hstack((col, col)))
```



&lt;/div&gt;

[14]: # write here

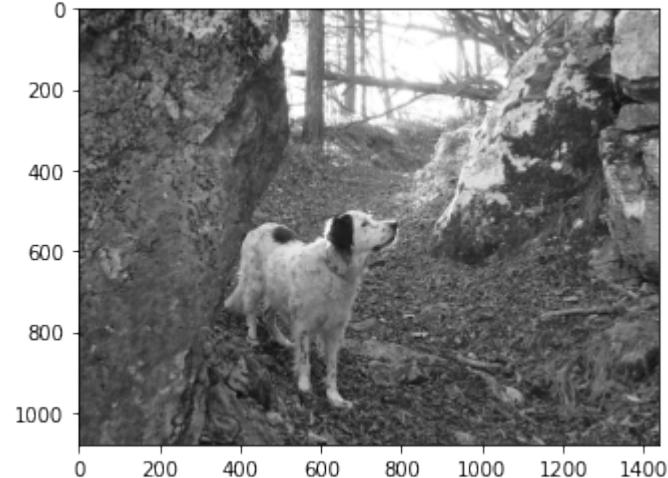


### flip

A handy method for mirroring is `flip`<sup>360</sup>:

[15]: `gs(np.flip(gimg, axis=1))`

<sup>360</sup> <https://numpy.org/doc/stable/reference/generated/numpy.flip.html>

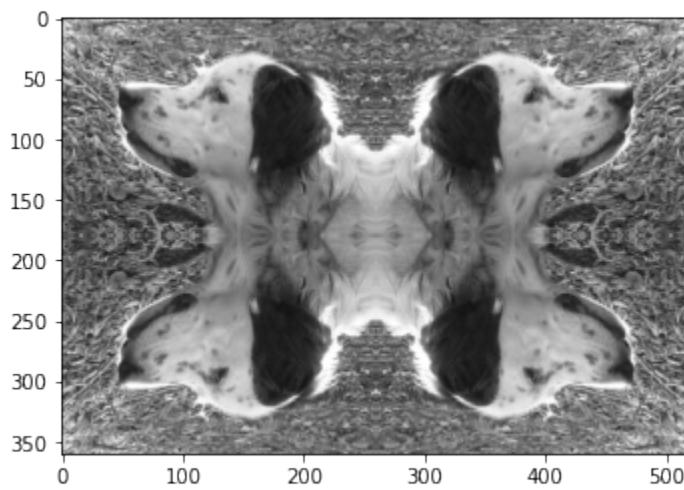


### Exercise - Hall of mirrors

Try to replicate somehow the head, pointing it in different directions as in the example

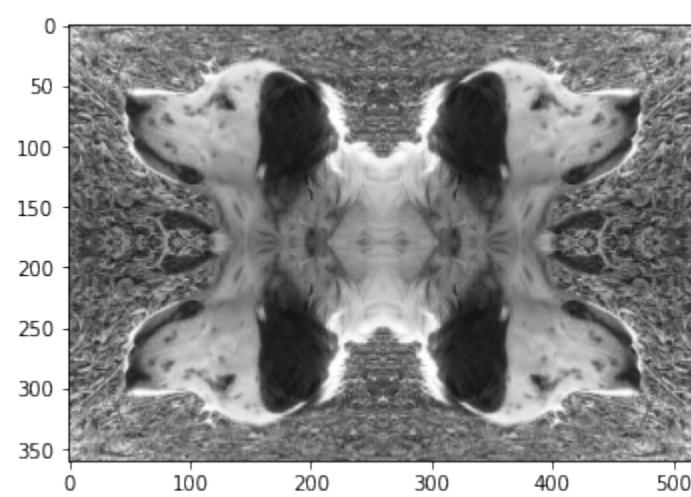
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[16]: # write here
head = gimg[470:650,600:860]
col1 = np.vstack((head, np.flip(head, axis=0)))
gs(col1)
col2 = np.flip(col1, axis=0)
gs(np.hstack((col1,np.flip(col2))))
```



</div>

```
[16]: # write here
```



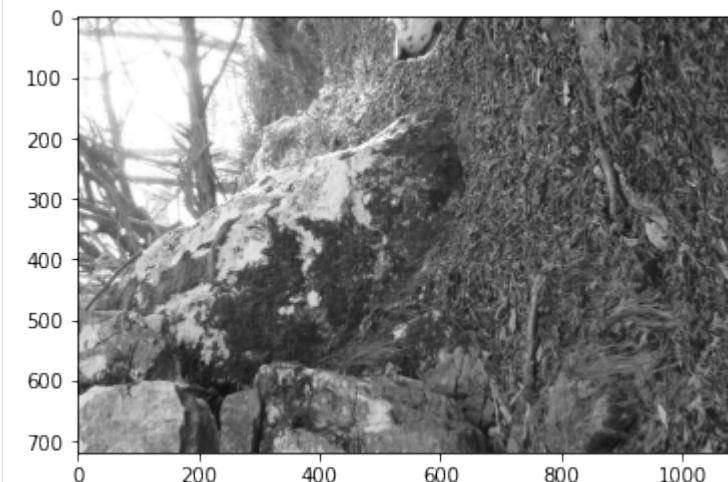
### Exercise - The nose from above

Do some googling and find an appropriate method for obtaining this:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

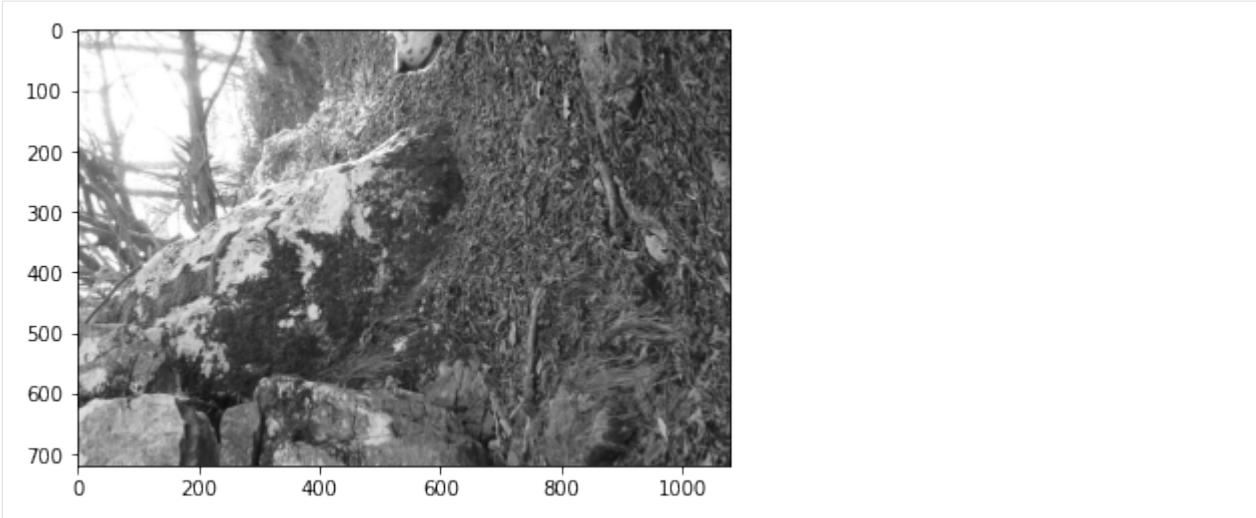
[17]: # write here

```
gs(np.rot90(gimg[:, :gimg.shape[1] // 2]))
```



</div>

[17]: # write here



## Writing arrays

We can write into an array using square brackets:

```
[18]: arr = np.array([5, 9, 4, 8, 6])
```

```
[19]: arr[0] = 7
```

```
[20]: arr
```

```
[20]: array([7, 9, 4, 8, 6])
```

So far, nothing special. Let's try to make a slice:

```
[21]: # 0 1 2 3 4
arr1 = np.array([5, 9, 4, 8, 6])
arr2 = arr1[1:3]
arr2
```

```
[21]: array([9, 4])
```

```
[22]: arr2[0] = 7
```

```
[23]: arr2
```

```
[23]: array([7, 4])
```

```
[24]: arr1 # the original was modified !!!
```

```
[24]: array([5, 7, 4, 8, 6])
```

**WARNING: SLICE CELLS IN NUMPY ARE POINTERS TO ORIGINAL CELLS!**

To prevent problems, you can create a *deep copy* by using the `copy` method:

```
[25]: #0 1 2 3 4
arr1 = np.array([5,9,4,8,6])
arr2 = arr1[1:3].copy()
arr2
[25]: array([9, 4])
```

```
[26]: arr2[0] = 7
```

```
[27]: arr2
```

```
[27]: array([7, 4])
```

```
[28]: arr1 # remained the same
[28]: array([5, 9, 4, 8, 6])
```

## Writing into images

Let's go back to images. First note that `gimg` was generated by calling `pt.imshow`, which set it as READ-ONLY:

```
gimg[0,0] = 255 # NOT POSSIBLE WITH LOADED IMAGES!

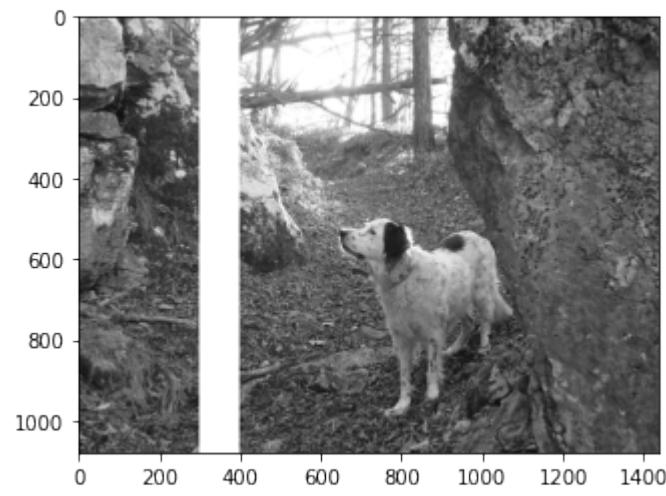
ValueError Traceback (most recent call last)
<ipython-input-186-7d21dd84cac2> in <module>()
----> 1 img[0,0,0] = 4 # NOT POSSIBLE!
ValueError: assignment destination is read-only
```

If we want something we can write into, we need to perform a **deep copy**:

```
[29]: mimg = gimg.copy() # *DEEP* COPY
mimg[0,0] = 255 # the copy is writable
mimg[0,0]
[29]: 255
```

If we want to set an entire slice to a constant value, we can write like this:

```
[30]: mimg[:, 300:400] = 255
[31]: gs(mimg)
```



### Exercise - Stripes

Write a program that given top-left coordinates `tl` and bottom-right coordinates `br` creates a NEW image `nimg` with lines drawn like in the example:

- use a width of 5 pixels

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

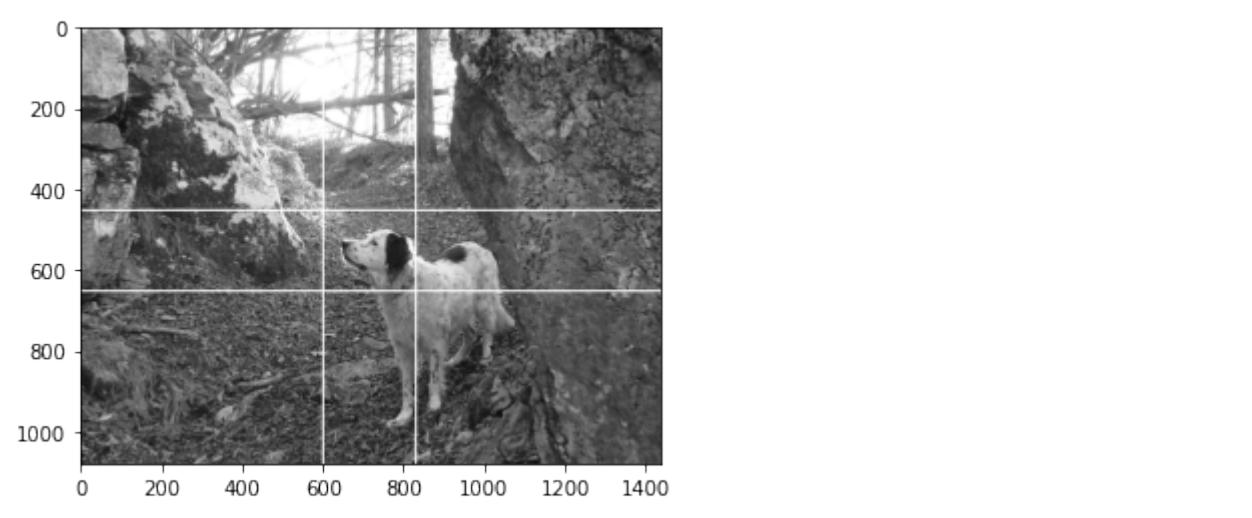
[32] :

```
tl = (450, 600)
br = (650, 830)

write here

nimg = gimg.copy() # *DEEP* COPY
nimg[tl[0]:tl[0]+5, :] = 255
nimg[br[0]:br[0]+5, :] = 255
nimg[:, tl[1]:tl[1]+5] = 255
nimg[:, br[1]:br[1]+5] = 255

gs(nimg)
```

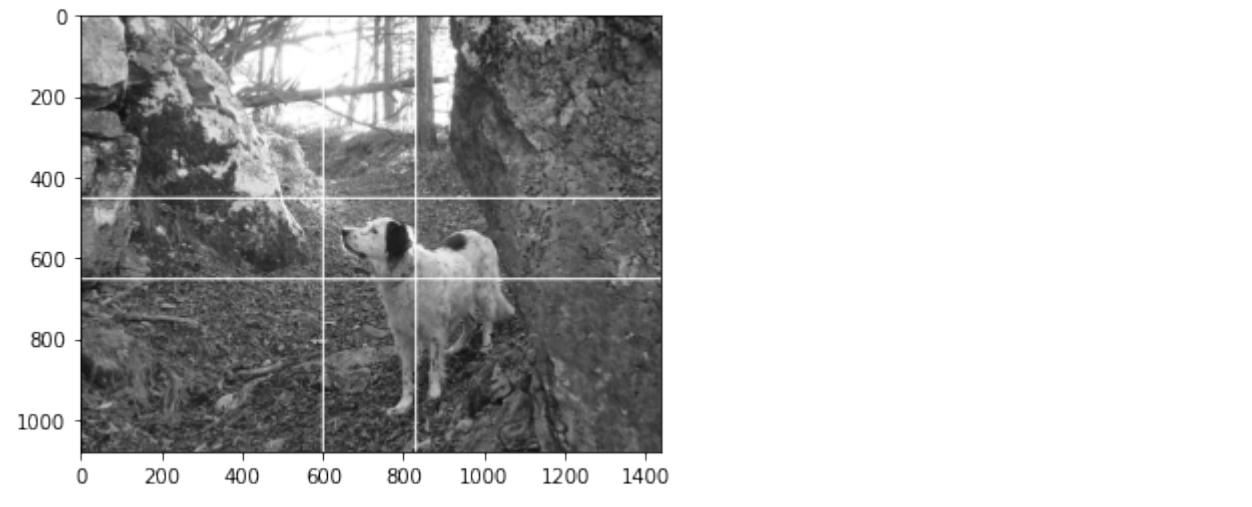


</div>

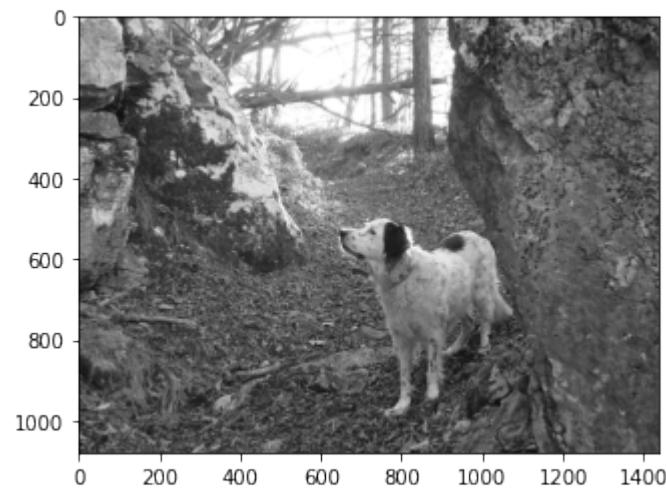
[32]:

```
tl = (450, 600)
br = (650, 830)

write here
```



[33]: `gs(gimg) # original must NOT change!`



### In a dark integer night

Let's say we want to darken the scene. One simple approach would be to divide all the numbers by two:

```
[34]: gs(gimg // 2)
```



```
[35]: gimg // 2
```

```
[35]: array([[104, 104, 105, ..., 58, 59, 58],
 [107, 107, 107, ..., 56, 58, 58],
 [108, 108, 108, ..., 52, 55, 57],
 ...,
 [18, 16, 15, ..., 36, 33, 32],
 [21, 18, 15, ..., 35, 32, 30],
 [18, 15, 12, ..., 34, 31, 30]], dtype=uint8)
```

If we divide by floats we get an array of floats:

```
[36]: gimg / 3.14
```

```
[36]: array([[66.56050955, 66.56050955, 66.87898089, ..., 37.2611465 ,
 37.57961783, 37.2611465],
 [68.15286624, 68.15286624, 68.47133758, ..., 35.66878981,
 36.94267516, 37.2611465],
 [69.10828025, 69.10828025, 69.10828025, ..., 33.43949045,
 35.03184713, 36.30573248],
 ...,
 [11.46496815, 10.50955414, 9.55414013, ..., 22.92993631,
 21.33757962, 20.38216561],
 [13.37579618, 11.46496815, 9.87261146, ..., 22.29299363,
 20.70063694, 19.42675159],
 [11.78343949, 9.87261146, 7.6433121 , ..., 21.65605096,
 20.06369427, 19.10828025]])
```

To go back to unsigned bytes, you can use astype:

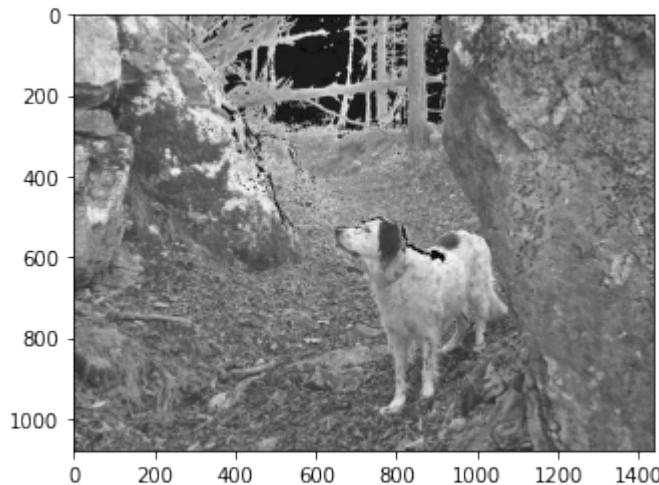
```
[37]: (gimg / 3.0).astype(np.uint8)
[37]: array([[69, 69, 70, ..., 39, 39, 39],
 [71, 71, 71, ..., 37, 38, 39],
 [72, 72, 72, ..., 35, 36, 38],
 ...,
 [12, 11, 10, ..., 24, 22, 21],
 [14, 12, 10, ..., 23, 21, 20],
 [12, 10, 8, ..., 22, 21, 20]], dtype=uint8)
```

We used division because it guarantees we will never go below zero, which is important when working with unsigned bytes as we're doing here. Let's see what happens when we violate the datatype bounds.

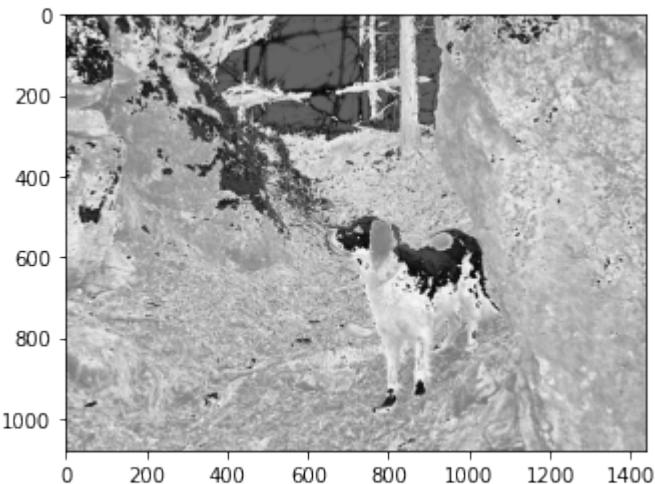
## The Integer Shining

Intuitively, if we want more light we can try increasing the matrices values but something terrible hides in the shadows....

```
[38]: gs(gimg + 30) # mmm something looks wrong ...
```



```
[39]: gs(gimg + 100) # even worse!
```



Something really bad happened:

```
[40]: gimg + 100
[40]: array([[53, 53, 54, ..., 217, 218, 217],
 [58, 58, 59, ..., 212, 216, 217],
 [61, 61, 61, ..., 205, 210, 214],
 ...,
 [136, 133, 130, ..., 172, 167, 164],
 [142, 136, 131, ..., 170, 165, 161],
 [137, 131, 124, ..., 168, 163, 160]], dtype=uint8)
```

Why do we get values less than < 100 ??

This is not so weird, technically it's called integer overflow and is the way CPU works with byte sized integers, so most programming languages actually behave like this. In regular Python you don't notice it because standard Python allows for arbitrary sized integers, but that comes at a big performance cost that Numpy cannot afford, so in a sense we can say Numpy is 'closer to the metal' of the CPU.

Let's see a simpler example:

```
[41]: arr = np.zeros(3, dtype=np.uint8) # unsigned 8 bit byte, values from 0 to 255
 ↪included
```

```
[42]: arr
```

```
[42]: array([0, 0, 0], dtype=uint8)
```

```
[43]: arr[0] = 255
```

```
[44]: arr
```

```
[44]: array([255, 0, 0], dtype=uint8)
```

```
[45]: arr[0] += 1 # cycles back to zero
```

```
[46]: arr
```

```
[46]: array([0, 0, 0], dtype=uint8)
```

```
[47]: arr[0] -= 1 # cycles forward to 255
```

```
[48]: arr
```

```
[48]: array([255, 0, 0], dtype=uint8)
```

Going back to the image, how could we prevent exceeding the limit of 255?

`np.minimum` compares arrays cell by cell:

```
[49]: np.minimum(np.array([5, 7, 2]), np.array([9, 4, 8]))
```

```
[49]: array([5, 4, 2])
```

As well as matrices:

```
[50]: m1 = np.array([[5, 7, 2],
 [8, 3, 1]])
m2 = np.array([[9, 4, 8],
 [6, 0, 3]])
np.minimum(m1, m2)
```

```
[50]: array([[5, 4, 2],
 [6, 0, 1]])
```

If you pass a constant, it will automatically compare all matrix cells against that constant:

```
[51]: np.minimum(m1, 2)
```

```
[51]: array([[2, 2, 2],
 [2, 2, 1]])
```

### Exercise - Be bright

Now try writing some code which enhances scene luminosity by adding `light=125` without distortions (you may still see some pixellation due to the fact we have taken just one color channel from the original image)

- **DO NOT** exceed 255 value for cells - if you see dark spots in your image where before there was white (i.e. background sky), it means color cycled back to small values!
- **DO NOT** write stuff like `gimg + light`, this would surely exceed the 255 bound !!
- **MUST** have unsigned bytes as cells type

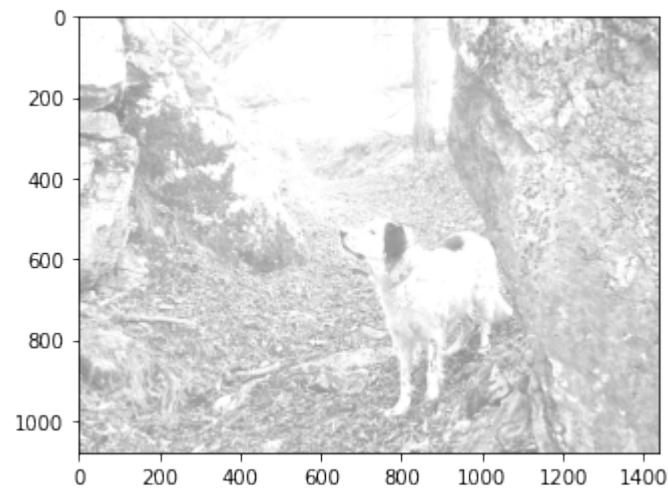
**HINT 1:** if direct sum is not the way, which safe operations are there which surely won't provoke any overflow?

**HINT 2:** you will need more than one step to solve the exercise

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

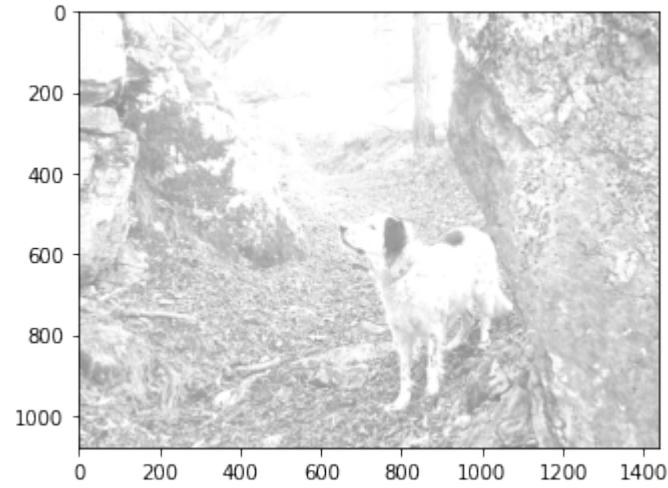
data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[52]: light=125
write here
gs(gimg + np.minimum(255 - gimg, light))
```



</div>

```
[52]: light=125
write here
```

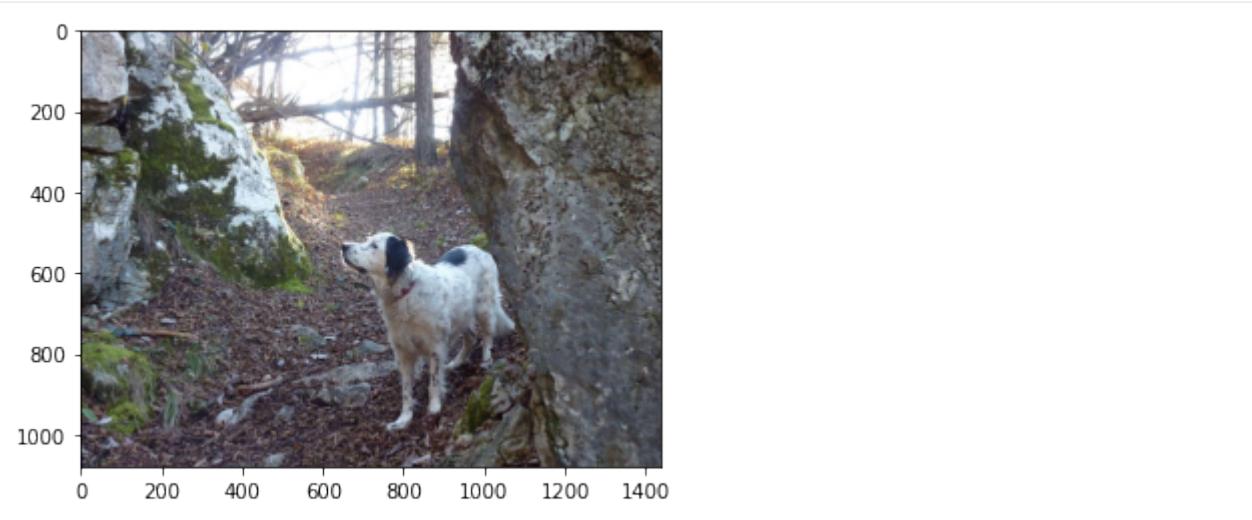


## RGB - Get colorful

Let's get a third dimension for representing colors. Our new third dimension will have three planes of integers, in this order:

- 0: Red
- 1: Green
- 2: Blue

```
[53]: plt.imshow(img)
[53]: <matplotlib.image.AxesImage at 0x7ff584617990>
```



```
[54]: type(img)
```

```
[54]: numpy.ndarray
```

```
[55]: img.shape
```

```
[55]: (1080, 1440, 3)
```

Each pixel is represented by three integer values:

```
[56]: print(img)
```

```
[[[209 223 236]
 [209 223 236]
 [210 224 237]
 ...
 [117 132 139]
 [118 132 141]
 [117 131 140]]]
```

```
[[214 228 241]
 [214 228 241]
 [215 229 242]
 ...
 [112 127 134]
 [116 131 138]
 [117 131 140]]]
```

```
[[217 229 243]
 [217 229 243]
 [217 229 243]
 ...
 [105 120 127]
 [110 125 132]
 [114 129 136]]]
```

```
...
 [[36 28 49]]
```

(continues on next page)

(continued from previous page)

```
[33 25 46]
[30 22 43]
...
[72 78 90]
[67 73 87]
[64 70 84]]]

[[42 34 55]
[36 28 49]
[31 23 44]
...
[70 76 88]
[65 71 85]
[61 67 81]]]

[[37 29 50]
[31 23 44]
[24 16 37]
...
[68 74 86]
[63 69 83]
[60 66 80]]]
```

Given a pixel coordinates, like `0, 0`, we can extract the color with a third coordinate like this:

```
[57]: img[0,0,0] # red
[57]: 209

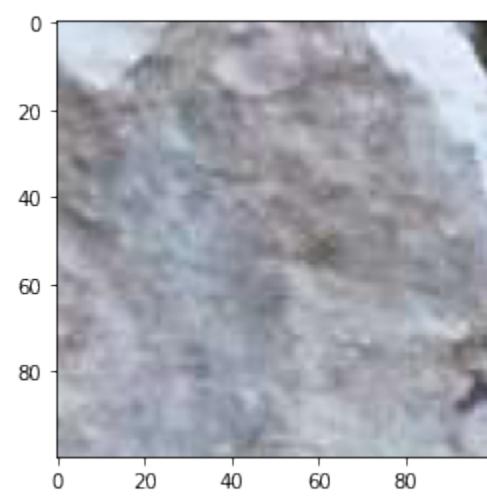
[58]: img[0,0,1] # green
[58]: 223

[59]: img[0,0,2] # blue
[59]: 236

[60]: img[0,0] # result is an array with three RGB colors
[60]: array([209, 223, 236], dtype=uint8)
```

### Exercise - Focus - top left

```
[61]: plt.imshow(img[:100,:100,:])
[61]: <matplotlib.image.AxesImage at 0x7ff5845b1dd0>
```

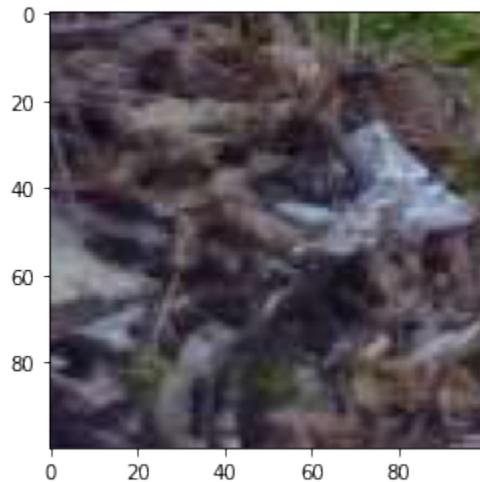


### Exercise - Focus - bottom - left

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[62]: # write here
plt.imshow(img[-100:,:100,:])
```

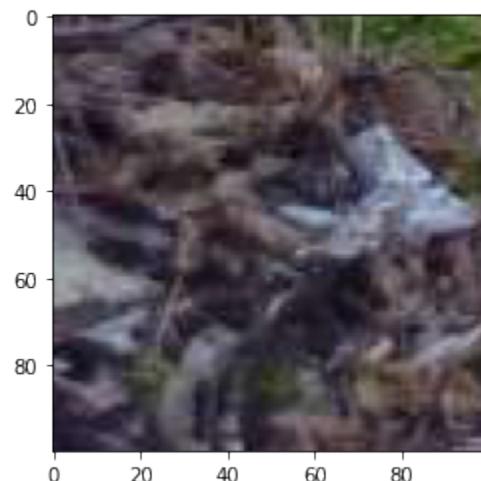
```
[62]: <matplotlib.image.AxesImage at 0x7ff5844b2f90>
```



```
</div>
```

```
[62]: # write here
```

```
[62]: <matplotlib.image.AxesImage at 0x7ff5844b2f90>
```

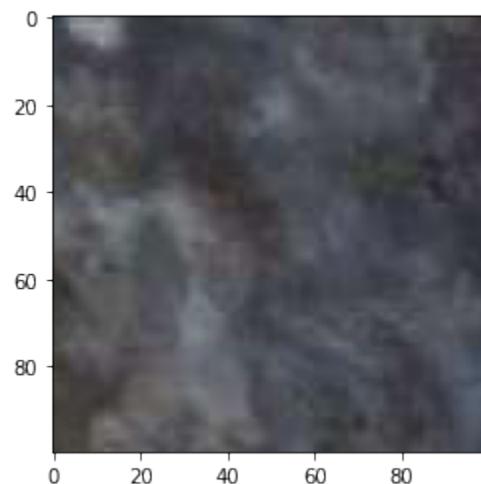


### Exercise - Focus - bottom - right

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[63]: # write here
plt.imshow(img[-100:,-100:,:])
```

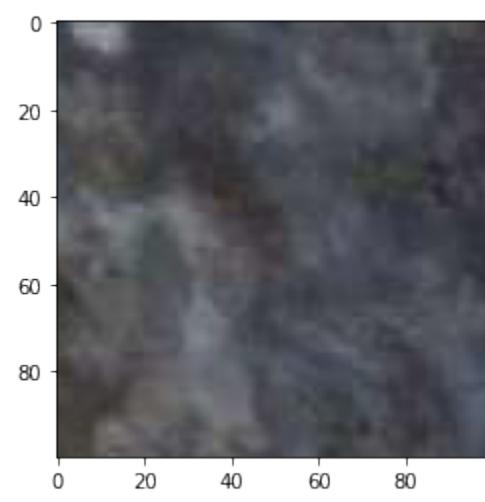
```
[63]: <matplotlib.image.AxesImage at 0x7ff5842d5090>
```



```
</div>
```

```
[63]: # write here
```

```
[63]: <matplotlib.image.AxesImage at 0x7ff5842d5090>
```

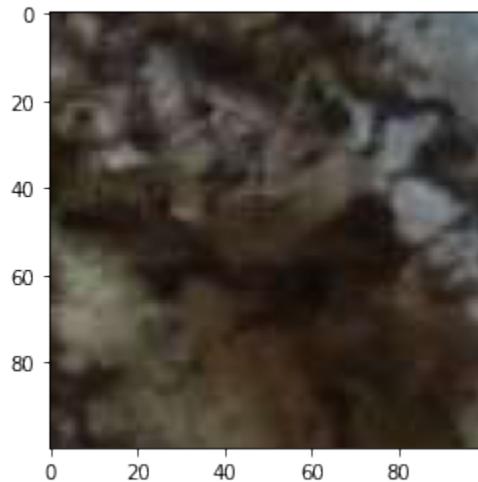


### Exercise - Focus - top - right

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[64]: # write here
plt.imshow(img[:100,-100:,:])
```

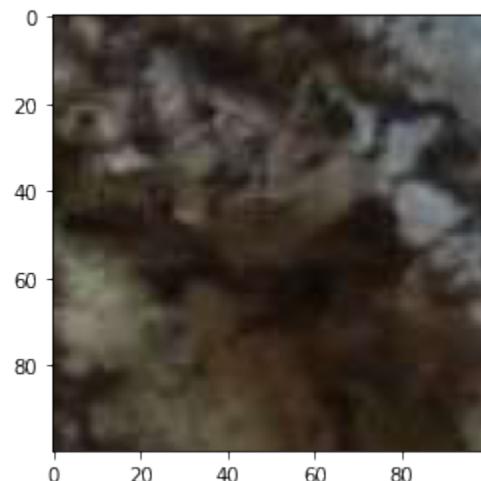
[64]: <matplotlib.image.AxesImage at 0x7ff58432c990>



</div>

```
[64]: # write here
```

[64]: <matplotlib.image.AxesImage at 0x7ff58432c990>

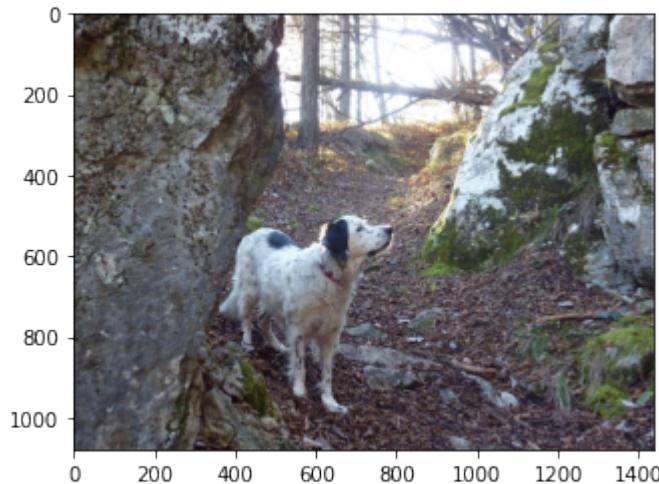


### Exercise - Look the other way

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[65]: # write here
plt.imshow(np.fliplr(img))
```

```
[65]: <matplotlib.image.AxesImage at 0x7ff5845f71d0>
```



```
</div>
```

```
[65]: # write here
```

```
[65]: <matplotlib.image.AxesImage at 0x7ff5845f71d0>
```

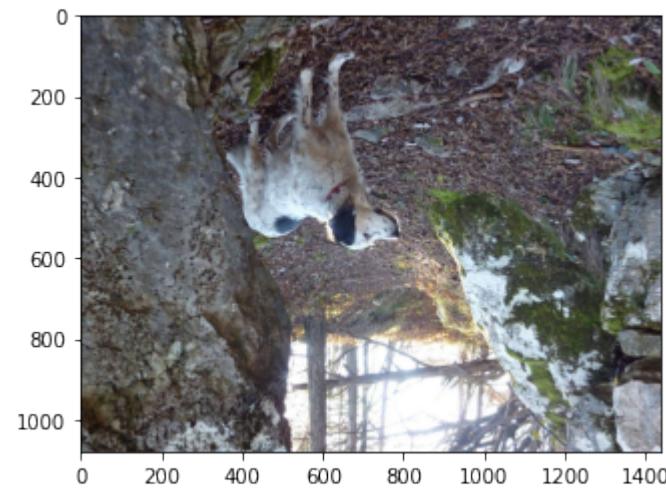


### Exercise - Upside down world

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[66]: # write here
plt.imshow(np.fliplr(np.flipud(img)))
```

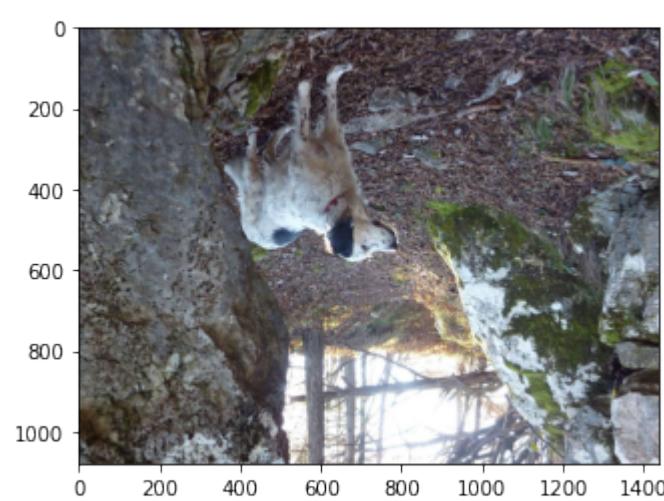
```
[66]: <matplotlib.image.AxesImage at 0x7ff58447e590>
```



```
</div>
```

```
[66]: # write here
```

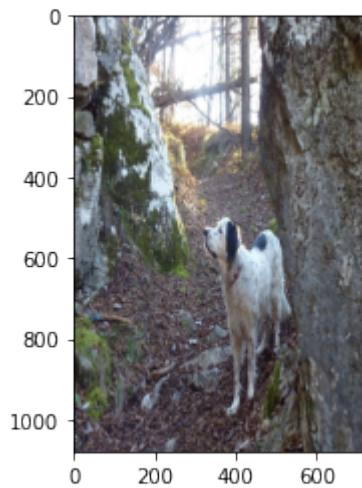
```
[66]: <matplotlib.image.AxesImage at 0x7ff58447e590>
```



### Exercise - Shrinking Walls - X

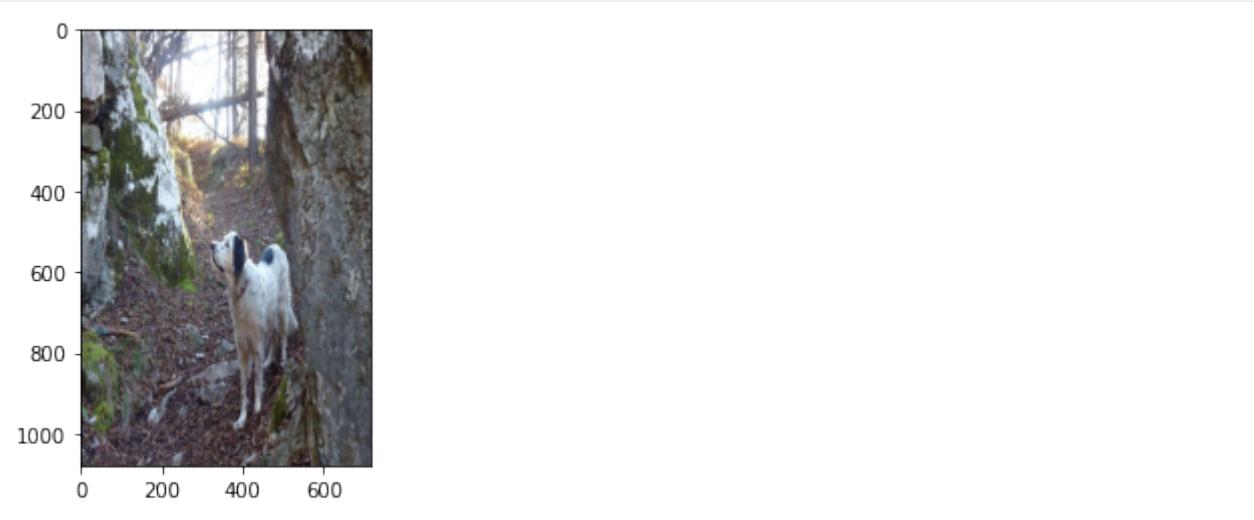
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[67]: # write here
plt.imshow(img[:,::2,:])
[67]: <matplotlib.image.AxesImage at 0x7ff584486e50>
```



</div>

```
[67]: # write here
[67]: <matplotlib.image.AxesImage at 0x7ff584486e50>
```



### Exercise - Shrinking Walls - Y

Show solution  
Hide

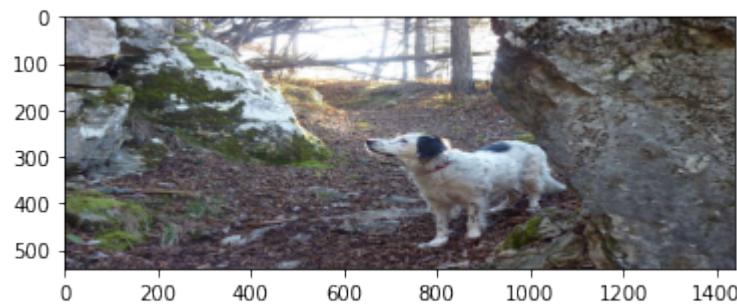
```
[68]: # write here
plt.imshow(img[:, :, :])
[68]: <matplotlib.image.AxesImage at 0x7ff58427e7d0>
```



```
</div>
```

```
[68]: # write here

[68]: <matplotlib.image.AxesImage at 0x7ff58427e7d0>
```



### Exercise - Shrinking World

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

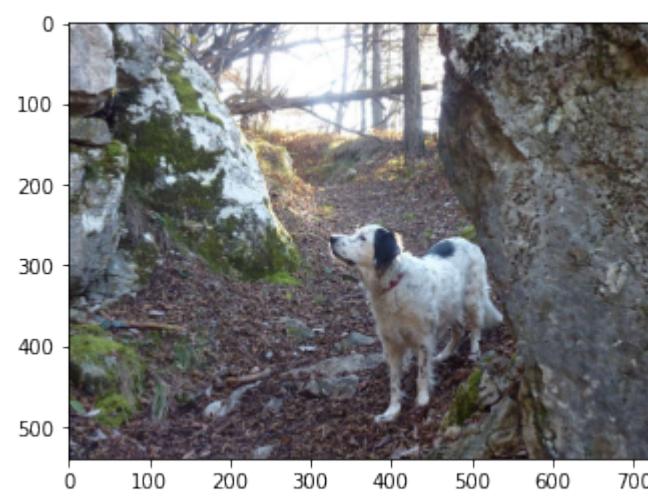
```
[69]: # write here
plt.imshow(img[::2, ::2, :])
[69]: <matplotlib.image.AxesImage at 0x7ff58420f1d0>
```



</div>

```
[69]: # write here

[69]: <matplotlib.image.AxesImage at 0x7ff58420f1d0>
```

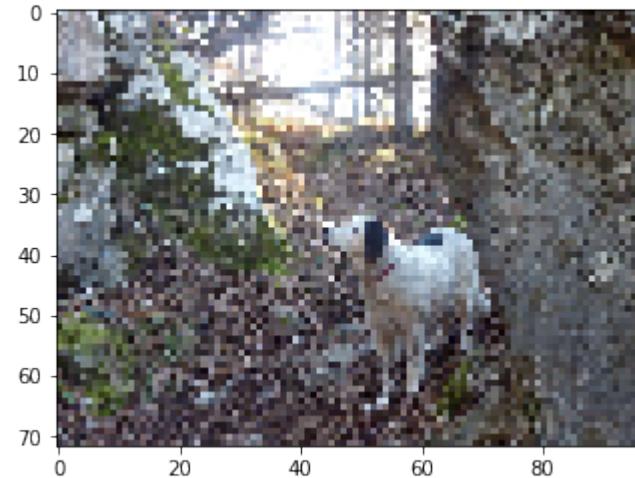


### Exercise - Pixelate

Show solutionHide

```
[70]: # write here
plt.imshow(img[::15, ::15, :])
```

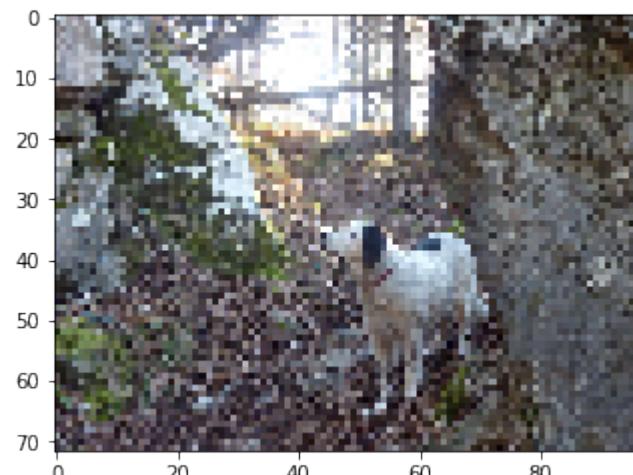
```
[70]: <matplotlib.image.AxesImage at 0x7ff58406df10>
```



```
</div>
```

```
[70]: # write here
```

```
[70]: <matplotlib.image.AxesImage at 0x7ff58406df10>
```



### Exercise - Feeling Red

Create a NEW image where you only see red

Show solutionHide>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[71]: # write here
mimg = img.copy()
mimg[:, :, 2] = 0
mimg[:, :, 1] = 0
plt.imshow(mimg)
```

[71]: <matplotlib.image.AxesImage at 0x7ff57efd5990>



</div>

```
[71]: # write here
```

```
[71]: <matplotlib.image.AxesImage at 0x7ff57efd5990>
```



### Exercise - Feeling Green

Create a NEW image where you only see green

Show solutionHide

```
[72]: # write here
mimg = img.copy()
mimg[:, :, 0] = 0
mimg[:, :, 2] = 0
plt.imshow(mimg)
```

```
[72]: <matplotlib.image.AxesImage at 0x7ff57ef5f5d0>
```



</div>

```
[72]: # write here
```

```
[72]: <matplotlib.image.AxesImage at 0x7ff57ef5f5d0>
```



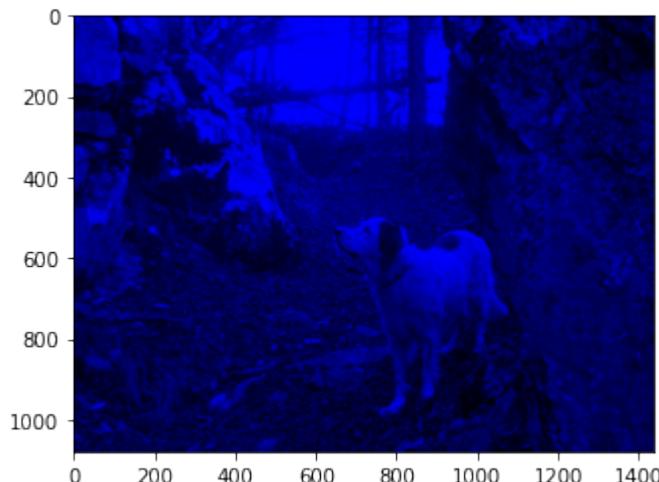
### Exercise - Feeling Blue

Create a NEW image where you only see blue

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[73]: # write here
mimg = img.copy()
mimg[:, :, 0] = 0
mimg[:, :, 1] = 0
plt.imshow(mimg)
```

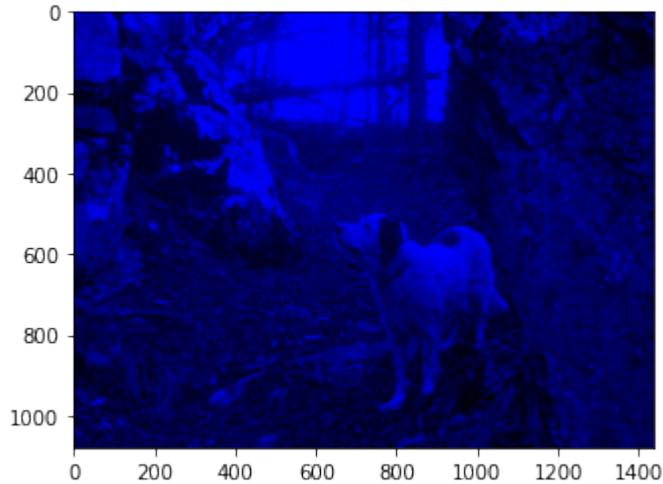
```
[73]: <matplotlib.image.AxesImage at 0x7ff57eeeaa510>
```



```
</div>
```

```
[73]: # write here
```

```
[73]: <matplotlib.image.AxesImage at 0x7ff57eee510>
```



### Exercise - No Red

Create a NEW image without red

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[74]: # write here
mimg = img.copy()
mimg[:, :, 0] = 0
plt.imshow(mimg)
```

```
[74]: <matplotlib.image.AxesImage at 0x7ff57ee771d0>
```



</div>

```
[74]: # write here

[74]: <matplotlib.image.AxesImage at 0x7ff57ee771d0>
```



### Exercise - No Green

Create a NEW image without green

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[75]: # write here
mimg = img.copy()
mimg[:, :, 1] = 0
plt.imshow(mimg)
```

```
[75]: <matplotlib.image.AxesImage at 0x7ff57ede1fd0>
```



```
</div>
```

```
[75]: # write here
```

```
[75]: <matplotlib.image.AxesImage at 0x7ff57ede1fd0>
```



### Exercise - No Blue

Create a NEW image without blue

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[76]: # write here
mimg = img.copy()
mimg[:, :, 2] = 0
plt.imshow(mimg)
```

```
[76]: <matplotlib.image.AxesImage at 0x7ff57ed01390>
```



</div>

[76]: # write here

[76]: &lt;matplotlib.image.AxesImage at 0x7ff57ed01390&gt;



### Exercise - Feeling Gray again

Given an RGB image, set all the values equal to red channel

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[77]: # write here

```
mimg = img.copy()
mimg[:, :, 1] = mimg[:, :, 0]
mimg[:, :, 2] = mimg[:, :, 0]
plt.imshow(mimg)
print(mimg)
```

```
[[209 209 209]
 [209 209 209]
 [210 210 210]
 ...
 [117 117 117]
 [118 118 118]
 [117 117 117]]
```

```
[[214 214 214]
 [214 214 214]
 [215 215 215]
 ...
 [112 112 112]
 [116 116 116]
 [117 117 117]]
```

```
[[217 217 217]
 [217 217 217]
 [217 217 217]]
```

(continues on next page)

(continued from previous page)

```

...
[105 105 105]
[110 110 110]
[114 114 114]]
```

...

```

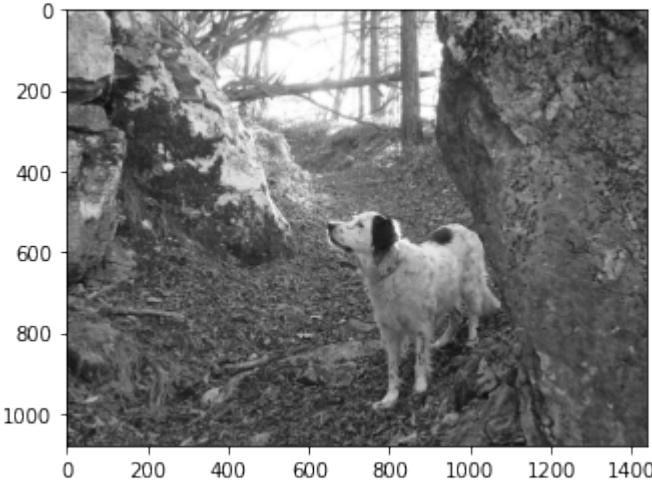
[[36 36 36]
 [33 33 33]
 [30 30 30]
 ...
 [[72 72 72]
 [67 67 67]
 [64 64 64]]]
```

```

[[42 42 42]
 [36 36 36]
 [31 31 31]
 ...
 [[70 70 70]
 [65 65 65]
 [61 61 61]]]
```

```

[[37 37 37]
 [31 31 31]
 [24 24 24]
 ...
 [[68 68 68]
 [63 63 63]
 [60 60 60]]]]
```



&lt;/div&gt;

[77]: # write here

```

[[[209 209 209]
 [209 209 209]]]
```

(continues on next page)

(continued from previous page)

[210 210 210]

...

[117 117 117]

[118 118 118]

[117 117 117]]

[[214 214 214]

[214 214 214]

[215 215 215]

...

[112 112 112]

[116 116 116]

[117 117 117]]

[[217 217 217]

[217 217 217]

[217 217 217]

...

[105 105 105]

[110 110 110]

[114 114 114]]

...

[[ 36 36 36]

[ 33 33 33]

[ 30 30 30]

...

[ 72 72 72]

[ 67 67 67]

[ 64 64 64]]

[[ 42 42 42]

[ 36 36 36]

[ 31 31 31]

...

[ 70 70 70]

[ 65 65 65]

[ 61 61 61]]

[[ 37 37 37]

[ 31 31 31]

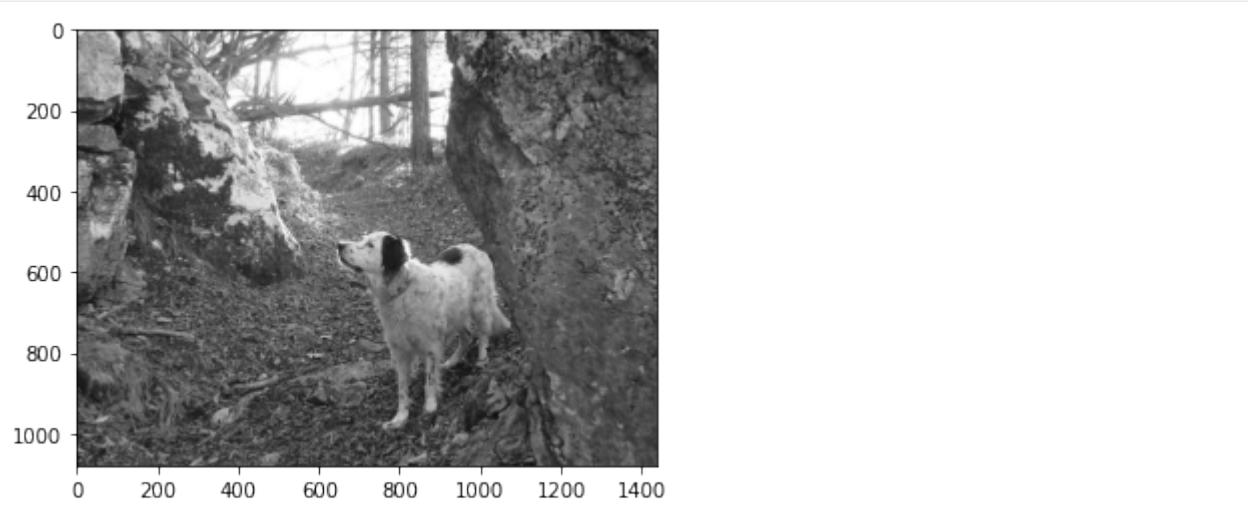
[ 24 24 24]

...

[ 68 68 68]

[ 63 63 63]

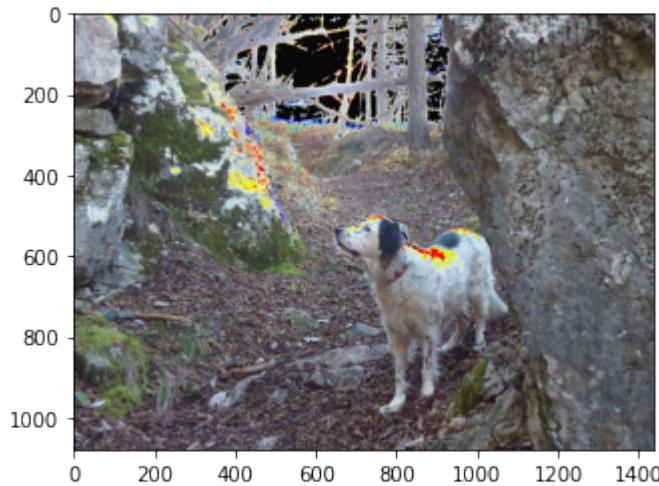
[ 60 60 60]]]



### Exercise - Beyond the limit

... weird things happen:

```
[78]: plt.imshow(img + 10)
[78]: <matplotlib.image.AxesImage at 0x7ff57ec72b50>
```



```
[79]: mimg = img.copy()
mimg[0,0,0] = 255 # limit !!
mimg[0,0,0]
```

```
[79]: 255
```

```
[80]: mimg[0,0,0] += 1 # integer overflow, cycles back - note it does not happen in
regular Python !
```

```
[81]: mimg[0,0,0]
```

```
[81]: 0
```

Note this is not so weird, technically this is called overflow and us the way CPU works with byte sized integers, so most programming languages actually behave like this.

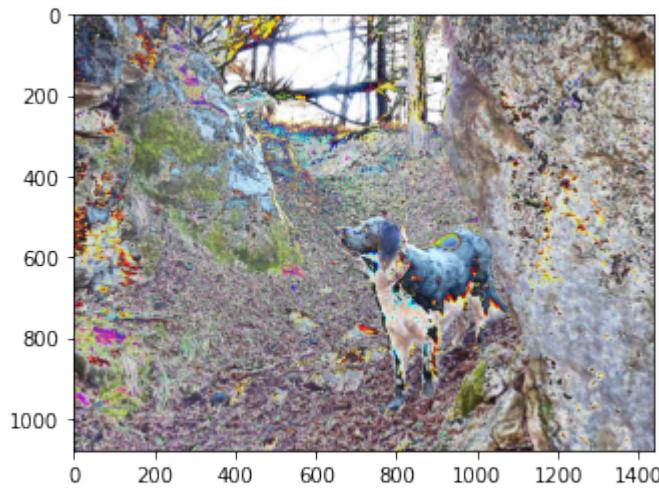
You can get the same problem when subtracting:

```
[82]: mimg[0,0,0] = 0 # limit !!
mimg[0,0,0] -= 1 # integer overflow , cycles forward
mimg[0,0,0]
```

```
[82]: 255
```

```
[83]: plt.imshow(img + img)
```

```
[83]: <matplotlib.image.AxesImage at 0x7ff57ebef550>
```



```
[84]: plt.imshow(img) # + operator didn't change original image
```

```
[84]: <matplotlib.image.AxesImage at 0x7ff57eb55250>
```



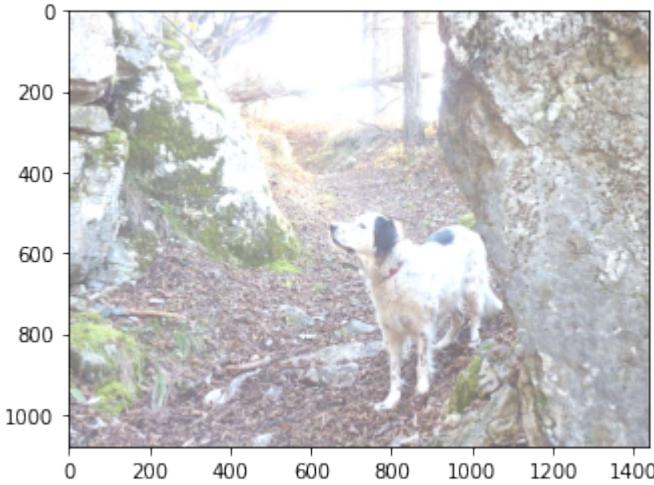
### Exercise - Gimme light

Increment all the RGB values of `light`, **without overflowing**

Show solutionHide>Show solution</a><div class="jupman-sol-jupman-sol-code" style="display:none">

```
[85]: light = 100

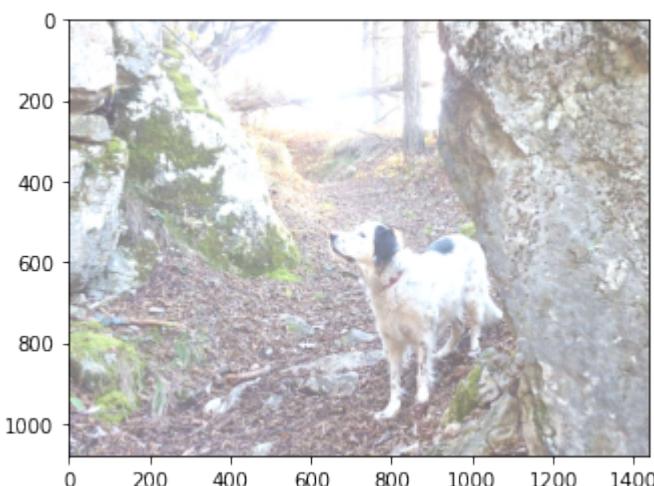
write here
plt.imshow(img + np.minimum(255 - img, light))
[85]: <matplotlib.image.AxesImage at 0x7ff57eaf8590>
```



</div>

```
[85]: light = 100

write here
[85]: <matplotlib.image.AxesImage at 0x7ff57eaf8590>
```



### Exercise - When the darkness comes - with a warning

Decrement all values by light. As a first attempt, a result with a warning might be considered acceptable.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[86]: light = -50
write here
plt.imshow(img + np.minimum(255 - img, light))

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
↪[0..255] for integers).
```

[86]: <matplotlib.image.AxesImage at 0x7ff57ea00510>

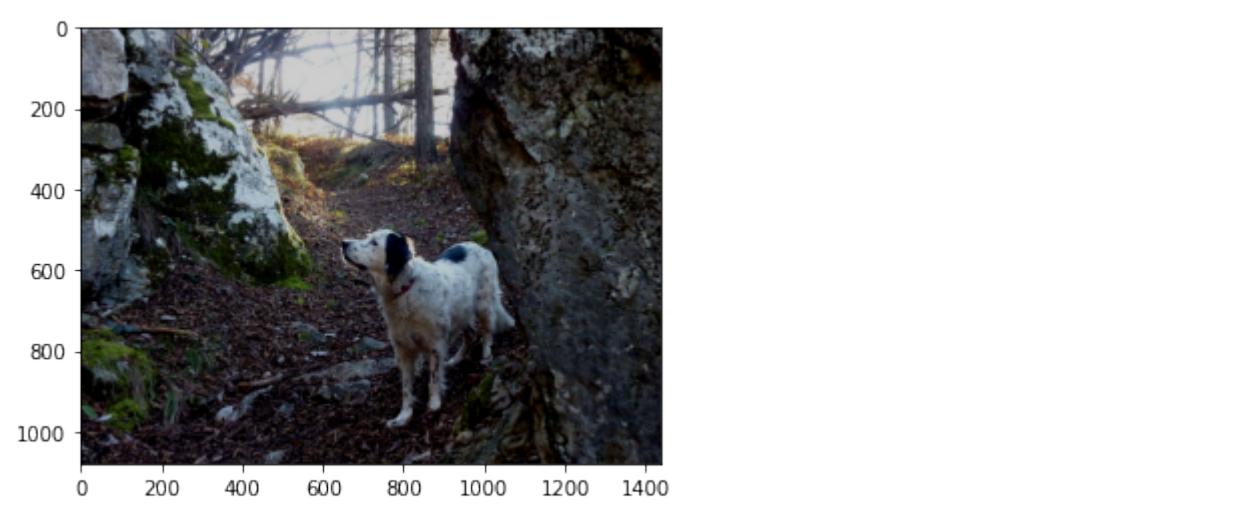


</div>

```
[86]: light = -50
write here

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
↪[0..255] for integers).
```

[86]: <matplotlib.image.AxesImage at 0x7ff57ea00510>



### Exercise - When the darkness comes - without a warning

Decrement all RGB values by light, **without overflowing nor warnings**

Show solutionHide>

```
[87]: light=50
write here
plt.imshow(img - np.minimum(img, light))
```

[87]: <matplotlib.image.AxesImage at 0x7ff57e9eb0>



</div>

```
[87]: light=50
write here
```

```
[87]: <matplotlib.image.AxesImage at 0x7ff57e9ebcd0>
```



### Exercise - Fade to black

Fade the gray picture to black from left to right. Try using `np.linspace` and `np.tile`

First create the `horiz_fade`:

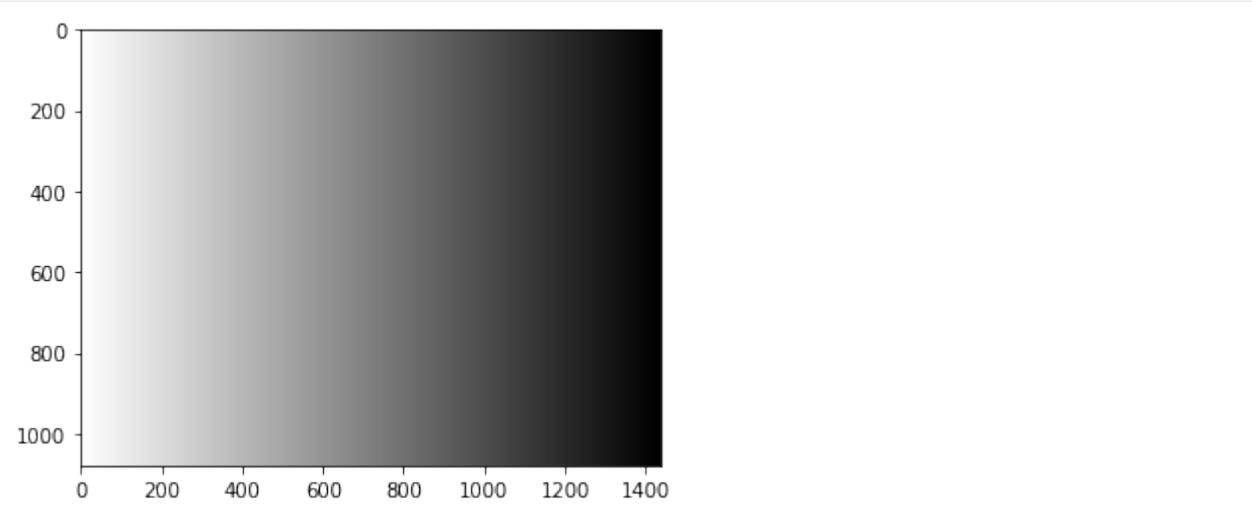
```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[88]: # write here
ls = np.linspace(255, 0, gimg.shape[1])
horiz_fade = np.tile(ls, (gimg.shape[0], 1))
```

```
</div>
```

```
[88]: # write here
```

```
[89]: gs(horiz_fade)
```



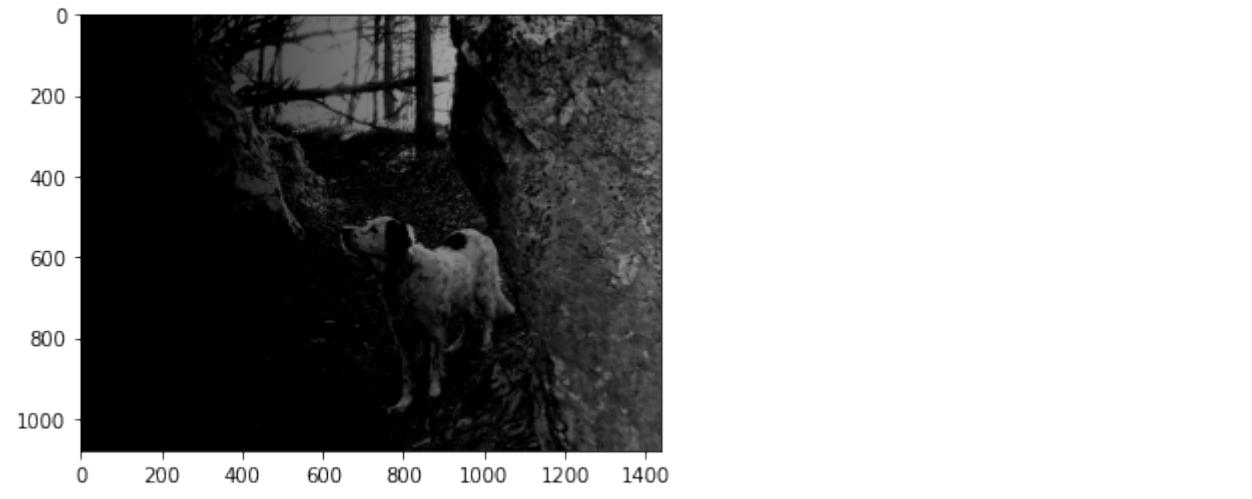
Then apply the fade - notice that by ‘applying’ we mean subtracting the fade (so white in the fade will actually correspond to dark in the picture)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

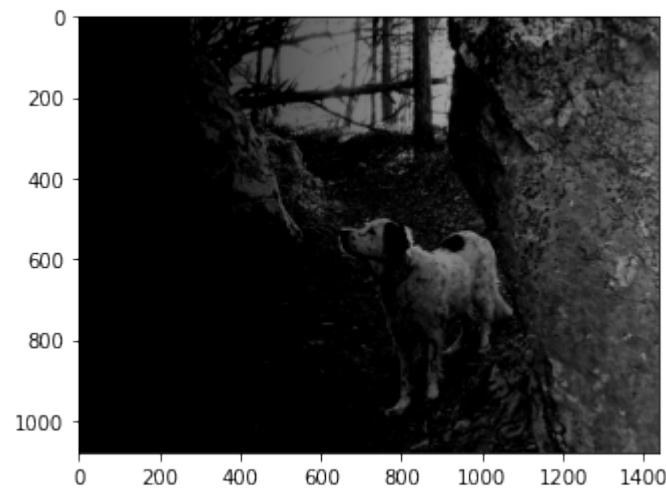
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[90]: # write here
gs(gimg - np.minimum(gimg, horiz_fade))
```



</div>

```
[90]: # write here
```



### Exercise - vertical fade

(harder) First create a `vertical_fade`:

Show solutionHide>

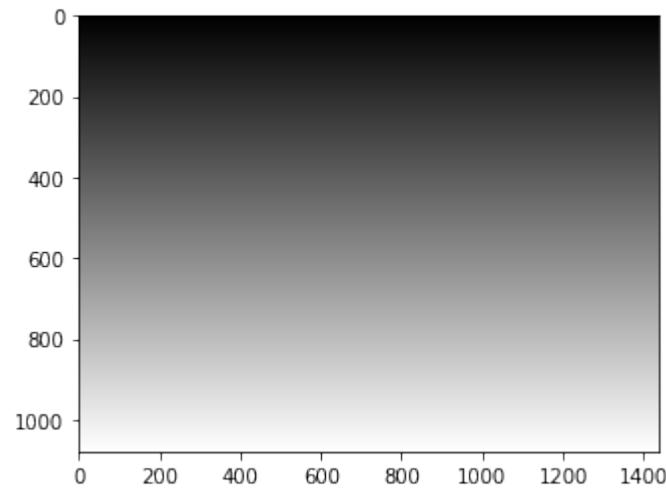
```
[91]: # write here

ls = np.linspace(0,255,gimg.shape[0])
vertical_fade = np.repeat(ls, gimg.shape[1]).reshape(gimg.shape[0], gimg.shape[1])
```

</div>

```
[91]: # write here
```

```
[92]: gs(vertical_fade)
```

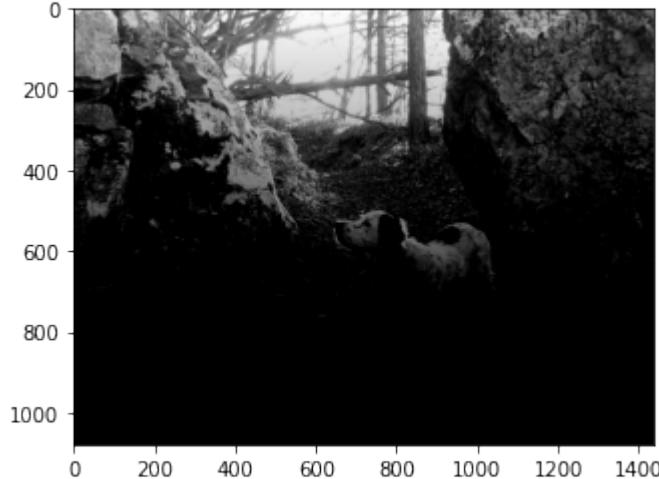


Then apply the fade:

```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

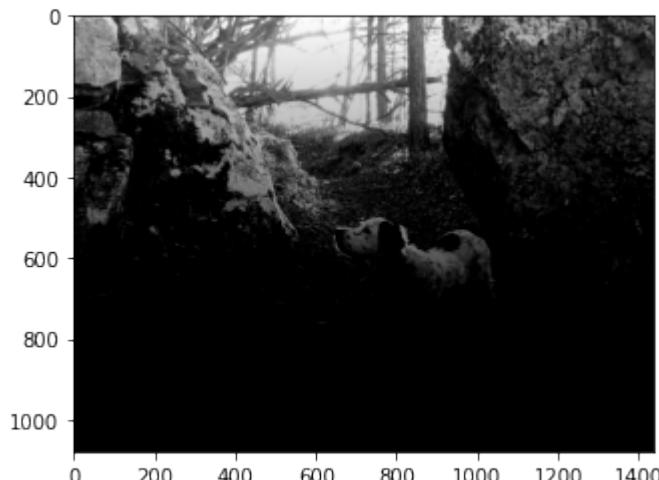
```
[93]: # write here
```

```
gs(gimg - np.minimum(gimg, vertical_fade))
```



```
</div>
```

```
[93]: # write here
```



## 8.3 Pandas

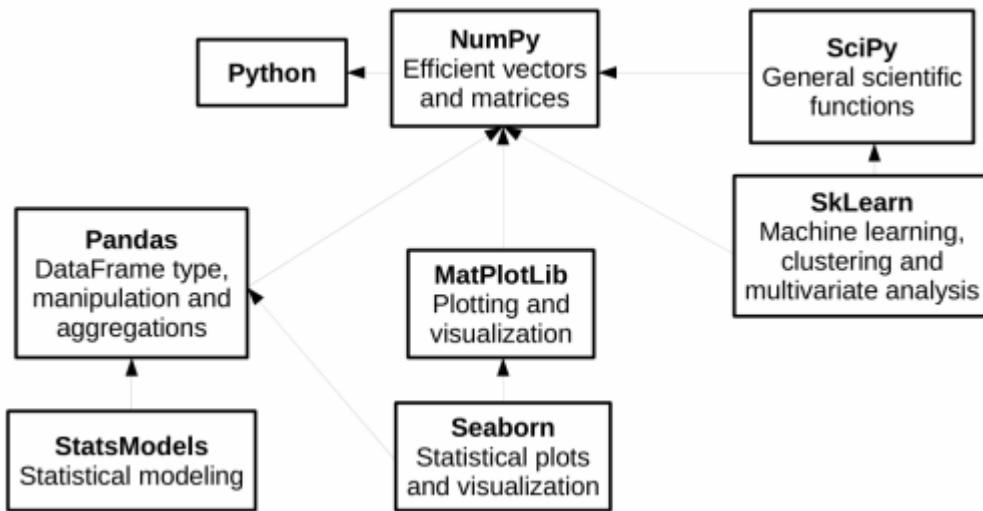
### 8.3.1 Analytics with Pandas : 1. introduction

[Download exercises zip](#)

Browse files online<sup>361</sup>

Python gives powerful tools for data analysis - among the main ones we find [Pandas](#)<sup>362</sup>, which gives fast and flexible data structures, especially for interactive data analysis.

Pandas reuses existing libraries we've already seen, such as Numpy:



In this first part of the tutorial we will see:

- filtering and transformation operations on Pandas dataframes
- plotting with Matplotlib
- Examples with AstroPi dataset
- Exercises with meteotrentino and other datasets

#### 1. What to do

1. unzip exercises in a folder, you should get something like this:

```
pandas
pandas1-intro.ipynb
pandas1-intro-sol.ipynb
pandas2-advanced.ipynb
pandas2-advanced-sol.ipynb
pandas3-chal.ipynb
jupman.py
```

<sup>361</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/pandas>

<sup>362</sup> <https://pandas.pydata.org/>

**WARNING 1:** to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then browser.
3. The browser should show a file list: navigate the list and open the notebook pandas/pandas1.ipynb

**WARNING 2:** DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

4. Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## Check installation

First let's see if you have already installed pandas on your system, try executing this cell with Ctrl + Enter:

```
[1]: import pandas as pd
```

If you saw no error messages, you can skip installation, otherwise do this:

- Anaconda - open Anaconda Prompt and issue this:

```
conda install pandas
```

- Without Anaconda (--user installs in your home):

```
python3 -m pip install --user pandas
```

## Which pandas should I use?

In this tutorial we adopt version 1 of pandas which is based upon numpy, as at present (2023) it's the most common one and usually the tutorials you find around refer to this version. We should also consider that version 2 was released on April 2023 which is more efficient, can optionally support the PyArrow<sup>363</sup> engine and has better support for 'nullable' types.

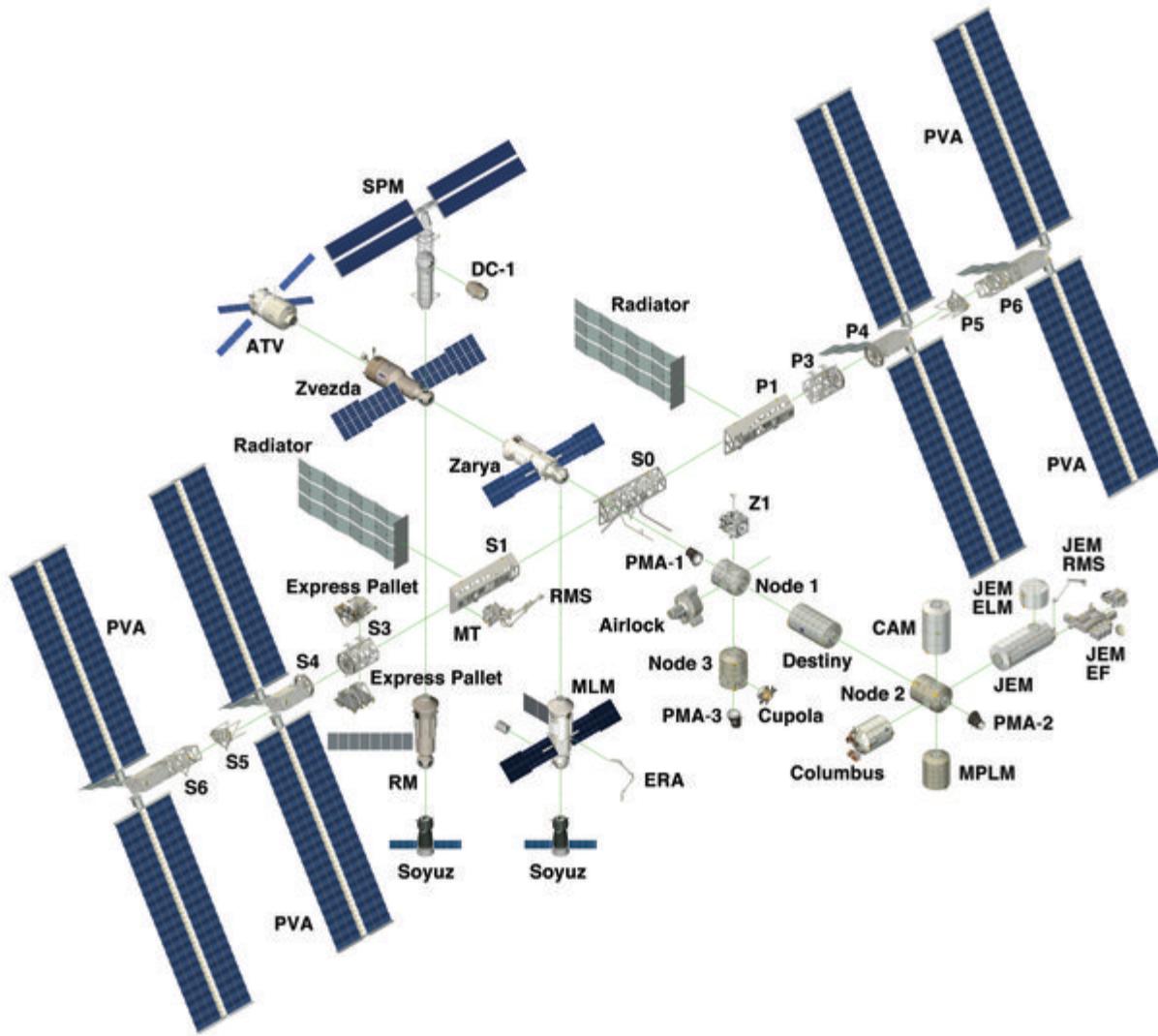
## 2. Data analysis of Astro Pi

Let's try analyzing data recorded on a RaspberryPi electronic board installed on the International Space Station (ISS). Data was downloaded from here:

<https://projects.raspberrypi.org/en/projects/astro-pi-flight-data-analysis>

In the website it's possible to find a detailed description of data gathered by sensors, in the month of February 2016 (one record each 10 seconds).

<sup>363</sup> <https://levelup.gitconnected.com/welcoming-pandas-2-0-194094e4275b>



## 2.1 Let's import the file

The method `read_csv` imports data from a CSV file and saves them in a DataFrame structure.

In this exercise we shall use the file `astropi.csv` (slightly modified by replacing `ROW_ID` column with the `time_stamp`)

```
[2]: import pandas as pd # we import pandas and for ease we rename it to 'pd'
import numpy as np # we import numpy and for ease we rename it to 'np'

remember the encoding !
df = pd.read_csv('astropi.csv', encoding='UTF-8')
df.info()
```

<code>&lt;class 'pandas.core.frame.DataFrame'&gt;</code>		
RangeIndex: 110869 entries, 0 to 110868		
Data columns (total 19 columns):		
# Column Non-Null Count Dtype		
---	-----	-----

(continues on next page)

(continued from previous page)

```

0 time_stamp 110869 non-null object
1 temp_cpu 110869 non-null float64
2 temp_h 110869 non-null float64
3 temp_p 110869 non-null float64
4 humidity 110869 non-null float64
5 pressure 110869 non-null float64
6 pitch 110869 non-null float64
7 roll 110869 non-null float64
8 yaw 110869 non-null float64
9 mag_x 110869 non-null float64
10 mag_y 110869 non-null float64
11 mag_z 110869 non-null float64
12 accel_x 110869 non-null float64
13 accel_y 110869 non-null float64
14 accel_z 110869 non-null float64
15 gyro_x 110869 non-null float64
16 gyro_y 110869 non-null float64
17 gyro_z 110869 non-null float64
18 reset 110869 non-null int64
dtypes: float64(17), int64(1), object(1)
memory usage: 16.1+ MB

```

## 2.2 Memory

Pandas loads the dataset from hard disk to your computer RAM memory (which in 2023 is typically 8 gigabytes). If by chance your dataset were bigger than available RAM, you would get an error and should start thinking about other tools to perform your analysis. You might also get troubles when making copies of the dataframe. It's then very important to understand how much the dataset actually occupies in RAM. If you look at the bottom, you will see written *memory usage: 16.1 MB* but **pay attention to that** +: Pandas is telling us the dataset occupies in RAM *at least* 16.1 Mb, but the actual dimension could be greater.

To see the actual occupied space, try adding the parameter `memory_usage="deep"` to the `df.info` call. The parameter is option because according to the dataset it might take some time to calculate. Do you notice any difference?

- How much space is taken by the original file on your disk? Try to find it by looking in Windows Explorer.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"  
data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```

write here
df.info(memory_usage="deep")

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 19 columns):
 # Column Non-Null Count Dtype

 0 time_stamp 110869 non-null object
 1 temp_cpu 110869 non-null float64
 2 temp_h 110869 non-null float64
 3 temp_p 110869 non-null float64
 4 humidity 110869 non-null float64
 5 pressure 110869 non-null float64
 6 pitch 110869 non-null float64
 7 roll 110869 non-null float64

```

(continues on next page)

(continued from previous page)

```

8 yaw 110869 non-null float64
9 mag_x 110869 non-null float64
10 mag_y 110869 non-null float64
11 mag_z 110869 non-null float64
12 accel_x 110869 non-null float64
13 accel_y 110869 non-null float64
14 accel_z 110869 non-null float64
15 gyro_x 110869 non-null float64
16 gyro_y 110869 non-null float64
17 gyro_z 110869 non-null float64
18 reset 110869 non-null int64
dtypes: float64(17), int64(1), object(1)
memory usage: 23.3 MB

```

&lt;/div&gt;

[3]:

# write here

## 2.3 Dimensions

We can quickly see rows and columns of the dataframe with the attribute `shape`:

**NOTE:** `shape` is **not** followed by round parenthesis !

[4]: df.shape

[4]: (110869, 19)

## 2.4 Let's explore!

[5]: df

```

[5]: time_stamp temp_cpu temp_h temp_p humidity pressure \
0 2016-02-16 10:44:40 31.88 27.57 25.01 44.94 1001.68
1 2016-02-16 10:44:50 31.79 27.53 25.01 45.12 1001.72
2 2016-02-16 10:45:00 31.66 27.53 25.01 45.12 1001.72
3 2016-02-16 10:45:10 31.69 27.52 25.01 45.32 1001.69
4 2016-02-16 10:45:20 31.66 27.54 25.01 45.18 1001.71
... ...
110864 2016-02-29 09:24:21 31.56 27.52 24.83 42.94 1005.83
110865 2016-02-29 09:24:30 31.55 27.50 24.83 42.72 1005.85
110866 2016-02-29 09:24:41 31.58 27.50 24.83 42.83 1005.85
110867 2016-02-29 09:24:50 31.62 27.50 24.83 42.81 1005.88
110868 2016-02-29 09:25:00 31.57 27.51 24.83 42.94 1005.86

 pitch roll yaw mag_x mag_y mag_z accel_x \
0 1.49 52.25 185.21 -46.422753 -8.132907 -12.129346 -0.000468
1 1.03 53.73 186.72 -48.778951 -8.304243 -12.943096 -0.000614
2 1.24 53.57 186.21 -49.161878 -8.470832 -12.642772 -0.000569
3 1.57 53.63 186.03 -49.341941 -8.457380 -12.615509 -0.000575
4 0.85 53.66 186.46 -50.056683 -8.122609 -12.678341 -0.000548

```

(continues on next page)

(continued from previous page)

```

...
110864 1.58 49.93 129.60 -15.169673 -27.642610 1.563183 -0.000682
110865 1.89 49.92 130.51 -15.832622 -27.729389 1.785682 -0.000736
110866 2.09 50.00 132.04 -16.646212 -27.719479 1.629533 -0.000647
110867 2.88 49.69 133.00 -17.270447 -27.793136 1.703806 -0.000835
110868 2.17 49.77 134.18 -17.885872 -27.824149 1.293345 -0.000787

 accel_y accel_z gyro_x gyro_y gyro_z reset
0 0.019439 0.014569 0.000942 0.000492 -0.000750 20
1 0.019436 0.014577 0.000218 -0.000005 -0.000235 0
2 0.019359 0.014357 0.000395 0.000600 -0.000003 0
3 0.019383 0.014409 0.000308 0.000577 -0.000102 0
4 0.019378 0.014380 0.000321 0.000691 0.000272 0
...
110864 0.017743 0.014646 -0.000264 0.000206 0.000196 0
110865 0.017570 0.014855 0.000143 0.000199 -0.000024 0
110866 0.017657 0.014799 0.000537 0.000257 0.000057 0
110867 0.017635 0.014877 0.000534 0.000456 0.000195 0
110868 0.017261 0.014380 0.000459 0.000076 0.000030 0

[110869 rows x 19 columns]

```

Note the first bold numerical column is an integer index that was automatically assigned by Pandas when the dataset got loaded. Note also it starts from zero. If we wanted, we could set a different index but we won't do it in this tutorial.

`head()` method gives back the first datasets:

```
[6]: df.head()
[6]:
 time_stamp temp_cpu temp_h temp_p humidity pressure pitch \
0 2016-02-16 10:44:40 31.88 27.57 25.01 44.94 1001.68 1.49
1 2016-02-16 10:44:50 31.79 27.53 25.01 45.12 1001.72 1.03
2 2016-02-16 10:45:00 31.66 27.53 25.01 45.12 1001.72 1.24
3 2016-02-16 10:45:10 31.69 27.52 25.01 45.32 1001.69 1.57
4 2016-02-16 10:45:20 31.66 27.54 25.01 45.18 1001.71 0.85

 roll yaw mag_x mag_y mag_z accel_x accel_y \
0 52.25 185.21 -46.422753 -8.132907 -12.129346 -0.000468 0.019439
1 53.73 186.72 -48.778951 -8.304243 -12.943096 -0.000614 0.019436
2 53.57 186.21 -49.161878 -8.470832 -12.642772 -0.000569 0.019359
3 53.63 186.03 -49.341941 -8.457380 -12.615509 -0.000575 0.019383
4 53.66 186.46 -50.056683 -8.122609 -12.678341 -0.000548 0.019378

 accel_z gyro_x gyro_y gyro_z reset
0 0.014569 0.000942 0.000492 -0.000750 20
1 0.014577 0.000218 -0.000005 -0.000235 0
2 0.014357 0.000395 0.000600 -0.000003 0
3 0.014409 0.000308 0.000577 -0.000102 0
4 0.014380 0.000321 0.000691 0.000272 0
```

`tail()` method gives back last rows:

```
[7]: df.tail()
[7]:
 time_stamp temp_cpu temp_h temp_p humidity pressure \
110864 2016-02-29 09:24:21 31.56 27.52 24.83 42.94 1005.83
110865 2016-02-29 09:24:30 31.55 27.50 24.83 42.72 1005.85
110866 2016-02-29 09:24:41 31.58 27.50 24.83 42.83 1005.85
```

(continues on next page)

(continued from previous page)

110867	2016-02-29	09:24:50	31.62	27.50	24.83	42.81	1005.88
110868	2016-02-29	09:25:00	31.57	27.51	24.83	42.94	1005.86
<hr/>							
110864	1.58	49.93	129.60	-15.169673	-27.642610	1.563183	-0.000682
110865	1.89	49.92	130.51	-15.832622	-27.729389	1.785682	-0.000736
110866	2.09	50.00	132.04	-16.646212	-27.719479	1.629533	-0.000647
110867	2.88	49.69	133.00	-17.270447	-27.793136	1.703806	-0.000835
110868	2.17	49.77	134.18	-17.885872	-27.824149	1.293345	-0.000787
<hr/>							
110864	0.017743	0.014646	-0.000264	0.000206	0.000196	0	
110865	0.017570	0.014855	0.000143	0.000199	-0.000024	0	
110866	0.017657	0.014799	0.000537	0.000257	0.000057	0	
110867	0.017635	0.014877	0.000534	0.000456	0.000195	0	
110868	0.017261	0.014380	0.000459	0.000076	0.000030	0	

## 2.5 Some stats

The `describe` method gives you much summary info on the fly:

- rows counting
- the average
- standard deviation<sup>364</sup>
- quantiles<sup>365</sup>
- minimum and maximum

[8] : `df.describe()`

count	110869.000000	110869.000000	110869.000000	110869.000000	\
mean	32.236259	28.101773	25.543272	46.252005	
std	0.360289	0.369256	0.380877	1.907273	
min	31.410000	27.200000	24.530000	42.270000	
25%	31.960000	27.840000	25.260000	45.230000	
50%	32.280000	28.110000	25.570000	46.130000	
75%	32.480000	28.360000	25.790000	46.880000	
max	33.700000	29.280000	26.810000	60.590000	
<hr/>					
count	110869.000000	110869.000000	110869.000000	110869.000000	\
mean	1008.126788	2.770553	51.807973	200.90126	
std	3.093485	21.848940	2.085821	84.47763	
min	1001.560000	0.000000	30.890000	0.01000	
25%	1006.090000	1.140000	51.180000	162.43000	
50%	1007.650000	1.450000	51.950000	190.58000	
75%	1010.270000	1.740000	52.450000	256.34000	
max	1021.780000	360.000000	359.400000	359.98000	
<hr/>					
mag_x		mag_y	mag_z	accel_x	\

(continues on next page)

<sup>364</sup> [https://en.wikipedia.org/wiki/Standard\\_deviati](https://en.wikipedia.org/wiki/Standard_deviati)

<sup>365</sup> <https://en.wikipedia.org/wiki/Quantile>

(continued from previous page)

count	110869.000000	110869.000000	110869.000000	110869.000000		
mean	-19.465265	-1.174493	-6.004529	-0.000630		
std	28.120202	15.655121	8.552481	0.000224		
min	-73.046240	-43.810030	-41.163040	-0.025034		
25%	-41.742792	-12.982321	-11.238430	-0.000697		
50%	-21.339485	-1.350467	-5.764400	-0.000631		
75%	7.299000	11.912456	-0.653705	-0.000567		
max	33.134748	37.552135	31.003047	0.018708		
	accel_y	accel_z	gyro_x	gyro_y	\	
count	110869.000000	110869.000000	1.108690e+05	110869.000000		
mean	0.018504	0.014512	-8.959493e-07	0.000007		
std	0.000604	0.000312	2.807614e-03	0.002456		
min	-0.005903	-0.022900	-3.037930e-01	-0.378412		
25%	0.018009	0.014349	-2.750000e-04	-0.000278		
50%	0.018620	0.014510	-3.000000e-06	-0.000004		
75%	0.018940	0.014673	2.710000e-04	0.000271		
max	0.041012	0.029938	2.151470e-01	0.389499		
	gyro_z	reset				
count	1.108690e+05	110869.000000				
mean	-9.671594e-07	0.000180				
std	2.133104e-03	0.060065				
min	-2.970800e-01	0.000000				
25%	-1.200000e-04	0.000000				
50%	-1.000000e-06	0.000000				
75%	1.190000e-04	0.000000				
max	2.698760e-01	20.000000				

**QUESTION:** is there some missing field from the table produced by describe? Why is it not included?

With corr method we can see the correlation between DataFrame columns.

[9]:	df.corr()								
[9]:		temp_cpu	temp_h	temp_p	humidity	pressure	pitch	\	
	temp_cpu	1.000000	0.986872	0.991672	-0.297081	0.038065	0.008076		
	temp_h	0.986872	1.000000	0.993260	-0.281422	0.070882	0.005145		
	temp_p	0.991672	0.993260	1.000000	-0.288373	0.035496	0.006750		
	humidity	-0.297081	-0.281422	-0.288373	1.000000	0.434374	0.004050		
	pressure	0.038065	0.070882	0.035496	0.434374	1.000000	0.003018		
	pitch	0.008076	0.005145	0.006750	0.004050	0.003018	1.000000		
	roll	-0.171644	-0.199628	-0.163685	0.101304	0.011815	0.087941		
	yaw	-0.117972	-0.117870	-0.118463	0.031664	-0.051697	-0.011611		
	mag_x	0.005145	0.000428	0.004338	-0.035146	-0.040183	0.013331		
	mag_y	-0.285192	-0.276276	-0.283427	0.077897	-0.074578	0.006133		
	mag_z	-0.120838	-0.098864	-0.114407	0.076424	0.092352	0.000540		
	accel_x	-0.023582	-0.032188	-0.018047	-0.009741	0.013556	0.043285		
	accel_y	-0.446358	-0.510126	-0.428884	0.226281	-0.115642	0.009015		
	accel_z	-0.029155	-0.043213	-0.036505	0.005281	-0.221208	-0.039146		
	gyro_x	0.002511	0.001771	0.001829	0.004345	-0.000611	0.066618		
	gyro_y	0.005947	0.005020	0.006127	0.003457	-0.002493	-0.015034		
	gyro_z	-0.001250	-0.001423	-0.001623	0.001298	-0.000615	0.049340		
	reset	-0.002970	-0.004325	-0.004205	-0.002066	-0.006259	-0.000176		
		roll	yaw	mag_x	mag_y	mag_z	accel_x	\	
		temp_cpu	-0.171644	-0.117972	0.005145	-0.285192	-0.120838	-0.023582	

(continues on next page)

(continued from previous page)

temp_h	-0.199628	-0.117870	0.000428	-0.276276	-0.098864	-0.032188
temp_p	-0.163685	-0.118463	0.004338	-0.283427	-0.114407	-0.018047
humidity	0.101304	0.031664	-0.035146	0.077897	0.076424	-0.009741
pressure	0.011815	-0.051697	-0.040183	-0.074578	0.092352	0.013556
pitch	0.087941	-0.011611	0.013331	0.006133	0.000540	0.043285
roll	1.000000	0.095354	-0.020947	0.060297	-0.080620	0.116637
yaw	0.095354	1.000000	0.257971	0.549394	-0.328360	0.006943
mag_x	-0.020947	0.257971	1.000000	0.001239	-0.213070	-0.006629
mag_y	0.060297	0.549394	0.001239	1.000000	-0.266351	0.014057
mag_z	-0.080620	-0.328360	-0.213070	-0.266351	1.000000	0.024718
accel_x	0.116637	0.006943	-0.006629	0.014057	0.024718	1.000000
accel_y	0.462630	0.044157	0.027921	0.051619	-0.083914	0.095286
accel_z	-0.167905	-0.013634	0.021524	-0.053016	-0.061317	-0.262305
gyro_x	-0.115873	0.003106	-0.004954	0.001239	-0.008470	0.035314
gyro_y	-0.002509	0.003665	-0.004429	0.001063	-0.009557	0.103449
gyro_z	-0.214202	0.004020	-0.005052	0.001530	-0.008997	0.197740
reset	0.000636	-0.000558	-0.002879	-0.001335	-0.002151	0.002173
	accel_y	accel_z	gyro_x	gyro_y	gyro_z	reset
temp_cpu	-0.446358	-0.029155	0.002511	0.005947	-0.001250	-0.002970
temp_h	-0.510126	-0.043213	0.001771	0.005020	-0.001423	-0.004325
temp_p	-0.428884	-0.036505	0.001829	0.006127	-0.001623	-0.004205
humidity	0.226281	0.005281	0.004345	0.003457	0.001298	-0.002066
pressure	-0.115642	-0.221208	-0.000611	-0.002493	-0.000615	-0.006259
pitch	0.009015	-0.039146	0.066618	-0.015034	0.049340	-0.000176
roll	0.462630	-0.167905	-0.115873	-0.002509	-0.214202	0.000636
yaw	0.044157	-0.013634	0.003106	0.003665	0.004020	-0.000558
mag_x	0.027921	0.021524	-0.004954	-0.004429	-0.005052	-0.002879
mag_y	0.051619	-0.053016	0.001239	0.001063	0.001530	-0.001335
mag_z	-0.083914	-0.061317	-0.008470	-0.009557	-0.008997	-0.002151
accel_x	0.095286	-0.262305	0.035314	0.103449	0.197740	0.002173
accel_y	1.000000	0.120215	0.043263	-0.046463	0.009541	0.004648
accel_z	0.120215	1.000000	0.078315	-0.075625	0.057075	0.000554
gyro_x	0.043263	0.078315	1.000000	-0.248968	0.337553	0.001009
gyro_y	-0.046463	-0.075625	-0.248968	1.000000	0.190112	0.000593
gyro_z	0.009541	0.057075	0.337553	0.190112	1.000000	-0.001055
reset	0.004648	0.000554	0.001009	0.000593	-0.001055	1.000000

## 2.6 Guardiamo le colonne

columns property gives the column headers:

```
[10]: df.columns
[10]: Index(['time_stamp', 'temp_cpu', 'temp_h', 'temp_p', 'humidity', 'pressure',
 'pitch', 'roll', 'yaw', 'mag_x', 'mag_y', 'mag_z', 'accel_x', 'accel_y',
 'accel_z', 'gyro_x', 'gyro_y', 'gyro_z', 'reset'],
 dtype='object')
```

As you see in the above, the type of the found object is not a list, but a special container defined by pandas:

```
[11]: type(df.columns)
[11]: pandas.core.indexes.base.Index
```

Nevertheless, we can access the elements of this container using indeces within the squared parenthesis:

```
[12]: df.columns[0]
[12]: 'time_stamp'
```

```
[13]: df.columns[1]
[13]: 'temp_cpu'
```

## 2.7 What is a column?

We can access a column like this:

```
[14]: df['humidity']
[14]:
0 44.94
1 45.12
2 45.12
3 45.32
4 45.18
...
110864 42.94
110865 42.72
110866 42.83
110867 42.81
110868 42.94
Name: humidity, Length: 110869, dtype: float64
```

Even handier, we can use the dot notation:

```
[15]: df.humidity
[15]:
0 44.94
1 45.12
2 45.12
3 45.32
4 45.18
...
110864 42.94
110865 42.72
110866 42.83
110867 42.81
110868 42.94
Name: humidity, Length: 110869, dtype: float64
```

**WARNING:** they look like two columns, but it's actually only one!

The sequence of numbers on the left is the integer index that Pandas automatically assigned to the dataset when it was created (notice it starts from zero).

**WARNING: Careful about spaces!:**

In case the field name has spaces (es. 'blender rotations'), **do not** use the dot notation, instead use squared bracket notation seen above (ie: df[['blender rotations']])

The type of a column is Series:

```
[16]: type(df.humidity)
[16]: pandas.core.series.Series
```

Some operations also work on single columns, i.e. .describe():

```
[17]: df.humidity.describe()

[17]: count 110869.000000
 mean 46.252005
 std 1.907273
 min 42.270000
 25% 45.230000
 50% 46.130000
 75% 46.880000
 max 60.590000
Name: humidity, dtype: float64
```

## 2.8 Exercise - meteo info

⊕ a) Create a new dataframe called meteo by importing the data from file `meteo.csv`, which contains the meteo data of Trento from November 2017 (source: [www.meteotrentino.it<sup>366</sup>](http://www.meteotrentino.it)). **IMPORTANT:** assign the dataframe to a variable called `meteo` (so we avoid confusion with AstroPi dataframe)

b) Visualize info about this dataframe

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: # write here - create dataframe

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
print("COLUMNS:", ', '.join(meteo.columns))
print()
print(meteo.columns)
print()
print("INFO:")
print(meteo.info())
print()
print("FIRST ROWS:")

print(meteo.head())

COLUMNS: Date, Pressure, Rain, Temp

Index(['Date', 'Pressure', 'Rain', 'Temp'], dtype='object')

INFO:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2878 entries, 0 to 2877
Data columns (total 4 columns):
 # Column Non-Null Count Dtype
--- -- -- -- --
 0 Date 2878 non-null object
```

(continues on next page)

<sup>366</sup> <https://www.meteotrentino.it>

(continued from previous page)

```

1 Pressure 2878 non-null float64
2 Rain 2878 non-null float64
3 Temp 2878 non-null float64
dtypes: float64(3), object(1)
memory usage: 90.1+ KB
None

FIRST ROWS:
 Date Pressure Rain Temp
0 01/11/2017 00:00 995.4 0.0 5.4
1 01/11/2017 00:15 995.5 0.0 6.0
2 01/11/2017 00:30 995.5 0.0 5.9
3 01/11/2017 00:45 995.7 0.0 5.4
4 01/11/2017 01:00 995.7 0.0 5.3

```

&lt;/div&gt;

```
[18]: # write here - create dataframe
```

COLUMNS: Date, Pressure, Rain, Temp

Index(['Date', 'Pressure', 'Rain', 'Temp'], dtype='object')

INFO:

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2878 entries, 0 to 2877
Data columns (total 4 columns):
 # Column Non-Null Count Dtype
 --- -----
 0 Date 2878 non-null object
 1 Pressure 2878 non-null float64
 2 Rain 2878 non-null float64
 3 Temp 2878 non-null float64
dtypes: float64(3), object(1)
memory usage: 90.1+ KB
None

```

FIRST ROWS:

	Date	Pressure	Rain	Temp
0	01/11/2017 00:00	995.4	0.0	5.4
1	01/11/2017 00:15	995.5	0.0	6.0
2	01/11/2017 00:30	995.5	0.0	5.9
3	01/11/2017 00:45	995.7	0.0	5.4
4	01/11/2017 01:00	995.7	0.0	5.3

### 3. Matplotlib review

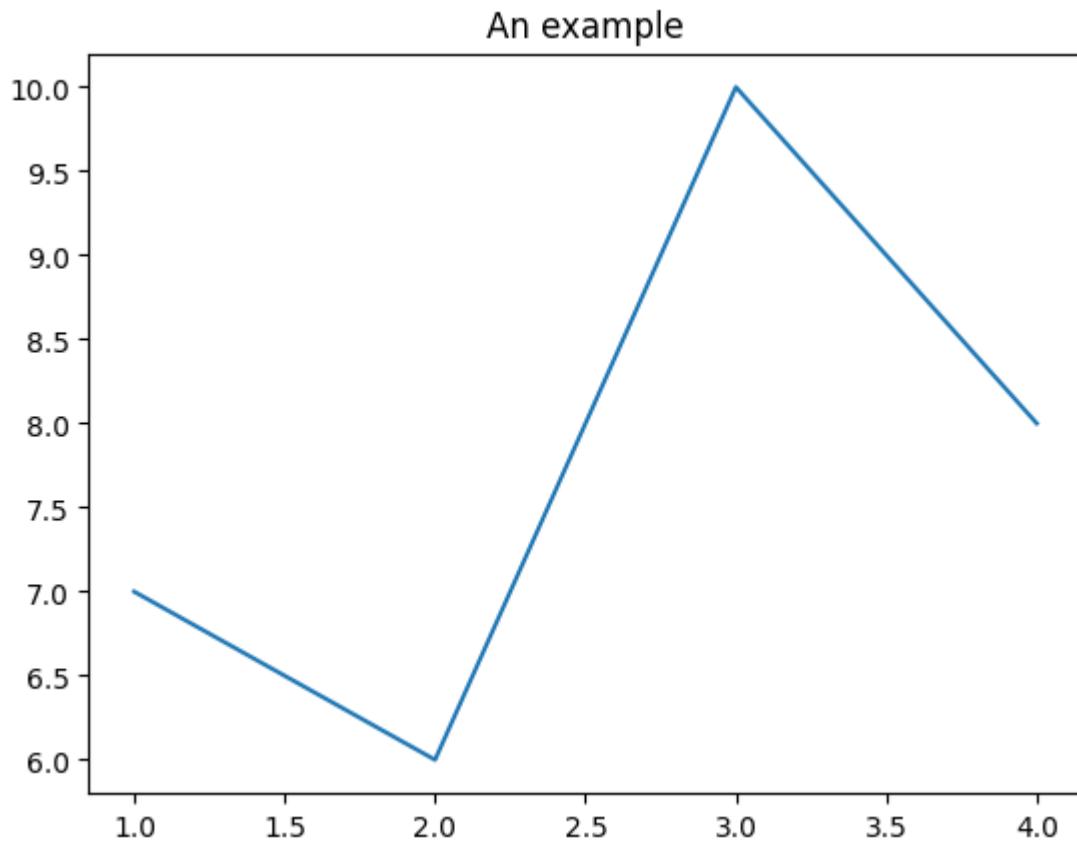
We've already seen Matplotlib in the part on visualization<sup>367</sup>, and today we use Matplotlib<sup>368</sup> to display data.

#### 3.1 An example

Let's take again an example, with the *Matlab approach*. We will plot a line passing two lists of coordinates, one for xs and one for ys:

```
[19]: import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

xs = [1, 2, 3, 4]
ys = [7, 6, 10, 8]
plt.plot(xs, ys) # we can directly pass xs and ys lists
plt.title('An example')
plt.show()
```



We can also create the series with numpy. Let's try a parabola:

<sup>367</sup> <https://en.softpython.org/visualization/visualization1-sol.html>

<sup>368</sup> <http://matplotlib.org>

```
[20]: import numpy as np
xs = np.arange(0., 5., 0.1)
'**' is the power operator in Python, NOT '^'
ys = xs**2
```

Let's use the `type` function to understand which data types are `xs` and `ys`:

```
[21]: type(xs)
```

```
[21]: numpy.ndarray
```

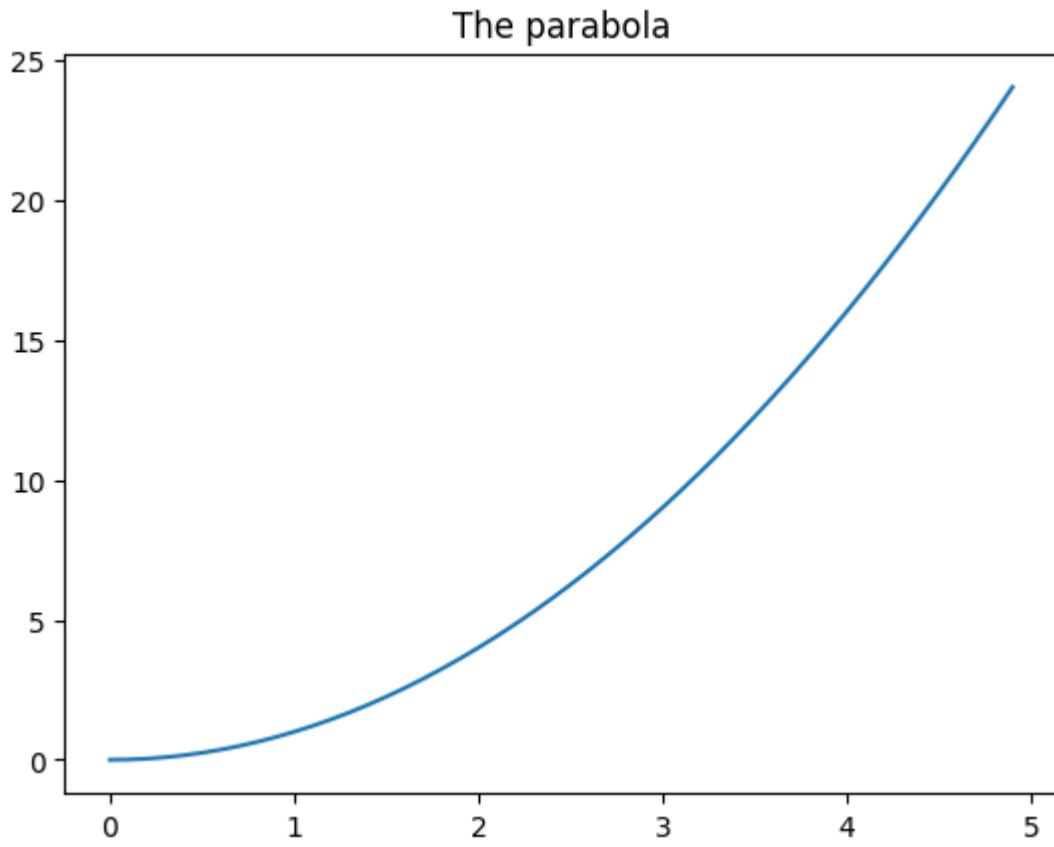
```
[22]: type(ys)
```

```
[22]: numpy.ndarray
```

Hence we have NumPy arrays.

Let's plot it:

```
[23]: plt.title('The parabola')
plt.plot(xs, ys);
```



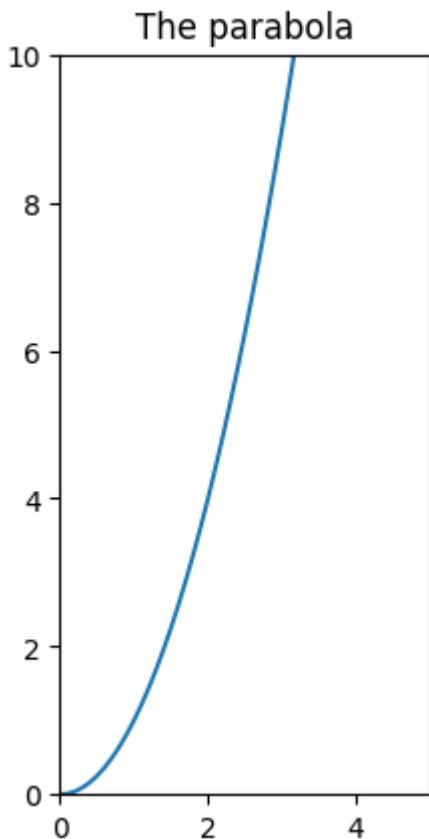
If we want the same units in both x and y axis, we can use the `gca`<sup>369</sup> function

To set x and y limits, we can use `xlim` e `ylim`:

<sup>369</sup> [https://matplotlib.org/api/\\_as\\_gen/matplotlib.pyplot.gca.html?highlight=matplotlib%20pyplot%20gca#matplotlib.pyplot.gca](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.gca.html?highlight=matplotlib%20pyplot%20gca#matplotlib.pyplot.gca)

```
[24]: plt.xlim([0, 5])
plt.ylim([0,10])
plt.title('The parabola')

plt.gca().set_aspect('equal')
plt.plot(xs,ys);
```



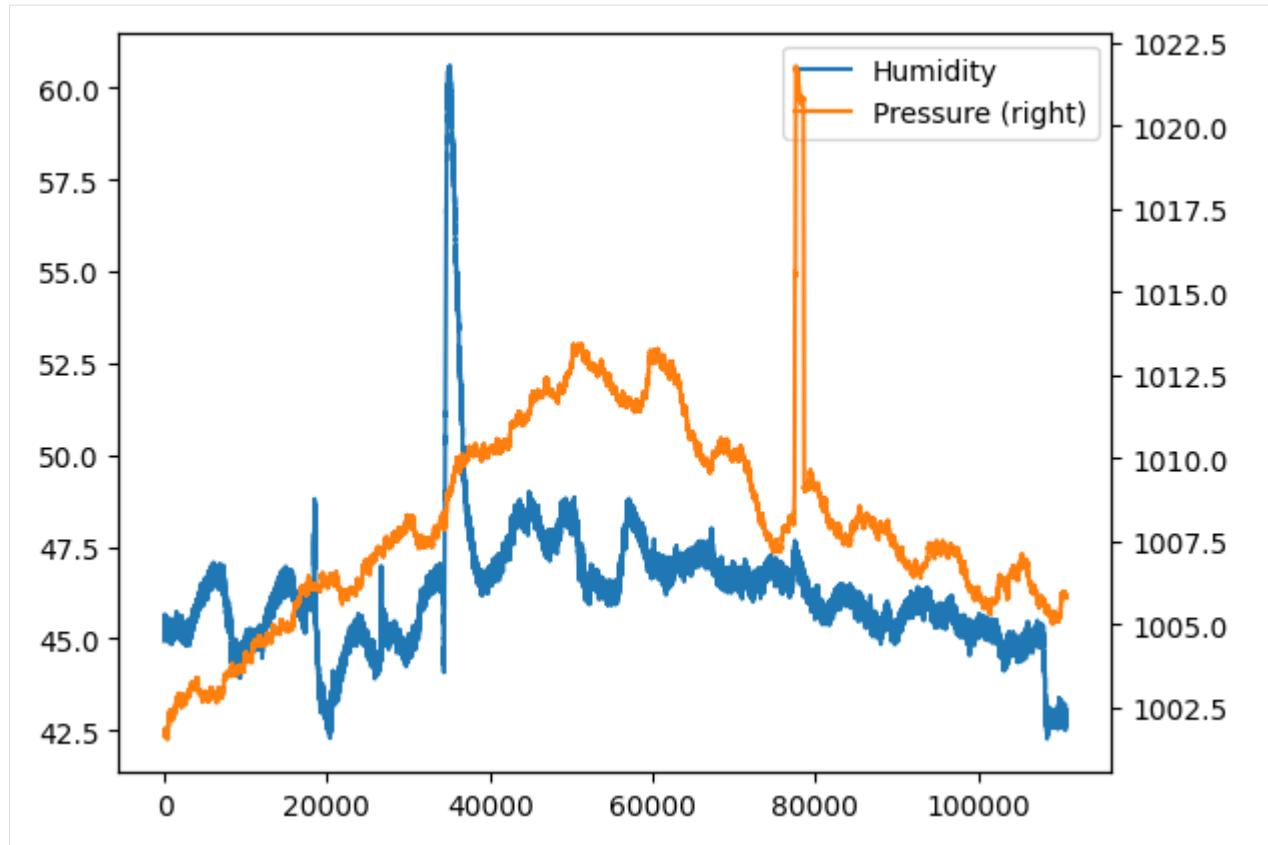
### 3.2 Matplotlib plots from pandas datastructures

We can get plots directly from pandas data structures using the *matlab style*. Let's make a simple example, for more complex cases we refer to `DataFrame.plot` documentation<sup>370</sup>.

In case of big quantity of data, it may be useful to have a qualitative idea of data by putting them in a plot:

```
[25]: df.humidity.plot(label="Humidity", legend=True)
with secondary_y=True we display number on y axis
of graph on the right
df.pressure.plot(secondary_y=True, label="Pressure", legend=True);
```

<sup>370</sup> <https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html>

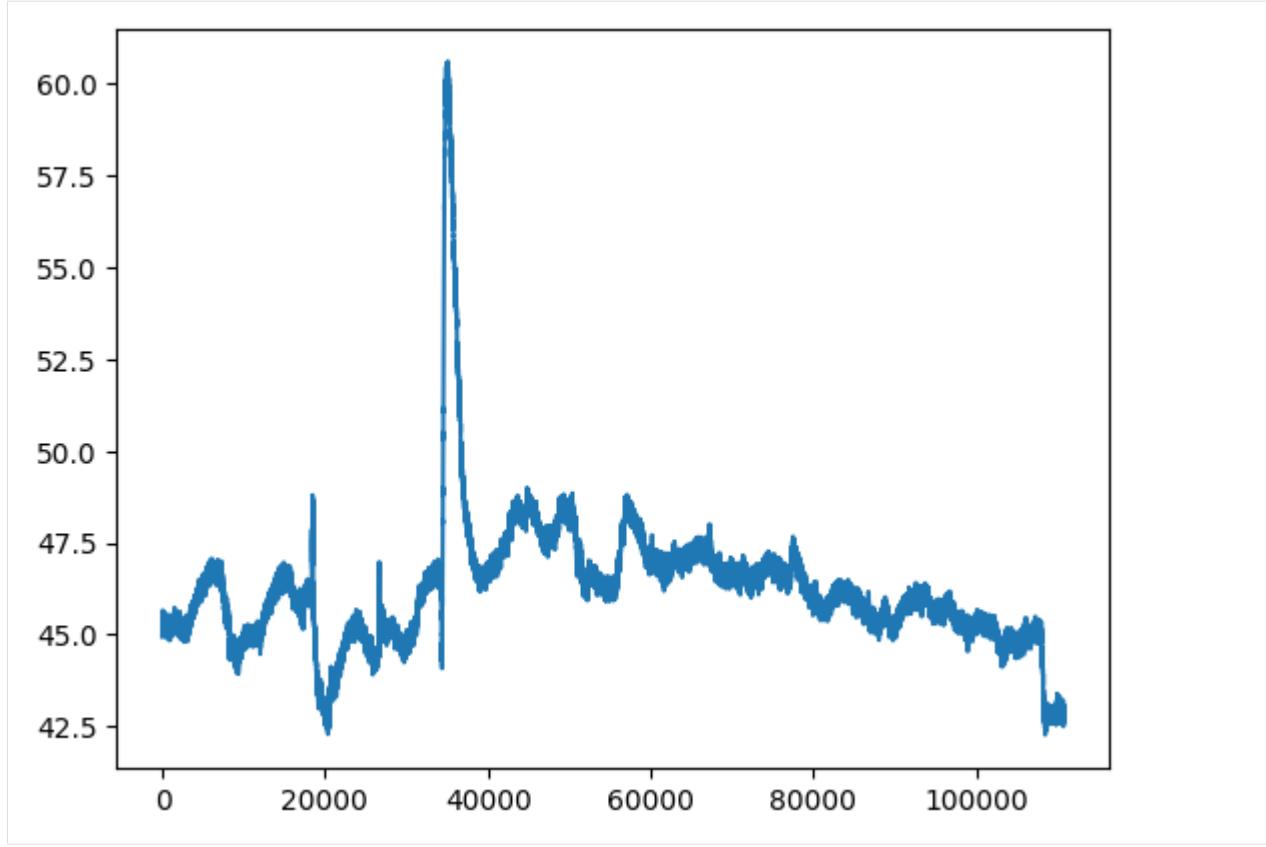


If we want, we can always directly use the original function `plt.plot`, it's sufficient to pass a sequence for the x coordinates and another one for the ys. For example, if we wanted to replicate the above example for humidity, for the x coordinates we could extract the dataframe index which is an iterable:

```
[26]: df.index
[26]: RangeIndex(start=0, stop=110869, step=1)
```

and then pass it to `plt.plot` as first parameter. As second parameter, we can directly pass the humidity Series: since it's also an iterable, Python will automatically be able to get the cells values:

```
[27]: plt.plot(df.index, df['humidity'])
plt.show() # prevents visualization of weird characters
```



## 4. Operations on rows

If we consider the rows of a dataset, typically we will want to index, filter and order them.

### 4.1 Indexing integers

We report here the simplest indexing with row numbers.

To obtain the i-th series you can use the method `iloc[i]` (here we reuse AstroPi dataset) :

```
[28]: df.iloc[6]
```

```
[28]: time_stamp 2016-02-16 10:45:41
temp_cpu 31.68
temp_h 27.53
temp_p 25.01
humidity 45.31
pressure 1001.7
pitch 0.63
roll 53.55
yaw 186.1
mag_x -50.447346
mag_y -7.937309
mag_z -12.188574
accel_x -0.00051
accel_y 0.019264
```

(continues on next page)

(continued from previous page)

```
accel_z 0.014528
gyro_x -0.000111
gyro_y 0.00032
gyro_z 0.000222
reset 0
Name: 6, dtype: object
```

It's possible to select a dataframe of contiguous positions by using *slicing*, as we already did for [strings<sup>371</sup>](#) and [lists<sup>372</sup>](#)

For example, here we select the rows from 5th *included* to 7-th *excluded* :

```
[29]: df.iloc[5:7]

[29]: time_stamp temp_cpu temp_h temp_p humidity pressure pitch \
5 2016-02-16 10:45:30 31.69 27.55 25.01 45.12 1001.67 0.85
6 2016-02-16 10:45:41 31.68 27.53 25.01 45.31 1001.70 0.63

 roll yaw mag_x mag_y mag_z accel_x accel_y \
5 53.53 185.52 -50.246476 -8.343209 -11.938124 -0.000536 0.019453
6 53.55 186.10 -50.447346 -7.937309 -12.188574 -0.000510 0.019264

 accel_z gyro_x gyro_y gyro_z reset
5 0.014380 0.000273 0.000494 -0.000059 0
6 0.014528 -0.000111 0.000320 0.000222 0
```

By filtering the rows we can ‘zoom in’ the dataset, selecting for example the rows between the 12500th (included) and the 15000th (excluded) in the new dataframe df2:

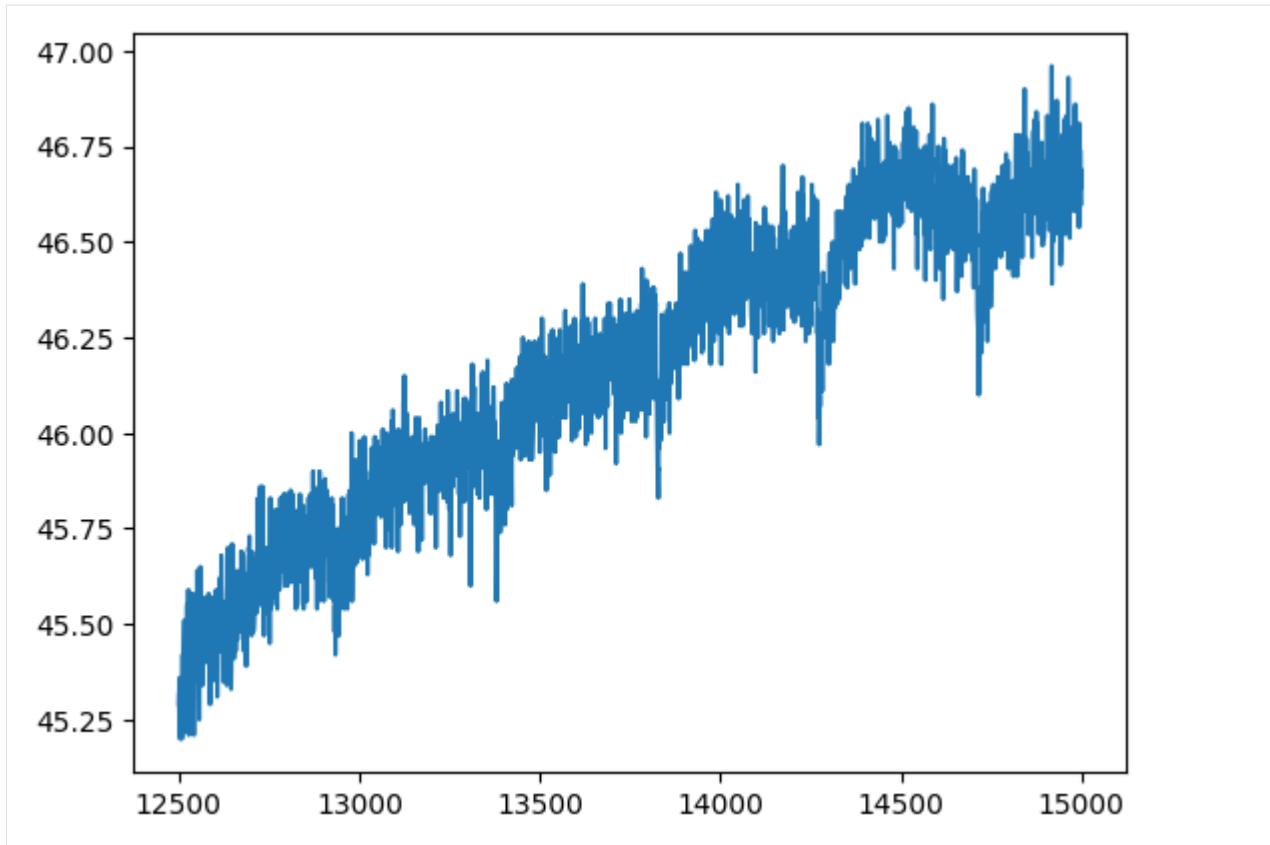
```
[30]: df2=df.iloc[12500:15000]
```

```
[31]: plt.plot(df2.index, df2['humidity'])
```

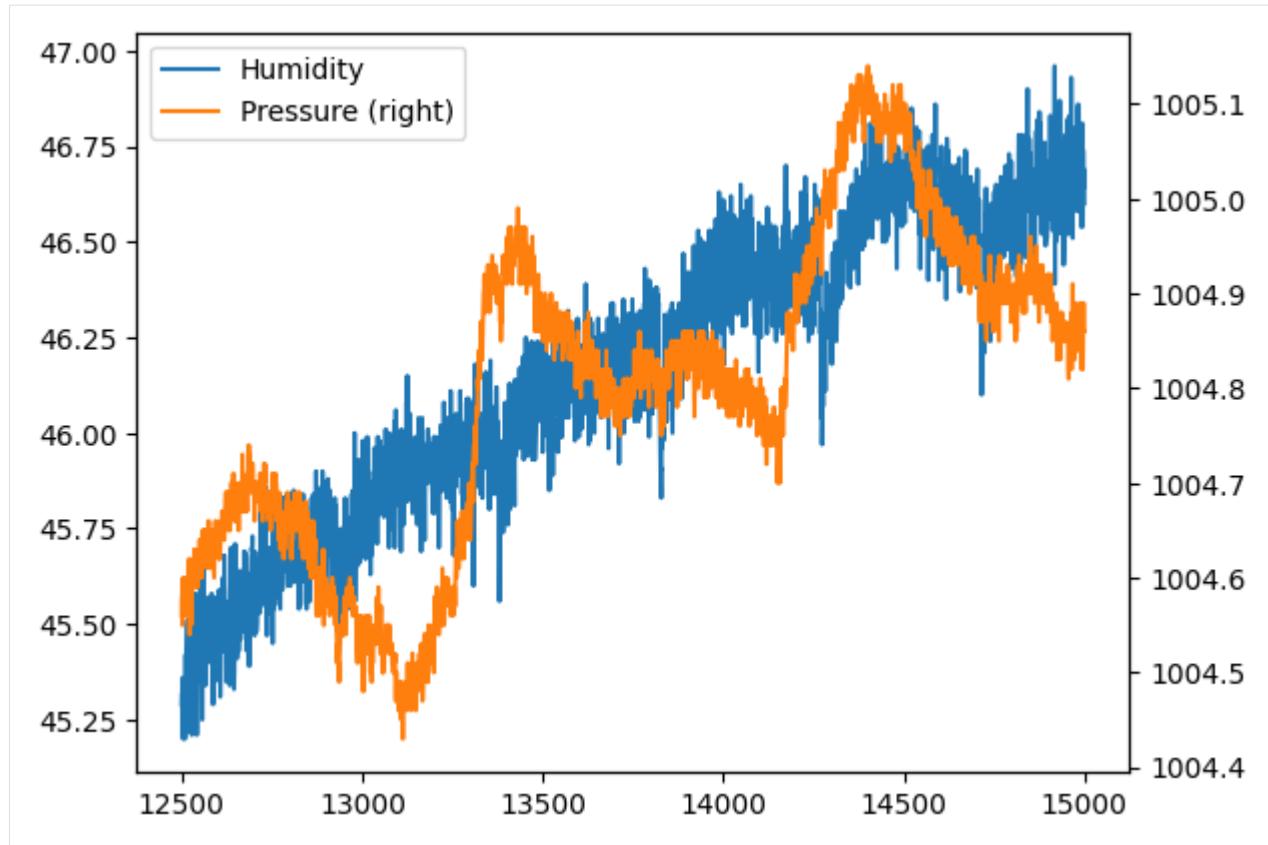
```
[31]: [matplotlib.lines.Line2D at 0x7fbb1c78ae90>]
```

<sup>371</sup> <https://en.softpython.org/strings/strings2-sol.html#Slices>

<sup>372</sup> <https://en.softpython.org/lists/lists2-sol.html#Slices>



```
[32]: df2.humidity.plot(label="Humidity", legend=True)
df2.pressure.plot(secondary_y=True, label="Pressure", legend=True)
plt.show() # prevents visualization of weird characters
```



### Difference between `iloc` and `loc`

`iloc` always uses an integer and returns always the row in the natural order of the dataframe we are inspecting.

`loc` on the other hand searches in the *index assigned by pandas*, which is the one you can see in bold when we show the dataset.

Apparently they look similar but the difference between them becomes evident whenever we act on filtered dataframes. Let's check the first rows of the filtered dataframe `df2`:

```
[33]: df2.head()
```

	time_stamp	temp_cpu	temp_h	temp_p	humidity	pressure	\
12500	2016-02-17 21:44:31	31.87	27.7	25.15	45.29	1004.56	
12501	2016-02-17 21:44:40	31.84	27.7	25.16	45.32	1004.58	
12502	2016-02-17 21:44:51	31.83	27.7	25.15	45.23	1004.55	
12503	2016-02-17 21:45:00	31.83	27.7	25.15	45.36	1004.58	
12504	2016-02-17 21:45:10	31.83	27.7	25.15	45.20	1004.60	

	pitch	roll	yaw	mag_x	mag_y	mag_z	accel_x	\
12500	0.85	52.78	357.18	30.517177	2.892431	0.371669	-0.000618	
12501	0.97	52.73	357.32	30.364154	2.315241	0.043272	-0.001196	
12502	1.40	52.84	357.76	29.760987	1.904932	0.037701	-0.000617	
12503	2.14	52.84	357.79	29.882673	1.624020	-0.249268	-0.000723	
12504	1.76	52.98	357.78	29.641547	1.532007	-0.336724	-0.000664	

	accel_y	accel_z	gyro_x	gyro_y	gyro_z	reset
--	---------	---------	--------	--------	--------	-------

(continues on next page)

(continued from previous page)

12500	0.019318	0.014503	-0.000135	-0.000257	0.000121	0
12501	0.019164	0.014545	0.000254	0.000497	-0.000010	0
12502	0.019420	0.014672	0.000192	0.000081	0.000024	0
12503	0.019359	0.014691	0.000597	0.000453	-0.000118	0
12504	0.019245	0.014673	0.000373	0.000470	-0.000130	0

Let's consider number 0, in this case:

- `.iloc[0]` selects the initial row
- `.loc[0]` selects the row at *pandas index* with value zero

**QUESTION:** in case of `df2`, what's the initial row? What's its *pandas index*?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** la riga iniziale ha *indice di pandas* 12500

</div>

Let's see the difference in results.

`df2.loc[0]` will actually find the zeroth row:

[34]: `df2.iloc[0]`

```
[34]: time_stamp 2016-02-17 21:44:31
temp_cpu 31.87
temp_h 27.7
temp_p 25.15
humidity 45.29
pressure 1004.56
pitch 0.85
roll 52.78
yaw 357.18
mag_x 30.517177
mag_y 2.892431
mag_z 0.371669
accel_x -0.000618
accel_y 0.019318
accel_z 0.014503
gyro_x -0.000135
gyro_y -0.000257
gyro_z 0.000121
reset 0
Name: 12500, dtype: object
```

`df2.loc[0]` instead will miserably fail:

```
df2.loc[0]

ValueError Traceback (most recent call last)
~/local/lib/python3.7/site-packages/pandas/core/indexes/range.py in get_loc(self, key, method, tolerance)
 384 try:
--> 385 return self._range.index(new_key)
 386 except ValueError as err:
ValueError: 0 is not in range
```

Let's try using the *pandas index* for the zeroth row:

```
[35]: df2.loc[12500]

[35]: time_stamp 2016-02-17 21:44:31
temp_cpu 31.87
temp_h 27.7
temp_p 25.15
humidity 45.29
pressure 1004.56
pitch 0.85
roll 52.78
yaw 357.18
mag_x 30.517177
mag_y 2.892431
mag_z 0.371669
accel_x -0.000618
accel_y 0.019318
accel_z 0.014503
gyro_x -0.000135
gyro_y -0.000257
gyro_z 0.000121
reset 0
Name: 12500, dtype: object
```

As expected, this was correctly found.

## 4.2 Filtering

It's possible to filter data by according to a condition data should satisfy, which can be expressed by indicating a column and a comparison operator. For example:

```
[36]: df.humidity < 45.2

[36]: 0 True
1 True
2 True
3 False
4 True
...
110864 True
110865 True
110866 True
110867 True
110868 True
Name: humidity, Length: 110869, dtype: bool
```

We see it's a series of values `True` or `False`, according to `df.humidity` being less than `45.2`. What's the type of this result?

```
[37]: type(df.humidity < 45.2)

[37]: pandas.core.series.Series
```

## Combining filters

It's possible to combine conditions like we already did in Numpy filtering<sup>373</sup>: for example by using the special operator conjunction &

If we write `(df.humidity > 45.0) & (df.humidity < 45.2)` we obtain a series of values True or False, whether `df.humidity` is at the same time greater or equal than 45.0 and less or equal of 45.2

```
[38]: type((df.humidity > 45.0) & (df.humidity < 45.2))
```

```
[38]: pandas.core.series.Series
```

## Applying a filter

If we want complete rows of the dataframe which satisfy the condition, we can write like this:

**IMPORTANT:** we use `df` externally from expression `df[ ]` starting and closing the square bracket parenthesis to tell Python we want to filter the `df` dataframe, and use again `df` *inside* the parenthesis to tell on *which* columns and *which* rows we want to filter

```
[39]: df[(df.humidity > 45.0) & (df.humidity < 45.2)]
```

```
[39]: time_stamp temp_cpu temp_h temp_p humidity pressure \
1 2016-02-16 10:44:50 31.79 27.53 25.01 45.12 1001.72
2 2016-02-16 10:45:00 31.66 27.53 25.01 45.12 1001.72
4 2016-02-16 10:45:20 31.66 27.54 25.01 45.18 1001.71
5 2016-02-16 10:45:30 31.69 27.55 25.01 45.12 1001.67
10 2016-02-16 10:46:20 31.68 27.53 25.00 45.16 1001.72
...
108001 2016-02-29 01:23:30 32.32 28.20 25.57 45.05 1005.74
108003 2016-02-29 01:23:50 32.28 28.18 25.57 45.10 1005.76
108004 2016-02-29 01:24:00 32.30 28.18 25.57 45.11 1005.74
108006 2016-02-29 01:24:20 32.29 28.19 25.57 45.02 1005.73
108012 2016-02-29 01:25:21 32.27 28.19 25.57 45.01 1005.72

 pitch roll yaw mag_x mag_y mag_z accel_x \
1 1.03 53.73 186.72 -48.778951 -8.304243 -12.943096 -0.000614
2 1.24 53.57 186.21 -49.161878 -8.470832 -12.642772 -0.000569
4 0.85 53.66 186.46 -50.056683 -8.122609 -12.678341 -0.000548
5 0.85 53.53 185.52 -50.246476 -8.343209 -11.938124 -0.000536
10 1.32 53.52 186.24 -51.616473 -6.818130 -11.860839 -0.000530
...
108001 1.32 50.04 338.15 15.549799 -1.424077 -9.087291 -0.000754
108003 1.65 50.03 338.91 15.134025 -1.776843 -8.806690 -0.000819
108004 1.70 50.21 338.19 14.799790 -1.695364 -8.895130 -0.000739
108006 0.81 49.81 339.24 14.333920 -2.173228 -8.694976 -0.000606
108012 0.44 50.34 342.01 14.364146 -2.974811 -8.287531 -0.000754

 accel_y accel_z gyro_x gyro_y gyro_z reset
1 0.019436 0.014577 0.000218 -0.000005 -0.000235 0
2 0.019359 0.014357 0.000395 0.000600 -0.000003 0
4 0.019378 0.014380 0.000321 0.000691 0.000272 0
5 0.019453 0.014380 0.000273 0.000494 -0.000059 0
```

(continues on next page)

<sup>373</sup> <https://en.softpython.org/matrices-numpy/matrices-numpy1-sol.html#Filtering>

(continued from previous page)

```

10 0.019477 0.014500 0.000268 0.001194 0.000106 0
...
108001 0.017375 0.014826 0.000908 0.000447 0.000149 0
108003 0.017378 0.014974 0.000048 -0.000084 -0.000039 0
108004 0.017478 0.014792 -0.000311 -0.000417 -0.000008 0
108006 0.017275 0.014725 -0.000589 -0.000443 -0.000032 0
108012 0.017800 0.014704 -0.000033 -0.000491 0.000309 0

[7275 rows x 19 columns]

```

**Another example:** if we want to search the record(s) where pressure is maximal, we use `values` property of the series on which we calculate the maximal value:

```
[40]: df[df.humidity == df.humidity.values.max()]
[40]:
 time_stamp temp_cpu temp_h temp_p humidity pressure \
35068 2016-02-20 12:57:40 31.58 27.41 24.83 60.59 1008.91
35137 2016-02-20 13:09:20 31.60 27.50 24.89 60.59 1008.97

 pitch roll yaw mag_x mag_y mag_z accel_x \
35068 1.86 51.78 192.83 -53.325819 10.641053 -6.898934 -0.000657
35137 1.78 51.91 208.49 -29.012379 14.546882 -8.387606 -0.000811

 accel_y accel_z gyro_x gyro_y gyro_z reset
35068 0.018981 0.014993 0.000608 0.000234 -0.000063 0
35137 0.019145 0.015148 0.000038 -0.000182 0.000066 0

```

**QUESTION:** if you remember, when talking about the basics of floats, we said<sup>374</sup> that comparing floats with equality is actually a bad thing. Do you remember why? Does it really matters in this case?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"< data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** when dealing with float numbers, two mathematically sound computations that in theory should produce the same result can unfortunately produce slightly different numbers (es. `0.1 + 0.2 == 0.7 - 0.4` will give you... `False`). But in this case we can almost safely assume pandas is just comparing raw data without performing other potentially problematic computations.

</div>

### 4.3 Sorting

To obtain a NEW dataframe sorted according to one or more columns, we can use the `sort_values` method:

```
[41]: df.sort_values('pressure', ascending=False).head()
[41]:
 time_stamp temp_cpu temp_h temp_p humidity pressure \
77602 2016-02-25 12:13:20 32.44 28.31 25.74 47.57 1021.78
77601 2016-02-25 12:13:10 32.45 28.30 25.74 47.26 1021.75
77603 2016-02-25 12:13:30 32.44 28.30 25.74 47.29 1021.75
77604 2016-02-25 12:13:40 32.43 28.30 25.74 47.39 1021.75
77608 2016-02-25 12:14:20 32.42 28.29 25.74 47.36 1021.73

 pitch roll yaw mag_x mag_y mag_z accel_x \
77602 1.10 51.82 267.39 -0.797428 10.891803 -15.728202 -0.000612
```

(continues on next page)

<sup>374</sup> <https://en.softpython.org/basics/basics3-floats-sol.html#Reals---equality>

(continued from previous page)

77601	1.53	51.76	266.12	-1.266335	10.927442	-15.690558	-0.000661
77603	1.86	51.83	268.83	-0.320795	10.651441	-15.565123	-0.000648
77604	1.78	51.54	269.41	-0.130574	10.628383	-15.488983	-0.000672
77608	0.86	51.89	272.77	0.952025	10.435951	-16.027235	-0.000607
	accel_y	accel_z	gyro_x	gyro_y	gyro_z	reset	
77602	0.018170	0.014295	-0.000139	-0.000179	-0.000298	0	
77601	0.018357	0.014533	0.000152	0.000459	-0.000298	0	
77603	0.018290	0.014372	0.000049	0.000473	-0.000029	0	
77604	0.018154	0.014602	0.000360	0.000089	-0.000002	0	
77608	0.018186	0.014232	-0.000260	-0.000059	-0.000187	0	

#### 4.4 Exercise - Meteo stats

⊕ Analyze data from Dataframe `meteo` to find:

- values of average pression, minimal and maximal
- average temperature
- the dates of rainy days

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[42]: # write here
print("Average pressure : %s" % meteo.Pressure.values.mean())
print("Minimal pressure : %s" % meteo.Pressure.values.min())
print("Maximal pressure : %s" % meteo.Pressure.values.max())
print("Average temperature : %s" % meteo.Temp.values.mean())
meteo[(meteo.Rain > 0)]
```

```
Average pressure : 986.3408269631689
Minimal pressure : 966.3
Maximal pressure : 998.3
Average temperature : 6.410701876302988
```

```
[42]:
```

	Date	Pressure	Rain	Temp
433	05/11/2017 12:15	979.2	0.2	8.6
435	05/11/2017 12:45	978.9	0.2	8.4
436	05/11/2017 13:00	979.0	0.2	8.4
437	05/11/2017 13:15	979.1	0.8	8.2
438	05/11/2017 13:30	979.0	0.6	8.2
...	...	...	...	...
2754	29/11/2017 17:15	976.1	0.2	0.9
2755	29/11/2017 17:30	975.9	0.2	0.9
2802	30/11/2017 05:15	971.3	0.2	1.3
2803	30/11/2017 05:30	971.3	0.2	1.1
2804	30/11/2017 05:45	971.5	0.2	1.1

```
[107 rows x 4 columns]
```

</div>

```
[42]: # write here
```

```
Average pressure : 986.3408269631689
Minimal pressure : 966.3
Maximal pressure : 998.3
Average temperature : 6.410701876302988
```

```
[42]: Date Pressure Rain Temp
433 05/11/2017 12:15 979.2 0.2 8.6
435 05/11/2017 12:45 978.9 0.2 8.4
436 05/11/2017 13:00 979.0 0.2 8.4
437 05/11/2017 13:15 979.1 0.8 8.2
438 05/11/2017 13:30 979.0 0.6 8.2
...
2754 29/11/2017 17:15 976.1 0.2 0.9
2755 29/11/2017 17:30 975.9 0.2 0.9
2802 30/11/2017 05:15 971.3 0.2 1.3
2803 30/11/2017 05:30 971.3 0.2 1.1
2804 30/11/2017 05:45 971.5 0.2 1.1
```

[107 rows x 4 columns]

## 4.5 Exercise - meteo plot

⊕ Put in a plot the temperature from dataframe *meteo*:

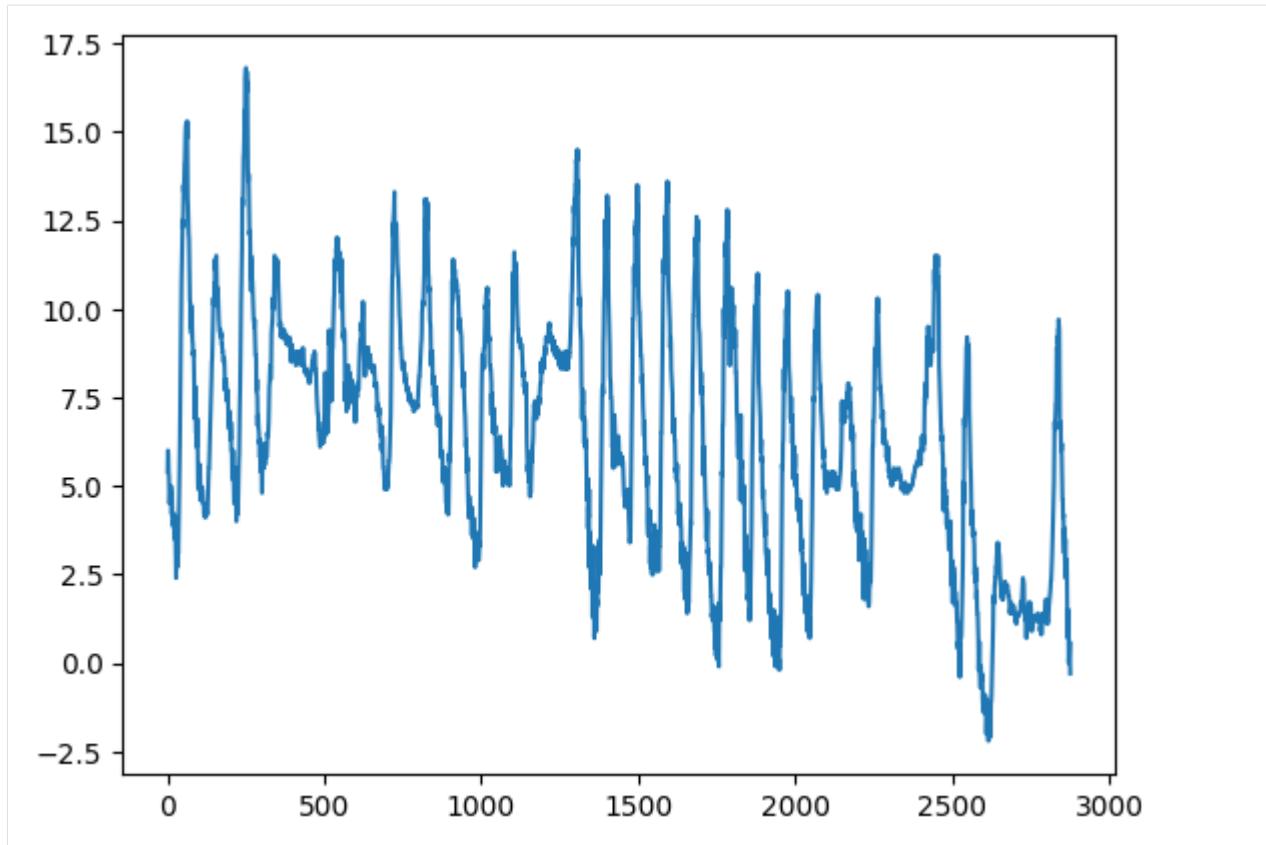
```
[43]: import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

write here
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

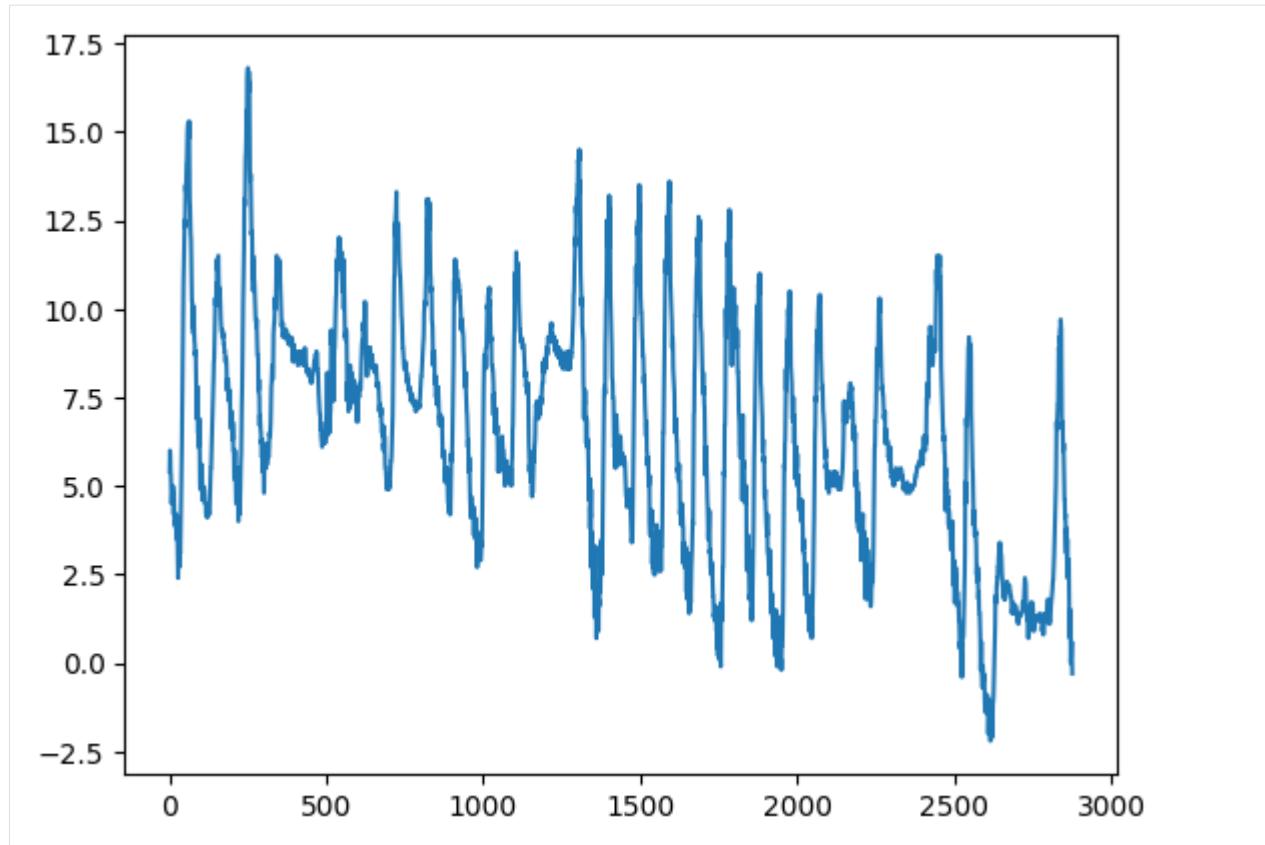
```
[44]: # SOLUTION
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

meteo.Temp.plot()
plt.show()
```



</div>

[44] :



#### 4.6 Exercise - Meteo pressure and raining

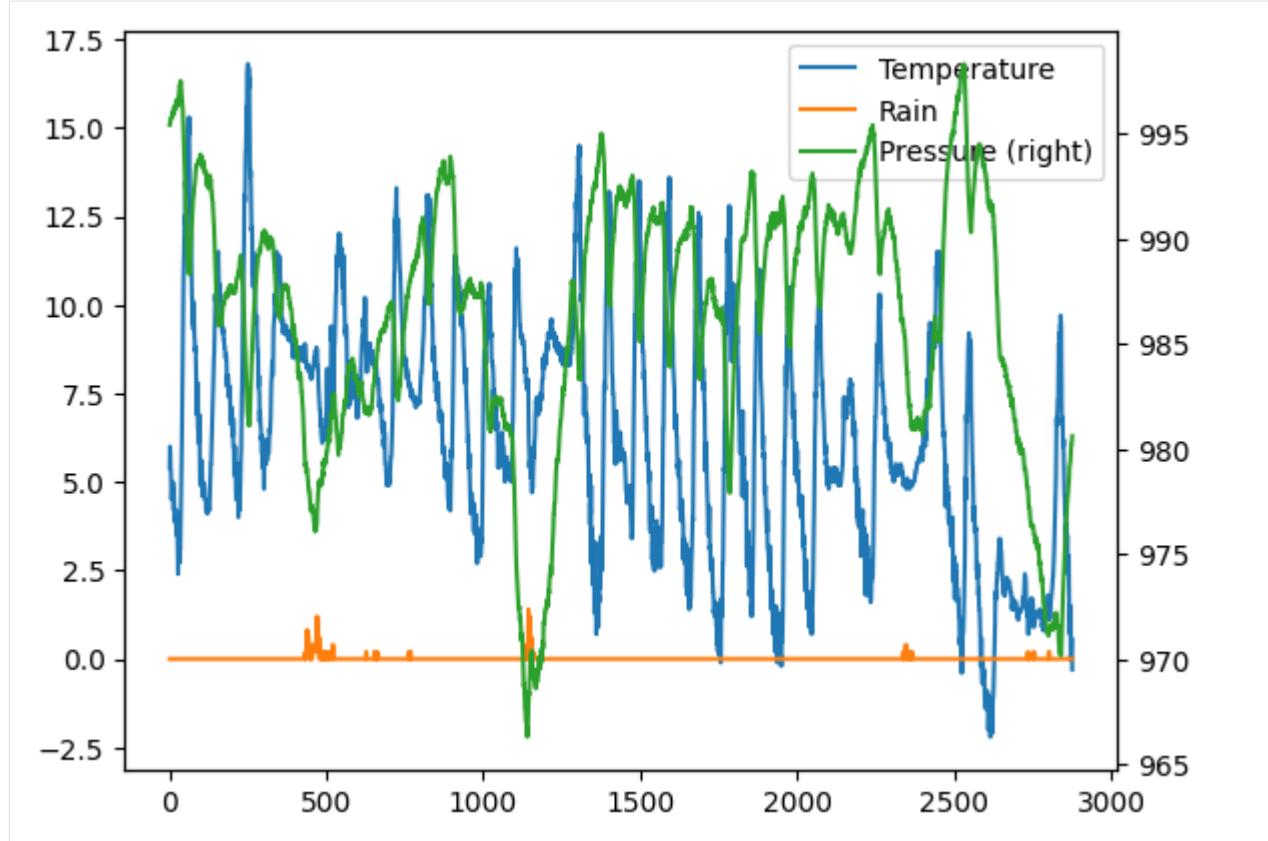
⊕ In the same plot as above show the pressure and amount of raining.

```
[45]: # write here
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

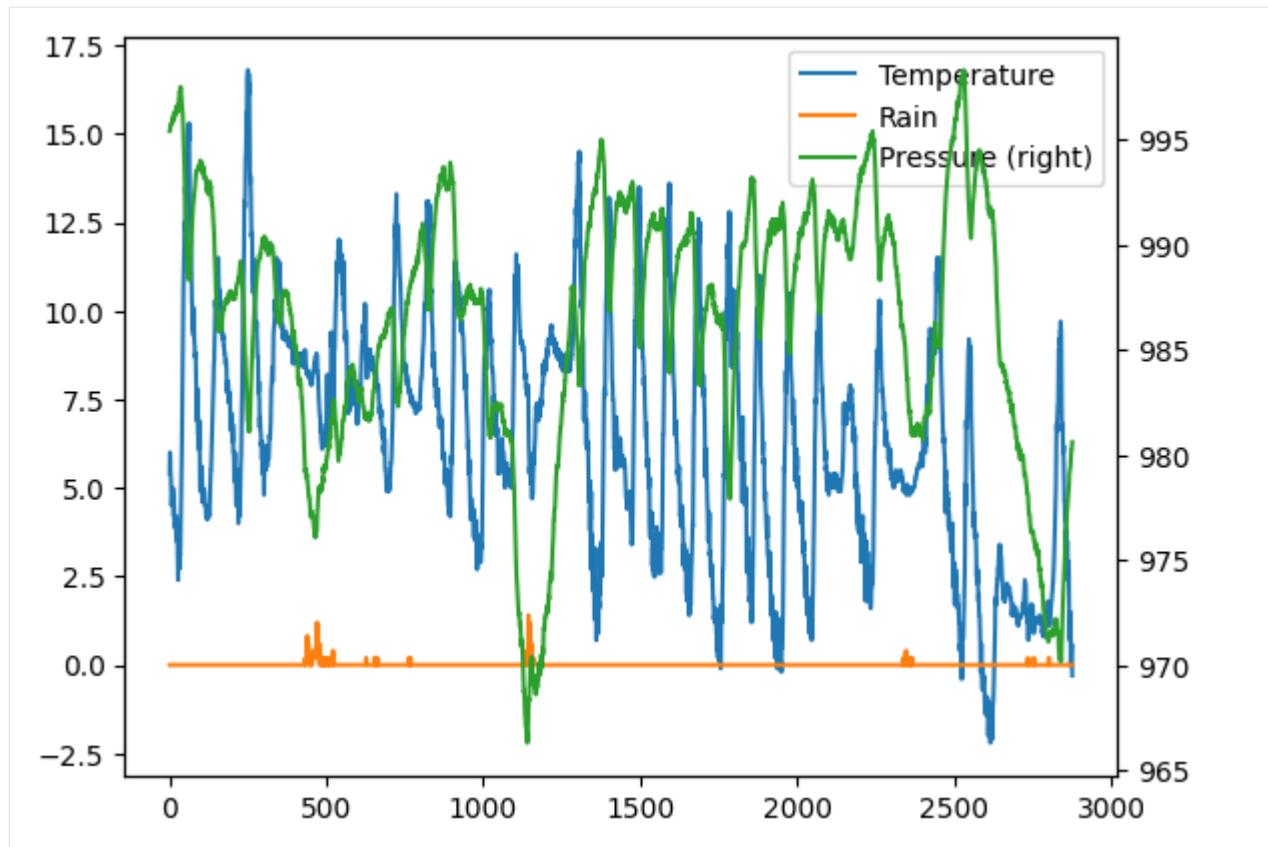
```
[46]: # SOLUTION
```

```
meteo.Temp.plot(label="Temperature", legend=True)
meteo.Rain.plot(label="Rain", legend=True)
meteo.Pressure.plot(secondary_y=True, label="Pressure", legend=True);
plt.show()
```



</div>

[46] :



## 5. Object values and strings

In general, when we want to manipulate objects of a known type, say strings which have type `str`, we can write `.str` after a series and then treat the result like it were a single string, using any operator (es: slicing) or method that particular class allows us, plus others provided by pandas

Text in particular can be manipulated in many ways, for more details see pandas documentation<sup>375)</sup>

### 5.1 Filter by textual values

When we want to filter by text values, we can use `.str.contains`, here for example we select all the samples in the last days of february (which have timestamp containing 2016-02-2) :

```
[47]: df[df['time_stamp'].str.contains('2016-02-2')]
```

	time_stamp	temp_cpu	temp_h	temp_p	humidity	pressure
30442	2016-02-20 00:00:00	32.30	28.12	25.59	45.05	1008.01
30443	2016-02-20 00:00:10	32.25	28.13	25.59	44.82	1008.02
30444	2016-02-20 00:00:41	33.07	28.13	25.59	45.08	1008.09
30445	2016-02-20 00:00:50	32.63	28.10	25.60	44.87	1008.07
30446	2016-02-20 00:01:01	32.55	28.11	25.60	44.94	1008.07
...	...	...	...	...	...	...
110864	2016-02-29 09:24:21	31.56	27.52	24.83	42.94	1005.83
110865	2016-02-29 09:24:30	31.55	27.50	24.83	42.72	1005.85

(continues on next page)

<sup>375</sup> <https://pandas.pydata.org/pandas-docs/stable/text.html>

(continued from previous page)

110866	2016-02-29	09:24:41	31.58	27.50	24.83	42.83	1005.85	
110867	2016-02-29	09:24:50	31.62	27.50	24.83	42.81	1005.88	
110868	2016-02-29	09:25:00	31.57	27.51	24.83	42.94	1005.86	
	pitch	roll	yaw	mag_x	mag_y	mag_z	accel_x	\
30442	1.47	51.82	51.18	9.215883	-12.947023	4.066202	-0.000612	
30443	0.81	51.53	52.21	8.710130	-13.143595	3.499386	-0.000718	
30444	0.68	51.69	57.36	7.383435	-13.827667	4.438656	-0.000700	
30445	1.42	52.13	59.95	7.292313	-13.999682	4.517029	-0.000657	
30446	1.41	51.86	61.83	6.699141	-14.065591	4.448778	-0.000678	
...	...	...	...	...	...	...	...	
110864	1.58	49.93	129.60	-15.169673	-27.642610	1.563183	-0.000682	
110865	1.89	49.92	130.51	-15.832622	-27.729389	1.785682	-0.000736	
110866	2.09	50.00	132.04	-16.646212	-27.719479	1.629533	-0.000647	
110867	2.88	49.69	133.00	-17.270447	-27.793136	1.703806	-0.000835	
110868	2.17	49.77	134.18	-17.885872	-27.824149	1.293345	-0.000787	
	accel_y	accel_z	gyro_x	gyro_y	gyro_z	reset		
30442	0.018792	0.014558	-0.000042	0.000275	0.000157	0		
30443	0.019290	0.014667	0.000260	0.001011	0.000149	0		
30444	0.018714	0.014598	0.000299	0.000343	-0.000025	0		
30445	0.018857	0.014565	0.000160	0.000349	-0.000190	0		
30446	0.018871	0.014564	-0.000608	-0.000381	-0.000243	0		
...	...	...	...	...	...	...		
110864	0.017743	0.014646	-0.000264	0.000206	0.000196	0		
110865	0.017570	0.014855	0.000143	0.000199	-0.000024	0		
110866	0.017657	0.014799	0.000537	0.000257	0.000057	0		
110867	0.017635	0.014877	0.000534	0.000456	0.000195	0		
110868	0.017261	0.014380	0.000459	0.000076	0.000030	0		

[80427 rows x 19 columns]

**WARNING: DON'T use operator in:**

You may be tempted to use it for filtering but you will soon discover it doesn't work. The reason is it produces *only one* value but when filtering we want a series of booleans with *n* values, one per row.

[48]: # Try check what happens when using it:

## 5.2 Extracting strings

To extract only the day from timestamp column, we can use str and slice operator with square brackets:

[50]: df['time\_stamp'].str[8:10]

```
0 16
1 16
2 16
3 16
4 16
 ..
110864 29
```

(continues on next page)

(continued from previous page)

```
110865 29
110866 29
110867 29
110868 29
Name: time_stamp, Length: 110869, dtype: object
```

## 6. Operations on columns

Let's see now how to select, add and trasform columns.

### 6.1 - Selecting columns

If we want a subset of columns, we can express the names in a list like this:

**NOTE:** inside the external square brackets ther is a simple list without df!

```
[51]: df[['temp_h', 'temp_p', 'time_stamp']]
```

	temp_h	temp_p	time_stamp
0	27.57	25.01	2016-02-16 10:44:40
1	27.53	25.01	2016-02-16 10:44:50
2	27.53	25.01	2016-02-16 10:45:00
3	27.52	25.01	2016-02-16 10:45:10
4	27.54	25.01	2016-02-16 10:45:20
...	...	...	...
110864	27.52	24.83	2016-02-29 09:24:21
110865	27.50	24.83	2016-02-29 09:24:30
110866	27.50	24.83	2016-02-29 09:24:41
110867	27.50	24.83	2016-02-29 09:24:50
110868	27.51	24.83	2016-02-29 09:25:00

[110869 rows x 3 columns]

As always selecting the columns doens't change the original dataframe:

```
[52]: df.head()
```

	time_stamp	temp_cpu	temp_h	temp_p	humidity	pressure	pitch	\
0	2016-02-16 10:44:40	31.88	27.57	25.01	44.94	1001.68	1.49	
1	2016-02-16 10:44:50	31.79	27.53	25.01	45.12	1001.72	1.03	
2	2016-02-16 10:45:00	31.66	27.53	25.01	45.12	1001.72	1.24	
3	2016-02-16 10:45:10	31.69	27.52	25.01	45.32	1001.69	1.57	
4	2016-02-16 10:45:20	31.66	27.54	25.01	45.18	1001.71	0.85	

	roll	yaw	mag_x	mag_y	mag_z	accel_x	accel_y	\
0	52.25	185.21	-46.422753	-8.132907	-12.129346	-0.000468	0.019439	
1	53.73	186.72	-48.778951	-8.304243	-12.943096	-0.000614	0.019436	
2	53.57	186.21	-49.161878	-8.470832	-12.642772	-0.000569	0.019359	
3	53.63	186.03	-49.341941	-8.457380	-12.615509	-0.000575	0.019383	
4	53.66	186.46	-50.056683	-8.122609	-12.678341	-0.000548	0.019378	

	accel_z	gyro_x	gyro_y	gyro_z	reset
0	0.014569	0.000942	0.000492	-0.000750	20
1	0.014577	0.000218	-0.000005	-0.000235	0
2	0.014357	0.000395	0.000600	-0.000003	0

(continues on next page)

(continued from previous page)

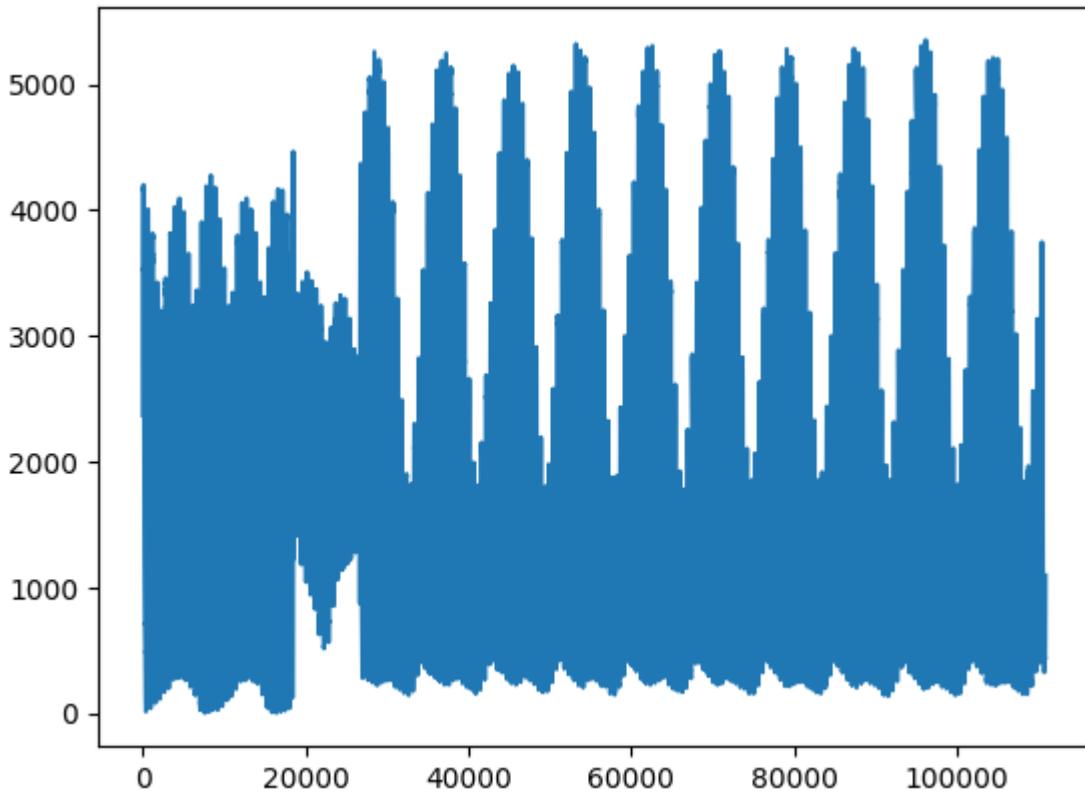
3	0.014409	0.000308	0.000577	-0.000102	0
4	0.014380	0.000321	0.000691	0.000272	0

## 6.2 - Adding columns

It's possible to obtain new columns by calculating them from other columns in a very natural way. For example, we get new column `mag_tot`, that is the absolute magnetic field taken from space station by `mag_x`, `mag_y`, e `mag_z`, and then plot it:

```
[53]: df['mag_tot'] = df['mag_x']**2 + df['mag_y']**2 + df['mag_z']**2
```

```
[54]: df.mag_tot.plot()
plt.show()
```



Let's find when the magnetic field was maximal:

```
[55]: df['time_stamp'][(df.mag_tot == df.mag_tot.values.max())]
```

```
[55]: 96156 2016-02-27 16:12:31
Name: time_stamp, dtype: object
```

Try filling in the value found on the website [iss tracker.com/historical](http://www.iss tracker.com/historical)<sup>376</sup>, you should find the positions where the magnetic field is at the highest.

<sup>376</sup> <http://www.iss tracker.com/historical>

### 6.2.1 Exercise: Meteo temperature in Fahrenheit

In `meteo` dataframe, create a column `Temp (Fahrenheit)` with the temperature measured in Fahrenheit degrees.

Formula to calculate conversion from Celsius degrees (C):

$$\text{Fahrenheit} = \frac{9}{5}C + 32$$

```
[56]: # write here
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[57]: # SOLUTION
```

```
print()
print("***** SOLUTION OUTPUT *****")
meteo['Temp (Fahrenheit)'] = meteo['Temp'] * 9/5 + 32
meteo.head()
```

```
***** SOLUTION OUTPUT *****
```

	Date	Pressure	Rain	Temp	Temp (Fahrenheit)
0	01/11/2017 00:00	995.4	0.0	5.4	41.72
1	01/11/2017 00:15	995.5	0.0	6.0	42.80
2	01/11/2017 00:30	995.5	0.0	5.9	42.62
3	01/11/2017 00:45	995.7	0.0	5.4	41.72
4	01/11/2017 01:00	995.7	0.0	5.3	41.54

</div>

```
[57]:
```

```
***** SOLUTION OUTPUT *****
```

	Date	Pressure	Rain	Temp	Temp (Fahrenheit)
0	01/11/2017 00:00	995.4	0.0	5.4	41.72
1	01/11/2017 00:15	995.5	0.0	6.0	42.80
2	01/11/2017 00:30	995.5	0.0	5.9	42.62
3	01/11/2017 00:45	995.7	0.0	5.4	41.72
4	01/11/2017 01:00	995.7	0.0	5.3	41.54

### 6.2.2 Exercise - Pressure vs Temperature

Pressure should be directly proportional to temperature in a closed environment [Gay-Lussac's law](#)<sup>377</sup>:

$$\frac{P}{T} = k$$

Does this holds true for `meteo` dataset? Try to find out by direct calculation of the formula and compare with `corr()` method results.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

<sup>377</sup> [https://en.wikipedia.org/wiki/Gay-Lussac%27s\\_law](https://en.wikipedia.org/wiki/Gay-Lussac%27s_law)

```
[58]: # SOLUTION

as expected, in an open environment there is not much linear correlation
print(meteo.corr())
meteo['Pressure'] / meteo['Temp']

Pressure Pressure Rain Temp Temp (Fahrenheit)
Pressure 1.000000 -0.270345 -0.214149 -0.214149
Rain -0.270345 1.000000 0.025227 0.025227
Temp -0.214149 0.025227 1.000000 1.000000
Temp (Fahrenheit) -0.214149 0.025227 1.000000 1.000000

[58]: 0 184.333333
1 165.916667
2 168.728814
3 184.388889
4 187.867925
...
2873 4900.000000
2874 1960.400000
2875 1633.666667
2876 4902.500000
2877 -3268.666667
Length: 2878, dtype: float64
```

</div>

```
[58]:
```

### 6.3 Scrivere in colonne filtrate con loc

The `loc` property allows to filter rows according to a property and select a column, which can be new. In this case, for rows where cpu temperature is too high, we write `True` value in the fields of the column with header '`cpu_too_hot`' :

```
[59]: df.loc[(df.temp_cpu > 31.68), 'cpu_too_hot'] = True
```

Let's see the resulting table (scroll until the end to see the new column). We note the values from the rows we did not filter are represented with `NaN`<sup>378</sup>, which literally means *not a number* :

```
[60]: df.head()

[60]: time_stamp temp_cpu temp_h temp_p humidity pressure pitch \
0 2016-02-16 10:44:40 31.88 27.57 25.01 44.94 1001.68 1.49
1 2016-02-16 10:44:50 31.79 27.53 25.01 45.12 1001.72 1.03
2 2016-02-16 10:45:00 31.66 27.53 25.01 45.12 1001.72 1.24
3 2016-02-16 10:45:10 31.69 27.52 25.01 45.32 1001.69 1.57
4 2016-02-16 10:45:20 31.66 27.54 25.01 45.18 1001.71 0.85

 roll yaw mag_x ... mag_z accel_x accel_y accel_z \
0 52.25 185.21 -46.422753 ... -12.129346 -0.000468 0.019439 0.014569
1 53.73 186.72 -48.778951 ... -12.943096 -0.000614 0.019436 0.014577
2 53.57 186.21 -49.161878 ... -12.642772 -0.000569 0.019359 0.014357
3 53.63 186.03 -49.341941 ... -12.615509 -0.000575 0.019383 0.014409
4 53.66 186.46 -50.056683 ... -12.678341 -0.000548 0.019378 0.014380
```

(continues on next page)

<sup>378</sup> <https://en.softpython.org/matrices-numpy/matrices-numpy-sol.html#NaNs-and-infinities>

(continued from previous page)

	gyro_x	gyro_y	gyro_z	reset	mag_tot	cpu_too_hot
0	0.000942	0.000492	-0.000750	20	2368.337207	True
1	0.000218	-0.000005	-0.000235	0	2615.870247	True
2	0.000395	0.000600	-0.000003	0	2648.484927	NaN
3	0.000308	0.000577	-0.000102	0	2665.305485	True
4	0.000321	0.000691	0.000272	0	2732.388620	NaN

[5 rows x 21 columns]

Pandas is a very flexible library, and gives several methods to obtain the same results.

For example, if we want to write in a column some values where rows satisfy a given criteria, and other values when the condition isn't satisfied, we can use a single command `np.where`

Proviamo ad aggiungere una colonna `controllo_pressione` che mi dice se la pressione è below or above the average (scroll till the end to see it):

```
[61]: avg_pressure = df.pressure.values.mean()
```

```
[62]: df['check_p'] = np.where(df.pressure <= avg_pressure, 'below', 'over')
```

Let's select some rows where we know the variation is evident:

[63]:	df.iloc[29735:29745]							
[63]:		time_stamp	temp_cpu	temp_h	temp_p	humidity	pressure	\
	29735	2016-02-19 22:00:51	32.24	28.03	25.52	44.55	1008.11	
	29736	2016-02-19 22:01:00	32.18	28.05	25.52	44.44	1008.11	
	29737	2016-02-19 22:01:11	32.22	28.04	25.52	44.40	1008.12	
	29738	2016-02-19 22:01:20	32.18	28.04	25.52	44.38	1008.14	
	29739	2016-02-19 22:01:30	32.24	28.03	25.52	44.43	1008.10	
	29740	2016-02-19 22:01:40	32.26	28.04	25.52	44.37	1008.11	
	29741	2016-02-19 22:01:50	32.22	28.04	25.52	44.49	1008.15	
	29742	2016-02-19 22:02:01	32.21	28.04	25.52	44.48	1008.13	
	29743	2016-02-19 22:02:10	32.23	28.05	25.52	44.45	1008.11	
	29744	2016-02-19 22:02:21	32.24	28.05	25.52	44.60	1008.12	
		pitch	roll	yaw	mag_x	...	accel_x	accel_y
	29735	1.83	52.10	272.50	0.666420	...	-0.000630	0.018846
	29736	1.16	51.73	273.26	1.028125	...	-0.000606	0.018716
	29737	2.10	52.16	274.66	1.416078	...	-0.000736	0.018774
	29738	1.38	52.01	275.22	1.702723	...	-0.000595	0.018928
	29739	1.42	51.98	275.80	1.910006	...	-0.000619	0.018701
	29740	1.47	52.08	277.11	2.413142	...	-0.000574	0.018719
	29741	1.60	52.17	278.52	2.929722	...	-0.000692	0.018716
	29742	1.47	52.24	279.44	3.163792	...	-0.000639	0.019034
	29743	1.88	51.81	280.36	3.486707	...	-0.000599	0.018786
	29744	1.26	51.83	281.22	3.937303	...	-0.000642	0.018701
		gyro_x	gyro_y	gyro_z	reset	mag_tot	cpu_too_hot	check_p
	29735	0.000127	0.000002	0.000234	0	481.466349	True	below
	29736	0.000536	0.000550	0.000103	0	476.982306	True	below
	29737	0.000717	0.000991	0.000309	0	484.654588	True	below
	29738	0.000068	0.000222	0.000034	0	485.716793	True	over
	29739	-0.000093	-0.000080	0.000018	0	481.830794	True	below
	29740	0.000451	0.000524	0.000078	0	486.220778	True	below

(continues on next page)

(continued from previous page)

29741	0.000670	0.000455	0.000109	0	480.890508	True	over
29742	0.000221	0.000553	0.000138	0	483.919953	True	over
29743	-0.000020	0.000230	0.000134	0	476.163984	True	below
29744	0.000042	0.000156	0.000071	0	478.775309	True	below

[10 rows x 22 columns]

## 6.3 Transforming columns

Suppose we want to convert all values of column temperature which are floats to integers.

We know that to convert a single float to an integer there the predefined python function `int`

[64]: `int(23.7)`

[64]: 23

How to apply such function to *all* the elements of the column `humidity`.

To do so, we can call the `transform` method and pass to it the function object `int` *as a parameter*

**NOTE:** there are no round parenthesis after `int` !!!

[65]: `df['humidity'].transform(int)`

```
[65]: 0 44
 1 45
 2 45
 3 45
 4 45
 ..
110864 42
110865 42
110866 42
110867 42
110868 42
Name: humidity, Length: 110869, dtype: int64
```

Just to be clear what *passing a function* means, let's see other two *completely equivalent* ways we could have used to pass the function.

**Defining a function:** We could have defined a function `myf` like this (notice the function MUST RETURN something !)

[66]: `def myf(x):
 return int(x)``df['humidity'].transform(myf)`

```
[66]: 0 44
 1 45
 2 45
 3 45
 4 45
 ..
110864 42
110865 42
```

(continues on next page)

(continued from previous page)

```
110866 42
110867 42
110868 42
Name: humidity, Length: 110869, dtype: int64
```

**lambda function:** We could have used as well a `lambda` function<sup>379</sup>, that is, a function without a name which is defined on one line:

```
[67]: df['humidity'].transform(lambda x: int(x))

[67]: 0 44
1 45
2 45
3 45
4 45
...
110864 42
110865 42
110866 42
110867 42
110868 42
Name: humidity, Length: 110869, dtype: int64
```

Regardless of the way we choose to pass the function, `transform` method does NOT change the original dataframe:

```
[68]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 22 columns):
 # Column Non-Null Count Dtype

 0 time_stamp 110869 non-null object
 1 temp_cpu 110869 non-null float64
 2 temp_h 110869 non-null float64
 3 temp_p 110869 non-null float64
 4 humidity 110869 non-null float64
 5 pressure 110869 non-null float64
 6 pitch 110869 non-null float64
 7 roll 110869 non-null float64
 8 yaw 110869 non-null float64
 9 mag_x 110869 non-null float64
 10 mag_y 110869 non-null float64
 11 mag_z 110869 non-null float64
 12 accel_x 110869 non-null float64
 13 accel_y 110869 non-null float64
 14 accel_z 110869 non-null float64
 15 gyro_x 110869 non-null float64
 16 gyro_y 110869 non-null float64
 17 gyro_z 110869 non-null float64
 18 reset 110869 non-null int64
 19 mag_tot 110869 non-null float64
 20 cpu_too_hot 105315 non-null object
 21 check_p 110869 non-null object
dtypes: float64(18), int64(1), object(3)
memory usage: 18.6+ MB
```

<sup>379</sup> <https://en.softpython.org/functions/fun1-intro-sol.html#Lambda-functions>

If we want to add a new column, say `humidity_int`, we have to explicitly assign the result of `transform` to a new series:

```
[69]: df['humidity_int'] = df['humidity'].transform(lambda x: int(x))
```

Notice how pandas automatically infers type `int64` for the newly created column:

```
[70]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 23 columns):
 # Column Non-Null Count Dtype

 0 time_stamp 110869 non-null object
 1 temp_cpu 110869 non-null float64
 2 temp_h 110869 non-null float64
 3 temp_p 110869 non-null float64
 4 humidity 110869 non-null float64
 5 pressure 110869 non-null float64
 6 pitch 110869 non-null float64
 7 roll 110869 non-null float64
 8 yaw 110869 non-null float64
 9 mag_x 110869 non-null float64
 10 mag_y 110869 non-null float64
 11 mag_z 110869 non-null float64
 12 accel_x 110869 non-null float64
 13 accel_y 110869 non-null float64
 14 accel_z 110869 non-null float64
 15 gyro_x 110869 non-null float64
 16 gyro_y 110869 non-null float64
 17 gyro_z 110869 non-null float64
 18 reset 110869 non-null int64
 19 mag_tot 110869 non-null float64
 20 cpu_too_hot 105315 non-null object
 21 check_p 110869 non-null object
 22 humidity_int 110869 non-null int64
dtypes: float64(18), int64(2), object(3)
memory usage: 19.5+ MB
```

## 7. Other exercises

### 7.1 Exercise - Air pollutants

Let's try analyzing the hourly data from air quality monitoring stations from Autonomous Province of Trento.

Source: [dati.trentino.it](https://dati.trentino.it)<sup>380</sup>

---

<sup>380</sup> <https://dati.trentino.it/dataset/qualita-dell-aria-rilevazioni-delle-stazioni-monitoraggio>

### 7.1.1 - load the file

⊕ Load the file `aria.csv` in pandas

**IMPORTANT 1:** put the dataframe into the variable `aria`, so not to confuse it with the previous datasets.

**IMPORTANT 2:** use encoding '`'latin-1'`' (otherwise you might get weird load errors according to your operating system)

**IMPORTANT 3:** if you also receive other strange errors, try adding the parameter `engine=python`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[71]:

```
write here

import pandas as pd
import numpy as np

remember the encoding !
aria = pd.read_csv('aria.csv', encoding='latin-1')
aria.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20693 entries, 0 to 20692
Data columns (total 6 columns):
 # Column Non-Null Count Dtype
--- --
 0 Stazione 20693 non-null object
 1 Inquinante 20693 non-null object
 2 Data 20693 non-null object
 3 Ora 20693 non-null int64
 4 Valore 20693 non-null float64
 5 Unità di misura 20693 non-null object
dtypes: float64(1), int64(1), object(4)
memory usage: 970.1+ KB
```

</div>

[71]:

```
write here

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20693 entries, 0 to 20692
Data columns (total 6 columns):
 # Column Non-Null Count Dtype
--- --
 0 Stazione 20693 non-null object
 1 Inquinante 20693 non-null object
 2 Data 20693 non-null object
 3 Ora 20693 non-null int64
 4 Valore 20693 non-null float64
 5 Unità di misura 20693 non-null object
dtypes: float64(1), int64(1), object(4)
memory usage: 970.1+ KB
```

### 7.1.2 - pollutants average

⊕ find the average of PM10 pollutants at Parco S. Chiara (average on all days). You should obtain the value 11.385752688172044

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[72]: # write here

```
aria[(aria.Stazione == 'Parco S. Chiara') & (aria.Inquinante == 'PM10')].Valore.
 ↪values.mean()
```

[72]: 11.385752688172044

</div>

[72]: # write here

[72]: 11.385752688172044

### 7.1.3 - PM10 chart

⊕ Use plt.plot as seen in a *previous example* (so by directly passing the relevant Pandas series), show in a chart the values of PM10 during May 7th, 2019.

[75]: # write here

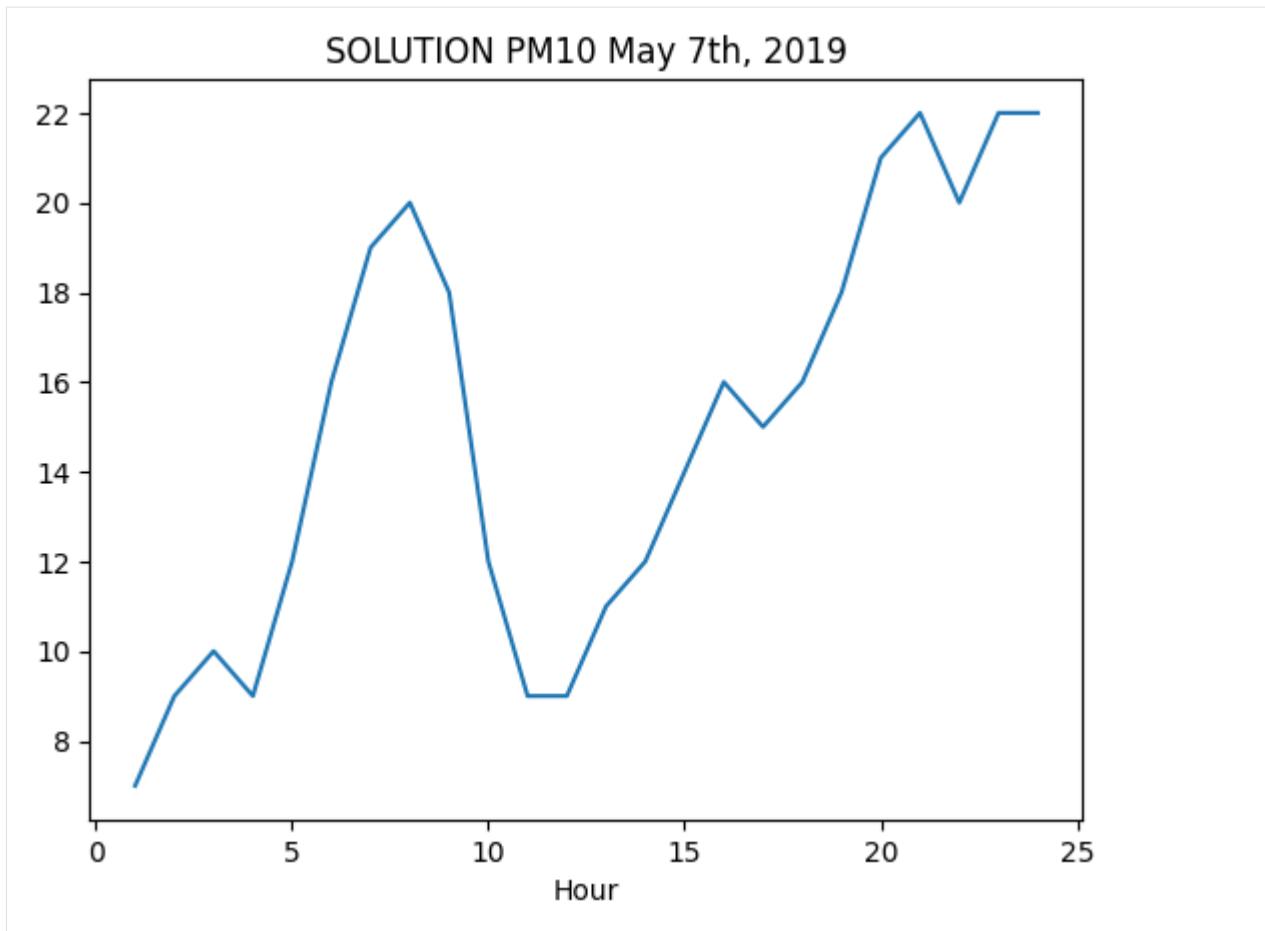
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[76]: # SOLUTION

```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

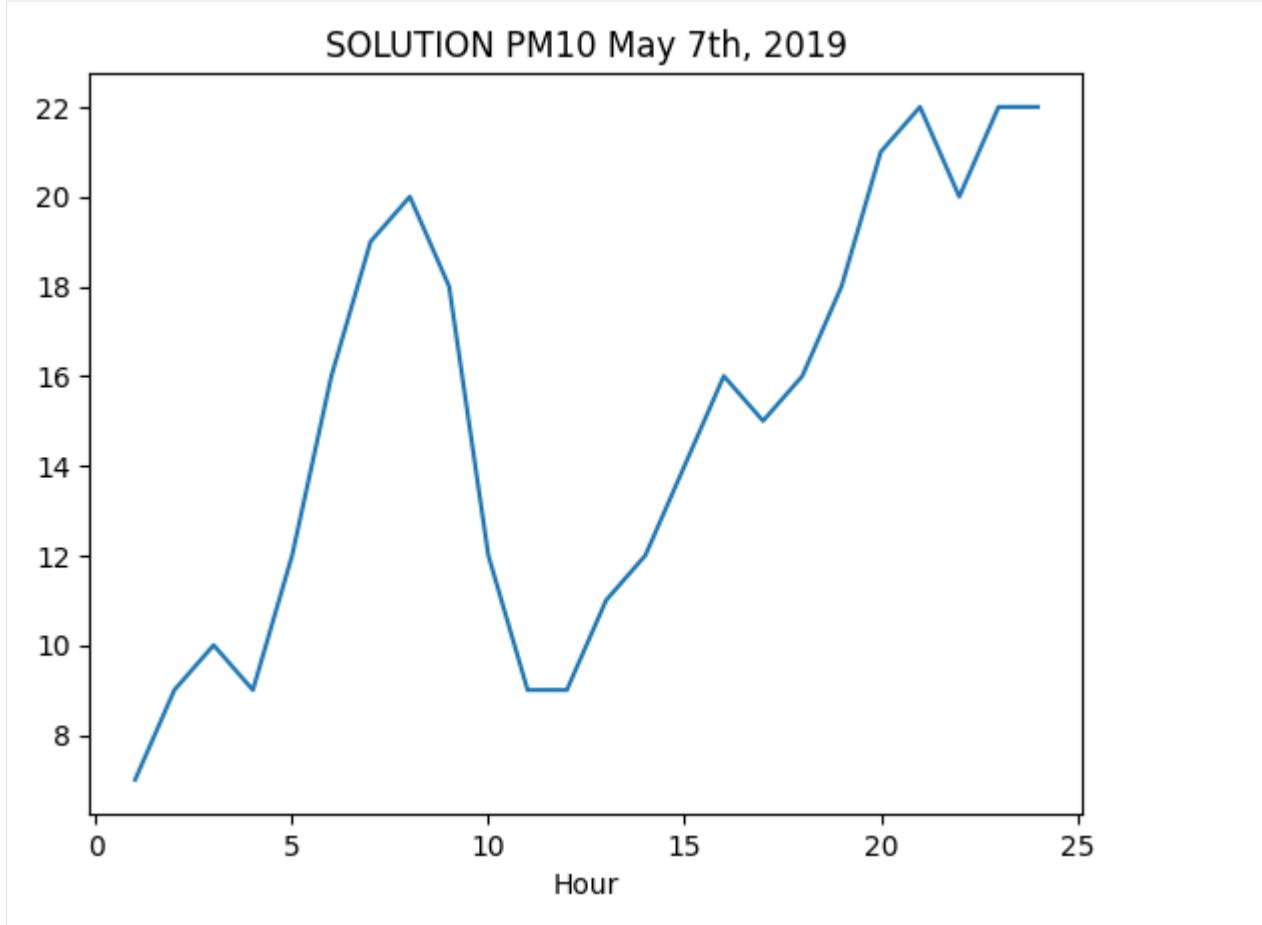
filtered = aria[(aria.Stazione == 'Parco S. Chiara') & (aria.Inquinante == 'PM10') &
 ↪(aria.Data == '2019-05-07')]

plt.plot(filtered['Ora'], filtered['Valore'])
plt.title('SOLUTION PM10 May 7th, 2019')
plt.xlabel('Hour')
plt.show()
```



</div>

[76] :



## 7.2 Exercise - Game of Thrones

Open with Pandas the file `game-of-thrones.csv` which holds episodes from various years.

- use UTF-8 encoding
- **IMPORTANT:** place the dataframe into the variable `game`, so not to confuse it with previous dataframes

Data source: [Kaggle<sup>381</sup>](#) - License: [CC0: Public Domain<sup>382</sup>](#)

### 7.2.1 Exercise - fan

You are given a dictionary `favorite` with the most liked episodes of a group of people, who unfortunately don't remember exactly the various titles which are often incomplete: Select the favorite episodes of Paolo and Chiara.

- assume the capitalization in `favorite` is the correct one
- **NOTE:** the dataset contains insidious double quotes around the titles, but if you write the code in the right way it shouldn't be a problem

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

<sup>381</sup> <https://www.kaggle.com/datasets/bakar31/game-of-thronesgot>

<sup>382</sup> <https://creativecommons.org/publicdomain/zero/1.0/>

```
[77]:

import pandas as pd

import numpy as np

favorite = {

 "Paolo" : 'Winter Is',

 "Chiara" : 'Wolf and the Lion',

 "Anselmo" : 'Fire and',

 "Letizia" : 'Garden of'

}

write here

game = pd.read_csv('game-of-thrones.csv', encoding='UTF-8')

titolidf = game[(game["Title"].str.contains(favorite['Paolo'])) | (game["Title"].

 ↪str.contains(favorite['Chiara']))]

titolidf

[77]:

 No. overall No. in season Season Title \
0 1 1 1 "Winter Is Coming"
4 5 5 1 "The Wolf and the Lion"

 Directed by Written by Novel(s) adapted \
0 Tim Van Patten David Benioff & D. B. Weiss A Game of Thrones
4 Brian Kirk David Benioff & D. B. Weiss A Game of Thrones

 Original air date U.S. viewers(millions) Imdb rating
0 17-Apr-11 2.22 9.1
4 15-May-11 2.58 9.1
```

&lt;/div&gt;

```
[77]:

import pandas as pd

import numpy as np

favorite = {

 "Paolo" : 'Winter Is',

 "Chiara" : 'Wolf and the Lion',

 "Anselmo" : 'Fire and',

 "Letizia" : 'Garden of'

}

write here

[77]:

 No. overall No. in season Season Title \
0 1 1 1 "Winter Is Coming"
4 5 5 1 "The Wolf and the Lion"

 Directed by Written by Novel(s) adapted \
0 Tim Van Patten David Benioff & D. B. Weiss A Game of Thrones
4 Brian Kirk David Benioff & D. B. Weiss A Game of Thrones
```

(continues on next page)

(continued from previous page)

	Original air date	U.S. viewers(millions)	Imdb rating
0	17-Apr-11	2.22	9.1
4	15-May-11	2.58	9.1

## 7.2.2 Esercise - first airing

Select all the episodes which have been aired the first time in a given year (Original air date column)

- NOTE: year is given as an int

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"  
data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[78]: year = 17

write here
annidf = game[game['Original air date'].str[-2:] == str(year)]
annidf
```

	No.	overall	No. in season	Season	Title \
61	62		2	7	"Stormborn"
62	63		3	7	"The Queen's Justice"
63	64		4	7	"The Spoils of War"
64	65		5	7	"Eastwatch"
65	66		6	7	"Beyond the Wall"
66	67		7	7	"The Dragon and the Wolf"

	Directed by	Written by \
61	Mark Mylod	Bryan Cogman
62	Mark Mylod	David Benioff & D. B. Weiss
63	Matt Shakman	David Benioff & D. B. Weiss
64	Matt Shakman	Dave Hill
65	Alan Taylor	David Benioff & D. B. Weiss
66	Jeremy Podeswa	David Benioff & D. B. Weiss

	Novel(s) adapted	Original air date \
61	Outline from A Dream of Spring and original co...	23-Jul-17
62	Outline from A Dream of Spring and original co...	30-Jul-17
63	Outline from A Dream of Spring and original co...	6-Aug-17
64	Outline from A Dream of Spring and original co...	13-Aug-17
65	Outline from A Dream of Spring and original co...	20-Aug-17
66	Outline from A Dream of Spring and original co...	27-Aug-17

	U.S. viewers(millions)	Imdb rating
61	9.27	8.9
62	9.25	9.2
63	10.17	9.8
64	10.72	8.8
65	10.24	9.0
66	12.07	9.4

```
</div>

[78]: year = 17

write here
```

(continues on next page)

(continued from previous page)

[78]:	No. overall	No. in season	Season	Title \
61	62	2	7	"Stormborn"
62	63	3	7	"The Queen's Justice"
63	64	4	7	"The Spoils of War"
64	65	5	7	"Eastwatch"
65	66	6	7	"Beyond the Wall"
66	67	7	7	"The Dragon and the Wolf"
				Directed by \
61	Mark Mylod			Bryan Cogman
62	Mark Mylod	David Benioff & D. B. Weiss		
63	Matt Shakman	David Benioff & D. B. Weiss		
64	Matt Shakman		Dave Hill	
65	Alan Taylor	David Benioff & D. B. Weiss		
66	Jeremy Podeswa	David Benioff & D. B. Weiss		
				Novel(s) adapted Original air date \
61	Outline from A Dream of Spring and original co...			23-Jul-17
62	Outline from A Dream of Spring and original co...			30-Jul-17
63	Outline from A Dream of Spring and original co...			6-Aug-17
64	Outline from A Dream of Spring and original co...			13-Aug-17
65	Outline from A Dream of Spring and original co...			20-Aug-17
66	Outline from A Dream of Spring and original co...			27-Aug-17
				U.S. viewers(millions) Imdb rating
61		9.27	8.9	
62		9.25	9.2	
63		10.17	9.8	
64		10.72	8.8	
65		10.24	9.0	
66		12.07	9.4	

### 7.3 Exercise - Healthcare facilities

⊕⊕ Let's examine the dataset [SANSTRUT001.csv](#) which contains the healthcare facilities of Trentino region, and for each tells the type of assistance it offers (clinical activity, diagnostics, etc), the code and name of the communality where it is located.

Data source: [dati.trentino.it](#)<sup>383</sup> Licenza: [Creative Commons Attribution 4.0](#)<sup>384</sup>

Write a function which takes as input a town code and a text string, opens the file **with pandas** (encoding UTF-8) and:

1. PRINTS also the number of found rows
2. RETURNS a dataframe with selected only the rows having that town code and which contain the string in the column ASSISTENZA. The returned dataset must have only the columns STRUTTURA, ASSISTENZA, COD\_COMUNE, COMUNE.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

<sup>383</sup> <https://dati.trentino.it/dataset/strutture-sanitarie-dell-azienda-sanitaria-e-convenzione>

<sup>384</sup> <http://creativecommons.org/licenses/by/4.0/deed.it>

```
[79]: import pandas as pd
import numpy as np

def strutsan(cod_comune, assistenza):

 print('***** SOLUTION')
 strudf = pd.read_csv('SANSTRUT001.csv', encoding='UTF-8')
 res = strudf[((strudf['COD_COMUNE'] == cod_comune) & strudf['ASSISTENZA'].str.
 ↪contains(assistenza))]

 print("Found", res.shape[0], "facilities")
 return res[['STRUUTURA', 'ASSISTENZA', 'COD_COMUNE', 'COMUNE']]
```

</div>

```
[79]: import pandas as pd
import numpy as np

def strutsan(cod_comune, assistenza):
 raise Exception('TODO IMPLEMENT ME !')
```

```
[80]: strutsan(22050, '') # no ASSISTENZA filter
```

```
***** SOLUTION
Found 6 facilities
```

```
[80]: STRUUTURA \
0 PRESIDIO OSPEDALIERO DI CAVALESE
1 PRESIDIO OSPEDALIERO DI CAVALESE
2 PRESIDIO OSPEDALIERO DI CAVALESE
3 CENTRO SALUTE MENTALE CAVALESE
4 CENTRO DIALISI CAVALESE
5 CONSULTORIO CAVALESE

 ASSISTENZA COD_COMUNE COMUNE
0 ATTIVITA` CLINICA 22050 CAVALESE
1 DIAGNOSTICA STRUMENTALE E PER IMMAGINI 22050 CAVALESE
2 ATTIVITA` DI LABORATORIO 22050 CAVALESE
3 ASSISTENZA PSICHIATRICA 22050 CAVALESE
4 ATTIVITA` CLINICA 22050 CAVALESE
5 ATTIVITA` DI CONSULTORIO MATERNO-INFANTILE 22050 CAVALESE
```

```
[81]: strutsan(22205, 'CLINICA')
```

```
***** SOLUTION
Found 16 facilities
```

```
[81]: STRUUTURA ASSISTENZA \
59 PRESIDIO OSPEDALIERO S.CHIARA ATTIVITA` CLINICA
62 CENTRO DIALISI TRENTO ATTIVITA` CLINICA
63 POLIAMBULATORI S.CHIARA ATTIVITA` CLINICA
64 PRESIDIO OSPEDALIERO VILLA IGEA ATTIVITA` CLINICA
73 OSPEDALE CLASSIFICATO S.CAMILLO ATTIVITA` CLINICA
84 NEUROPSICHIATRIA INFANTILE - UONPI 1 ATTIVITA` CLINICA
87 CASA DI CURA VILLA BIANCA SPA ATTIVITA` CLINICA
90 CENTRO SERVIZI SANITARI ATTIVITA` CLINICA
```

(continues on next page)

(continued from previous page)

93		PSICOLOGIA CLINICA	ATTIVITA` CLINICA
122	ASSOCIAZIONE TRENTINA SCLEROSI MULTIPLA, ONLUS	ATTIVITA` CLINICA	
123	ANFFAS TRENTO ONLUS	ATTIVITA` CLINICA	
124	COOPERATIVA SOCIALE IRIFOR DEL TRENTINO ONLUS	ATTIVITA` CLINICA	
126	AGSAT ASSOCIAZIONE GENITORI SOGGETTI AUTISTICI...	ATTIVITA` CLINICA	
127	AZIENDA PUBBLICA SERVIZI ALLA PERSONA - RSA PO...	ATTIVITA` CLINICA	
130	CST TRENTO	ATTIVITA` CLINICA	
133	A.P.S.P. 'BEATO DE TSCHIDERER' - AMB. LOGO-AUD...	ATTIVITA` CLINICA	
	COD_COMUNE	COMUNE	
59	22205	TRENTO	
62	22205	TRENTO	
63	22205	TRENTO	
64	22205	TRENTO	
73	22205	TRENTO	
84	22205	TRENTO	
87	22205	TRENTO	
90	22205	TRENTO	
93	22205	TRENTO	
122	22205	TRENTO	
123	22205	TRENTO	
124	22205	TRENTO	
126	22205	TRENTO	
127	22205	TRENTO	
130	22205	TRENTO	
133	22205	TRENTO	

[82]: strutsan(22205, 'LABORATORIO')

```
***** SOLUTION
Found 5 facilities
```

	STRUTTURA	ASSISTENZA	COD_COMUNE	\
61	PRESIDIO OSPEDALIERO S.CHIARA	ATTIVITA` DI LABORATORIO	22205	
85	LABORATORI ADIGE SRL	ATTIVITA` DI LABORATORIO	22205	
86	LABORATORIO DRUSO SRL	ATTIVITA` DI LABORATORIO	22205	
89	CASA DI CURA VILLA BIANCA SPA	ATTIVITA` DI LABORATORIO	22205	
92	CENTRO SERVIZI SANITARI	ATTIVITA` DI LABORATORIO	22205	
	COMUNE			
61	TRENTO			
85	TRENTO			
86	TRENTO			
89	TRENTO			
92	TRENTO			

## Continue

Go on with advanced operations<sup>385</sup> worksheet

[ ]:

<sup>385</sup> <https://en.softpython.org/pandas/pandas2-advanced-sol.html>

## 8.3.2 Analytics with Pandas : 2. Advanced operations

### Download exercises zip

Browse files online<sup>386</sup>

Let's see how to do more advanced operations with pandas, like grouping with groupby, joining tables with merge, and perform geospatial analysis with GeoPandas (only mentioned).

We chose to collect such topics in this notebook as typically while executing these operations problems are more likely to arise and thus some further internet search is required.

### 1. Grouping

#### Reference:

- PythonDataScienceHandbook: Aggregation and Grouping<sup>387</sup>

To group items and perform statistics on each group, you can use the groupby method.

Let's see an example of a possible grouping. First we reload again the astropi.csv described in the previous tutorial<sup>388</sup>

```
[1]: import pandas as pd
import numpy as np

df = pd.read_csv('astropi.csv', encoding='UTF-8')
```

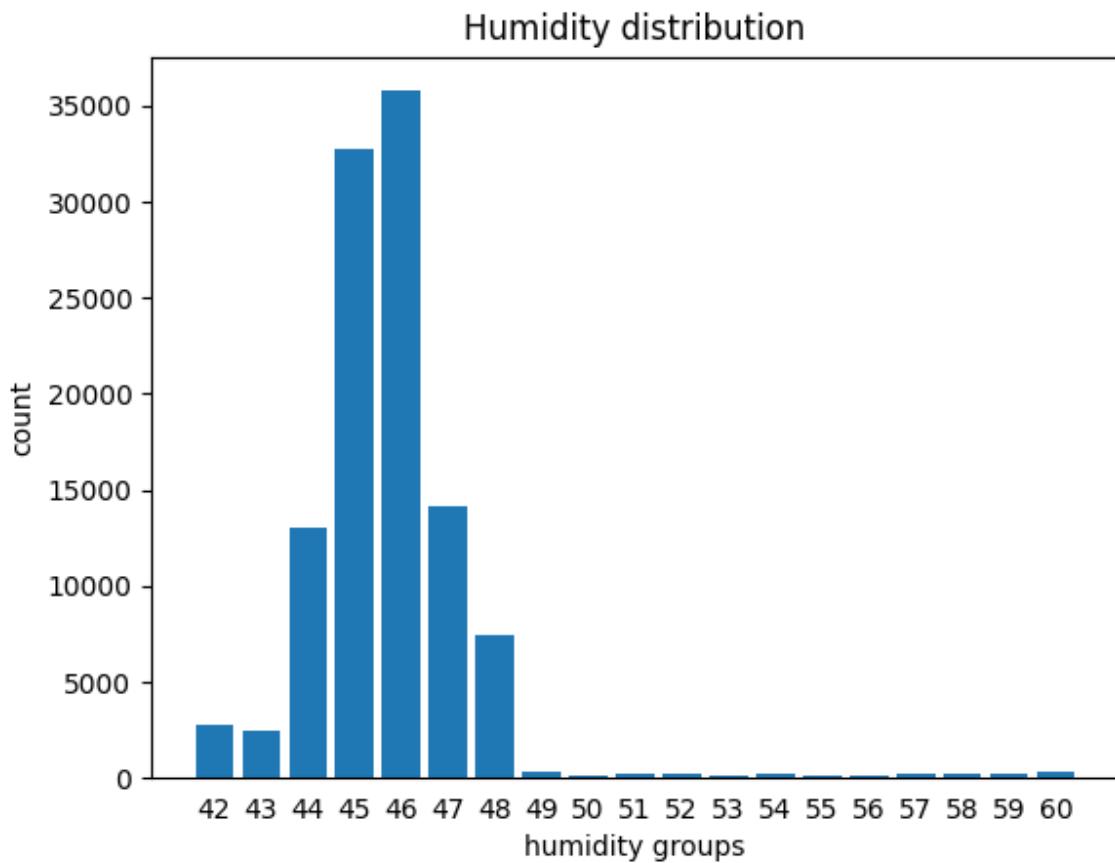
Suppose we want to calculate how many readings of humidity fall into the interval defined by each integer value of humidity humidity\_int, so to be able to plot a bar chart like this (actually there are faster methods with numpy<sup>389</sup> for making histograms but here we follow the step by step approach)

<sup>386</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/pandas>

<sup>387</sup> <https://jakevdp.github.io/PythonDataScienceHandbook/03.08-aggregation-and-grouping.html>

<sup>388</sup> <https://en.softpython.org/pandas/pandas1-intro-sol.html>

<sup>389</sup> <https://stackoverflow.com/a/13130357>



### 1.1 Let's see a group

To get an initial idea, we could start checking only the rows that belong to the group 42, that is have a humidity value lying between 42.0 included until 43.0 excluded. We can use the `transform`<sup>390</sup> method as previously seen, noting that group 42 holds 2776 rows:

[2]:	df[ df['humidity'].transform(int) == 42]
[2]:	time_stamp temp_cpu temp_h temp_p humidity pressure \
	19222 2016-02-18 16:37:00 33.18 28.96 26.51 42.99 1006.10
	19619 2016-02-18 17:43:50 33.34 29.06 26.62 42.91 1006.30
	19621 2016-02-18 17:44:10 33.38 29.06 26.62 42.98 1006.28
	19655 2016-02-18 17:49:51 33.37 29.07 26.62 42.94 1006.28
	19672 2016-02-18 17:52:40 33.33 29.06 26.62 42.93 1006.24
	... ... ... ... ... ...
	110864 2016-02-29 09:24:21 31.56 27.52 24.83 42.94 1005.83
	110865 2016-02-29 09:24:30 31.55 27.50 24.83 42.72 1005.85
	110866 2016-02-29 09:24:41 31.58 27.50 24.83 42.83 1005.85
	110867 2016-02-29 09:24:50 31.62 27.50 24.83 42.81 1005.88
	110868 2016-02-29 09:25:00 31.57 27.51 24.83 42.94 1005.86
	pitch roll yaw mag_x mag_y mag_z accel_x \

(continues on next page)

<sup>390</sup> <https://en.softpython.org/pandas/pandas1-intro-sol.html#6.5-Transforming-columns>

(continued from previous page)

19222	1.19	53.23	313.69	9.081925	-32.244905	-35.135448	-0.000581
19619	1.50	52.54	194.49	-53.197113	-4.014863	-20.257249	-0.000439
19621	1.01	52.89	195.39	-52.911983	-4.207085	-20.754475	-0.000579
19655	0.93	53.21	203.76	-43.124080	-8.181511	-29.151436	-0.000432
19672	1.34	52.71	206.97	-36.893841	-10.130503	-31.484077	-0.000551
...	...	...	...	...	...	...	...
110864	1.58	49.93	129.60	-15.169673	-27.642610	1.563183	-0.000682
110865	1.89	49.92	130.51	-15.832622	-27.729389	1.785682	-0.000736
110866	2.09	50.00	132.04	-16.646212	-27.719479	1.629533	-0.000647
110867	2.88	49.69	133.00	-17.270447	-27.793136	1.703806	-0.000835
110868	2.17	49.77	134.18	-17.885872	-27.824149	1.293345	-0.000787
	accel_y	accel_z	gyro_x	gyro_y	gyro_z	reset	
19222	0.018936	0.014607	0.000563	0.000346	-0.000113	0	
19619	0.018838	0.014526	-0.000259	0.000323	-0.000181	0	
19621	0.018903	0.014580	0.000415	-0.000232	0.000400	0	
19655	0.018919	0.014608	0.000182	0.000341	0.000015	0	
19672	0.018945	0.014794	-0.000378	-0.000013	-0.000101	0	
...	...	...	...	...	...	...	
110864	0.017743	0.014646	-0.000264	0.000206	0.000196	0	
110865	0.017570	0.014855	0.000143	0.000199	-0.000024	0	
110866	0.017657	0.014799	0.000537	0.000257	0.000057	0	
110867	0.017635	0.014877	0.000534	0.000456	0.000195	0	
110868	0.017261	0.014380	0.000459	0.000076	0.000030	0	
	[2776 rows x 19 columns]						

## 1.2 groupby

We can generalize and associate to each integer group the amount of rows belonging to that group with the `groupby` method. First let's make a column holding the integer humidity value for each group:

```
[3]: df['humidity_int'] = df['humidity'].transform(lambda x: int(x))
```

```
[4]: df[['time_stamp', 'humidity_int', 'humidity']].head()
```

	time_stamp	humidity_int	humidity
0	2016-02-16 10:44:40	44	44.94
1	2016-02-16 10:44:50	45	45.12
2	2016-02-16 10:45:00	45	45.12
3	2016-02-16 10:45:10	45	45.32
4	2016-02-16 10:45:20	45	45.18

Then we can call `groupby` by writing down:

- first the column where to group (`humidity_int`)
- second the column where to calculate the statistics
- finally the statistics to be performed, in this case `.count()` (other common ones are `sum()`, `min()`, `max()`, `mean()`...)

```
[5]: df.groupby(['humidity_int'])['humidity'].count()
```

```
[5]: humidity_int
42 2776
```

(continues on next page)

(continued from previous page)

```

43 2479
44 13029
45 32730
46 35775
47 14176
48 7392
49 297
50 155
51 205
52 209
53 128
54 224
55 164
56 139
57 183
58 237
59 271
60 300
Name: humidity, dtype: int64

```

Note the result is a Series:

```
[6]: result = df.groupby(['humidity_int'])['humidity'].count()
```

```
[7]: type(result)
```

```
[7]: pandas.core.series.Series
```

Since we would like a customized bar chart, for the sake of simplicity we could use the native `plt.plot` function of `matplotlib`, for which we will need one sequence for *xs* coordinates and another one for the *ys*.

The sequence for *xs* can be extracted from the index of the Series:

```
[8]: result.index
```

```
[8]: Int64Index([42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
 59, 60],
 dtype='int64', name='humidity_int')
```

For the *ys* sequence we can directly use the Series like this:

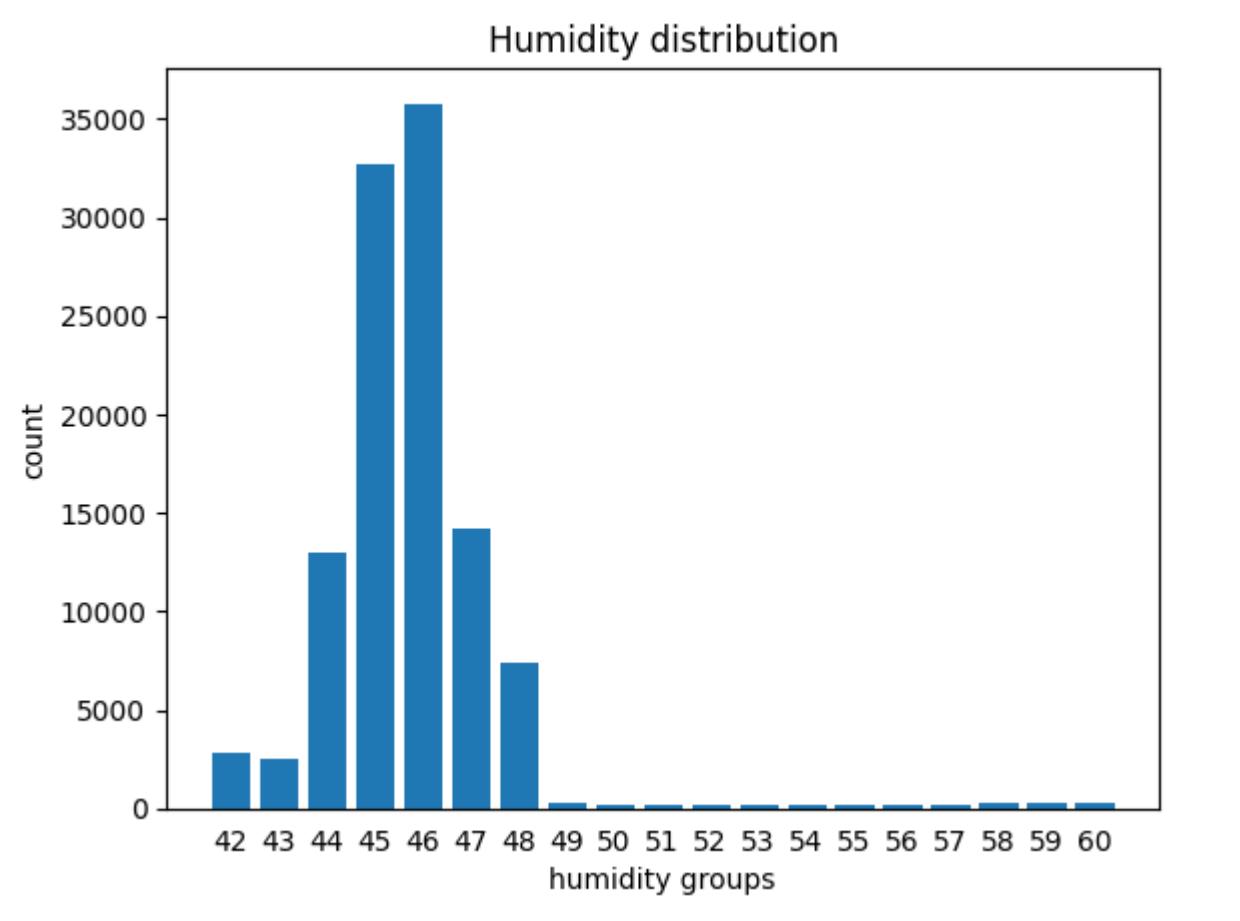
```
[9]: %matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

plt.bar(result.index, result)

plt.xlabel('humidity groups')
plt.ylabel('count')
plt.title('Humidity distribution')

plt.xticks(result.index, result.index) # shows labels as integers
plt.tick_params(bottom=False) # removes the little bottom lines

plt.show()
```



### 1.3 Modifying a dataframe by plugging in the result of a grouping

Notice we've got only 19 rows in the grouped series:

```
[10]: df.groupby(['humidity_int'])['humidity'].count()
```

```
[10]: humidity_int
42 2776
43 2479
44 13029
45 32730
46 35775
47 14176
48 7392
49 297
50 155
51 205
52 209
53 128
54 224
55 164
56 139
57 183
```

(continues on next page)

(continued from previous page)

```
58 237
59 271
60 300
Name: humidity, dtype: int64
```

How could we fill the whole original table, assigning to each row the count of its own group?

We can use `transform` like this:

```
[11]: df.groupby(['humidity_int'])['humidity'].transform('count')

[11]: 0 13029
1 32730
2 32730
3 32730
4 32730
...
110864 2776
110865 2776
110866 2776
110867 2776
110868 2776
Name: humidity, Length: 110869, dtype: int64
```

As usual, `group_by` does not modify the dataframe, if we want the result stored in the dataframe we need to assign the result to a new column:

```
[12]: df['humidity_counts'] = df.groupby(['humidity_int'])['humidity'].transform('count')
```

```
[13]: df
[13]: time_stamp temp_cpu temp_h temp_p humidity pressure \
0 2016-02-16 10:44:40 31.88 27.57 25.01 44.94 1001.68
1 2016-02-16 10:44:50 31.79 27.53 25.01 45.12 1001.72
2 2016-02-16 10:45:00 31.66 27.53 25.01 45.12 1001.72
3 2016-02-16 10:45:10 31.69 27.52 25.01 45.32 1001.69
4 2016-02-16 10:45:20 31.66 27.54 25.01 45.18 1001.71
...
110864
110865 2016-02-29 09:24:21 31.56 27.52 24.83 42.94 1005.83
110866 2016-02-29 09:24:30 31.55 27.50 24.83 42.72 1005.85
110867 2016-02-29 09:24:41 31.58 27.50 24.83 42.83 1005.85
110868 2016-02-29 09:24:50 31.62 27.50 24.83 42.81 1005.88
110869 2016-02-29 09:25:00 31.57 27.51 24.83 42.94 1005.86

 pitch roll yaw mag_x ... mag_z accel_x accel_y \
0 1.49 52.25 185.21 -46.422753 ... -12.129346 -0.000468 0.019439
1 1.03 53.73 186.72 -48.778951 ... -12.943096 -0.000614 0.019436
2 1.24 53.57 186.21 -49.161878 ... -12.642772 -0.000569 0.019359
3 1.57 53.63 186.03 -49.341941 ... -12.615509 -0.000575 0.019383
4 0.85 53.66 186.46 -50.056683 ... -12.678341 -0.000548 0.019378
...
110864 1.58 49.93 129.60 -15.169673 ... 1.563183 -0.000682 0.017743
110865 1.89 49.92 130.51 -15.832622 ... 1.785682 -0.000736 0.017570
110866 2.09 50.00 132.04 -16.646212 ... 1.629533 -0.000647 0.017657
110867 2.88 49.69 133.00 -17.270447 ... 1.703806 -0.000835 0.017635
110868 2.17 49.77 134.18 -17.885872 ... 1.293345 -0.000787 0.017261
```

(continues on next page)

(continued from previous page)

	accel_z	gyro_x	gyro_y	gyro_z	reset	humidity_int	\
0	0.014569	0.000942	0.000492	-0.000750	20	44	
1	0.014577	0.000218	-0.000005	-0.000235	0	45	
2	0.014357	0.000395	0.000600	-0.000003	0	45	
3	0.014409	0.000308	0.000577	-0.000102	0	45	
4	0.014380	0.000321	0.000691	0.000272	0	45	
...	...	...	...	...	...	...	
110864	0.014646	-0.000264	0.000206	0.000196	0	42	
110865	0.014855	0.000143	0.000199	-0.000024	0	42	
110866	0.014799	0.000537	0.000257	0.000057	0	42	
110867	0.014877	0.000534	0.000456	0.000195	0	42	
110868	0.014380	0.000459	0.000076	0.000030	0	42	
	humidity_counts						
0		13029					
1		32730					
2		32730					
3		32730					
4		32730					
...		...					
110864		2776					
110865		2776					
110866		2776					
110867		2776					
110868		2776					
[110869 rows x 21 columns]							

## 1.4 Exercise - meteo pressure intervals

⊕⊕⊕ The dataset `meteo.csv` contains the weather data of Trento, November 2017 (source: [www.meteotrentino.it](http://www.meteotrentino.it)<sup>391</sup>). We would like to subdivide the pressure readings into three intervals A (low), B (medium), C (high), and count how many readings have been made for each interval.

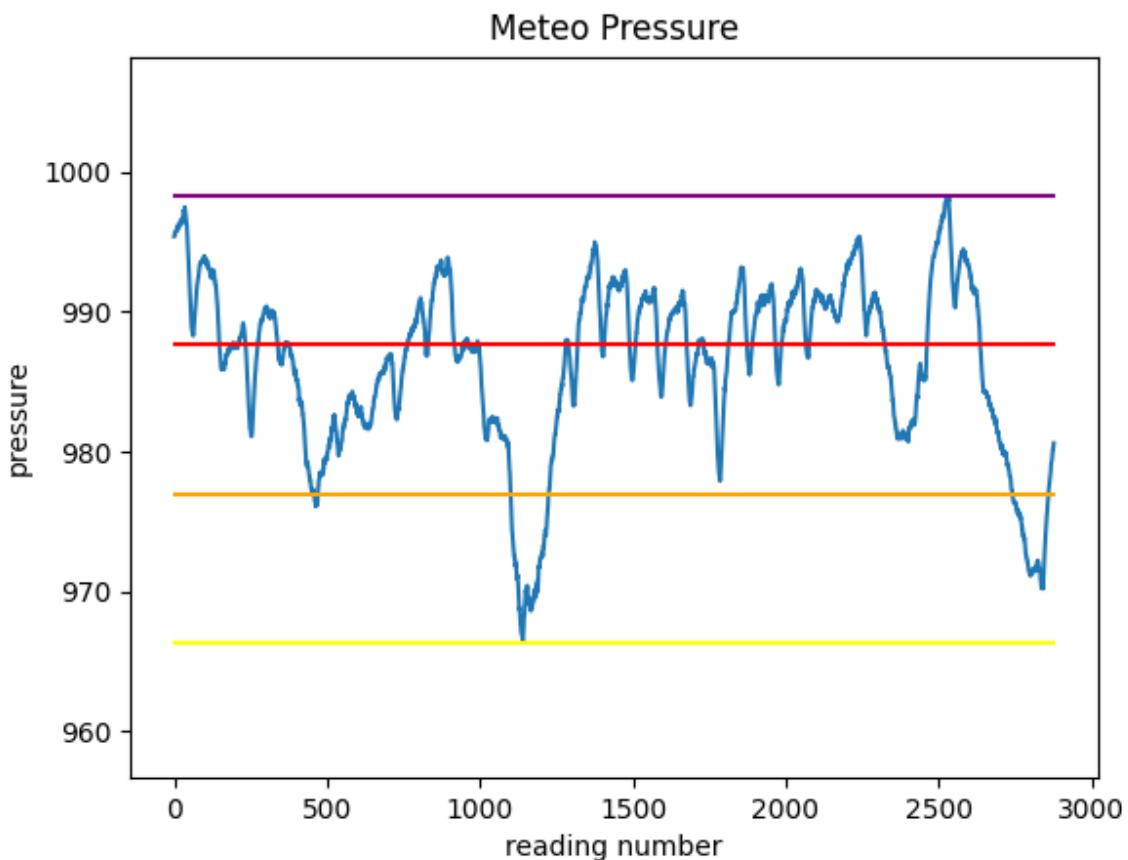
**IMPORTANT:** assign the dataframe to a variable called `meteo` so to avoid confusion with other dataframes

---

<sup>391</sup> <https://www.meteotrentino.it>

### 1.4.1 Where are the intervals?

First, let's find the pressure values for these 3 intervals and plot them as segments, so to end up with a chart like this:



Before doing the plot, we will need to know at which height we should plot the segments.

Load the dataset with pandas, calculate the following variables and PRINT them

- use UTF-8 as encoding
- round values with `round` function
- the excursion is the difference between minimum and maximum
- note `intervalC` coincides with the maximum

**DO NOT** use `min` and `max` as variable names (they are reserved functions!!)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[14]: import pandas as pd

write here

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
minimum = meteo['Pressure'].min()
```

(continues on next page)

(continued from previous page)

```
maximum = meteo['Pressure'].max()
excursion = maximum - minimum
intervalA,intervalB,intervalC = minimum + excursion/3.0, minimum + excursion*2.0/3.0,_
→minimum + excursion
intervalA,intervalB,intervalC = round(intervalA,2), round(intervalB,2),
→round(intervalC,2)

print ('minimum:',minimum)
print ('maximum:', maximum)
print ('excursion:', excursion)
print ('intervalA:', intervalA)
print ('intervalB:', intervalB)
print ('intervalC:', intervalC)

minimum: 966.3
maximum: 998.3
excursion: 32.0
intervalA: 976.97
intervalB: 987.63
intervalC: 998.3
```

&lt;/div&gt;

[14]: import pandas as pd

# write here

```
minimum: 966.3
maximum: 998.3
excursion: 32.0
intervalA: 976.97
intervalB: 987.63
intervalC: 998.3
```

## 1.4.2 Segments plot

Try now to plot the chart of pressure and the 4 horizontal segments.

- to overlay the segments with different colors, just make repeated calls to plt.plot
- a segment is defined by two points: so just find the coordinates of those two points..
- try leaving some space above and below the chart

**REMEMBER** title and labels

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[15]: # write here

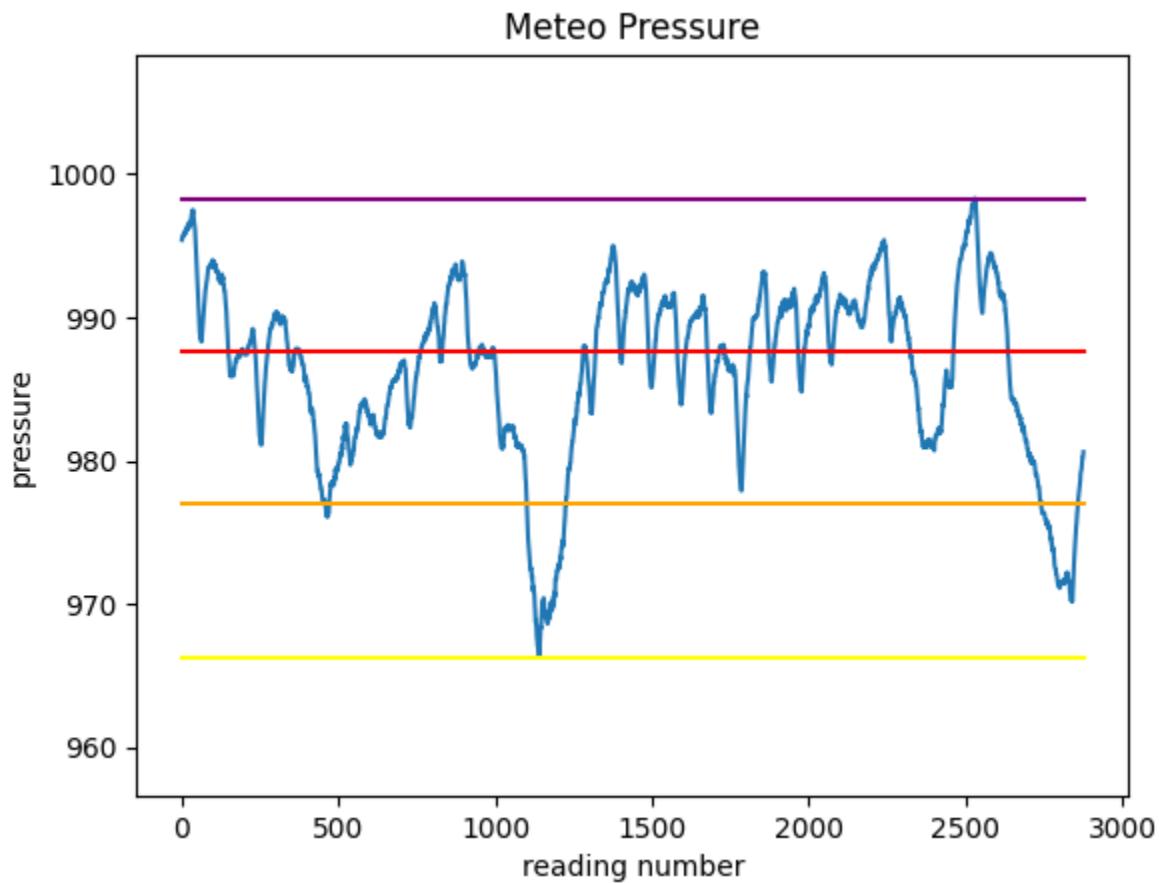
```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
plt.ylim(minimum*0.99,maximum*1.01) # sspace below and above
```

(continues on next page)

(continued from previous page)

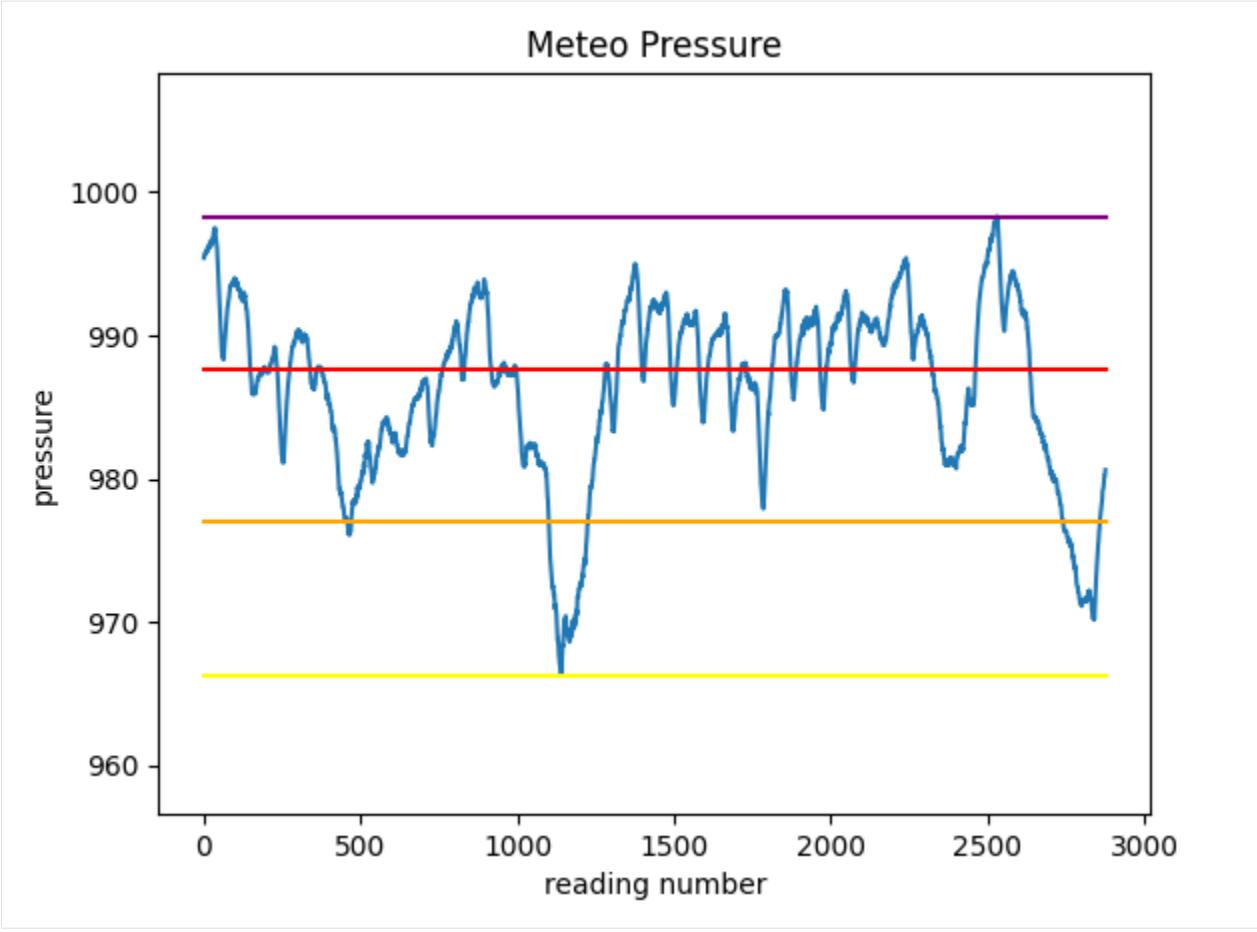
```
meteo['Pressure'].plot()
plt.plot([0,meteo.shape[0]], [minimum, minimum], color="yellow")
plt.plot([0,meteo.shape[0]], [intervalA,intervalA], color="orange")
plt.plot([0,meteo.shape[0]], [intervalB,intervalB], color="red")
plt.plot([0,meteo.shape[0]], [intervalC,intervalC], color="purple")
plt.title('Meteo Pressure')
plt.xlabel('reading number')
plt.ylabel('pressure')

plt.plot()
plt.show()
```



&lt;/div&gt;

[15]: # write here



### 1.4.3 Assigning the intervals

We literally made a picture of where the intervals are located - let's now ask ourselves how many readings have been done for each interval.

First, try creating a column which assigns to each reading the interval where it belongs to.

- **HINT 1:** use `transform`
- **HINT 2:** in the function you are going to define, do **not** recalculate inside values such as minimum, maximum, intervals etc because it would slow down Pandas. Instead, use the variables we've already defined - remember that `transform` rieexecutes the argument function for each row!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[16]: # write here

def fintervals(pressure):

 if pressure < intervalA:
 return "A (low)"
 elif pressure < intervalB :
 return "B (medium)"
```

(continues on next page)

(continued from previous page)

```
else:
 return "C (high)"
```

```
meteo['PressureInterval'] = meteo['Pressure'].transform(fintervals)
meteo
```

```
[16]:
```

	Date	Pressure	Rain	Temp	PressureInterval
0	01/11/2017 00:00	995.4	0.0	5.4	C (high)
1	01/11/2017 00:15	995.5	0.0	6.0	C (high)
2	01/11/2017 00:30	995.5	0.0	5.9	C (high)
3	01/11/2017 00:45	995.7	0.0	5.4	C (high)
4	01/11/2017 01:00	995.7	0.0	5.3	C (high)
...	...	...	...	...	...
2873	30/11/2017 23:00	980.0	0.0	0.2	B (medium)
2874	30/11/2017 23:15	980.2	0.0	0.5	B (medium)
2875	30/11/2017 23:30	980.2	0.0	0.6	B (medium)
2876	30/11/2017 23:45	980.5	0.0	0.2	B (medium)
2877	01/12/2017 00:00	980.6	0.0	-0.3	B (medium)

[2878 rows x 5 columns]

&lt;/div&gt;

```
[16]: # write here
```

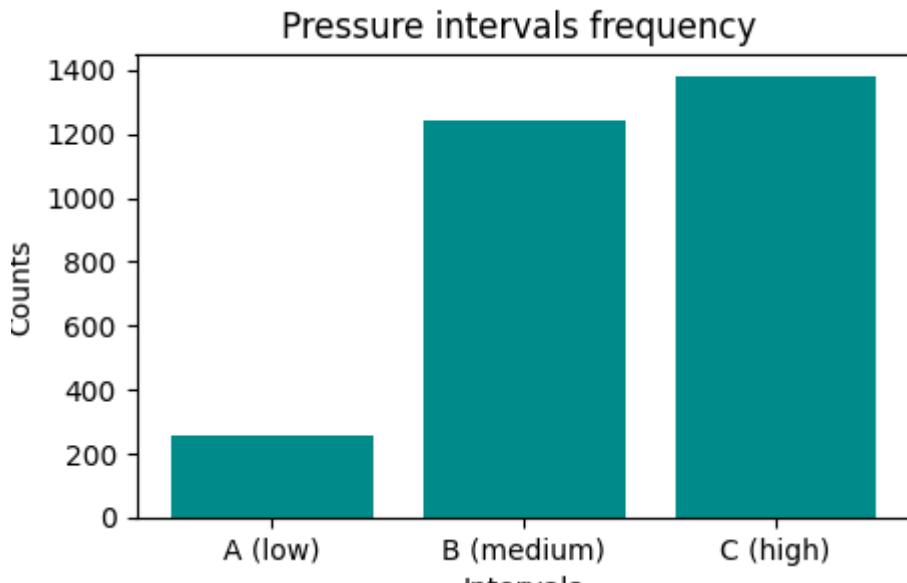
```
[16]:
```

	Date	Pressure	Rain	Temp	PressureInterval
0	01/11/2017 00:00	995.4	0.0	5.4	C (high)
1	01/11/2017 00:15	995.5	0.0	6.0	C (high)
2	01/11/2017 00:30	995.5	0.0	5.9	C (high)
3	01/11/2017 00:45	995.7	0.0	5.4	C (high)
4	01/11/2017 01:00	995.7	0.0	5.3	C (high)
...	...	...	...	...	...
2873	30/11/2017 23:00	980.0	0.0	0.2	B (medium)
2874	30/11/2017 23:15	980.2	0.0	0.5	B (medium)
2875	30/11/2017 23:30	980.2	0.0	0.6	B (medium)
2876	30/11/2017 23:45	980.5	0.0	0.2	B (medium)
2877	01/12/2017 00:00	980.6	0.0	-0.3	B (medium)

[2878 rows x 5 columns]

## 1.4.4 Grouping by intervals

We would like to have an histogram like this one:



- a. First, create a grouping to count occurrences:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[17]: # write here
meteo.groupby(['PressureInterval'])['Pressure'].count()
```

```
[17]: PressureInterval
A (low) 255
B (medium) 1243
C (high) 1380
Name: Pressure, dtype: int64
```

</div>

```
[17]: # write here
```

```
[17]: PressureInterval
A (low) 255
B (medium) 1243
C (high) 1380
Name: Pressure, dtype: int64
```

- b. Now plot it

- **NOTE:** the result of groupby is also a Series, so it's plottable as we've already seen...
- **REMEMBER** title and axis labels

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: %matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

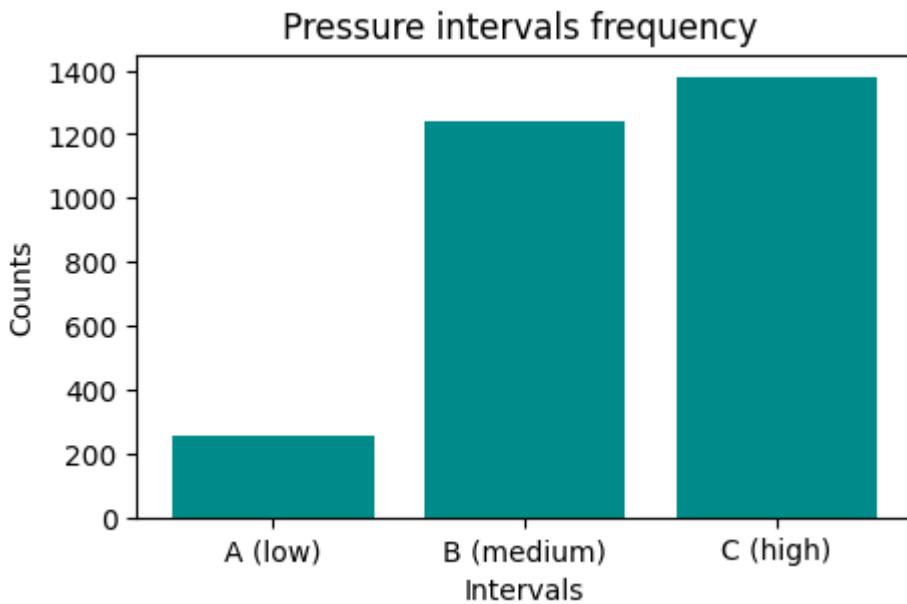
(continues on next page)

(continued from previous page)

```
write here

g = meteo.groupby(['PressureInterval'])['Pressure'].count()
plt.figure(figsize=(5, 3))
plt.title('Pressure intervals frequency ')
plt.xlabel('Intervals')
plt.ylabel('Counts')
plt.bar(g.index, g, color='darkcyan')

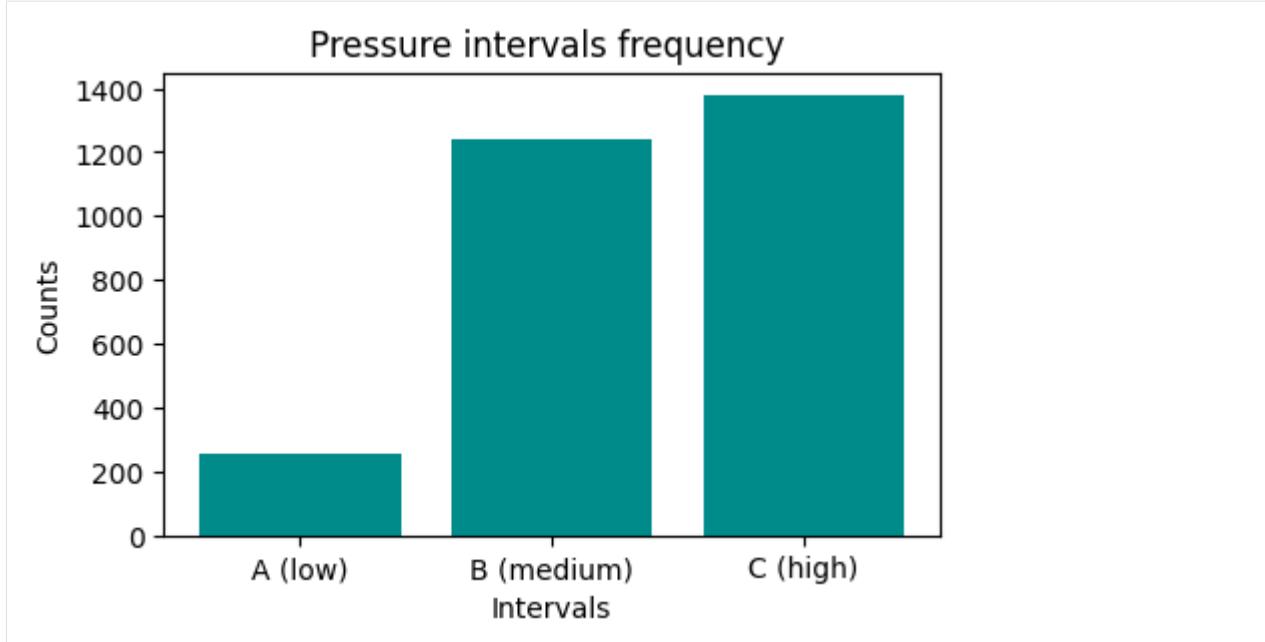
plt.show()
```



&lt;/div&gt;

```
[18]: %matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

write here
```



## 1.5 Exercise - meteo average temperature

⊕⊕⊕ Calculate the average temperature for each day, and show it in the plot, so to have a couple new columns like these:

Day	Avg_day_temp
01/11/2017	7.983333
01/11/2017	7.983333
01/11/2017	7.983333
.	.
.	.
02/11/2017	7.384375
02/11/2017	7.384375
02/11/2017	7.384375
.	.
.	.

**HINT 1:** add 'Day' column by extracting only the day from the date. To do it, use the function `.str` applied to all the column.

**HINT 2:** There are various ways to solve the exercise:

- Most performant and elegant is with `groupby` operator, see Pandas transform - more than meets the eye<sup>392</sup>
- As alternative, you may use a `for` to cycle through days. Typically, using a `for` is not a good idea with Pandas, as on large datasets it can take a lot to perform the updates. Still, since this dataset is small enough, you should get results in a decent amount of time.

[19]: # write here

```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

<sup>392</sup> <https://towardsdatascience.com/pandas-transform-more-than-meets-the-eye-928542b40b56>

```
[20]: # SOLUTION

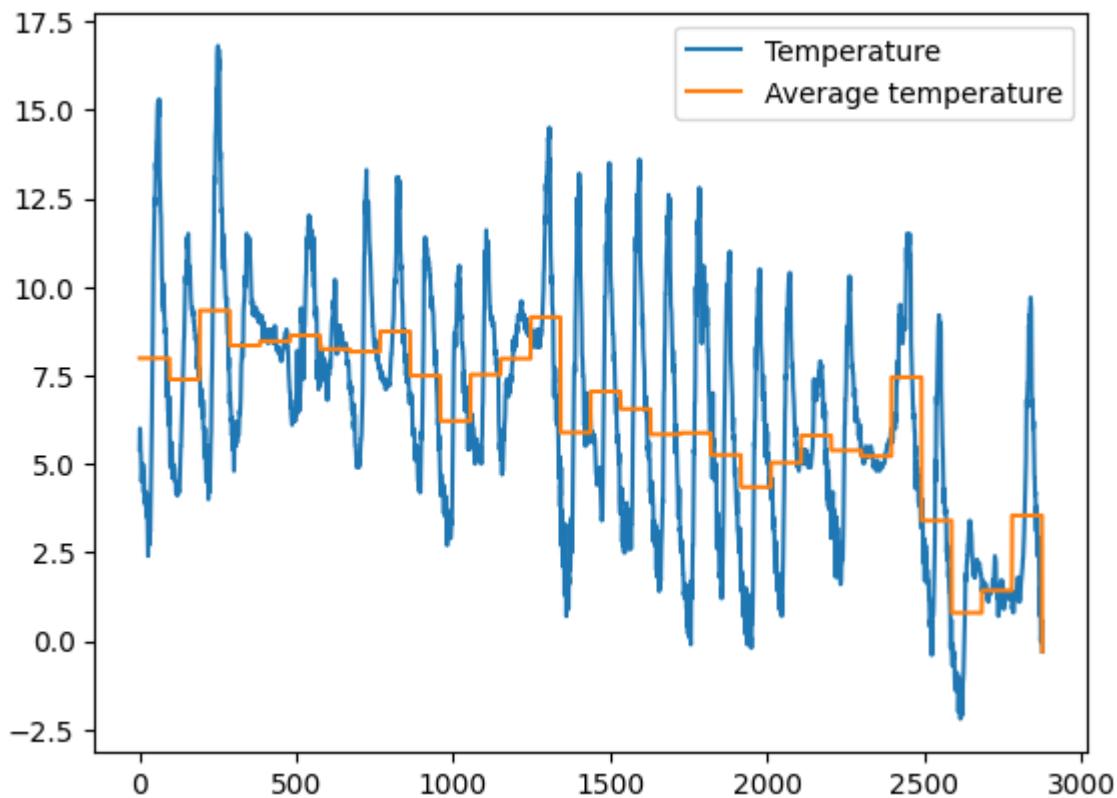
print()
print('****SOLUTION 1 (EFFICIENT) - best solution with groupby and transform')

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
meteo['Day'] = meteo['Date'].str[0:10]
.transform is needed to avoid getting a table with only 30 lines
meteo['Avg_day_temp'] = meteo.groupby('Day')['Temp'].transform('mean')

print(meteo.head())
meteo.Temp.plot(label="Temperature", legend=True)
meteo.Avg_day_temp.plot(label="Average temperature", legend=True)
plt.show()
```

\*\*\*\*SOLUTION 1 (EFFICIENT) - best solution with groupby and transform

	Date	Pressure	Rain	Temp	Day	Avg_day_temp
0	01/11/2017 00:00	995.4	0.0	5.4	01/11/2017	7.983333
1	01/11/2017 00:15	995.5	0.0	6.0	01/11/2017	7.983333
2	01/11/2017 00:30	995.5	0.0	5.9	01/11/2017	7.983333
3	01/11/2017 00:45	995.7	0.0	5.4	01/11/2017	7.983333
4	01/11/2017 01:00	995.7	0.0	5.3	01/11/2017	7.983333



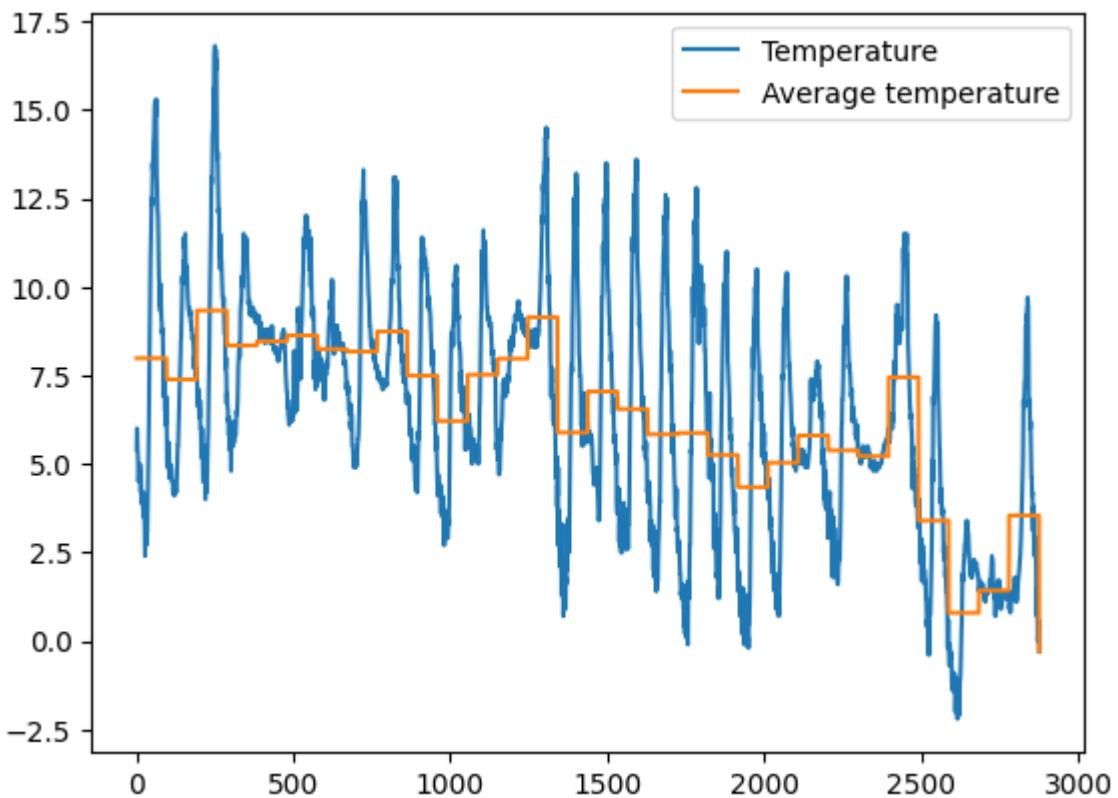
</div>

```
[20]:
```

(continues on next page)

(continued from previous page)

	Date	Pressure	Rain	Temp	Day	Avg_day_temp
0	01/11/2017 00:00	995.4	0.0	5.4	01/11/2017	7.983333
1	01/11/2017 00:15	995.5	0.0	6.0	01/11/2017	7.983333
2	01/11/2017 00:30	995.5	0.0	5.9	01/11/2017	7.983333
3	01/11/2017 00:45	995.7	0.0	5.4	01/11/2017	7.983333
4	01/11/2017 01:00	995.7	0.0	5.3	01/11/2017	7.983333



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[21]: # SOLUTION

print()
print('***** SOLUTION 2 (SLOW) - recalculate average for every row !')
print()

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
meteo['Day'] = meteo['Date'].str[0:10]

#print ("WITH DAY")
#print (meteo.head())
for day in meteo['Day']:
 avg_day_temp = meteo[(meteo.Day == day)].Temp.values.mean()
 meteo.loc[(meteo.Day == day), 'Avg_day_temp']= avg_day_temp

print(meteo.head())
meteo.Temp.plot(label="Temperature", legend=True)
```

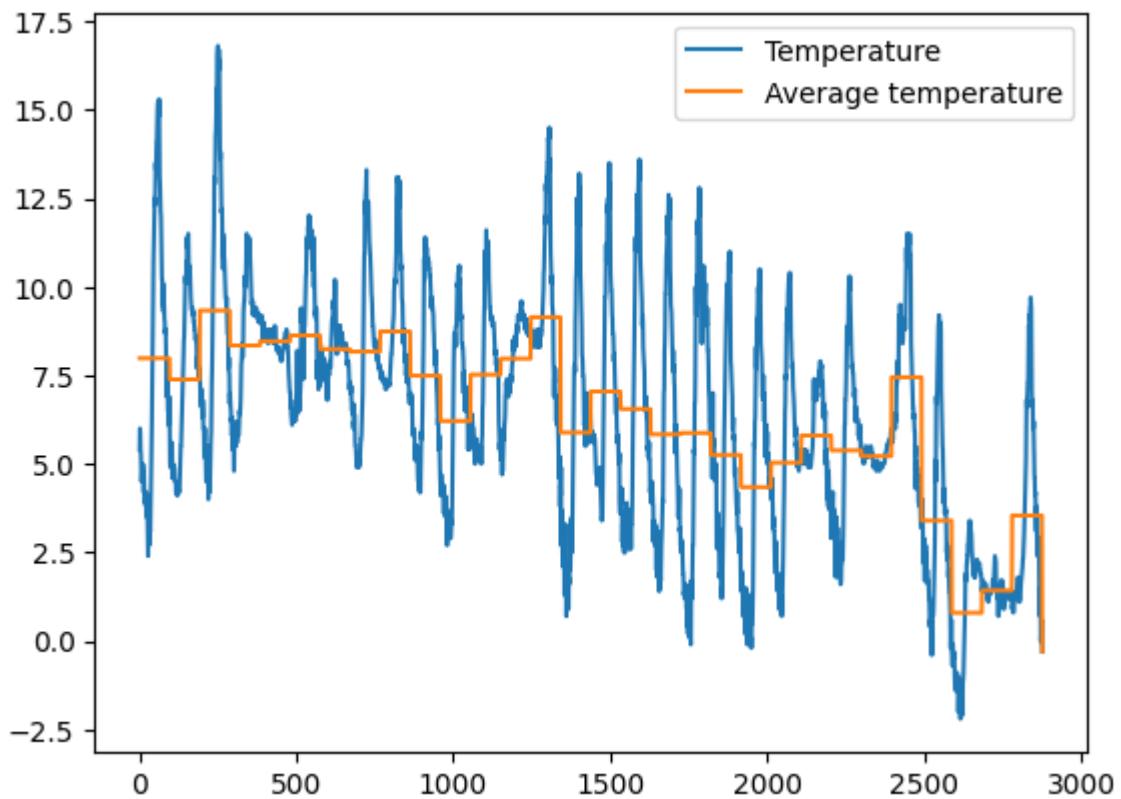
(continues on next page)

(continued from previous page)

```
meteo.Avg_day_temp.plot(label="Average temperature", legend=True)
plt.show()
```

\*\*\*\*\* SOLUTION 2 (SLOW) - recalculate average for every row !

	Date	Pressure	Rain	Temp	Day	Avg_day_temp
0	01/11/2017 00:00	995.4	0.0	5.4	01/11/2017	7.983333
1	01/11/2017 00:15	995.5	0.0	6.0	01/11/2017	7.983333
2	01/11/2017 00:30	995.5	0.0	5.9	01/11/2017	7.983333
3	01/11/2017 00:45	995.7	0.0	5.4	01/11/2017	7.983333
4	01/11/2017 01:00	995.7	0.0	5.3	01/11/2017	7.983333



</div>

[21]:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[22]: # SOLUTION

```
print()
print('***** SOLUTION 3 (STILL SLOW) - recalculate average only 30 times')
print(' by using a dictionary d_avg, faster but not yet optimal')
print()

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
```

(continues on next page)

(continued from previous page)

```

meteo['Day'] = meteo['Date'].str[0:10]
#print()
#print("WITH DAY")
#print(meteo.head())
d_avg = {}
for day in meteo['Day']:
 if day not in d_avg:
 d_avg[day] = meteo[meteo['Day'] == day] ['Temp'].mean()

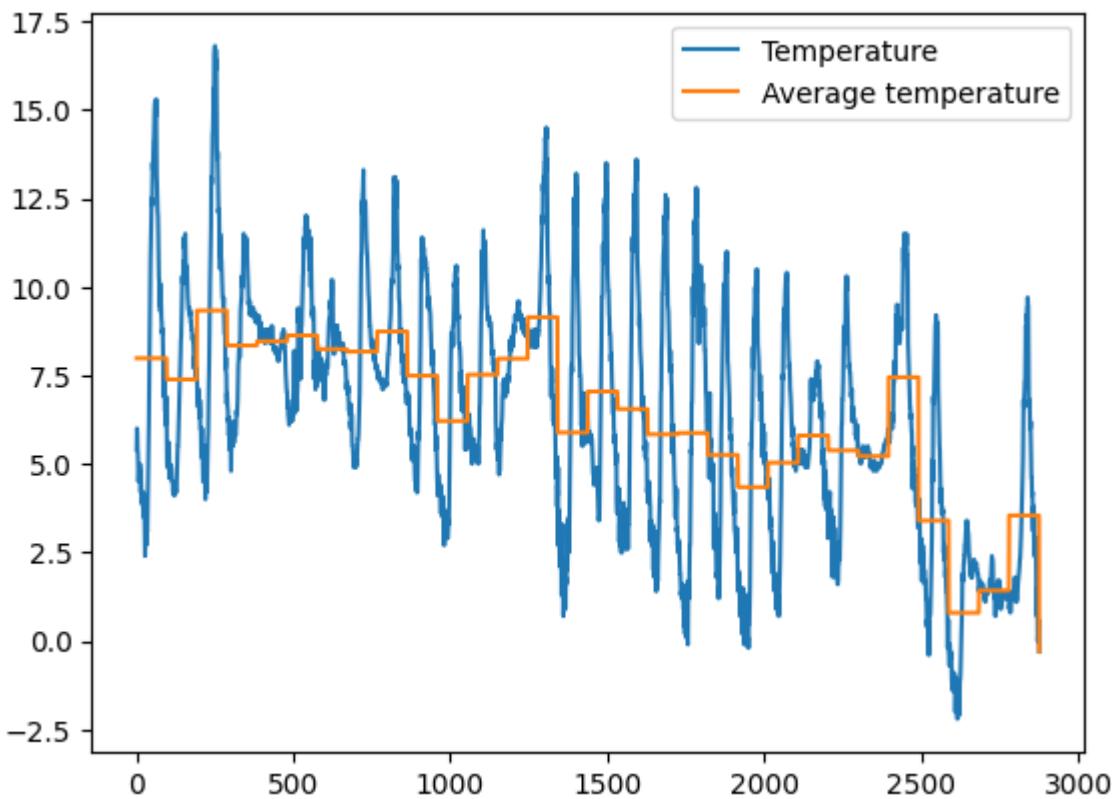
for day in meteo['Day']:
 meteo.loc[(meteo.Day == day), 'Avg_day_temp']= d_avg[day]

print(meteo.head())
meteo.Temp.plot(label="Temperature", legend=True)
meteo.Avg_day_temp.plot(label="Average temperature", legend=True)
plt.show()

```

\*\*\*\*\* SOLUTION 3 (STILL SLOW) - recalculate average only 30 times  
by using a dictionary d\_avg, faster but not yet optimal

	Date	Pressure	Rain	Temp	Day	Avg_day_temp
0	01/11/2017 00:00	995.4	0.0	5.4	01/11/2017	7.983333
1	01/11/2017 00:15	995.5	0.0	6.0	01/11/2017	7.983333
2	01/11/2017 00:30	995.5	0.0	5.9	01/11/2017	7.983333
3	01/11/2017 00:45	995.7	0.0	5.4	01/11/2017	7.983333
4	01/11/2017 01:00	995.7	0.0	5.3	01/11/2017	7.983333



&lt;/div&gt;

[22]:

## 2. Merging tables

Suppose we want to add a column with geographical position of the ISS. To do so, we would need to join our dataset with another one containing such information. Let's take for example the dataset `iss-coords.csv`

[23]: iss\_coords = pd.read\_csv('iss-coords.csv', encoding='UTF-8')

[24]: iss\_coords

```
[24]:
```

	timestamp	lat	lon
0	2016-01-01 05:11:30	-45.103458	14.083858
1	2016-01-01 06:49:59	-37.597242	28.931170
2	2016-01-01 11:52:30	17.126141	77.535602
3	2016-01-01 11:52:30	17.126464	77.535861
4	2016-01-01 14:54:08	7.259561	70.001561
..	...	...	...
333	2016-02-29 13:23:17	-51.077590	-31.093987
334	2016-02-29 13:44:13	30.688553	-135.403820
335	2016-02-29 13:44:13	30.688295	-135.403533
336	2016-02-29 18:44:57	27.608774	-130.198781
337	2016-02-29 21:36:47	27.325186	-129.893278

[338 rows x 3 columns]

We notice there is a `timestamp` column, which unfortunately has a slightly different name than `time_stamp` column (notice the underscore `_`) in original astropi dataset:

[25]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 21 columns):
 # Column Non-Null Count Dtype

 0 time_stamp 110869 non-null object
 1 temp_cpu 110869 non-null float64
 2 temp_h 110869 non-null float64
 3 temp_p 110869 non-null float64
 4 humidity 110869 non-null float64
 5 pressure 110869 non-null float64
 6 pitch 110869 non-null float64
 7 roll 110869 non-null float64
 8 yaw 110869 non-null float64
 9 mag_x 110869 non-null float64
 10 mag_y 110869 non-null float64
 11 mag_z 110869 non-null float64
 12 accel_x 110869 non-null float64
 13 accel_y 110869 non-null float64
 14 accel_z 110869 non-null float64
 15 gyro_x 110869 non-null float64
 16 gyro_y 110869 non-null float64
 17 gyro_z 110869 non-null float64
 18 reset 110869 non-null int64
 19 humidity_int 110869 non-null int64
```

(continues on next page)

(continued from previous page)

```
20 humidity_counts 110869 non-null int64
dtypes: float64(17), int64(3), object(1)
memory usage: 17.8+ MB
```

To merge datasets according to the columns, we can use the command `merge` like this:

```
[26]: # remember merge produces a NEW dataframe

geo_astropi = df.merge(iss_coords, left_on='time_stamp', right_on='timestamp')

merge will add both time_stamp and timestamp columns,
so we remove the duplicate column `timestamp`
geo_astropi = geo_astropi.drop('timestamp', axis=1)
```

```
[27]: geo_astropi
```

```
time_stamp temp_cpu temp_h temp_p humidity pressure pitch \
0 2016-02-19 03:49:00 32.53 28.37 25.89 45.31 1006.04 1.31
1 2016-02-19 14:30:40 32.30 28.12 25.62 45.57 1007.42 1.49
2 2016-02-19 14:30:40 32.30 28.12 25.62 45.57 1007.42 1.49
3 2016-02-21 22:14:11 32.21 28.05 25.50 47.36 1012.41 0.67
4 2016-02-23 23:40:50 32.32 28.18 25.61 47.45 1010.62 1.14
5 2016-02-24 10:05:51 32.39 28.26 25.70 46.83 1010.51 0.61
6 2016-02-25 00:23:01 32.38 28.18 25.62 46.52 1008.28 0.90
7 2016-02-27 01:43:10 32.42 28.34 25.76 45.72 1006.79 0.57
8 2016-02-27 01:43:10 32.42 28.34 25.76 45.72 1006.79 0.57
9 2016-02-28 09:48:40 32.62 28.62 26.02 45.15 1006.06 1.12

roll yaw mag_x ... accel_y accel_z gyro_x gyro_y \
0 51.63 34.91 21.125001 ... 0.018851 0.014607 0.000060 -0.000304
1 52.29 333.49 16.083471 ... 0.018687 0.014502 0.000208 -0.000499
2 52.29 333.49 16.083471 ... 0.018687 0.014502 0.000208 -0.000499
3 52.40 27.57 15.441683 ... 0.018800 0.014136 -0.000015 -0.000159
4 51.41 33.68 11.994554 ... 0.018276 0.014124 0.000368 0.000368
5 51.91 287.86 6.554283 ... 0.018352 0.014344 -0.000664 -0.000518
6 51.77 30.80 9.947132 ... 0.018502 0.014366 0.000290 0.000314
7 49.85 10.57 7.805606 ... 0.017930 0.014378 -0.000026 -0.000013
8 49.85 10.57 7.805606 ... 0.017930 0.014378 -0.000026 -0.000013
9 50.44 301.74 10.348327 ... 0.017620 0.014725 -0.000358 -0.000301

gyro_z reset humidity_int humidity_counts lat lon
0 0.000046 0 45 32730 31.434741 52.917464
1 0.000034 0 45 32730 -46.620658 -57.311657
2 0.000034 0 45 32730 -46.620477 -57.311138
3 0.000221 0 47 14176 19.138359 -140.211489
4 0.000030 0 47 14176 4.713819 80.261665
5 0.000171 0 46 35775 -46.061583 22.246025
6 -0.000375 0 46 35775 47.047346 137.958918
7 -0.000047 0 45 32730 -41.049112 30.193004
8 -0.000047 0 45 32730 -8.402991 -100.981726
9 -0.000061 0 45 32730 50.047523 175.566751

[10 rows x 23 columns]
```

### Exercise 2.1 - better merge

If you notice, above table does have `lat` and `lon` columns, but has very few rows. Why ? Try to merge the tables in some meaningful way so to have all the original rows and all cells of `lat` and `lon` filled.

- For other merging strategies, read about attribute `how` in [Why And How To Use Merge With Pandas in Python](#)<sup>393</sup>
- To fill missing values don't use fancy interpolation techniques, just put the station position in that given day or hour

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[28]: # write here

geo_astropi = df.merge(iss_coords, left_on='time_stamp', right_on='timestamp', how=
 ↪'left')

pd.merge_ordered(df, iss_coords, fill_method='ffill', how='left', left_on='time_stamp'
 ↪', right_on='timestamp')
geo_astropi
```

	time_stamp	temp_cpu	temp_h	temp_p	humidity	pressure	\		
0	2016-02-16 10:44:40	31.88	27.57	25.01	44.94	1001.68			
1	2016-02-16 10:44:50	31.79	27.53	25.01	45.12	1001.72			
2	2016-02-16 10:45:00	31.66	27.53	25.01	45.12	1001.72			
3	2016-02-16 10:45:10	31.69	27.52	25.01	45.32	1001.69			
4	2016-02-16 10:45:20	31.66	27.54	25.01	45.18	1001.71			
...	...	...	...	...	...	...			
110866	2016-02-29 09:24:21	31.56	27.52	24.83	42.94	1005.83			
110867	2016-02-29 09:24:30	31.55	27.50	24.83	42.72	1005.85			
110868	2016-02-29 09:24:41	31.58	27.50	24.83	42.83	1005.85			
110869	2016-02-29 09:24:50	31.62	27.50	24.83	42.81	1005.88			
110870	2016-02-29 09:25:00	31.57	27.51	24.83	42.94	1005.86			
	pitch	roll	yaw	mag_x	...	accel_z	gyro_x	gyro_y	\
0	1.49	52.25	185.21	-46.422753	...	0.014569	0.000942	0.000492	
1	1.03	53.73	186.72	-48.778951	...	0.014577	0.000218	-0.000005	
2	1.24	53.57	186.21	-49.161878	...	0.014357	0.000395	0.000600	
3	1.57	53.63	186.03	-49.341941	...	0.014409	0.000308	0.000577	
4	0.85	53.66	186.46	-50.056683	...	0.014380	0.000321	0.000691	
...	...	...	...	...	...	...	...	...	
110866	1.58	49.93	129.60	-15.169673	...	0.014646	-0.000264	0.000206	
110867	1.89	49.92	130.51	-15.832622	...	0.014855	0.000143	0.000199	
110868	2.09	50.00	132.04	-16.646212	...	0.014799	0.000537	0.000257	
110869	2.88	49.69	133.00	-17.270447	...	0.014877	0.000534	0.000456	
110870	2.17	49.77	134.18	-17.885872	...	0.014380	0.000459	0.000076	
	gyro_z	reset	humidity_int	humidity_counts	timestamp	lat	lon		
0	-0.000750	20	44	13029		NaN	NaN	NaN	
1	-0.000235	0	45	32730		NaN	NaN	NaN	
2	-0.000003	0	45	32730		NaN	NaN	NaN	
3	-0.000102	0	45	32730		NaN	NaN	NaN	
4	0.000272	0	45	32730		NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	
110866	0.000196	0	42	2776		NaN	NaN	NaN	
110867	-0.000024	0	42	2776		NaN	NaN	NaN	
110868	0.000057	0	42	2776		NaN	NaN	NaN	

(continues on next page)

<sup>393</sup> <https://towardsdatascience.com/why-and-how-to-use-merge-with-pandas-in-python-548600f7e738>

(continued from previous page)

110869	0.000195	0	42	2776	NaN	NaN	NaN
110870	0.000030	0	42	2776	NaN	NaN	NaN

[110871 rows x 24 columns]

&lt;/div&gt;

```
[28]: # write here
```

	time_stamp	temp_cpu	temp_h	temp_p	humidity	pressure	\		
0	2016-02-16 10:44:40	31.88	27.57	25.01	44.94	1001.68			
1	2016-02-16 10:44:50	31.79	27.53	25.01	45.12	1001.72			
2	2016-02-16 10:45:00	31.66	27.53	25.01	45.12	1001.72			
3	2016-02-16 10:45:10	31.69	27.52	25.01	45.32	1001.69			
4	2016-02-16 10:45:20	31.66	27.54	25.01	45.18	1001.71			
...	...	...	...	...	...	...			
110866	2016-02-29 09:24:21	31.56	27.52	24.83	42.94	1005.83			
110867	2016-02-29 09:24:30	31.55	27.50	24.83	42.72	1005.85			
110868	2016-02-29 09:24:41	31.58	27.50	24.83	42.83	1005.85			
110869	2016-02-29 09:24:50	31.62	27.50	24.83	42.81	1005.88			
110870	2016-02-29 09:25:00	31.57	27.51	24.83	42.94	1005.86			
	pitch	roll	yaw	mag_x	...	accel_z	gyro_x	gyro_y	\
0	1.49	52.25	185.21	-46.422753	...	0.014569	0.000942	0.000492	
1	1.03	53.73	186.72	-48.778951	...	0.014577	0.000218	-0.000005	
2	1.24	53.57	186.21	-49.161878	...	0.014357	0.000395	0.000600	
3	1.57	53.63	186.03	-49.341941	...	0.014409	0.000308	0.000577	
4	0.85	53.66	186.46	-50.056683	...	0.014380	0.000321	0.000691	
...	...	...	...	...	...	...	...	...	
110866	1.58	49.93	129.60	-15.169673	...	0.014646	-0.000264	0.000206	
110867	1.89	49.92	130.51	-15.832622	...	0.014855	0.000143	0.000199	
110868	2.09	50.00	132.04	-16.646212	...	0.014799	0.000537	0.000257	
110869	2.88	49.69	133.00	-17.270447	...	0.014877	0.000534	0.000456	
110870	2.17	49.77	134.18	-17.885872	...	0.014380	0.000459	0.000076	
	gyro_z	reset	humidity_int	humidity_counts	timestamp	lat	lon		
0	-0.000750	20	44	13029		NaN	NaN	NaN	
1	-0.000235	0	45	32730		NaN	NaN	NaN	
2	-0.000003	0	45	32730		NaN	NaN	NaN	
3	-0.000102	0	45	32730		NaN	NaN	NaN	
4	0.000272	0	45	32730		NaN	NaN	NaN	
...	...	...	...	...	...	...	...		
110866	0.000196	0	42	2776		NaN	NaN	NaN	
110867	-0.000024	0	42	2776		NaN	NaN	NaN	
110868	0.000057	0	42	2776		NaN	NaN	NaN	
110869	0.000195	0	42	2776		NaN	NaN	NaN	
110870	0.000030	0	42	2776		NaN	NaN	NaN	

[110871 rows x 24 columns]

### 3. GeoPandas

You can easily manipulate geographical data with [GeoPandas<sup>394</sup>](#) library. For some nice online tutorial, we refer to [Geospatial Analysis and Representation for Data Science<sup>395</sup>](#) course website @ master in Data Science University of Trento, by Maurizio Napolitano (FBK)

#### Continue

Go on with the [challenges<sup>396</sup>](#) worksheet

## 8.3.3 Pandas - 3. The Land of Poets Challenge

#### Download exercises

For a digital humanities project you need to display Italian poets by filtering a csv table according to various criteria. This challenge will be only about querying with pandas, which is something you might find convenient to do during exams for quickly understanding datasets content (using pandas will always be optional, you will never be asked to perform complex modifications with it)

You are given a dataset taken from [Wikidata<sup>397</sup>](#), a project by the Wikimedia foundation which aims to store only machine-readable data, like numbers, strings, and so on interlinked with many references. Each entity in Wikidata has an identifier, for example Dante Alighieri is the entity [Q1067<sup>398</sup>](#) and Florence is [Q2044<sup>399</sup>](#)

Wikidata can be queried using the SPARQL language: the data was obtained with [this query<sup>400</sup>](#) and downloaded in CSV format (among the many which can be chosen). Even if not necessary for the purposes of the exercise, you are invited to play a bit with the interface, like trying different visualizations (i.e. try select map in the middle-left corner) - or see other examples<sup>401</sup>

#### What to do

1. If you haven't already, install Pandas:

Anaconda:

```
conda install pandas
```

Without Anaconda (--user installs in your home):

```
python3 -m pip install --user pandas
```

2. unzip exercises in a folder, you should get something like this:

<sup>394</sup> <https://geopandas.org/>

<sup>395</sup> [https://napo.github.io/geospatial\\_course\\_uniin/](https://napo.github.io/geospatial_course_uniin/)

<sup>396</sup> <https://en.softpython.org/pandas/pandas3-chal.html>

<sup>397</sup> [https://www.wikidata.org/wiki/Wikidata:Main\\_Page](https://www.wikidata.org/wiki/Wikidata:Main_Page)

<sup>398</sup> <https://www.wikidata.org/wiki/Q1067>

<sup>399</sup> <https://www.wikidata.org/wiki/2044>

<sup>400</sup> [https://query.wikidata.org/#%23defaultView%3AMap%7B%22hide%22%3A%20%5B%22%3Fcoord%22%5D%7D%0ASELECT%20%3Fsubj%20%3FsubjLabel%20%3Fplace%20%3FplaceLabel%20%3Fcoord%20%3Fbirthyear%0AWHERE%20%7B%0A%20%20%20%3Fsubj%20wdt%3AP106%20wd%3AQ49757%20.%0A%20%20%20%3Fsubj%20wdt%3AP19%20%3Fplace%20.%0A%20%20%20%3Fplace%20wdt%3AP17%20wd%3AQ38%20.%0A%20%20%20%3Fplace%20wdt%3AP625%20%3Fcoord%20.%0A%20%20%20OPTIONAL%20%7B%20%3Fsubj%20wdt%3AP569%20%3Fdob%20%7D%0A%20%20%20BIND%28YEAR%28%3Fdob%29%20as%20%20%3Fbirthyear%29%0ASERVICE%20wikibase%3Alabel%20%7B%20%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO\\_LANGUAGE%5D%2Cen%22%20%7D%0A%7D](https://query.wikidata.org/#%23defaultView%3AMap%7B%22hide%22%3A%20%5B%22%3Fcoord%22%5D%7D%0ASELECT%20%3Fsubj%20%3FsubjLabel%20%3Fplace%20%3FplaceLabel%20%3Fcoord%20%3Fbirthyear%0AWHERE%20%7B%0A%20%20%20%3Fsubj%20wdt%3AP106%20wd%3AQ49757%20.%0A%20%20%20%3Fsubj%20wdt%3AP19%20%3Fplace%20.%0A%20%20%20%3Fplace%20wdt%3AP17%20wd%3AQ38%20.%0A%20%20%20%3Fplace%20wdt%3AP625%20%3Fcoord%20.%0A%20%20%20OPTIONAL%20%7B%20%3Fsubj%20wdt%3AP569%20%3Fdob%20%7D%0A%20%20%20BIND%28YEAR%28%3Fdob%29%20as%20%20%3Fbirthyear%29%0ASERVICE%20wikibase%3Alabel%20%7B%20%20bd%3AserviceParam%20wikibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22%20%7D%0A%7D)

<sup>401</sup> [https://www.wikidata.org/wiki/Wikidata:SPARQL\\_query\\_service/queries/examples](https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples)

```
pandas
 pandas1-sol.ipynb
 pandas1.ipynb
 pandas2-sol.ipynb
 pandas2.ipynb
 pandas3-chal.ipynb
jupman.py
```

**WARNING 1:** to correctly visualize the notebook, it MUST be in an unzipped folder !

3. open Jupyter Notebook from that folder. Two things should open, first a console and then browser.
4. The browser should show a file list: navigate the list and open the notebook `pandas3-chal.ipynb`

**WARNING 2:** DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

5. Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### Load the dataset

First load the dataset `italian-poets.csv` in pandas dataframe `df`

- USE UTF-8 as encoding

```
[1]: # write here
```

### Tell me more

Show some info about the dataset

```
[2]: # write here
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3198 entries, 0 to 3197
Data columns (total 6 columns):
 # Column Non-Null Count Dtype
--- -- ----- ----
 0 subj 3198 non-null object
 1 subjLabel 3198 non-null object
```

(continues on next page)

(continued from previous page)

```

2 place 3198 non-null object
3 placeLabel 3198 non-null object
4 coord 3198 non-null object
5 birthyear 3070 non-null float64
dtypes: float64(1), object(5)
memory usage: 150.0+ KB

```

## Getting in shape

Show the rows and the columns counts:

```
[3]: # write here

rows: 3198
columns: 6
```

## 10 rows

Display first 10 rows

```
[4]: # write here

[4]: subj subjLabel \
0 http://www.wikidata.org/entity/Q8797 Aemilius Macer
1 http://www.wikidata.org/entity/Q8833 Gaius Maecenas
2 http://www.wikidata.org/entity/Q5592 Michelangelo
3 http://www.wikidata.org/entity/Q6197 Horace
4 http://www.wikidata.org/entity/Q7170 Sallust
5 http://www.wikidata.org/entity/Q7198 Ovid
6 http://www.wikidata.org/entity/Q7728 Grazia Deledda
7 http://www.wikidata.org/entity/Q7803 Bronzino
8 http://www.wikidata.org/entity/Q8796 Sandra Lombardi
9 http://www.wikidata.org/entity/Q8800 Gaius Maecenas Melissus

 place placeLabel \
0 http://www.wikidata.org/entity/Q2028 Verona
1 http://www.wikidata.org/entity/Q13378 Arezzo
2 http://www.wikidata.org/entity/Q52069 Caprese Michelangelo
3 http://www.wikidata.org/entity/Q52691 Venosa
4 http://www.wikidata.org/entity/Q177061 Amiternum
5 http://www.wikidata.org/entity/Q50157 Sulmona
6 http://www.wikidata.org/entity/Q13649 Nuoro
7 http://www.wikidata.org/entity/Q2044 Florence
8 http://www.wikidata.org/entity/Q220 Rome
9 http://www.wikidata.org/entity/Q20571 Spoleto

 coord birthyear
0 Point(10.992777777 45.438611111) NaN
1 Point(11.878055555 43.463055555) NaN
2 Point(11.985833333 43.640833333) 1475.0
```

(continues on next page)

(continued from previous page)

```
3 Point(15.816666666 40.966666666) -64.0
4 Point(13.305769 42.400776) -85.0
5 Point(13.926198 42.048025) -42.0
6 Point(9.3280792 40.3200621) 1871.0
7 Point(11.254166666 43.771388888) 1503.0
8 Point(12.482777777 41.893055555) 1946.0
9 Point(12.733333333 42.733333333) -100.0
```

## Born in Verona

Display all people born in Verona

```
[5]: # write here
```

```
[5]: subj \
0 http://www.wikidata.org/entity/Q8797
135 http://www.wikidata.org/entity/Q163079
232 http://www.wikidata.org/entity/Q318593
256 http://www.wikidata.org/entity/Q539577
375 http://www.wikidata.org/entity/Q1236766
436 http://www.wikidata.org/entity/Q620193
755 http://www.wikidata.org/entity/Q2293943
764 http://www.wikidata.org/entity/Q1587432
858 http://www.wikidata.org/entity/Q3290043
891 http://www.wikidata.org/entity/Q3611735
1035 http://www.wikidata.org/entity/Q3638918
1090 http://www.wikidata.org/entity/Q3663490
1098 http://www.wikidata.org/entity/Q3665350
1143 http://www.wikidata.org/entity/Q3741666
1169 http://www.wikidata.org/entity/Q3746475
1393 http://www.wikidata.org/entity/Q3762433
1459 http://www.wikidata.org/entity/Q3766734
1489 http://www.wikidata.org/entity/Q3767945
1566 http://www.wikidata.org/entity/Q3768974
1694 http://www.wikidata.org/entity/Q4015300
1771 http://www.wikidata.org/entity/Q3081061
1869 http://www.wikidata.org/entity/Q3837018
1935 http://www.wikidata.org/entity/Q3846323
2211 http://www.wikidata.org/entity/Q6999870
2322 http://www.wikidata.org/entity/Q15432608
2361 http://www.wikidata.org/entity/Q15726796
2390 http://www.wikidata.org/entity/Q16574305
2460 http://www.wikidata.org/entity/Q17341090
2530 http://www.wikidata.org/entity/Q18945280
2531 http://www.wikidata.org/entity/Q18945373
2617 http://www.wikidata.org/entity/Q19597229
2634 http://www.wikidata.org/entity/Q20671732
2651 http://www.wikidata.org/entity/Q23014868
2841 http://www.wikidata.org/entity/Q30126093
2850 http://www.wikidata.org/entity/Q30303339
2872 http://www.wikidata.org/entity/Q28778065
2885 http://www.wikidata.org/entity/Q30308589
```

(continues on next page)

(continued from previous page)

	subjLabel \
0	Aemilius Macer
135	Catullus
232	Girolamo Fracastoro
256	Guarino da Verona
375	Ippolito Pindemonte
436	Aleardo Aleardi
755	Cristina Ali Farah
764	Francesco Scipione, marchese di Maffei
858	Marco Antonio Zucchi
891	Alida Airaghi
1035	Berto Barbarani
1090	Caterina Bon Brenzoni
1098	Cesare Betteloni
1143	Federico Ceruti
1169	Flavio Ermini
1393	Giambattista Spolverini
1459	Giovanni Battista Pighi
1489	Giovanni Pindemonte
1566	Girolamo Pompei
1694	Vittorio Betteloni
1771	Francesco Pona
1869	Lorenzo Montano
1935	Marco Ongaro
2211	Rudy De Cadaval
2322	Ortensio Mauro
2361	Teresa Albarelli
2390	Luigi Nogarola
2460	Giovanni Ceriotto
2530	Francesco degli Allegri
2531	Giorgio Summaripa
2617	Giambattista Mutinelli
2634	Pietro Caliari
2651	Ilario Casarotti
2841	Girolamo Orti Manara
2850	Paolo Zazzaroni
2872	Angela Nogarola
2885	Bartolomeo Tortoletti
	place placeLabel \
0	http://www.wikidata.org/entity/Q2028 Verona
135	http://www.wikidata.org/entity/Q2028 Verona
232	http://www.wikidata.org/entity/Q2028 Verona
256	http://www.wikidata.org/entity/Q2028 Verona
375	http://www.wikidata.org/entity/Q2028 Verona
436	http://www.wikidata.org/entity/Q2028 Verona
755	http://www.wikidata.org/entity/Q2028 Verona
764	http://www.wikidata.org/entity/Q2028 Verona
858	http://www.wikidata.org/entity/Q2028 Verona
891	http://www.wikidata.org/entity/Q2028 Verona
1035	http://www.wikidata.org/entity/Q2028 Verona
1090	http://www.wikidata.org/entity/Q2028 Verona
1098	http://www.wikidata.org/entity/Q2028 Verona
1143	http://www.wikidata.org/entity/Q2028 Verona
1169	http://www.wikidata.org/entity/Q2028 Verona
1393	http://www.wikidata.org/entity/Q2028 Verona
1459	http://www.wikidata.org/entity/Q2028 Verona

(continues on next page)

(continued from previous page)

1489	http://www.wikidata.org/entity/Q2028	Verona
1566	http://www.wikidata.org/entity/Q2028	Verona
1694	http://www.wikidata.org/entity/Q2028	Verona
1771	http://www.wikidata.org/entity/Q2028	Verona
1869	http://www.wikidata.org/entity/Q2028	Verona
1935	http://www.wikidata.org/entity/Q2028	Verona
2211	http://www.wikidata.org/entity/Q2028	Verona
2322	http://www.wikidata.org/entity/Q2028	Verona
2361	http://www.wikidata.org/entity/Q2028	Verona
2390	http://www.wikidata.org/entity/Q2028	Verona
2460	http://www.wikidata.org/entity/Q2028	Verona
2530	http://www.wikidata.org/entity/Q2028	Verona
2531	http://www.wikidata.org/entity/Q2028	Verona
2617	http://www.wikidata.org/entity/Q2028	Verona
2634	http://www.wikidata.org/entity/Q2028	Verona
2651	http://www.wikidata.org/entity/Q2028	Verona
2841	http://www.wikidata.org/entity/Q2028	Verona
2850	http://www.wikidata.org/entity/Q2028	Verona
2872	http://www.wikidata.org/entity/Q2028	Verona
2885	http://www.wikidata.org/entity/Q2028	Verona

		coord	birthyear
0	Point(10.992777777 45.438611111)	NAN	
135	Point(10.992777777 45.438611111)	-83.0	
232	Point(10.992777777 45.438611111)	1478.0	
256	Point(10.992777777 45.438611111)	1374.0	
375	Point(10.992777777 45.438611111)	1753.0	
436	Point(10.992777777 45.438611111)	1812.0	
755	Point(10.992777777 45.438611111)	1973.0	
764	Point(10.992777777 45.438611111)	1675.0	
858	Point(10.992777777 45.438611111)	1750.0	
891	Point(10.992777777 45.438611111)	1953.0	
1035	Point(10.992777777 45.438611111)	1872.0	
1090	Point(10.992777777 45.438611111)	1813.0	
1098	Point(10.992777777 45.438611111)	1808.0	
1143	Point(10.992777777 45.438611111)	1532.0	
1169	Point(10.992777777 45.438611111)	1947.0	
1393	Point(10.992777777 45.438611111)	1695.0	
1459	Point(10.992777777 45.438611111)	1898.0	
1489	Point(10.992777777 45.438611111)	1751.0	
1566	Point(10.992777777 45.438611111)	1731.0	
1694	Point(10.992777777 45.438611111)	1840.0	
1771	Point(10.992777777 45.438611111)	1595.0	
1869	Point(10.992777777 45.438611111)	1893.0	
1935	Point(10.992777777 45.438611111)	1956.0	
2211	Point(10.992777777 45.438611111)	1933.0	
2322	Point(10.992777777 45.438611111)	1634.0	
2361	Point(10.992777777 45.438611111)	1788.0	
2390	Point(10.992777777 45.438611111)	1669.0	
2460	Point(10.992777777 45.438611111)	1883.0	
2530	Point(10.992777777 45.438611111)	1495.0	
2531	Point(10.992777777 45.438611111)	1435.0	
2617	Point(10.992777777 45.438611111)	1747.0	
2634	Point(10.992777777 45.438611111)	1841.0	
2651	Point(10.992777777 45.438611111)	1772.0	
2841	Point(10.992777777 45.438611111)	1769.0	
2850	Point(10.992777777 45.438611111)	NAN	

(continues on next page)

(continued from previous page)

2872	Point(10.992777777 45.438611111)	1380.0
2885	Point(10.992777777 45.438611111)	1560.0

## How many people in Verona

Display how many people were born in Verona

```
[6]: # write here
```

```
[6]: 37
```

## Python is everywhere

Show poets born in Catania in the year -500

- mind the minus
- I swear we did not altered the dataset in any way :-)

```
[7]: # write here
```

```
[7]: subj subjLabel \
2231 http://www.wikidata.org/entity/Q7263938 Python of Catana

 place placeLabel \
2231 http://www.wikidata.org/entity/Q1903 Catania

 coord birthyear
2231 Point(15.087269444 37.502669444) -500.0
```

## Verona after 1500

Display all people born in Verona after the year 1500

```
[8]: # write here
```

```
[8]: subj \
375 http://www.wikidata.org/entity/Q1236766
436 http://www.wikidata.org/entity/Q620193
755 http://www.wikidata.org/entity/Q2293943
764 http://www.wikidata.org/entity/Q1587432
858 http://www.wikidata.org/entity/Q3290043
891 http://www.wikidata.org/entity/Q3611735
1035 http://www.wikidata.org/entity/Q3638918
1090 http://www.wikidata.org/entity/Q3663490
1098 http://www.wikidata.org/entity/Q3665350
1143 http://www.wikidata.org/entity/Q3741666
1169 http://www.wikidata.org/entity/Q3746475
```

(continues on next page)

(continued from previous page)

```

1393 http://www.wikidata.org/entity/Q3762433
1459 http://www.wikidata.org/entity/Q3766734
1489 http://www.wikidata.org/entity/Q3767945
1566 http://www.wikidata.org/entity/Q3768974
1694 http://www.wikidata.org/entity/Q4015300
1771 http://www.wikidata.org/entity/Q3081061
1869 http://www.wikidata.org/entity/Q3837018
1935 http://www.wikidata.org/entity/Q3846323
2211 http://www.wikidata.org/entity/Q6999870
2322 http://www.wikidata.org/entity/Q15432608
2361 http://www.wikidata.org/entity/Q15726796
2390 http://www.wikidata.org/entity/Q16574305
2460 http://www.wikidata.org/entity/Q17341090
2617 http://www.wikidata.org/entity/Q19597229
2634 http://www.wikidata.org/entity/Q20671732
2651 http://www.wikidata.org/entity/Q23014868
2841 http://www.wikidata.org/entity/Q30126093
2885 http://www.wikidata.org/entity/Q30308589

```

```

 subjLabel \
375 Ippolito Pindemonte
436 Aleardo Aleardi
755 Cristina Ali Farah
764 Francesco Scipione, marchese di Maffei
858 Marco Antonio Zucchi
891 Alida Airaghi
1035 Berto Barbarani
1090 Caterina Bon Brenzoni
1098 Cesare Betteloni
1143 Federico Ceruti
1169 Flavio Ermini
1393 Giambattista Spolverini
1459 Giovanni Battista Pighi
1489 Giovanni Pindemonte
1566 Girolamo Pompei
1694 Vittorio Betteloni
1771 Francesco Pona
1869 Lorenzo Montano
1935 Marco Ongaro
2211 Rudy De Cadaval
2322 Ortensio Mauro
2361 Teresa Albarelli
2390 Luigi Nogarola
2460 Giovanni Ceriotto
2617 Giambattista Mutinelli
2634 Pietro Caliari
2651 Ilario Casarotti
2841 Girolamo Orti Manara
2885 Bartolomeo Tortoletti

```

```

 place placeLabel \
375 http://www.wikidata.org/entity/Q2028 Verona
436 http://www.wikidata.org/entity/Q2028 Verona
755 http://www.wikidata.org/entity/Q2028 Verona
764 http://www.wikidata.org/entity/Q2028 Verona
858 http://www.wikidata.org/entity/Q2028 Verona
891 http://www.wikidata.org/entity/Q2028 Verona

```

(continues on next page)

(continued from previous page)

1035	http://www.wikidata.org/entity/Q2028	Verona
1090	http://www.wikidata.org/entity/Q2028	Verona
1098	http://www.wikidata.org/entity/Q2028	Verona
1143	http://www.wikidata.org/entity/Q2028	Verona
1169	http://www.wikidata.org/entity/Q2028	Verona
1393	http://www.wikidata.org/entity/Q2028	Verona
1459	http://www.wikidata.org/entity/Q2028	Verona
1489	http://www.wikidata.org/entity/Q2028	Verona
1566	http://www.wikidata.org/entity/Q2028	Verona
1694	http://www.wikidata.org/entity/Q2028	Verona
1771	http://www.wikidata.org/entity/Q2028	Verona
1869	http://www.wikidata.org/entity/Q2028	Verona
1935	http://www.wikidata.org/entity/Q2028	Verona
2211	http://www.wikidata.org/entity/Q2028	Verona
2322	http://www.wikidata.org/entity/Q2028	Verona
2361	http://www.wikidata.org/entity/Q2028	Verona
2390	http://www.wikidata.org/entity/Q2028	Verona
2460	http://www.wikidata.org/entity/Q2028	Verona
2617	http://www.wikidata.org/entity/Q2028	Verona
2634	http://www.wikidata.org/entity/Q2028	Verona
2651	http://www.wikidata.org/entity/Q2028	Verona
2841	http://www.wikidata.org/entity/Q2028	Verona
2885	http://www.wikidata.org/entity/Q2028	Verona
	coord	birthyear
375	Point(10.992777777 45.438611111)	1753.0
436	Point(10.992777777 45.438611111)	1812.0
755	Point(10.992777777 45.438611111)	1973.0
764	Point(10.992777777 45.438611111)	1675.0
858	Point(10.992777777 45.438611111)	1750.0
891	Point(10.992777777 45.438611111)	1953.0
1035	Point(10.992777777 45.438611111)	1872.0
1090	Point(10.992777777 45.438611111)	1813.0
1098	Point(10.992777777 45.438611111)	1808.0
1143	Point(10.992777777 45.438611111)	1532.0
1169	Point(10.992777777 45.438611111)	1947.0
1393	Point(10.992777777 45.438611111)	1695.0
1459	Point(10.992777777 45.438611111)	1898.0
1489	Point(10.992777777 45.438611111)	1751.0
1566	Point(10.992777777 45.438611111)	1731.0
1694	Point(10.992777777 45.438611111)	1840.0
1771	Point(10.992777777 45.438611111)	1595.0
1869	Point(10.992777777 45.438611111)	1893.0
1935	Point(10.992777777 45.438611111)	1956.0
2211	Point(10.992777777 45.438611111)	1933.0
2322	Point(10.992777777 45.438611111)	1634.0
2361	Point(10.992777777 45.438611111)	1788.0
2390	Point(10.992777777 45.438611111)	1669.0
2460	Point(10.992777777 45.438611111)	1883.0
2617	Point(10.992777777 45.438611111)	1747.0
2634	Point(10.992777777 45.438611111)	1841.0
2651	Point(10.992777777 45.438611111)	1772.0
2841	Point(10.992777777 45.438611111)	1769.0
2885	Point(10.992777777 45.438611111)	1560.0

## First Antonio

Display all people with Antonio as first name

```
[9]: # write here
```

```
[9]:
```

	subj	subjLabel \
47	http://www.wikidata.org/entity/Q266482	Antonio Bonfini
48	http://www.wikidata.org/entity/Q266482	Antonio Bonfini
77	http://www.wikidata.org/entity/Q348311	Antonio Tebaldeo
120	http://www.wikidata.org/entity/Q470067	Antonio Fogazzaro
203	http://www.wikidata.org/entity/Q524960	Antonio Ghislanzoni
...	...	...
2881	http://www.wikidata.org/entity/Q30250615	Antonio Bruni
2917	http://www.wikidata.org/entity/Q42941837	Antonio Decio
2979	http://www.wikidata.org/entity/Q56166956	Antonio Rossetti
3060	http://www.wikidata.org/entity/Q54860414	Antonio Ricci
3135	http://www.wikidata.org/entity/Q94075340	Antonio Gasparinetti
	place	placeLabel \
47	http://www.wikidata.org/entity/Q3415	Ancona
48	http://www.wikidata.org/entity/Q3897778	Patrignone
77	http://www.wikidata.org/entity/Q13362	Ferrara
120	http://www.wikidata.org/entity/Q6537	Vicenza
203	http://www.wikidata.org/entity/Q6237	Lecco
...	...	...
2881	http://www.wikidata.org/entity/Q52019	Manduria
2917	http://www.wikidata.org/entity/Q176180	Orte
2979	http://www.wikidata.org/entity/Q51313	Vasto
3060	http://www.wikidata.org/entity/Q51240	Guardiagrele
3135	http://www.wikidata.org/entity/Q46503	Ponte di Piave
	coord	birthyear
47	Point(13.516666666 43.616666666)	1427.0
48	Point(13.60926 42.98027)	1427.0
77	Point(11.619865 44.835297)	1463.0
120	Point(11.55 45.55)	1842.0
203	Point(9.4 45.85)	1824.0
...	...	...
2881	Point(17.634166666 40.402777777)	1593.0
2917	Point(12.386111111 42.460277777)	1560.0
2979	Point(14.708219444 42.111588888)	1770.0
3060	Point(14.221591666 42.189222222)	1952.0
3135	Point(12.466666666 45.716666666)	1777.0

[85 rows x 6 columns]

## Some Antonio

Display all people with Antonio as one of the names (so also include 'Paolo Antonio Rolli')

[10]: # write here

```
[10]: subj subjLabel \
47 http://www.wikidata.org/entity/Q266482 Antonio Bonfini
48 http://www.wikidata.org/entity/Q266482 Antonio Bonfini
53 http://www.wikidata.org/entity/Q55433 Michelangelo Antonioni
77 http://www.wikidata.org/entity/Q348311 Antonio Tebaldeo
120 http://www.wikidata.org/entity/Q470067 Antonio Fogazzaro
...
2906 http://www.wikidata.org/entity/Q41566775 Carlo Antonio Bertelli
2917 http://www.wikidata.org/entity/Q42941837 Antonio Decio
2979 http://www.wikidata.org/entity/Q56166956 Antonio Rossetti
3060 http://www.wikidata.org/entity/Q54860414 Antonio Ricci
3135 http://www.wikidata.org/entity/Q94075340 Antonio Gasparinetti

 place placeLabel \
47 http://www.wikidata.org/entity/Q3415 Ancona
48 http://www.wikidata.org/entity/Q3897778 Patrignone
53 http://www.wikidata.org/entity/Q13362 Ferrara
77 http://www.wikidata.org/entity/Q13362 Ferrara
120 http://www.wikidata.org/entity/Q6537 Vicenza
...
2906 http://www.wikidata.org/entity/Q111705 Salò
2917 http://www.wikidata.org/entity/Q176180 Orte
2979 http://www.wikidata.org/entity/Q51313 Vasto
3060 http://www.wikidata.org/entity/Q51240 Guardiagrele
3135 http://www.wikidata.org/entity/Q46503 Ponte di Piave

 coord birthyear
47 Point(13.516666666 43.616666666) 1427.0
48 Point(13.60926 42.98027) 1427.0
53 Point(11.619865 44.835297) 1912.0
77 Point(11.619865 44.835297) 1463.0
120 Point(11.55 45.55) 1842.0
...
2906 Point(10.533333333 45.6) 1637.0
2917 Point(12.386111111 42.460277777) 1560.0
2979 Point(14.708219444 42.111588888) 1770.0
3060 Point(14.221591666 42.189222222) 1952.0
3135 Point(12.466666666 45.716666666) 1777.0

[110 rows x 6 columns]
```

## Cesares during 1800

Display all people named Cesare who were born in 1800 century

```
[11]: # write here
```

```
[11]: subj subjLabel \
389 http://www.wikidata.org/entity/Q1056872 Cesare Meano
1098 http://www.wikidata.org/entity/Q3665350 Cesare Betteloni
1101 http://www.wikidata.org/entity/Q3665409 Cesare De Titta
1105 http://www.wikidata.org/entity/Q3665495 Cesare Pascarella

 place placeLabel \
389 http://www.wikidata.org/entity/Q495 Turin
1098 http://www.wikidata.org/entity/Q2028 Verona
1101 http://www.wikidata.org/entity/Q51292 Sant'Eusonio del Sangro
1105 http://www.wikidata.org/entity/Q220 Rome

 coord birthyear
389 Point(7.7 45.066666666) 1899.0
1098 Point(10.992777777 45.438611111) 1808.0
1101 Point(14.333333333 42.166666666) 1862.0
1105 Point(12.482777777 41.893055555) 1858.0
```

## The old ones

Show poets in year of birth order

- **DO NOT** include in the result NaN values

**HINT:** see `pd.notnull`<sup>402</sup>

```
[12]: # write here
```

```
[12]: subj subjLabel \
292 http://www.wikidata.org/entity/Q332797 Stesichorus
293 http://www.wikidata.org/entity/Q332802 Ibucus
327 http://www.wikidata.org/entity/Q336115 Theognis of Megara
84 http://www.wikidata.org/entity/Q125551 Parmenides
2575 http://www.wikidata.org/entity/Q20002641 Glaucus of Rhegion
...
3104 http://www.wikidata.org/entity/Q78162153 Q78162153
2989 http://www.wikidata.org/entity/Q58995193 Giovanni Bertoglio
2986 http://www.wikidata.org/entity/Q58308029 Gio Evan
2374 http://www.wikidata.org/entity/Q14922292 Q14922292
552 http://www.wikidata.org/entity/Q1151356 D.B.P.I.T.

 place placeLabel \
292 http://www.wikidata.org/entity/Q54614 Gioia Tauro
293 http://www.wikidata.org/entity/Q8471 Reggio Calabria
327 http://www.wikidata.org/entity/Q1457477 Megara Hyblaea
84 http://www.wikidata.org/entity/Q272968 Velia
```

(continues on next page)

<sup>402</sup> <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.notnull.html>

(continued from previous page)

```

2575 http://www.wikidata.org/entity/Q8471 Reggio Calabria
...
3104 http://www.wikidata.org/entity/Q13678 ...
2989 http://www.wikidata.org/entity/Q495 Agrigento
2986 http://www.wikidata.org/entity/Q19300 Turin
2374 http://www.wikidata.org/entity/Q80652 Molfetta
552 http://www.wikidata.org/entity/Q220 Battipaglia
 Rome

 coord birthyear
292 Point(15.9 38.433333333) -629.0
293 Point(15.65 38.114438888) -600.0
327 Point(15.18194444 37.20388889) -569.0
84 Point(15.154444444 40.159444444) -514.0
2575 Point(15.65 38.114438888) -500.0
...
3104 Point(13.576547222 37.311075) 1985.0
2989 Point(7.7 45.066666666) 1986.0
2986 Point(16.6 41.2) 1988.0
2374 Point(14.983333333 40.616666666) 1991.0
552 Point(12.482777777 41.893055555) 2000.0

```

[3070 rows x 6 columns]

## Cities of poets

Find the 5 cities with most poets, sorted from most to least.

- use groupby and sort\_values methods

```
[13]: # write here
```

```

[13]: placeLabel
Rome 198
Florence 165
Milan 121
Naples 113
Venice 94
Name: subj, dtype: int64

```

## Most duplicated poets

Find first 8 duplicated poets

```
[14]: # write here
```

```

[14]: subjLabel
Sosiphanes 4
Alojz Rebula 4
Eliseo Calenzio 4
Giambattista Andreini 4
Tommaso Grossi 3

```

(continues on next page)

(continued from previous page)

Giovanni della Casa	3
Giuseppe Carpani	3
Aulus Gellius	3
Name: subj, dtype: int64	

## All duplicated poets

Print the number of all duplicated poets

**NOTE:** a Series object has **only one** column, even if they look two (the apparent other is the index) - so if you have a Series object you don't need to specify a column

```
[15]: # write here
```

```
There are 118 duplicated poets
```

## Northern poets

Find all the poets born north of a given town. In other words, look for town latitude (the second coordinate in coords), print it, and then filter the table.

- **DO NOT** put constants like 46.5 in your code!
- **DO NOT** add new columns for longitude and latitude
- **NOTE:** coord column holds just simple strings!
- **HINT:** to get an element at a given numerical index *i* of a filtered Series (regardless of the original dataframe row index), you need to use .iloc[i] property - note the square brackets!

```
[]:
```

```
[16]: town = 'Bolzano'
#town = 'Trento'
```

```
write here
```

```
Latitude of Bolzano : 46.5
```

		subj	subjLabel	\
27	http://www.wikidata.org/entity/Q45105	Oswald von Wolkenstein		
41	http://www.wikidata.org/entity/Q122070	Simon Lemnius		
42	http://www.wikidata.org/entity/Q122070	Simon Lemnius		
88	http://www.wikidata.org/entity/Q137683	Mary de Rachewiltz		
583	http://www.wikidata.org/entity/Q873784	Ignaz Vincenz Zingerle		
636	http://www.wikidata.org/entity/Q1705031	Josef Kostner		
637	http://www.wikidata.org/entity/Q1705031	Josef Kostner		
770	http://www.wikidata.org/entity/Q1996716	Norbert Conrad Kaser		
1905	http://www.wikidata.org/entity/Q3839806	Luigi Maierón		
2152	http://www.wikidata.org/entity/Q4505559	Karl Ziegler		
2741	http://www.wikidata.org/entity/Q24073666	Fedele Demetz		
2939	http://www.wikidata.org/entity/Q55471982	Roberta Dapunt		

(continues on next page)

(continued from previous page)

2940	http://www.wikidata.org/entity/Q55471982		Roberta Dapunt
3128	http://www.wikidata.org/entity/Q95185585		Anna Katharina Mair
		place	placeLabel \
27	http://www.wikidata.org/entity/Q1013962		Schöneck Castle
41	http://www.wikidata.org/entity/Q257965		Val Müstair
42	http://www.wikidata.org/entity/Q257965		Val Müstair
88	http://www.wikidata.org/entity/Q185541		Brixen
583	http://www.wikidata.org/entity/Q131605		Merano
636	http://www.wikidata.org/entity/Q255568		Urtijëi
637	http://www.wikidata.org/entity/Q255568		Urtijëi
770	http://www.wikidata.org/entity/Q185541		Brixen
1905	http://www.wikidata.org/entity/Q53240		Cercivento
2152	http://www.wikidata.org/entity/Q504216	St. Martin in Passeier	
2741	http://www.wikidata.org/entity/Q499129		Sëlva
2939	http://www.wikidata.org/entity/Q644159		Val Badia
2940	http://www.wikidata.org/entity/Q644159		Val Badia
3128	http://www.wikidata.org/entity/Q185541		Brixen
		coord birthyear	
27	Point(11.847977 46.820264)	1377.0	
41	Point(10.39009 46.60566)	1511.0	
42	Point(10.42 46.6)	1511.0	
88	Point(11.65 46.716666666)	1925.0	
583	Point(11.163888888 46.668888888)	1825.0	
636	Point(11.666666666 46.566666666)	1933.0	
637	Point(11.66748 46.57432)	1933.0	
770	Point(11.65 46.716666666)	1947.0	
1905	Point(12.983333333 46.533333333)	1954.0	
2152	Point(11.22727 46.78392)	1812.0	
2741	Point(11.76038 46.55472)	1850.0	
2939	Point(11.89917 46.68361)	1970.0	
2940	Point(11.9 46.683333333)	1970.0	
3128	Point(11.65 46.716666666)	1967.0	

## Papers please

Extract subject id (i.e. Q8797) and place id (i.e. Q2028) and MODIFY df by putting them into two new columns subj\_id and place\_id

[17]: # write here

[17]:		subj	subjLabel \
0	http://www.wikidata.org/entity/Q8797		Aemilius Macer
1	http://www.wikidata.org/entity/Q8833		Gaius Maecenas
2	http://www.wikidata.org/entity/Q5592		Michelangelo
3	http://www.wikidata.org/entity/Q6197		Horace
4	http://www.wikidata.org/entity/Q7170		Sallust
...		...	...
3193	http://www.wikidata.org/entity/Q99308713		Giovanni Marrasio
3194	http://www.wikidata.org/entity/Q100775377	Annemarie Innerebner	
3195	http://www.wikidata.org/entity/Q100775410	Nesti Lyro Wollek	
3196	http://www.wikidata.org/entity/Q99655533	Lorenza Meletti	

(continues on next page)

(continued from previous page)

```

3197 http://www.wikidata.org/entity/Q99912619 Nanni Falconi
 place placeLabel \
0 http://www.wikidata.org/entity/Q2028 Verona
1 http://www.wikidata.org/entity/Q13378 Arezzo
2 http://www.wikidata.org/entity/Q52069 Caprese Michelangelo
3 http://www.wikidata.org/entity/Q52691 Venosa
4 http://www.wikidata.org/entity/Q177061 Amiternum
...
3193 http://www.wikidata.org/entity/Q487174 ...
3194 http://www.wikidata.org/entity/Q6526 Noto
3195 http://www.wikidata.org/entity/Q2044 Bolzano
3196 http://www.wikidata.org/entity/Q95093 Florence
3197 http://www.wikidata.org/entity/Q391218 Bondeno
3197 http://www.wikidata.org/entity/Q391218 Pattada

 coord birthyear subj_id place_id
0 Point(10.992777777 45.438611111) NaN Q8797 Q2028
1 Point(11.878055555 43.463055555) NaN Q8833 Q13378
2 Point(11.985833333 43.640833333) 1475.0 Q5592 Q52069
3 Point(15.816666666 40.966666666) -64.0 Q6197 Q52691
4 Point(13.305769 42.400776) -85.0 Q7170 Q177061
...
3193 Point(15.083333333 36.883333333) 1405.0 Q99308713 Q487174
3194 Point(11.35 46.5) 1924.0 Q100775377 Q6526
3195 Point(11.254166666 43.771388888) 1875.0 Q100775410 Q2044
3196 Point(11.41542 44.88944) 1940.0 Q99655533 Q95093
3197 Point(9.11 40.582222222) 1950.0 Q99912619 Q391218

```

[3198 rows x 8 columns]

## Unknown poets

Find all the ids of nameless poets and put them in a **python list**.

- **DO NOT** use loops
- **NOTE** a Series object from the point of view of Python is just a sequence

[18]: # write here

```
[18]: ['Q4360247',
 'Q14922292',
 'Q19130448',
 'Q21207901',
 'Q19984452',
 'Q21209119',
 'Q21282215',
 'Q23673492',
 'Q29049430',
 'Q29052339',
 'Q31763467',
 'Q28465822',
 'Q48809843',
 'Q27553577',
```

(continues on next page)

(continued from previous page)

```
'Q48811051',
'Q48861610',
'Q55441810',
'Q47468550',
'Q50327630',
'Q50330028',
'Q55897192',
'Q65019765',
'Q51845316',
'Q60838260',
'Q64433131',
'Q71684946',
'Q93338246',
'Q59187521',
'Q61136330',
'Q61450547',
'Q52107491',
'Q61790603',
'Q61791394',
'Q61827513',
'Q61895377',
'Q59851133',
'Q59851150',
'Q62066746',
'Q66736238',
'Q66921487',
'Q85421610',
'Q61080035',
'Q87068357',
'Q64031897',
'Q64167386',
'Q64364409',
'Q69818426',
'Q64512266',
'Q78162153',
'Q78499894',
'Q88264630',
'Q89674973',
'Q94998318',
'Q94325725',
'Q84138681',
'Q98102965',
'Q80705985',
'Q81100287',
'Q81738068',
'Q83643534',
'Q83808244',
'Q96097742',
'Q96245247',
'Q96248786',
'Q95485499',
'Q99196947']
```

## Better unknown poets

Find all the ids, the birthplace and birthdate of nameless poets born after year 0, and put them in a **python list of tuples**.

- birthplaces must be integers - if not specified, put -1
- print also how many results were found
- **DO NOT** use loops nor list comprehensions

```
[19]: # write here
```

```
Found 66 results
```

```
[19]: [('Q4360247', 'Rome', 1907),
 ('Q14922292', 'Battipaglia', 1991),
 ('Q19130448', 'Vicenza', 1492),
 ('Q21207901', 'Aradeo', -1),
 ('Q19984452', 'Anghiari', -1),
 ('Q21209119', 'Giuliano Teatino', 1711),
 ('Q21282215', 'Palermo', 1590),
 ('Q23673492', 'Butera', -1),
 ('Q29049430', 'Fondi', -1),
 ('Q29052339', 'Taranto', 1733),
 ('Q31763467', 'Caltanissetta', 1755),
 ('Q28465822', 'Orvieto', 1700),
 ('Q48809843', 'Palermo', -1),
 ('Q27553577', 'Cavriana', 1250),
 ('Q48811051', 'Roccabernarda', 1550),
 ('Q48861610', 'Vittorio Veneto', 1452),
 ('Q55441810', 'Vernio', 1844),
 ('Q47468550', 'Florence', 1607),
 ('Q50327630', 'Rome', 1700),
 ('Q50330028', 'Rome', 1680),
 ('Q55897192', 'Narni', 1872),
 ('Q65019765', 'Afragola', 1853),
 ('Q51845316', 'Genoa', -1),
 ('Q60838260', 'Tuscany', -1),
 ('Q64433131', 'Castel Goffredo', -1),
 ('Q71684946', "Trezzo sull'Adda", -1),
 ('Q93338246', 'Viterbo', -1),
 ('Q59187521', 'Padua', 1850),
 ('Q61136330', 'Osilo', 1865),
 ('Q61450547', 'Siderno', 1847),
 ('Q52107491', 'Rome', 1680),
 ('Q61790603', 'Veneto', 1857),
 ('Q61791394', 'Piedimonte del Calvario', 1905),
 ('Q61827513', 'Grimacco', 1847),
 ('Q61895377', 'Como', 1731),
 ('Q59851133', 'Gorizia', 1883),
 ('Q59851150', 'Trieste', 1929),
 ('Q62066746', 'Bologna', 1926),
 ('Q66736238', 'Lucca', 1635),
 ('Q66921487', 'Alghero', 1914),
 ('Q85421610', 'Alghero', 1869),
 ('Q61080035', 'Greci', 1830),
 ('Q87068357', 'Polla', 1970),
 ('Q64031897', 'Castel Goffredo', 1445),
```

(continues on next page)

(continued from previous page)

```
('Q64167386', 'Sambiase', 1930),
('Q64364409', 'Corleone', 1752),
('Q69818426', 'Galati Mamertino', 1923),
('Q64512266', 'Rieti', 1890),
('Q78162153', 'Agrigento', 1985),
('Q78499894', 'San Giorgio a Cremano', 1932),
('Q88264630', 'Bari', 1930),
('Q89674973', 'Genoa', 1939),
('Q94998318', 'Trieste', 1939),
('Q94325725', 'Orune', 1908),
('Q84138681', 'Rome', -1),
('Q98102965', 'Florence', -1),
('Q80705985', 'Patti', 1879),
('Q81100287', 'Lecco', 1935),
('Q81738068', 'Rogolo', 1940),
('Q83643534', 'Alessandria', 1717),
('Q83808244', 'Cancello e Arnone', 1921),
('Q96097742', 'Milan', 1982),
('Q96245247', 'Lecce', 1919),
('Q96248786', 'Guardia Lombardi', 1970),
('Q95485499', 'Borgo San Martino', 1907),
('Q99196947', 'Tivoli', 1973)]
```

## 8.4 Relational data

### 8.4.1 Relational data 1 - introduction

#### Download exercises zip

Browse files online<sup>403</sup>

We live in a world of *relations* like John is *friend of* Paul or Mary *works at* CodeWizards. We can display them as networks or, as we call them in computer science, *graphs*. We will see some ways for storing graphs, like matrices, adjacency lists and also have a quick look at a specialized library called Networkx. Note in this book we limit ourselves to represent and manage relations with relatively simple programs, we won't deal with path exploration algorithms (no bfs, dfs etc) nor data formats specific for graphs (like rdf) which can get quite complex.

#### Required libraries

In order for visualizations to work, you need installed the python library networkx and pydot. Pydot is an interface to the non-python package GraphViz<sup>404</sup>.

##### Anaconda:

From Anaconda Prompt:

1. Install GraphViz:

```
conda install graphviz
```

2. Install python packages:

<sup>403</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/relational>

<sup>404</sup> <http://graphviz.org/>

```
conda install pydot networkx
```

### Ubuntu

From console:

1. Install PyGraphViz (note: you should use apt to install it, pip might give problems):

```
sudo apt-get install python3-pygraphviz
```

2. Install python packages:

```
python3 -m pip install --user pydot networkx
```

### What to do

- unzip exercises in a folder, you should get something like this:

```
relational
 relational1-intro.ipynb
 relational1-intro-sol.ipynb
 relational2-binrel.ipynb
 relational2-binrel-sol.ipynb
 relational3-simple-stats.ipynb
 relational3-simple-stats-sol.ipynb
 relational4-chal.ipynb
 jupman.py
 soft.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook relational/relational1-intro.ipynb

**WARNING 2:** DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## Graph definition

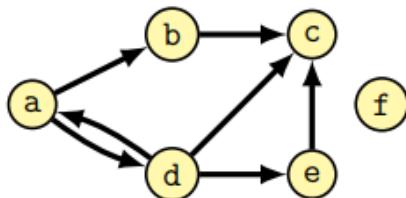
In computer science a *graph* is a set of vertices  $V$  (also called *nodes*) linked by a set of edges  $E$ . You can visualize nodes as circles and links as lines. If the graph is *undirected*, links are just lines, if the graph is *directed*, links are represented as arrows with a tip to show the direction:

### Directed and undirected graphs: definitions

#### Directed graph $G = (V, E)$

- $V$  is a set of *vertexes/nodes*
- $E$  is a set of *edges*, i.e. ordered pairs  $(u, v)$  of nodes

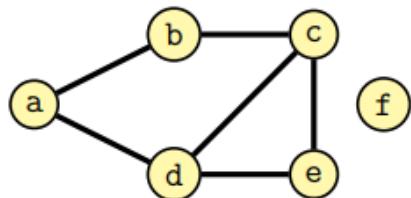
```
V = { a,b,c,d,e,f }
E = { (a,b),(a,d),(b,c),(d,a)
 (d,c),(d,e),(e,c) }
```



#### Undirected graph $G = (V, E)$

- $V$  is a set of *vertexes/nodes*
- $E$  is a set of *edges*, i.e. unordered pairs  $[u, v]$  of nodes

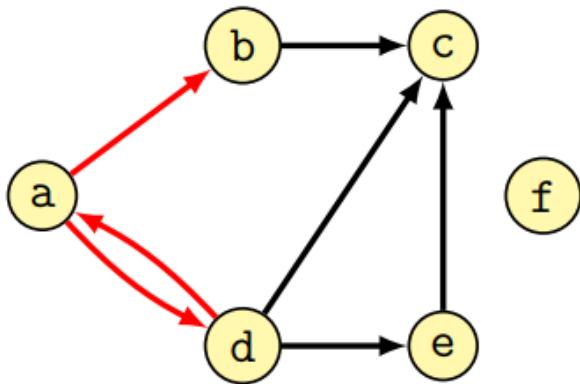
```
V = { a,b,c,d,e,f }
E = { [a,b],[a,d],[b,c],
 [c,d],[d,e],[c,e] }
```



Credits: slide by Dr Alberto Montresor

## Terminology

- Vertex  $v$  is **adjacent** to  $u$  if and only if  $(u, v) \in E$ .
- In an undirected graph, the adjacency relation is symmetric
- An edge  $(u, v)$  is said to be **incident** from  $u$  to  $v$

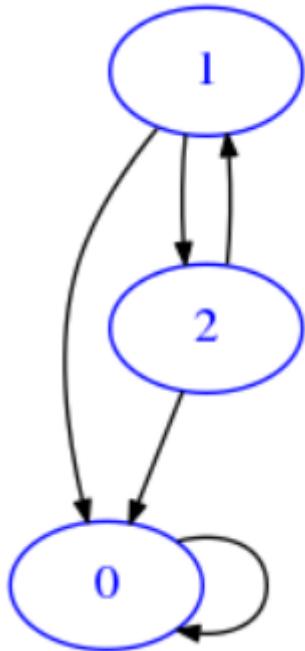


- $(a, b)$  is incident from  $a$  to  $b$
- $(a, d)$  is incident from  $a$  to  $d$
- $(d, a)$  is incident from  $d$  to  $a$
- $b$  is adjacent to  $a$
- $d$  is adjacent to  $a$
- $a$  is adjacent to  $d$

Credits: slide by Dr Alberto Montresor

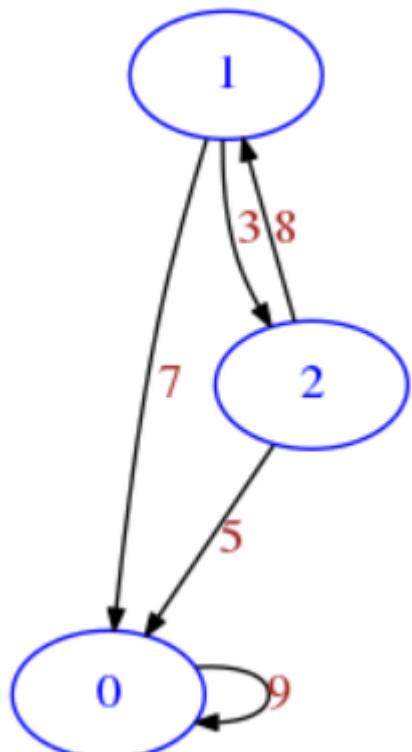
For our purposes, we will consider directed graphs (also called *digraphs*).

Usually we will indicate nodes with numbers going from zero included but optionally they can be labelled. Since we are dealing with directed graphs, we can have an arrow going for example from node 1 to node 2, but also another arrow going from node 2 to node 1. Furthermore, a node (for example node 0) can have a *cap*, that is an edge going to itself:



### Edge weights

Optionally, we will sometimes assign a *weight* to the edges, that is a number to be shown over the edges. So we can modify the previous example. Note we can have an arrow going from node 1 to node 2 with a weight which is different from the weight arrow from 2 to 1:



## Matrices

Here we will represent graphs as matrices, which performance-wise is particularly good when the matrix is *dense*, that is, has many entries different from zero. Otherwise, when you have a so-called *sparse* matrix (few non-zero entries), it is best to represent the graph with *adjacency list*, but we will deal with them later.

If you have a directed graph (digraph) with  $n$  vertices, you can represent it as an  $n \times n$  matrix by considering each row as vertex:

- A row at index  $i$  represents the outward links from node  $i$  to the other  $n$  nodes, with possibly node  $i$  itself included.
- A value of zero means there is no link to a given node.
- In general,  $\text{mat}[i][j]$  is the weight of the edge between node  $i$  to node  $j$

## Visualization examples

We defined a function `soft.draw_mat` to display matrices as graphs (you don't need to understand the internals, for now we won't go into depth about matrix visualizations).

If it doesn't work, see above *Required libraries paragraph*

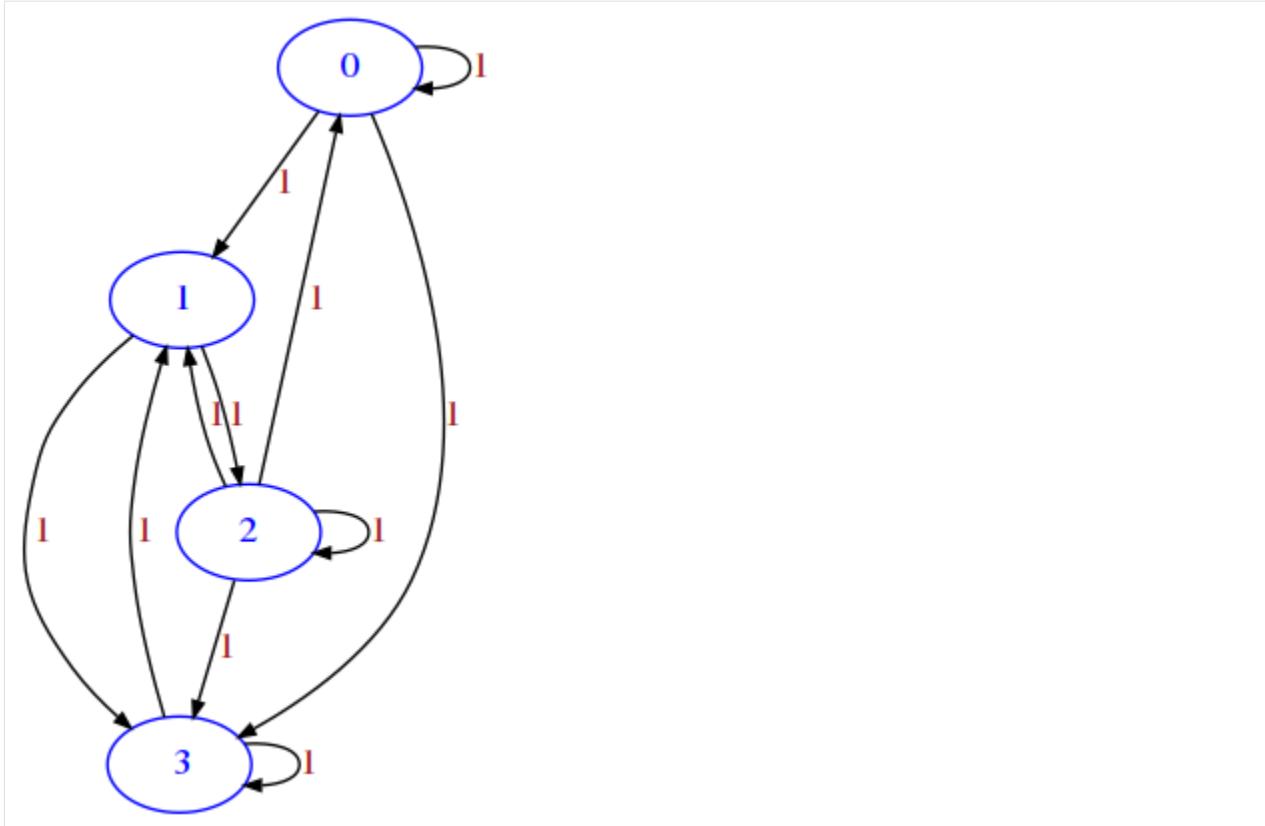
```
[2]: # PLEASE EXECUTE THIS CELL TO CHECK IF VISUALIZATION IS WORKING

notice links with weight zero are not shown)
all weights are set to 1

first need to import this
from soft import draw_mat

mat = [
 [1,1,0,1], # node 0 is linked to node 0 itself, node 1 and node 2
 [0,0,1,1], # node 1 is linked to node 2 and node 3
 [1,1,1,1], # node 2 is linked to node 0, node 1, node 2 itself and node 3
 [0,1,0,1] # node 3 is linked to node 1 and node 3 itself
]

draw_mat(mat)
```



### Saving a graph to an image file

If you want (or if you are not using Jupyter), optionally you can save the graph to a .png file by specifying the `save_to` filepath:

```
[3]: mat = [
 [1, 1],
 [0, 1]
]
draw_mat(mat, save_to='example.png')
```

Image saved to file: example.png



## Saving a graph to an dot file

You can also save a graph to the original *dot* language of GraphViz:

```
[4]: mat = [
 [1, 1],
 [0, 1]
]
draw_mat(mat, save_to='example.dot')

Dot saved to file: example.dot
```

Note no visualization occurs, as you probably might need this kind of output when GraphViz is not installed in your system and you want to display the file elsewhere.

There are lots of websites that take `.dot` and output images, for example [GraphvizOnline](#)<sup>405</sup>

We output here the file content, try to copy/paste it in the above website:

```
[5]: with open('example.dot') as f:
 print(f.read())

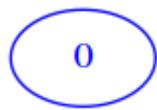
digraph {
scale=3;
style="dotted, rounded";
node [color=blue, fontcolor=blue];
edge [arrowsize="0.6", splines=curved, fontcolor=brown];
0;
1;
0 -> 0 [weight=1, label=1];
0 -> 1 [weight=1, label=1];
1 -> 1 [weight=1, label=1];
}
```

## Minimal graph

With this representation derived from matrices as we intend them (that is with at least one row and one column), the corresponding minimal graph can have only one node:

```
[6]: minimal = [
 [0]
]

draw_mat(minimal)
```



If we set the weight different from zero, the zeroeth node will link to itself (here we put the weight 5 in the link):

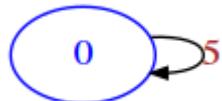
```
[7]: minimal = [
 [5]
]
```

(continues on next page)

<sup>405</sup> <https://dreampuf.github.io/GraphvizOnline>

(continued from previous page)

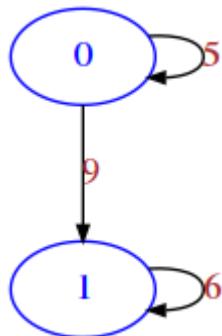
```
draw_mat (minimal)
```



### Graph with two nodes example

```
[8]: m = [
 [5,9], # node 0 links to node 0 itself with a weight of 5, and to node 1 with a
 ↪weight of 9
 [0,6], # node 1 links to node 1 with a weight of 6
]

draw_mat (m)
```



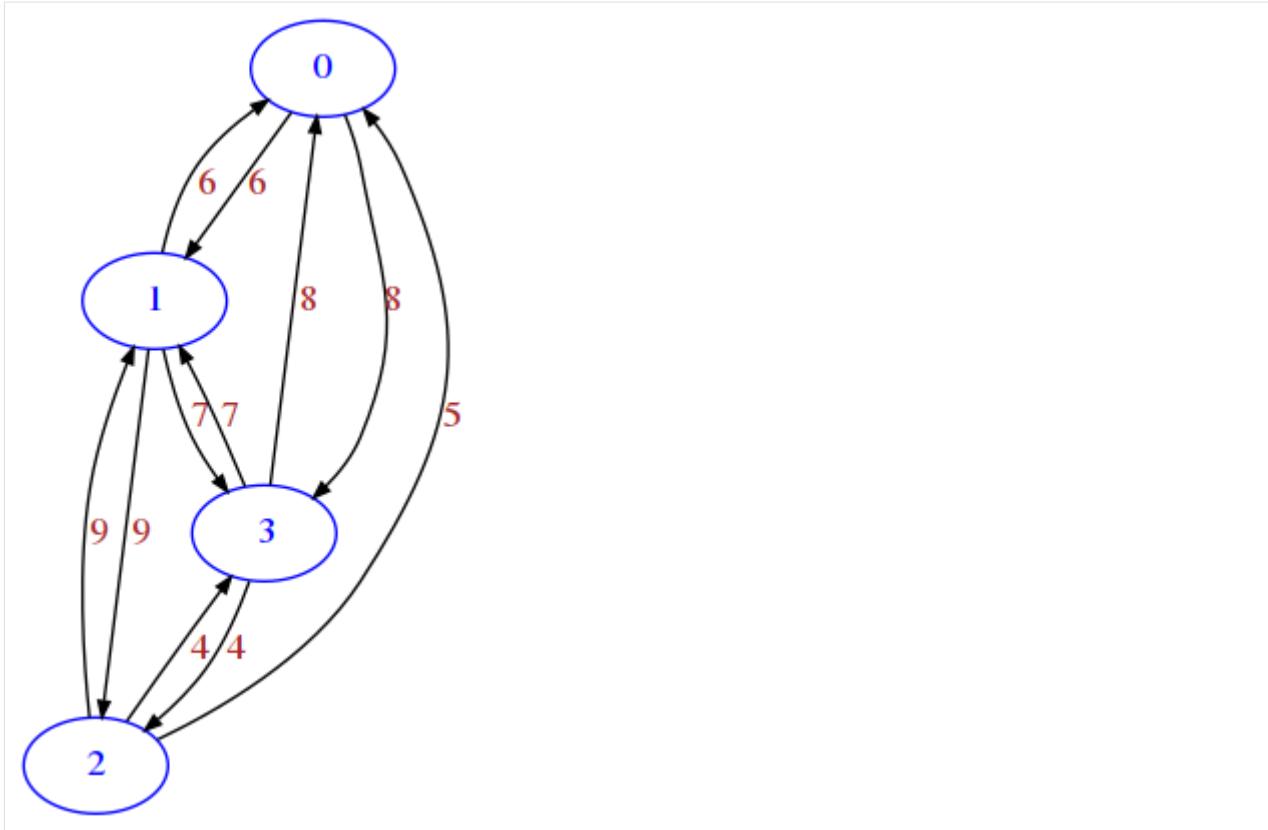
### Distance matrix

Depending on the problem at hand, it may be reasonable to change the weights. For example, on a road network the nodes could represent places and the weights could be the distances. If we assume it is possible to travel in both directions on all roads, we get a matrix symmetric along the diagonal, and we can call the matrix a *distance matrix*. Talking about the diagonal, for the special case of going from a place to itself, we set that street length to 0 (which make sense for street length but could give troubles for other purposes, for example if we give the numbers the meaning ‘is connected’ a place should always be connected to itself)

```
[9]: # distance matrix example

mat = [
 [0,6,0,8], # place 0 is linked to place 1 and place 2
 [6,0,9,7], # place 1 is linked to place 0, place 2 and place 3
 [5,9,0,4], # place 2 is linked to place 0, place 1 and place 3
 [8,7,4,0] # place 3 is linked to place 0, place 1 and place 2
]

draw_mat (mat)
```

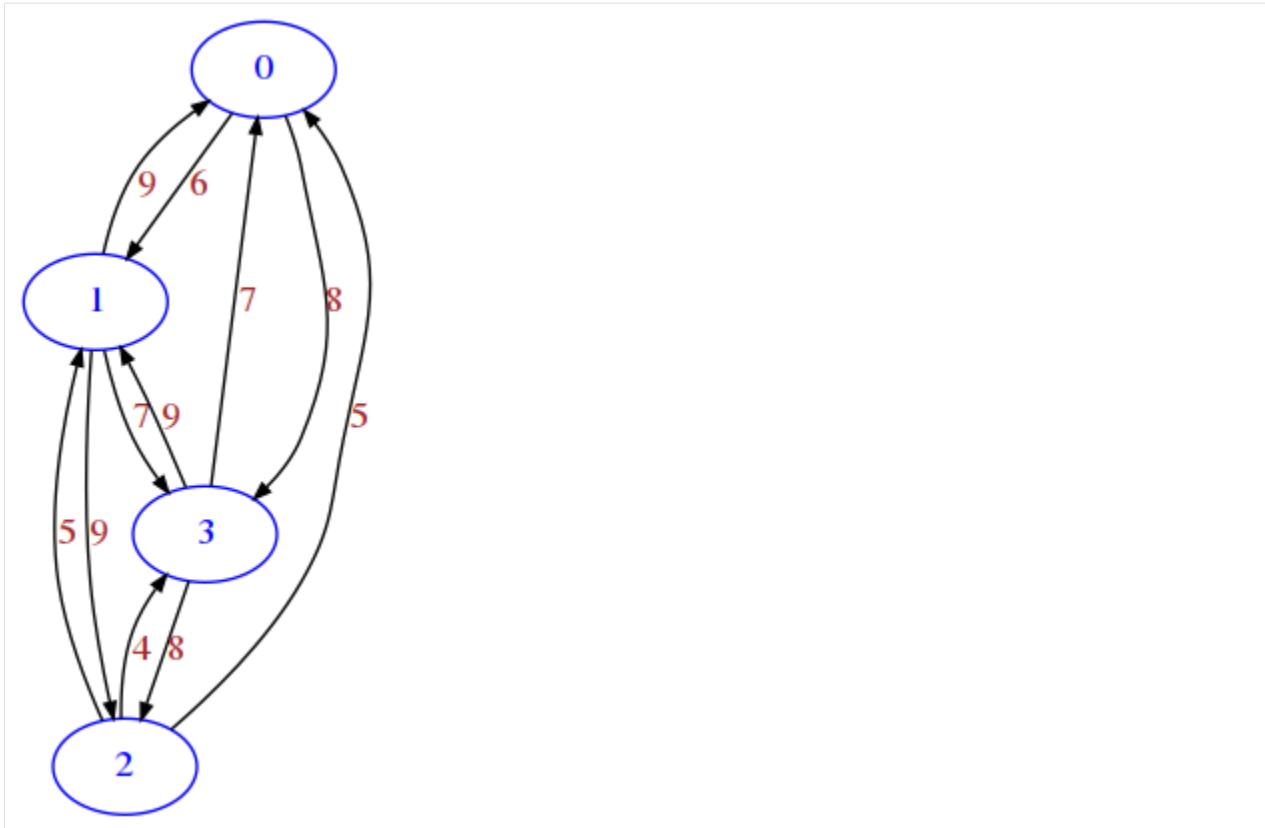


More realistic traffic road network, where going in one direction might take actually longer than going back, because of one-way streets and different routing times.

[10]:

```
mat = [
 [0, 6, 0, 8], # place 0 is linked to place 1 and place 2
 [9, 0, 9, 7], # place 1 is linked to place 0, place 2 and place 3
 [5, 5, 0, 4], # place 2 is linked to place 0, place 1 and place 3
 [7, 9, 8, 0] # place 3 is linked to place 0, place 1, place 2
]

draw_mat(mat)
```



### Boolean matrix example

If we are not interested at all in the weights, we might use only zeroes and ones as we did before. But this could have implications when doing operations on matrices, so sometimes it is better to use only True and False

```
[11]: mat = [
 [False, True, False],
 [False, True, True],
 [True, False, True],
]

draw_mat(mat)
```



## Matrix exercises

We are now ready to start implementing the following functions. Before even start implementation, for each try to interpret the matrix as a graph, drawing it on paper. When you're done implementing try to use `draw_mat` on the results. Notice that since `draw_mat` is a generic display function and knows nothing about the nature of the graph, sometimes it will not show the graph in the optimal way we humans would use.

### Exercise - line

⊗⊗ This function is similar to `diag`. As that one, you can implement it in two ways: you can use a double `for`, or a single one (much more efficient). What would be the graph representation of `line` ?

RETURN a matrix as lists of lists where node  $i$  must have an edge to node  $i + 1$  with weight 1

- Last node points to nothing
- $n$  must be  $\geq 1$ , otherwise raises `ValueError`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[12]: def line(n):

 if n < 1:
 raise ValueError("Invalid n %s" % n)
 ret = [[0]*n for i in range(n)]
 for i in range(n-1):
 ret[i][i+1] = 1
 return ret

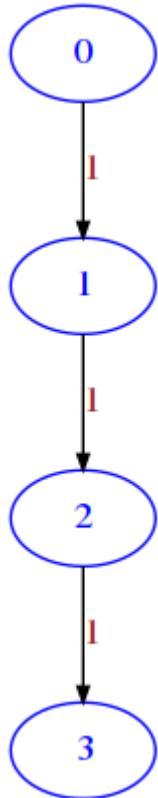
assert line(1) == [[0]]
assert line(2) == [[0,1], [0,0]]
assert line(3) == [[0,1,0], [0,0,1], [0,0,0]]
```

(continues on next page)

(continued from previous page)

```
[0,0,0]]

assert line(4) == [[0,1,0,0],
 [0,0,1,0],
 [0,0,0,1],
 [0,0,0,0]]
draw_mat(line(4))
```



&lt;/div&gt;

```
[12]: def line(n):
 raise Exception('TODO IMPLEMENT ME !')

assert line(1) == [[0]]

assert line(2) == [[0,1],
 [0,0]]
assert line(3) == [[0,1,0],
 [0,0,1],
 [0,0,0]]

assert line(4) == [[0,1,0,0],
 [0,0,1,0],
 [0,0,0,1],
 [0,0,0,0]]
draw_mat(line(4))
```

**Exercise - cross**

$\oplus\oplus$  RETURN a  $n \times n$  matrix filled with zeros except on the crossing lines.

- $n$  must be  $\geq 1$  and odd, otherwise a `ValueError` is thrown

Example for  $n=7$  :

```

0001000
0001000
0001000
1111111
0001000
0001000
0001000

```

Try to figure out how the resulting graph would look like (try to draw on paper, also notice that `draw_mat` will probably not draw the best possible representation)

[Show solution](#)  
[Hide](#)

```
[13]: def cross(n):
 if n < 1 or n % 2 == 0:
 raise ValueError("Invalid n %s" % n)
 ret = [[0]*n for i in range(n)]
```

(continues on next page)

(continued from previous page)

```

for i in range(n):
 ret[n//2][i] = 1
 ret[i][n//2] = 1
return ret

assert cross(1) == [
 [1]
]
assert cross(3) == [
 [0, 1, 0],
 [1, 1, 1],
 [0, 1, 0]
]

assert cross(5) == [
 [0, 0, 1, 0, 0],
 [0, 0, 1, 0, 0],
 [1, 1, 1, 1, 1],
 [0, 0, 1, 0, 0],
 [0, 0, 1, 0, 0]
]

```

&lt;/div&gt;

```
[13]: def cross(n):
 raise Exception('TODO IMPLEMENT ME !')

assert cross(1) == [
 [1]
]
assert cross(3) == [
 [0, 1, 0],
 [1, 1, 1],
 [0, 1, 0]
]

assert cross(5) == [
 [0, 0, 1, 0, 0],
 [0, 0, 1, 0, 0],
 [1, 1, 1, 1, 1],
 [0, 0, 1, 0, 0],
 [0, 0, 1, 0, 0]
]
```

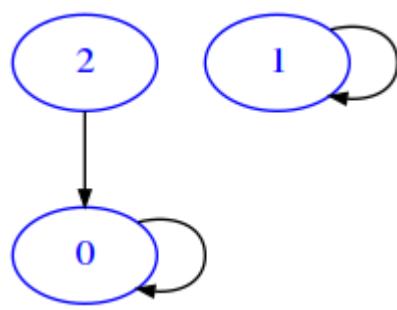
## union

When we talk about the *union* of two graphs, we intend the graph having union of vertexes of both graphs and having as edges the union of edges of both graphs. In this exercise, we have two graphs as list of lists with boolean edges. To simplify we suppose they have the same vertices but possibly different edges, and we want to calculate the union as a new graph.

For example, if we have a graph `ma` like this:

```
[14]: ma = [
 [True, False, False],
 [False, True, False],
 [True, False, False]
]
```

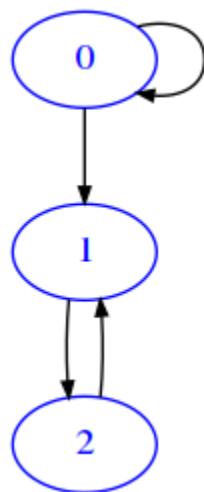
```
[15]: draw_mat (ma)
```



And another mb like this:

```
[16]: mb = [
 [True, True, False],
 [False, False, True],
 [False, True, False]
]
```

```
[17]: draw_mat (mb)
```



The result of calling `union (ma, mb)` will be the following:

```
[18]: res = [[True, True, False], [False, True, True], [True, True, False]]
```

which will be displayed as

```
[19]: draw_mat (res)
```



So we get same vertexes and edges from both ma and mb

### Exercise - union

$\oplus\oplus$  Takes two graphs represented as  $n \times n$  matrices of lists of lists with boolean edges, and RETURN a NEW matrix which is the union of both graphs

- if mata row number is different from matb, raises ValueError

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[20]: def union(mata, matb):

 if len(mata) != len(matb):
 raise ValueError("mata and matb have different row number a:%s b:%s!" %
 (len(mata), len(matb)))

 n = len(mata)

 ret = []
 for i in range(n):
 row = []
 ret.append(row)
 for j in range(n):
 row.append(mata[i][j] or matb[i][j])
 return ret

try:
 union([[False], [False]], [[False]])
 raise Exception("Shouldn't arrive here !")
except ValueError:
 "test passed"

try:
 union([[False]], [[False], [False]])

```

(continues on next page)

(continued from previous page)

```
 raise Exception("Shouldn't arrive here !")
except ValueError:
 "test passed"

ma1 = [[False]]
mb1 = [[False]]

assert union(ma1, mb1) == [[False]]

ma2 = [[False]]
mb2 = [[True]]

assert union(ma2, mb2) == [[True]]

ma3 = [[True]]
mb3 = [[False]]

assert union(ma3, mb3) == [[True]]

ma4 = [[True]]
mb4 = [[True]]

assert union(ma4, mb4) == [[True]]

ma5 = [[False, False, False],
 [False, False, False],
 [False, False, False]]
mb5 = [[True, False, True],
 [False, True, True],
 [False, False, False]]

assert union(ma5, mb5) == [[True, False, True],
 [False, True, True],
 [False, False, False]]

ma6 = [[True, False, True],
 [False, True, True],
 [False, False, False]]
mb6 = [[False, False, False],
 [False, False, False],
 [False, False, False]]

assert union(ma6, mb6) == [[True, False, True],
 [False, True, True],
 [False, False, False]]

ma7 = [[True, False, False],
 [False, True, False],
 [True, False, False]]
mb7 = [[True, True, False],
 [False, False, True],
 [False, True, False]]

assert union(ma7, mb7) == [[True, True, False],
 [False, True, True],
 [True, True, False]]
```

&lt;/div&gt;

```
[20]: def union(mata, matb):
 raise Exception('TODO IMPLEMENT ME !')

try:
 union([[False],[False]], [[False]])
 raise Exception("Shouldn't arrive here !")
except ValueError:
 "test passed"

try:
 union([[False]], [[False],[False]])
 raise Exception("Shouldn't arrive here !")
except ValueError:
 "test passed"

ma1 = [[False]]
mb1 = [[False]]

assert union(ma1, mb1) == [[False]]

ma2 = [[False]]
mb2 = [[True]]

assert union(ma2, mb2) == [[True]]

ma3 = [[True]]
mb3 = [[False]]

assert union(ma3, mb3) == [[True]]

ma4 = [[True]]
mb4 = [[True]]

assert union(ma4, mb4) == [[True]]

ma5 = [[False, False, False],
 [False, False, False],
 [False, False, False]]
mb5 = [[True, False, True],
 [False, True, True],
 [False, False, False]]

assert union(ma5, mb5) == [[True, False, True],
 [False, True, True],
 [False, False, False]]

ma6 = [[True, False, True],
 [False, True, True],
 [False, False, False]]
mb6 = [[False, False, False],
 [False, False, False],
 [False, False, False]]

assert union(ma6, mb6) == [[True, False, True],
 [False, True, True],
```

(continues on next page)

(continued from previous page)

```
[False, False, False]]
```

```
ma7 = [[True, False, False],
 [False, True, False],
 [True, False, False]]
mb7 = [[True, True, False],
 [False, False, True],
 [False, True, False]]

assert union(ma7, mb7) == [[True, True, False],
 [False, True, True],
 [True, True, False]]
```

## Subgraphs

If we interpret a matrix as graph, we may wonder when a graph A is a subgraph of another graph B, that is, when A nodes are a subset of B nodes and when A edges are a subset of B edges. For convenience, here we only consider graphs having the same n nodes both in A and B. Edges may instead vary. Graphs are represented as boolean matrices.

### Exercise - is\_subgraph

⊕⊕ RETURN True if A is a subgraph of B, that is, some or all of its edges also belong to B. A and B are boolean matrices of size nxn.

- If sizes don't match, raises ValueError

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[21]: def is_subgraph(mata, matb):

 n = len(mata)
 m = len(matb)
 if n != m:
 raise ValueError("A size %s and B size %s should match !" % (n,m))
 for i in range(n):
 for j in range(n):
 if mata[i][j] and not matb[i][j]:
 return False
 return True

the set of edges is empty
ma = [[False]]
the set of edges is empty
mb = [[False]]
an empty set is always a subset of an empty set
assert is_subgraph(ma, mb) == True

the set of edges is empty
ma = [[False]]
the set of edges contains one element
mb = [[True]]
an empty set is always a subset of any set, so function gives True
```

(continues on next page)

(continued from previous page)

```

assert is_subgraph(ma, mb) == True

ma = [[True]]
mb = [[True]]
assert is_subgraph(ma, mb) == True

ma = [[True]]
mb = [[False]]
assert is_subgraph(ma, mb) == False

ma = [[True, False],
 [True, False]]
mb = [[True, False],
 [True, True]]
assert is_subgraph(ma, mb) == True

ma = [[False, False, True],
 [True, True,True],
 [True, False,True]]
mb = [[True, False, True],
 [True, True,True],
 [True, True,True]]
assert is_subgraph(ma, mb) == True

```

&lt;/div&gt;

```
[21]: def is_subgraph(mata, matb):
 raise Exception('TODO IMPLEMENT ME !')

 # the set of edges is empty
 ma = [[False]]
 # the set of edges is empty
 mb = [[False]]
 # an empty set is always a subset of an empty set
 assert is_subgraph(ma, mb) == True

 # the set of edges is empty
 ma = [[False]]
 # the set of edges contains one element
 mb = [[True]]
 # an empty set is always a subset of any set, so function gives True
 assert is_subgraph(ma, mb) == True

 ma = [[True]]
 mb = [[True]]
 assert is_subgraph(ma, mb) == True

 ma = [[True]]
 mb = [[False]]
 assert is_subgraph(ma, mb) == False

 ma = [[True, False],
 [True, False]]
 mb = [[True, False],
 [True, True]]
 assert is_subgraph(ma, mb) == True
```

(continues on next page)

(continued from previous page)

```
ma = [[False, False, True],
 [True, True, True],
 [True, False, True]]
mb = [[True, False, True],
 [True, True, True],
 [True, True, True]]
assert is_subgraph(ma, mb) == True
```

### Exercise - remove\_node

⊕⊕ Here the function text is not so precise, as it is talking about nodes but you have to operate on a matrix. Can you guess exactly what you have to do ? In your experiments, try to draw the matrix before and after executing `remove_node`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[22]: def remove_node(mat, i):
 """ MODIFIES mat by removing node i.
 """

 del mat[i]
 for row in mat:
 del row[i]

m = [[3,5,2,5],
 [6,2,3,7],
 [4,2,1,2],
 [7,2,2,6]]

remove_node(m, 2)

assert len(m) == 3
for i in range(3):
 assert len(m[i]) == 3
```

</div>

```
[22]: def remove_node(mat, i):
 """ MODIFIES mat by removing node i.
 """

 raise Exception('TODO IMPLEMENT ME !')

m = [[3,5,2,5],
 [6,2,3,7],
 [4,2,1,2],
 [7,2,2,6]]

remove_node(m, 2)

assert len(m) == 3
for i in range(3):
 assert len(m[i]) == 3
```

## Exercise - utriang

⊕⊕⊕ You will try to create an upper triangular matrix of side n. What could possibly be the graph interpretation of such a matrix? Since draw\_mat is a generic drawing function it doesn't provide the best possible representation, try to draw on paper a more intuitive one.

RETURN a matrix of size nxn which is upper triangular, that is, has all nodes below the diagonal 0, while all the other nodes are set to 1

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[23]: def utriang(n):

 ret = []
 for i in range(n):
 row = []
 for j in range(n):
 if j < i:
 row.append(0)
 else:
 row.append(1)
 ret.append(row)
 return ret

assert utriang(1) == [[1]]
assert utriang(2) == [[1, 1],
 [0, 1]]
assert utriang(3) == [[1, 1, 1],
 [0, 1, 1],
 [0, 0, 1]]
assert utriang(4) == [[1, 1, 1, 1],
 [0, 1, 1, 1],
 [0, 0, 1, 1],
 [0, 0, 0, 1]]
```

</div>

```
[23]: def utriang(n):
 raise Exception('TODO IMPLEMENT ME !')

assert utriang(1) == [[1]]
assert utriang(2) == [[1, 1],
 [0, 1]]
assert utriang(3) == [[1, 1, 1],
 [0, 1, 1],
 [0, 0, 1]]
assert utriang(4) == [[1, 1, 1, 1],
 [0, 1, 1, 1],
 [0, 0, 1, 1],
 [0, 0, 0, 1]]
```

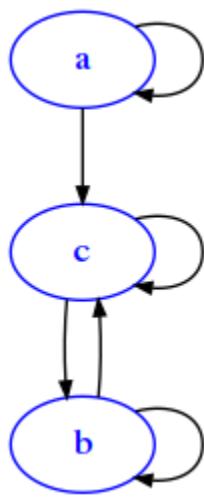
## Edge difference

The *edge difference* of two graphs `ediff(da, db)` is a graph with the edges of the first except the edges of the second. For simplicity, here we consider only graphs having the same vertices but possibly different edges. This time we will try operate on graphs represented as dictionaries of adjacency lists.

For example, if we have

```
[24]: da = {
 'a': ['a', 'c'],
 'b': ['b', 'c'],
 'c': ['b', 'c']
}

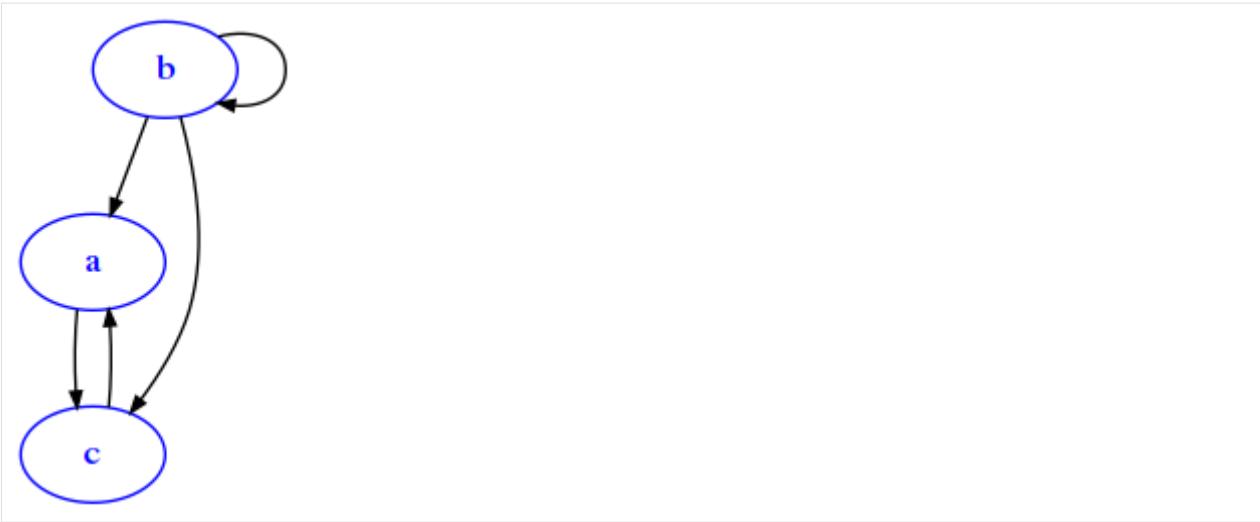
draw_adj(da)
```



and

```
[25]: db = {
 'a': ['c'],
 'b': ['a', 'b', 'c'],
 'c': ['a']
}

draw_adj(db)
```

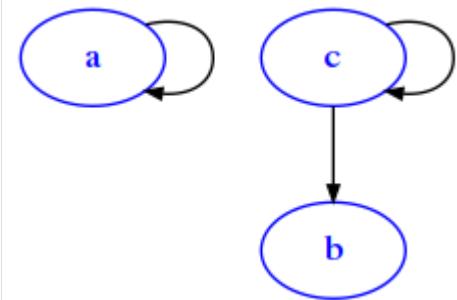


The result of calling `ediff(da, db)` will be:

```
[26]: res = {
 'a': ['a'],
 'b': [],
 'c': ['b', 'c']
 }
```

Which can be shown as

```
[27]: draw_adj(res)
```



### Exercise - ediff

$\oplus\oplus\oplus$  Takes two graphs as dictionaries of adjacency lists da and db, and RETURN a NEW graph as dictionary of adjacency lists, containing the same vertices of da, and the edges of da except the edges of db.

- As order of elements within the adjacency lists, use the same order as found in da.
- We assume all verteces in da and db are represented in the keys (even if they have no outgoing edge), and that da and db have the same keys

Example:

```
da = { 'a': ['a', 'c'],
 'b': ['b', 'c'],
 'c': ['b', 'c'] }
```

(continues on next page)

(continued from previous page)

```
 }

db = { 'a':['c'],
 'b':['a','b', 'c'],
 'c':['a']
 }

assert ediff(da, db) == { 'a':['a'],
 'b':[],
 'c':['b', 'c']
 }
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[28]: def ediff(da, db):

 ret = {}
 for key in da:
 ret[key] = []
 for target in da[key]:
 # not efficient but works for us
 # using sets would be better, see https://stackoverflow.com/a/6486483
 if target not in db[key]:
 ret[key].append(target)
 return ret

da1 = { 'a': [] }
db1 = { 'a': [] }
assert ediff(da1, db1) == { 'a': [] }

da2 = { 'a': [] }
db2 = { 'a': ['a'] }
assert ediff(da2, db2) == { 'a': [] }

da3 = { 'a': ['a'] }
db3 = { 'a': [] }
assert ediff(da3, db3) == { 'a': ['a'] }

da4 = { 'a': ['a'] }
db4 = { 'a': ['a'] }
assert ediff(da4, db4) == { 'a': [] }

da5 = { 'a':['b'],
 'b':[] }
db5 = { 'a':['b'],
 'b':[] }
assert ediff(da5, db5) == { 'a':[],
 'b':[] }

da6 = { 'a':['b'],
 'b':[] }
db6 = { 'a':[],
 'b':[] }
```

(continues on next page)

(continued from previous page)

```

assert ediff(da6, db6) == { 'a':['b'],
 'b':[] }

da7 = { 'a':['a','b'],
 'b':[] }
db7 = { 'a':['a'],
 'b':[] }
assert ediff(da7, db7) == { 'a':['b'],
 'b':[] }

da8 = { 'a':['a','b'],
 'b':['a'] }
db8 = { 'a':['a'],
 'b':['b'] }
assert ediff(da8, db8) == { 'a':['b'],
 'b':['a'] }

da9 = { 'a':['a','c'],
 'b':['b','c'],
 'c':['b','c'] }
db9 = { 'a':['c'],
 'b':['a','b','c'],
 'c':['a'] }
assert ediff(da9, db9) == { 'a':['a'],
 'b':[],
 'c':['b','c'] }

```

&lt;/div&gt;

```
[28]: def ediff(da,db):
 raise Exception('TODO IMPLEMENT ME !')

da1 = { 'a': [] }
db1 = { 'a': [] }
assert ediff(da1, db1) == { 'a': [] }

da2 = { 'a': [] }
db2 = { 'a': ['a'] }
assert ediff(da2, db2) == { 'a': [] }

da3 = { 'a': ['a'] }
db3 = { 'a': [] }
assert ediff(da3, db3) == { 'a': ['a'] }

da4 = { 'a': ['a'] }
db4 = { 'a': ['a'] }
assert ediff(da4, db4) == { 'a': [] }

da5 = { 'a':['b'],
 'b':[] }
db5 = { 'a':['b'],
 'b':[] }
assert ediff(da5, db5) == { 'a':[],
 'b':[] }

da6 = { 'a':['b'],
 'b':[] }

```

(continues on next page)

(continued from previous page)

```

db6 = { 'a':[],
 'b':[] }
assert ediff(da6, db6) == { 'a':['b'],
 'b':[] }

da7 = { 'a':['a','b'],
 'b':[] }
db7 = { 'a':['a'],
 'b':[] }
assert ediff(da7, db7) == { 'a':['b'],
 'b':[] }

da8 = { 'a':['a','b'],
 'b':['a'] }
db8 = { 'a':['a'],
 'b':['b'] }
assert ediff(da8, db8) == { 'a':['b'],
 'b':['a'] }

da9 = { 'a':['a','c'],
 'b':['b','c'],
 'c':['b','c'] }
db9 = { 'a':['c'],
 'b':['a','b','c'],
 'c':['a'] }
assert ediff(da9, db9) == { 'a':['a'],
 'b':[],
 'c':['b','c'] }

```

## Exercise - pyramid

⊕⊕⊕ The following function requires to create a matrix filled with non-zero numbers. Even if don't know exactly the network meaning, with this fact we can conclude that all nodes are linked to all others. A graph where this happens is called a *clique* (the Italian name is *cricca*)

Takes an odd number  $n \geq 1$  and RETURN a matrix as list of lists containing numbers displaced like this example for a pyramid of square 7:

```

1111111
1222221
1233321
1234321
1233321
1222221
1111111

```

- if  $n$  is even, raises ValueError

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol" jupman-sol-code" style="display:none">

```
[29]: def pyramid(n):

 if n % 2 == 0:
 raise ValueError("n should be odd, found instead %s" % n)
```

(continues on next page)

(continued from previous page)

```

ret = [[0]*n for i in range(n)]
for i in range(n//2 + 1):
 for j in range(n//2 +1):
 ret[i][j] = min(i, j) + 1
 ret[i][n-j-1] = min(i, j) + 1
 ret[n-i-1][j] = min(i, j) + 1
 ret[n-i-1][n-j-1] = min(i, j) + 1

ret[n//2][n//2] = n // 2 + 1
return ret

try:
 pyramid(4)
 raise Exception("SHOULD HAVE FAILED!")
except ValueError:
 "passed test"

assert pyramid(1) == [[1]]

assert pyramid(3) == [[1,1,1],
 [1,2,1],
 [1,1,1]]

assert pyramid(5) == [[1, 1, 1, 1, 1],
 [1, 2, 2, 2, 1],
 [1, 2, 3, 2, 1],
 [1, 2, 2, 2, 1],
 [1, 1, 1, 1, 1]]

```

&lt;/div&gt;

```
[29]: def pyramid(n):
 raise Exception('TODO IMPLEMENT ME !')

try:
 pyramid(4)
 raise Exception("SHOULD HAVE FAILED!")
except ValueError:
 "passed test"

assert pyramid(1) == [[1]]

assert pyramid(3) == [[1,1,1],
 [1,2,1],
 [1,1,1]]

assert pyramid(5) == [[1, 1, 1, 1, 1],
 [1, 2, 2, 2, 1],
 [1, 2, 3, 2, 1],
 [1, 2, 2, 2, 1],
 [1, 1, 1, 1, 1]]
```

## Adjacency lists

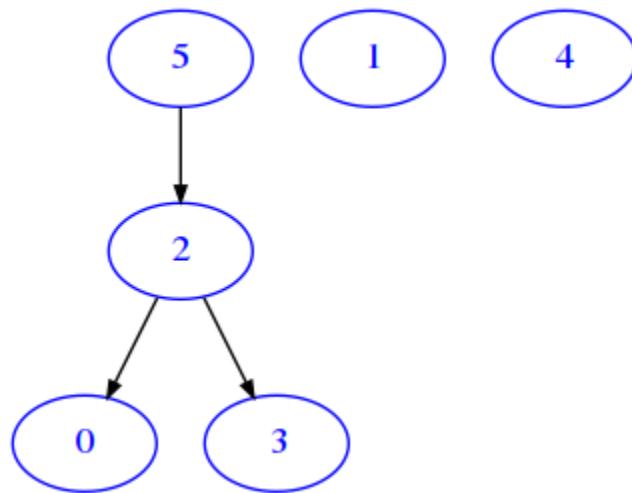
So far, we represented graphs as matrices, saying they are good when the graph is dense, that is any given node is likely to be connected to almost all other nodes - or equivalently, many cell entries in the matrix are different from zero. But if this is not the case, other representations might be needed. For example, we can represent a graph as a *adjacency lists*.

Let's look at this 6x6 boolean matrix:

```
[30]: m = [
 [False, False, False, False, False, False],
 [False, False, False, False, False, False],
 [True, False, False, True, False, False],
 [False, False, False, False, False, False],
 [False, False, False, False, False, False],
 [False, False, True, False, False, False]
]
```

We see just a few True, so by drawing it we don't expect to see many edges:

```
[31]: draw_mat(m)
```



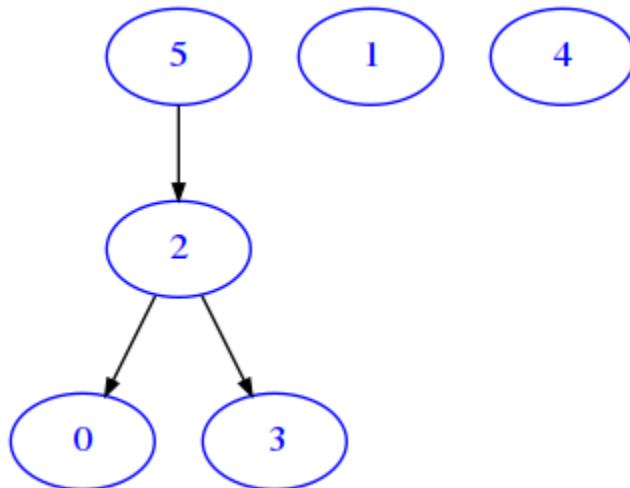
As a more compact representation, we might represent the data as a dictionary of *adjacency lists* where the keys are the node indexes and the to each node we associate a list with the target nodes it points to.

To reproduce the example above, we can write like this:

```
[32]: d = {
 0: [], # node 0 links to nothing
 1: [], # node 1 links to nothing
 2: [0, 3], # node 2 links to node 0 and 3
 3: [], # node 3 links to nothing
 4: [], # node 4 links to nothing
 5: [2] # node 5 links to node 2
}
```

In `soft.py`, we provide also a function `soft.draw_adj` to quickly inspect such data structure:

```
[33]: from soft import draw_adj
draw_adj(d)
```



As expected, the resulting graph is the same as for the equivalent matrix representation.

### Exercise - mat\_to\_adj

⊕⊕ Implement a function that takes a boolean nxn matrix and RETURN the equivalent representation as dictionary of adjacency lists. Remember that to create an empty dict you have to write `dict()`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[34]: def mat_to_adj(bool_mat):
 ret = dict()
 n = len(bool_mat)
 for i in range(n):
 ret[i] = []
 for j in range(n):
 if bool_mat[i][j]:
 ret[i].append(j)
 return ret

m1 = [[False]]
d1 = { 0:[] }
assert mat_to_adj(m1) == d1

m2 = [[True]]
d2 = { 0:[0] }
assert mat_to_adj(m2) == d2

m3 = [[False, False],
 [False, False]]
d3 = { 0:[],
```

(continues on next page)

(continued from previous page)

```
 1:[] }
assert mat_to_adj(m3) == d3

m4 = [[True,True],
 [True,True]]
d4 = { 0:[0,1],
 1:[0,1] }
assert mat_to_adj(m4) == d4

m5 = [[False,False],
 [False,True]]
d5 = { 0:[],
 1:[1] }
assert mat_to_adj(m5) == d5

m6 = [[True,False,False],
 [True, True,False],
 [False,True,False]]
d6 = { 0:[0],
 1:[0,1],
 2:[1] }
assert mat_to_adj(m6) == d6
```

&lt;/div&gt;

```
[34]: def mat_to_adj(bool_mat):
 raise Exception('TODO IMPLEMENT ME !')

m1 = [[False]]
d1 = { 0:[] }
assert mat_to_adj(m1) == d1

m2 = [[True]]
d2 = { 0:[0] }
assert mat_to_adj(m2) == d2

m3 = [[False,False],
 [False,False]]
d3 = { 0:[],
 1:[] }
assert mat_to_adj(m3) == d3

m4 = [[True,True],
 [True,True]]
d4 = { 0:[0,1],
 1:[0,1] }
assert mat_to_adj(m4) == d4

m5 = [[False,False],
 [False,True]]
d5 = { 0:[],
 1:[1] }
assert mat_to_adj(m5) == d5
```

(continues on next page)

(continued from previous page)

```
m6 = [[True, False, False],
 [True, True, False],
 [False, True, False]]
d6 = { 0:[0],
 1:[0, 1],
 2:[1] }
assert mat_to_adj(m6) == d6
```

**Exercise - mat\_ids\_to\_adj**

⊕⊕ Implement a function that takes a boolean nxn matrix and a list of immutable identifiers for the nodes, and RETURN the equivalent representation as dictionary of adjacency lists.

- If matrix is not nxn or ids length does not match n, raise ValueError

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[35]: def mat_ids_to_adj(bool_mat, ids):

 ret = dict()
 n = len(bool_mat)
 m = len(bool_mat[0])
 if n != m:
 raise ValueError('matrix is not nxn !')
 if n != len(ids):
 raise ValueError("Identifiers quantity is different from matrix size! ")
 for i in range(n):
 ret[ids[i]] = []
 for j in range(n):
 if bool_mat[i][j]:
 ret[ids[i]].append(ids[j])
 return ret

try:
 mat_ids_to_adj([[False, True]], ['a', 'b'])
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

try:
 mat_ids_to_adj([[False]], ['a', 'b'])
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

m1 = [[False]]
d1 = { 'a':[] }
assert mat_ids_to_adj(m1, ['a']) == d1

m2 = [[True]]
d2 = { 'a':['a'] }
```

(continues on next page)

(continued from previous page)

```

assert mat_ids_to_adj(m2, ['a']) == d2

m3 = [[False, False], [False, False]]
d3 = {'a':[], 'b':[]}
assert mat_ids_to_adj(m3, ['a', 'b']) == d3

m4 = [[True, True], [True, True]]
d4 = {'a':['a', 'b'], 'b':['a', 'b']}
assert mat_ids_to_adj(m4, ['a', 'b']) == d4

m5 = [[False, False], [False, True]]
d5 = {'a':[], 'b':['b']}
assert mat_ids_to_adj(m5, ['a', 'b']) == d5

m6 = [[True, False, False], [True, True, False], [False, True, False]]
d6 = {'a':['a'], 'b':['a', 'b'], 'c':['b']}
assert mat_ids_to_adj(m6, ['a', 'b', 'c']) == d6

```

&lt;/div&gt;

```

[35]: def mat_ids_to_adj(bool_mat, ids):
 raise Exception('TODO IMPLEMENT ME !')

 try:
 mat_ids_to_adj([[False, True]], ['a', 'b'])
 raise Exception("SHOULD HAVE FAILED !")
 except ValueError:
 "passed test"

 try:
 mat_ids_to_adj([[False]], ['a', 'b'])
 raise Exception("SHOULD HAVE FAILED !")
 except ValueError:
 "passed test"

m1 = [[False]]
d1 = {'a':[]}
assert mat_ids_to_adj(m1, ['a']) == d1

m2 = [[True]]
d2 = {'a':['a'] }

```

(continues on next page)

(continued from previous page)

```

assert mat_ids_to_adj(m2, ['a']) == d2

m3 = [[False, False],
 [False, False]]
d3 = { 'a':[],
 'b':[]
 }
assert mat_ids_to_adj(m3, ['a', 'b']) == d3

m4 = [[True, True],
 [True, True]]
d4 = { 'a':['a', 'b'],
 'b':['a', 'b']
 }
assert mat_ids_to_adj(m4, ['a', 'b']) == d4

m5 = [[False, False],
 [False, True]]
d5 = { 'a':[],
 'b':['b']
 }
assert mat_ids_to_adj(m5, ['a', 'b']) == d5

m6 = [[True, False, False],
 [True, True, False],
 [False, True, False]]
d6 = { 'a':['a'],
 'b':['a', 'b'],
 'c':['b']
 }
assert mat_ids_to_adj(m6, ['a', 'b', 'c']) == d6

```

### Exercise - adj\_to\_mat

Try now conversion from dictionary of adjacency list to matrix (this is a bit hard).

To solve this, the general idea is that you have to fill an nxn matrix to return. During the filling of a cell at row  $i$  and column  $j$ , you have to decide whether to put a `True` or a `False`. You should put `True` if in the `d` list value corresponding to the  $i$ -th key, there is contained a number equal to  $j$ . Otherwise, you should put `False`.

If you look at the tests, as inputs we are passing `OrderedDict`. The reason is that when we check the output matrix of your function, we want to be sure the matrix rows are ordered in a certain way.

But you have to assume `d` can contain arbitrary ids with no precise ordering, so:

1. first you should scan the dictionary and lists to save the mapping between indexes to ids in a separate list

**NOTE:** `d.keys()` is not exactly a list (does not allow access by index), so you must convert to list with this: `list(d.keys())`

2. then you should build the matrix to return, using the previously built list when needed.

⊕⊕⊕ Now implement a function that takes a dictionary of adjacency lists with arbitrary ids and RETURN its representation as an nxn boolean matrix

- assume all nodes are present as keys

- assume d is a simple dictionary (not necessarily an OrderedDict)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[36]: def adj_to_mat(d):

 ret = []
 n = len(d)
 ids_to_row_indexes = dict()
 # first maps row indexes to keys
 row_indexes_to_ids = list(d.keys()) # because d.keys() is *not* indexable !
 i = 0
 for key in d:
 row = []
 ret.append(row)
 for j in range(n):
 if row_indexes_to_ids[j] in d[key]:
 row.append(True)
 else:
 row.append(False)
 i += 1
 return ret

from collections import OrderedDict
od1 = OrderedDict([('a',[])])
m1 = [[False]]
assert adj_to_mat(od1) == m1

od2 = OrderedDict([('a',['a'])])
m2 = [[True]]

assert adj_to_mat(od2) == m2

od3 = OrderedDict([('a',['a', 'b']),
 ('b',['a', 'b'])])
m3 = [[True, True],
 [True, True]]
assert adj_to_mat(od3) == m3

od4 = OrderedDict([('a',[]),
 ('b',[])])

m4 = [[False, False],
 [False, False]]
assert adj_to_mat(od4) == m4

od5 = OrderedDict([('a',['a']),
 ('b',['a', 'b'])])

m5 = [[True, False],
 [True, True]]
assert adj_to_mat(od5) == m5

od6 = OrderedDict([('a',['a', 'c']),
 ('b',['c'])],
```

(continues on next page)

(continued from previous page)

```
('c', ['a', 'b'])])
m6 = [[True, False, True],
 [False, False, True],
 [True, True, False]]
assert adj_to_mat(od6) == m6
```

&lt;/div&gt;

```
[36]: def adj_to_mat(d):
 raise Exception('TODO IMPLEMENT ME !')

from collections import OrderedDict
od1 = OrderedDict([('a',[])])
m1 = [[False]]
assert adj_to_mat(od1) == m1

od2 = OrderedDict([('a',['a'])])
m2 = [[True]]

assert adj_to_mat(od2) == m2

od3 = OrderedDict([('a',['a','b']),
 ('b',['a','b'])])
m3 = [[True, True],
 [True, True]]
assert adj_to_mat(od3) == m3

od4 = OrderedDict([('a',[]),
 ('b',[])])

m4 = [[False, False],
 [False, False]]
assert adj_to_mat(od4) == m4

od5 = OrderedDict([('a',['a']),
 ('b',['a','b'])])

m5 = [[True, False],
 [True, True]]
assert adj_to_mat(od5) == m5

od6 = OrderedDict([('a',['a','c']),
 ('b',['c']),
 ('c',['a','b'])])
m6 = [[True, False, True],
 [False, False, True],
 [True, True, False]]
assert adj_to_mat(od6) == m6
```

**Exercise - table\_to\_adj**

Suppose you have a table expressed as a list of lists with headers like this:

```
[37]: m0 = [
 ['Identifier', 'Price', 'Quantity'],
 ['a', 1, 1],
 ['b', 5, 8],
 ['c', 2, 6],
 ['d', 8, 5],
 ['e', 7, 3]
]
```

where a, b, c etc are the row identifiers (imagine they represent items in a store), Price and Quantity are properties they might have. **NOTE:** here we put two properties, but they might have n properties !

We want to transform such table into a graph-like format as a dictionary of lists, which relates store items as keys to the properties they might have. To include in the list both the property identifier and its value, we will use tuples. So you need to write a function that transforms the above input into this:

```
[38]: res0 = {
 'a': [('Price', 1), ('Quantity', 1)],
 'b': [('Price', 5), ('Quantity', 8)],
 'c': [('Price', 2), ('Quantity', 6)],
 'd': [('Price', 8), ('Quantity', 5)],
 'e': [('Price', 7), ('Quantity', 3)]
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[39]: def table_to_adj(table):

 ret = {}
 headers = table[0]

 for row in table[1:]:
 lst = []
 for j in range(1, len(row)):
 lst.append((headers[j], row[j]))
 ret[row[0]] = lst
 return ret
```

```
m0 = [['I', 'P', 'Q']]
res0 = {}

assert res0 == table_to_adj(m0)

m1 = [
 ['Identifier', 'Price', 'Quantity'],
 ['a', 1, 1],
 ['b', 5, 8],
 ['c', 2, 6],
 ['d', 8, 5],
 ['e', 7, 3]
]
```

(continues on next page)

(continued from previous page)

```

res1 = {
 'a': [('Price', 1), ('Quantity', 1)],
 'b': [('Price', 5), ('Quantity', 8)],
 'c': [('Price', 2), ('Quantity', 6)],
 'd': [('Price', 8), ('Quantity', 5)],
 'e': [('Price', 7), ('Quantity', 3)]
}

assert res1 == table_to_adj(m1)

m2 = [
 ['I', 'P', 'Q'],
 ['a', 'x', 'y'],
 ['b', 'w', 'z'],
 ['c', 'z', 'x'],
 ['d', 'w', 'w'],
 ['e', 'y', 'x']
]
res2 = {
 'a': [('P', 'x'), ('Q', 'y')],
 'b': [('P', 'w'), ('Q', 'z')],
 'c': [('P', 'z'), ('Q', 'x')],
 'd': [('P', 'w'), ('Q', 'w')],
 'e': [('P', 'y'), ('Q', 'x')]
}

assert res2 == table_to_adj(m2)

m3 = [
 ['I', 'P', 'Q', 'R'],
 ['a', 'x', 'y', 'x'],
 ['b', 'z', 'x', 'y'],
]
res3 = {
 'a': [('P', 'x'), ('Q', 'y'), ('R', 'x')],
 'b': [('P', 'z'), ('Q', 'x'), ('R', 'y')]
}
assert res3 == table_to_adj(m3)

```

&lt;/div&gt;

```
[39]: def table_to_adj(table):
 raise Exception('TODO IMPLEMENT ME !')

m0 = [['I', 'P', 'Q']]
res0 = {}

assert res0 == table_to_adj(m0)

m1 = [
 ['Identifier', 'Price', 'Quantity'],
 ['a', 1, 1],
 ['b', 5, 8],
 ['c', 2, 6],
 ['d', 8, 5],
 ['e', 7, 3]
]
```

(continues on next page)

(continued from previous page)

```
]
res1 = {
 'a': [('Price', 1), ('Quantity', 1)],
 'b': [('Price', 5), ('Quantity', 8)],
 'c': [('Price', 2), ('Quantity', 6)],
 'd': [('Price', 8), ('Quantity', 5)],
 'e': [('Price', 7), ('Quantity', 3)]
}

assert res1 == table_to_adj(m1)

m2 = [
 ['I', 'P', 'Q'],
 ['a', 'x', 'y'],
 ['b', 'w', 'z'],
 ['c', 'z', 'x'],
 ['d', 'w', 'w'],
 ['e', 'y', 'x']
]
res2 = {
 'a': [('P', 'x'), ('Q', 'y')],
 'b': [('P', 'w'), ('Q', 'z')],
 'c': [('P', 'z'), ('Q', 'x')],
 'd': [('P', 'w'), ('Q', 'w')],
 'e': [('P', 'y'), ('Q', 'x')]
}

assert res2 == table_to_adj(m2)

m3 = [
 ['I', 'P', 'Q', 'R'],
 ['a', 'x', 'y', 'x'],
 ['b', 'z', 'x', 'y'],
]
res3 = {
 'a': [('P', 'x'), ('Q', 'y'), ('R', 'x')],
 'b': [('P', 'z'), ('Q', 'x'), ('R', 'y')]
}
assert res3 == table_to_adj(m3)
```

## Networkx

**Before continuing, make sure to have installed the *required libraries***

Networkx is a library to perform statistics on networks. For now, it will offer us a richer data structure where we can store the properties we want in nodes and also edges.

You can initialize networkx objects with the dictionary of adjacency lists we've already seen:

```
[40]:
import networkx as nx

notice with networkx if nodes are already referenced to in an adjacency list
you do not need to put them as keys:
```

(continues on next page)

(continued from previous page)

```
G=nx.DiGraph({
 'a':['b','c'], # node a links to b and c
 'b':['b','c', 'd'] # node b links to b itself, c and d
})
```

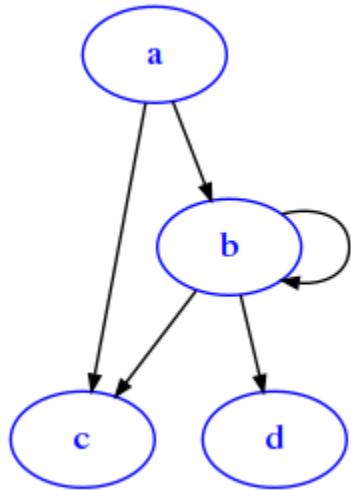
The resulting object is not a simple dict, but something more complex:

```
[41]: G
[41]: <networkx.classes.digraph.DiGraph at 0x7f5e09983f10>
```

To display it in a way uniform with the rest of the course, we developed a function called `soft.draw_nx`:

```
[42]: from soft import draw_nx
```

```
[43]: draw_nx(G)
```



From the picture above, we notice there are no weights displayed, because in networkx they are just considered optional attributes of edges.

To see all the attributes of an edge, you can write like this:

```
[44]: G['a']['b']
[44]: {}
```

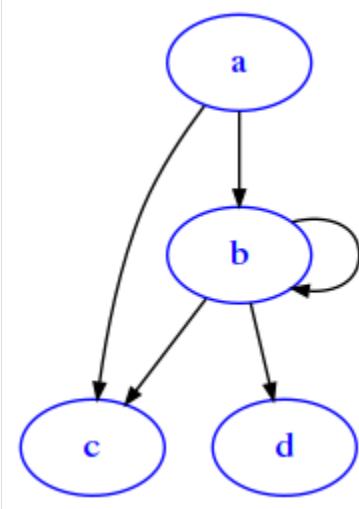
This graph has no attributes for the edge, so we get back an empty dict. If we wanted to add a weight of 123 to that particular a → b edge, you could write like this:

```
[45]: G['a']['b']['weight'] = 123
```

```
[46]: G['a']['b']
[46]: {'weight': 123}
```

Let's try to display it:

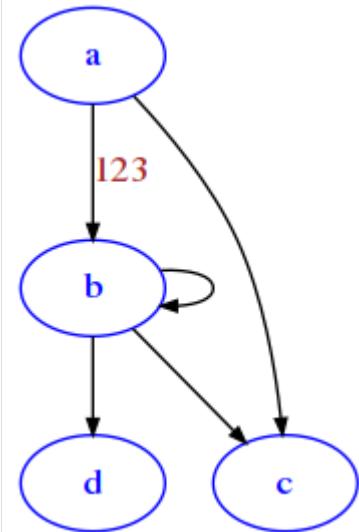
```
[47]: draw_nx(G)
```



We still don't see the weight as weight can be one of many properties: the only thing that gets displayed is the property label. So let's set label equal to the weight:

```
[48]: G['a']['b']['label'] = 123
```

```
[49]: draw_nx(G)
```



## Fancy networkx graphs

With networkx we can set additional attributes to embellish the resulting graph, here we show a bus network example.

```
[50]: G = nx.DiGraph()

we can force horizontal layout like this:

G.graph['graph'] = {
```

(continues on next page)

(continued from previous page)

```

 'rankdir':'LR',
 }

When we add nodes, we can identify them with an identifier like the
stop_id which is separate from the label, for example in some unfortunate
case two different stops can share the same label.

G.add_node('1', label='Trento',
 color='orange', fontcolor='black')
G.add_node('723', label='Rovereto',
 color='black', fontcolor='black')
G.add_node('870', label='Arco',
 color='black', fontcolor='black')
G.add_node('1180', label='Riva',
 color='black', fontcolor='blue')

IMPORTANT: edges connect stop_ids , NOT labels !!!!
G.add_edge('870','1')
G.add_edge('723','1')
G.add_edge('1','1180')

we can retrieve an edge like this:

edge = G['1']['1180']

and set attributes, like these:

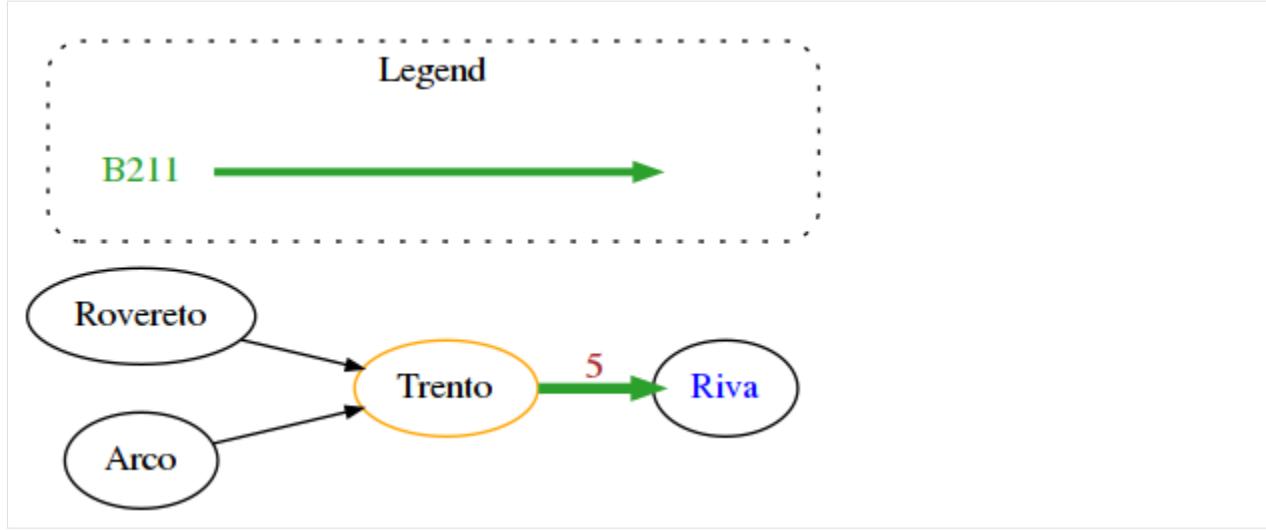
edge['weight'] = 5 # the actual weight (not shown!)
edge['label'] = str(5) # the label is a string
edge['color'] = '#2ca02c' # we can set some style for the edge, such as color
edge['penwidth']= 4 # and thickness

edge['route_short_name'] = 'B301' # we can add any attribute we want,
 # Note these custom ones won't show in the graph

legend = [{'label': 'B211', 'color': '#2ca02c'}]

draw_nx(G, legend)

```



## Converting networkx graphs

If you try to just output the string representation of the graph, networkx will give the empty string:

```
[51]: print(G)
```

```
[52]: str(G)
```

```
[52]: ''
```

```
[53]: repr(G)
```

```
[53]: '<networkx.classes.digraph.DiGraph object at 0x7f5e09a61b10>'
```

To convert to the dict of adjacency lists we know, you can use this method:

```
[54]: nx.to_dict_of_lists(G)
```

```
[54]: {'1': ['1180'], '723': ['1'], '870': ['1'], '1180': []}
```

The above works, but it doesn't convert additional edge info. For a complete conversion, use `nx.to_dict_of_dicts`

```
[55]: nx.to_dict_of_dicts(G)
```

```
[55]: {'1': {'1180': {'weight': 5,
 'label': '5',
 'color': '#2ca02c',
 'penwidth': 4,
 'route_short_name': 'B301'}},
 '723': {'1': {}},
 '870': {'1': {}},
 '1180': {}}
```

### Exercise - mat\_to\_nx

⊕⊕ Now try by yourself to convert a matrix as list of lists along with node ids (like *you did before*) into a networkx object.

This time, don't create a dictionary to pass it to `nx.DiGraph` constructor: instead, use networkx methods like `.add_edge` and `add_node`. For usage example, check the [networkx tutorial](#)<sup>406</sup>.

**QUESTION:** Do you need to explicitly call `add_node` before referring to some node with `add_edge` ?

Implement a function that given a real-valued  $n \times n$  matrix as list of lists, and a list of immutable identifiers for the nodes, RETURNS the corresponding graph in networkx format (as `nx.DiGraph`).

- If matrix is not  $n \times n$  or `ids` length does not match  $n$ , raise `ValueError`

**DO NOT** transform into a dict, use instead `add_` methods from networkx object!

**WARNING:** remember to set the weights labels AS STRINGS!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[56]: def mat_to_nx(mat, ids):

 G = nx.DiGraph()
 n = len(mat)
 m = len(mat[0])
 if n != m:
 raise ValueError('matrix is not nxn !')
 if n != len(ids):
 raise ValueError("Identifiers quantity is different from matrix size! ")
 for i in range(n):
 G.add_node(ids[i])
 for j in range(n):
 if mat[i][j] != 0:
 G.add_edge(ids[i], ids[j])
 G[ids[i]][ids[j]]['weight'] = mat[i][j]
 G[ids[i]][ids[j]]['label'] = str(mat[i][j])
 return G

try:
 mat_ids_to_adj([[0, 3]], ['a', 'b'])
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

try:
 mat_ids_to_adj([[0]], ['a', 'b'])
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

m1 = [[0]]
d1 = {'a': {}}

assert nx.to_dict_of_dicts(mat_to_nx(m1, ['a'])) == d1
```

(continues on next page)

<sup>406</sup> <https://networkx.github.io/documentation/stable/tutorial.html>

(continued from previous page)

```
m2 = [[7]]

d2 = {'a': {'a': {'weight': 7, 'label': '7'}}}
assert nx.to_dict_of_dicts(mat_to_nx(m2, ['a'])) == d2

m3 = [[0,0],
 [0,0]]

d3 = { 'a':{},
 'b':{}
 }
assert nx.to_dict_of_dicts(mat_to_nx(m3, ['a', 'b'])) == d3

m4 = [[7,9],
 [8,6]]

d4 = { 'a':{'a': {'weight':7,'label':'7'},
 'b' : {'weight':9,'label':'9'},
 },
 'b':{'a': {'weight':8,'label':'8'},
 'b' : {'weight':6,'label':'6'},
 }
 }

assert nx.to_dict_of_dicts(mat_to_nx(m4, ['a', 'b'])) == d4

m5 = [[0,0],
 [0,7]]

d5 = { 'a':{},
 'b':{
 'b' : {'weight':7,'label':'7'}
 }
 }

assert nx.to_dict_of_dicts(mat_to_nx(m5, ['a', 'b'])) == d5

m6 = [[7,0,0],
 [7,9,0],
 [0,7,0]]

d6 = { 'a':{
 'a' : {'weight':7,'label':'7'},
 },
 'b': {
 'a': {'weight':7,'label':'7'},
 'b' : {'weight':9,'label':'9'}
 },
 'c':{
 'b' : {'weight':7,'label':'7'}
 }
 }
```

(continues on next page)

(continued from previous page)

```

 }
 }

assert nx.to_dict_of_dicts(mat_to_nx(m6, ['a', 'b', 'c'])) == d6

```

&lt;/div&gt;

```
[56]: def mat_to_nx(mat, ids):
 raise Exception('TODO IMPLEMENT ME !')

try:
 mat_ids_to_adj([[0, 3]], ['a', 'b'])
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

try:
 mat_ids_to_adj([[0]], ['a', 'b'])
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

m1 = [[0]]

d1 = {'a': {}}

assert nx.to_dict_of_dicts(mat_to_nx(m1, ['a'])) == d1

m2 = [[7]]

d2 = {'a': {'a': {'weight': 7, 'label': '7'}}}
assert nx.to_dict_of_dicts(mat_to_nx(m2, ['a'])) == d2

m3 = [[0, 0],
 [0, 0]]

d3 = { 'a':{},
 'b':{}
 }
assert nx.to_dict_of_dicts(mat_to_nx(m3, ['a', 'b'])) == d3

m4 = [[7, 9],
 [8, 6]]

d4 = { 'a':{'a': {'weight':7,'label':'7'},
 'b' : {'weight':9,'label':'9'},
 },
 'b':{'a': {'weight':8,'label':'8'},
 'b' : {'weight':6,'label':'6'},
 }
 }

assert nx.to_dict_of_dicts(mat_to_nx(m4, ['a', 'b'])) == d4
```

(continues on next page)

(continued from previous page)

```
m5 = [[0,0],
 [0,7]]

d5 = { 'a':{},
 'b':{
 'b' : {'weight':7,'label':'7'},
 }
 }

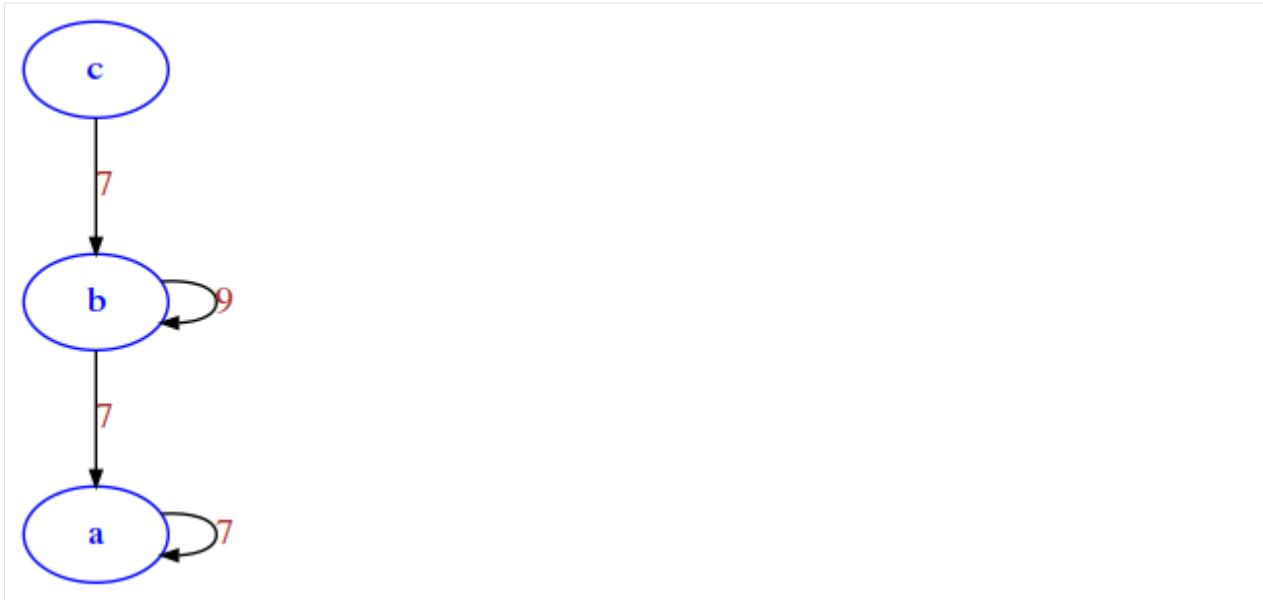
assert nx.to_dict_of_dicts(mat_to_nx(m5,['a','b'])) == d5

m6 = [[7,0,0],
 [7,9,0],
 [0,7,0]]

d6 = { 'a':{
 'a' : {'weight':7,'label':'7'},
 },
 'b': {
 'a': {'weight':7,'label':'7'},
 'b' : {'weight':9,'label':'9'}
 },
 'c':{
 'b' : {'weight':7,'label':'7'}
 }
 }

assert nx.to_dict_of_dicts(mat_to_nx(m6,['a','b','c'])) == d6
```

```
[57]: draw_nx(mat_to_nx([
 [7,0,0],
 [7,9,0],
 [0,7,0]
], ['a','b','c']))
```



## Continue

Go on with [binary relations](#)<sup>407</sup>

### 8.4.2 Relational data 2 - binary relations

[Download exercises zip](#)

[Browse files online](#)<sup>408</sup>

We can use graphs to model relations of many kinds, like *isCloseTo*, *isFriendOf*, *loves*, etc. Here we review some of them and their properties.

**Before going on, make sure to have read the first tutorial Relational data**<sup>409</sup>

## What to do

- unzip exercises in a folder, you should get something like this:

```

relational
 relational1-intro.ipynb
 relational1-intro-sol.ipynb
 relational2-binrel.ipynb
 relational2-binrel-sol.ipynb
 relational3-simple-stats.ipynb
 relational3-simple-stats-sol.ipynb
 relational4-chal.ipynb
 jupman.py
 soft.py

```

<sup>407</sup> <https://en.softpython.org/relational/relational2-binrel-sol.html>

<sup>408</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/relational>

<sup>409</sup> <https://en.softpython.org/relational/relational1-intro-sol.html>

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook relational/relational2-binrel.ipynb

**WARNING 2:** DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

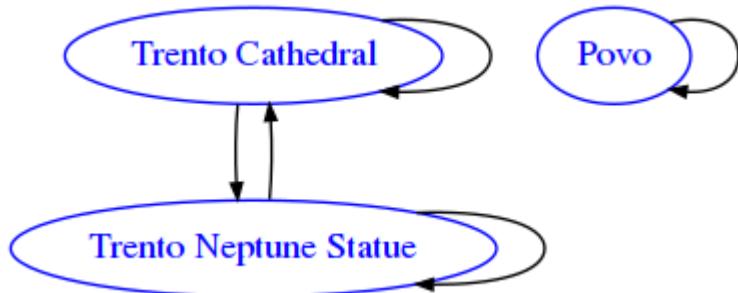
### Reflexive relations

A graph is reflexive when each node links to itself.

In real life, the typical reflexive relation could be “is close to”, supposing “close to” means being within a 100 meters distance. Obviously, any place is always close to itself, let’s see an example (Povo is a small town around Trento):

```
[2]: from soft import draw_adj

draw_adj({
 'Trento Cathedral' : ['Trento Cathedral', 'Trento Neptune Statue'],
 'Trento Neptune Statue' : ['Trento Neptune Statue', 'Trento Cathedral'],
 'Povo' : ['Povo'],
})
```



Some relations might not always be necessarily reflexive, like “did homeworks for”. You should always do your own homeworks, but to our dismay, university intelligence services caught some of you cheating. In the following example we expose the situation - due to privacy concerns, we identify students with numbers starting from zero included:

```
[3]: from soft import draw_mat

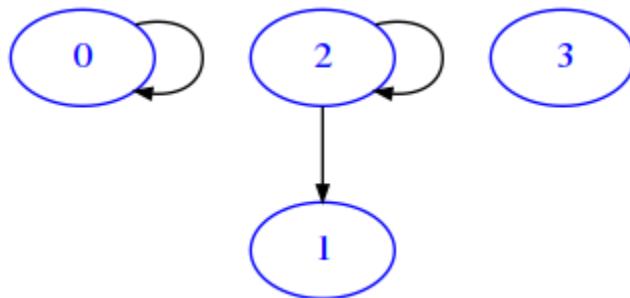
draw_mat(
 [
 [True, False, False, False],
 [False, True, False, False],
 [False, False, True, False],
 [False, False, False, True]
])
```

(continues on next page)

(continued from previous page)

```
[False, False, False, False],
[False, True, True, False],
[False, False, False, False],
]
```

)



From the graph above, we see student 0 and student 2 both did their own homeworks. Student 3 did no homeworks at all. Alarmingly, we notice student 2 did the homeworks for student 1. Resulting conspiracy shall be severely punished with a one year ban from having spritz at Emma's bar.

### Exercise - is\_reflexive\_mat

⊕⊕ Implement a function that RETURN `True` if  $n \times n$  boolean matrix `mat` as list of lists is reflexive, `False` otherwise.

A graph is *reflexive* when all nodes point to themselves.

- Please at least try to make the function efficient

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[4]: def is_reflexive_mat(mat):

 n = len(mat)
 for i in range(n):
 if not mat[i][i]:
 return False
 return True

assert is_reflexive_mat([[False]]) == False # m1
assert is_reflexive_mat([[True]]) == True # m2

assert is_reflexive_mat([[False, False],
 [False, False]]) == False # m3

assert is_reflexive_mat([[True, True],
 [True, True]]) == True # m4

assert is_reflexive_mat([[True, True],
 [False, True]]) == True # m5

assert is_reflexive_mat([[True, False],
 [True, True]]) == True # m6
```

(continues on next page)

(continued from previous page)

```
assert is_reflexive_mat([[True, True],
 [True, False]]) == False # m7

assert is_reflexive_mat([[False, True],
 [True, True]]) == False # m8

assert is_reflexive_mat([[False, True],
 [True, False]]) == False # m9

assert is_reflexive_mat([[False, False],
 [True, False]]) == False # m10

assert is_reflexive_mat([[False, True, True],
 [True, False, False],
 [True, True, True]]) == False # m11

assert is_reflexive_mat([[True, True, True],
 [True, True, True],
 [True, True, True]]) == True # m12
```

&lt;/div&gt;

```
[4]: def is_reflexive_mat(mat):
 raise Exception('TODO IMPLEMENT ME !')

assert is_reflexive_mat([[False]]) == False # m1
assert is_reflexive_mat([[True]]) == True # m2

assert is_reflexive_mat([[False, False],
 [False, False]]) == False # m3

assert is_reflexive_mat([[True, True],
 [True, True]]) == True # m4

assert is_reflexive_mat([[True, True],
 [False, True]]) == True # m5

assert is_reflexive_mat([[True, False],
 [True, True]]) == True # m6

assert is_reflexive_mat([[True, True],
 [True, False]]) == False # m7

assert is_reflexive_mat([[False, True],
 [True, True]]) == False # m8

assert is_reflexive_mat([[False, True],
 [True, False]]) == False # m9

assert is_reflexive_mat([[False, False],
 [True, False]]) == False # m10

assert is_reflexive_mat([[False, True, True],
 [True, False, False],
 [True, True, True]]) == False # m11
```

(continues on next page)

(continued from previous page)

```
assert is_reflexive_mat([[True, True, True],
 [True, True, True],
 [True, True, True]]) == True # m12
```

**Exercise - is\_reflexive\_adj**

⊕⊕ Implement now the same function for dictionaries of adjacency lists:

RETURN `True` if provided graph as dictionary of adjacency lists is reflexive, `False` otherwise.

- A graph is *reflexive* when all nodes point to themselves.
- Please at least try to make the function efficient.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[5]: def is_reflexive_adj(d):

 for v in d:
 if not v in d[v]:
 return False
 return True

assert is_reflexive_adj({ 'a':[] }) == False # d1
assert is_reflexive_adj({ 'a':['a'] }) == True # d2

assert is_reflexive_adj({ 'a':[],
 'b':[]
 }) == False # d3

assert is_reflexive_adj({ 'a':['a'],
 'b':['b']
 }) == True # d4

assert is_reflexive_adj({ 'a':['a','b'],
 'b':['b']
 }) == True # d5

assert is_reflexive_adj({ 'a':['a'],
 'b':['a','b']
 }) == True # d6

assert is_reflexive_adj({ 'a':['a','b'],
 'b':['a']
 }) == False # d7

assert is_reflexive_adj({ 'a':['b'],
 'b':['a','b']
 }) == False # d8

assert is_reflexive_adj({ 'a':['b'],
 'b':['a']
 })
```

(continues on next page)

(continued from previous page)

```
 }) == False # d9

assert is_reflexive_adj({ 'a':[],
 'b':['a']
 }) == False # d10

assert is_reflexive_adj({ 'a':['b','c'],
 'b':['a'],
 'c':['a','b','c']
 }) == False # d11

assert is_reflexive_adj({ 'a':['a','b','c'],
 'b':['a','b','c'],
 'c':['a','b','c']
 }) == True # d12
```

&lt;/div&gt;

```
[5]: def is_reflexive_adj(d):
 raise Exception('TODO IMPLEMENT ME !')

assert is_reflexive_adj({ 'a':[] }) == False # d1
assert is_reflexive_adj({ 'a':['a'] }) == True # d2

assert is_reflexive_adj({ 'a':[],
 'b':[]
 }) == False # d3

assert is_reflexive_adj({ 'a':['a'],
 'b':['b']
 }) == True # d4

assert is_reflexive_adj({ 'a':['a','b'],
 'b':['b']
 }) == True # d5

assert is_reflexive_adj({ 'a':['a'],
 'b':['a','b']
 }) == True # d6

assert is_reflexive_adj({ 'a':['a','b'],
 'b':['a']
 }) == False # d7

assert is_reflexive_adj({ 'a':['b'],
 'b':['a','b']
 }) == False # d8

assert is_reflexive_adj({ 'a':['b'],
 'b':['a']
 }) == False # d9

assert is_reflexive_adj({ 'a':[],
 'b':['a']
 }) == False # d10
```

(continues on next page)

(continued from previous page)

```
assert is_reflexive_adj({ 'a':['b','c'],
 'b':['a'],
 'c':['a','b','c']
 }) == False # d11

assert is_reflexive_adj({ 'a':['a','b','c'],
 'b':['a','b','c'],
 'c':['a','b','c']
 }) == True # d12
```

## Symmetric relations

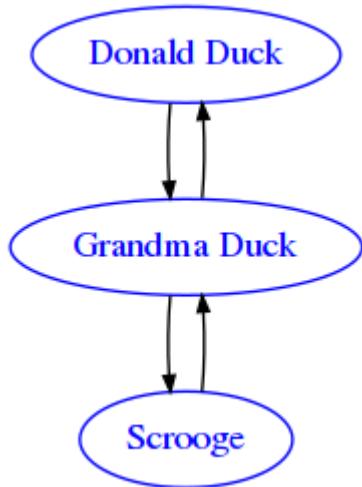
A graph is symmetric when for all nodes, if a node A links to another node B, there is also a link from node B to A.

In real life, the typical symmetric relation is “is friend of”. If you are friend to someone, that someone should be also be your friend.

For example, since Scrooge typically is not so friendly with his lazy nephew Donald Duck, but certainly both Scrooge and Donald Duck enjoy visiting the farm of Grandma Duck, we can model their friendship relation like this:

```
[6]: from soft import draw_adj

draw_adj({
 'Donald Duck' : ['Grandma Duck'],
 'Scrooge' : ['Grandma Duck'],
 'Grandma Duck' : ['Scrooge', 'Donald Duck'],
})
```



Not that Scrooge is not linked to Donald Duck, but this does not mean the whole graph cannot be considered symmetric. If you pay attention to the definition above, there is *if* written at the beginning: *if* a node A links to another node B, there is also a link from node B to A.

**QUESTION:** Looking purely at the above definition (so do *not* consider ‘is friend of’ relation), should a symmetric relation be necessarily reflexive?

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol" jupman-sol-question" style="display:none">

**ANSWER:** No, in a symmetric relation some nodes can be linked to themselves, while some other nodes may have no link to themselves. All we care about to check symmetry is links from a node to *other* nodes.

</div>

**QUESTION:** Think about the semantics of the specific “is friend of” relation: can you think of a social network where the relation is not shown as reflexive?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** In the particular case of “is friend to” relation is interesting, as it prompts us to think about the semantic meaning of the relation: obviously, everybody *should* be a friend of himself/herself - but if were to implement say a social network service like Facebook, it would look rather useless to show in your friends list the information that you are a friend of yourself.

</div>

**QUESTION:** Always talking about the specific semantics of “is friend of” relation: can you think about some case where it should be meaningful to store information about individuals *not* being friends of themselves ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

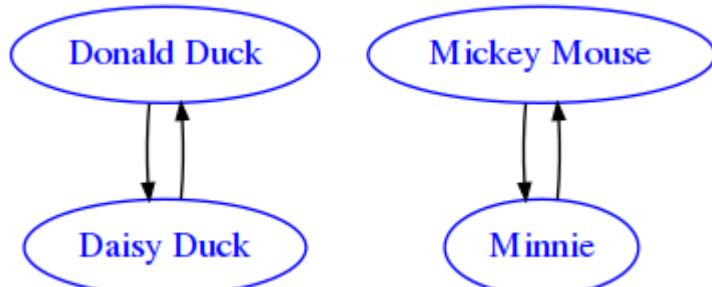
**ANSWER:** in real life it may always happen to find fringe cases - suppose you are given the task to model a network of possibly depressed people with self-harming tendencies. So always be sure your model correctly fits the problem at hand.

</div>

Some relations sometimes may or not be symmetric, depending on the graph at hand. Think about the relation *loves*. It is well known that Mickey Mouse loves Minnie and the sentiment is reciprocal, and Donald Duck loves Daisy Duck and the sentiment is reciprocal. We can conclude this particular graph is symmetrical:

[7]: `from soft import draw_adj`

```
draw_adj({
 'Donald Duck' : ['Daisy Duck'],
 'Daisy Duck' : ['Donald Duck'],
 'Mickey Mouse' : ['Minnie'],
 'Minnie' : ['Mickey Mouse']
})
```



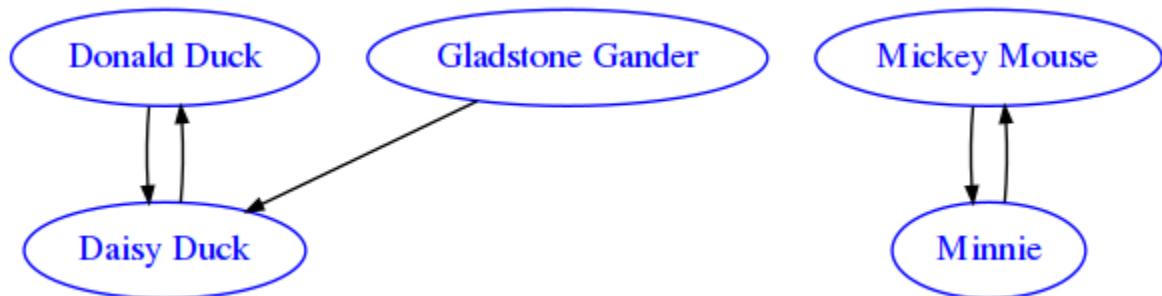
But what about this one? Donald Duck is not the only duck in town and sometimes a contender shows up: Gladstone Gander<sup>410</sup> (Gastone in Italian) also would like the attention of Daisy ( never mind in some comics he actually gets it when Donald Duck messes up big time):

<sup>410</sup> [https://en.wikipedia.org/wiki/Gladstone\\_Gander](https://en.wikipedia.org/wiki/Gladstone_Gander)

```
[8]: from soft import draw_adj

draw_adj({
 'Donald Duck' : ['Daisy Duck'],
 'Daisy Duck' : ['Donald Duck'],
 'Mickey Mouse' : ['Minnie'],
 'Minnie' : ['Mickey Mouse'],
 'Gladstone Gander' : ['Daisy Duck']

})
```



### Exercise - is\_symmetric\_mat

⊕⊕ Implement an automated procedure to check whether or not a graph is symmetrical, which given a matrix as a list of lists that RETURN True if nxn boolean matrix mat as list of lists is symmetric, False otherwise.

- A graph is symmetric when for all nodes, if a node A links to another node B, there is also a link from node B to A.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[9]: def is_symmetric_mat(mat):

 n = len(mat)
 for i in range(n):
 for j in range(n):
 if mat[i][j] and not mat[j][i]:
 return False
 return True

assert is_symmetric_mat([[False]]) == True # m1
assert is_symmetric_mat([[True]]) == True # m2

assert is_symmetric_mat([[False, False],
 [False, False]]) == True # m3

assert is_symmetric_mat([[True, True],
 [True, True]]) == True # m4

assert is_symmetric_mat([[True, True],
 [False, True]]) == False # m5
```

(continues on next page)

(continued from previous page)

```
assert is_symmetric_mat([[True, False],
 [True, True]]) == False # m6

assert is_symmetric_mat([[True, True],
 [True, False]]) == True # m7

assert is_symmetric_mat([[False, True],
 [True, True]]) == True # m8

assert is_symmetric_mat([[False, True],
 [True, False]]) == True # m9

assert is_symmetric_mat([[False, False],
 [True, False]]) == False # m10

assert is_symmetric_mat([[False, True, True],
 [True, False, False],
 [True, True, True]]) == False # m11

assert is_symmetric_mat([[False, True, True],
 [True, False, True],
 [True, True, True]]) == True # m12
```

&lt;/div&gt;

[9]:

```
def is_symmetric_mat(mat):
 raise Exception('TODO IMPLEMENT ME !')

assert is_symmetric_mat([[False]]) == True # m1
assert is_symmetric_mat([[True]]) == True # m2

assert is_symmetric_mat([[False, False],
 [False, False]]) == True # m3

assert is_symmetric_mat([[True, True],
 [True, True]]) == True # m4

assert is_symmetric_mat([[True, True],
 [False, True]]) == False # m5

assert is_symmetric_mat([[True, False],
 [True, True]]) == False # m6

assert is_symmetric_mat([[True, True],
 [True, False]]) == True # m7

assert is_symmetric_mat([[False, True],
 [True, True]]) == True # m8

assert is_symmetric_mat([[False, True],
 [True, False]]) == True # m9

assert is_symmetric_mat([[False, False],
 [True, False]]) == False # m10

assert is_symmetric_mat([[False, True, True],
```

(continues on next page)

(continued from previous page)

```
[True, False, False],
[True, True, True]]) == False # m11

assert is_symmetric_mat([[False, True, True],
 [True, False, True],
 [True, True, True]]) == True # m12
```

**Exercise - is\_symmetric\_adj**

⊕⊕ Now implement the same as before but for a dictionary of adjacency lists:

RETURN `True` if given dictionary of adjacency lists is symmetric, `False` otherwise.

- Assume all the nodes are represented in the keys.
- A graph is symmetric when for all nodes, if a node A links to another node B, there is also a link from node B to A.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[10]:

```
def is_symmetric_adj(d):

 for k in d:
 for v in d[k]:
 if not k in d[v]:
 return False
 return True

assert is_symmetric_adj({ 'a':[] }) == True # d1
assert is_symmetric_adj({ 'a':['a'] }) == True # d2

assert is_symmetric_adj({ 'a' : [],
 'b' : []
 }) == True # d3

assert is_symmetric_adj({ 'a' : ['a','b'],
 'b' : ['a','b']
 }) == True # d4

assert is_symmetric_adj({ 'a' : ['a','b'],
 'b' : ['b']
 }) == False # d5

assert is_symmetric_adj({ 'a' : ['a'],
 'b' : ['a','b']
 }) == False # d6

assert is_symmetric_adj({ 'a' : ['a','b'],
 'b' : ['a']
 }) == True # d7

assert is_symmetric_adj({ 'a' : ['b'],
 'b' : ['a','b']
 })
```

(continues on next page)

(continued from previous page)

```

 }) == True # d8

assert is_symmetric_adj({ 'a' : ['b'],
 'b' : ['a']
 }) == True # d9

assert is_symmetric_adj({ 'a' : [],
 'b' : ['a']
 }) == False # d10

assert is_symmetric_adj({ 'a' : ['b', 'c'],
 'b' : ['a'],
 'c' : ['a', 'b', 'c']
 }) == False # d11

assert is_symmetric_adj({ 'a' : ['b', 'c'],
 'b' : ['a', 'c'],
 'c' : ['a', 'b', 'c']
 }) == True # d12

```

&lt;/div&gt;

[10]:

```

def is_symmetric_adj(d):
 raise Exception('TODO IMPLEMENT ME !')

assert is_symmetric_adj({ 'a':[] }) == True # d1
assert is_symmetric_adj({ 'a':['a'] }) == True # d2

assert is_symmetric_adj({ 'a' : [],
 'b' : []
 }) == True # d3

assert is_symmetric_adj({ 'a' : ['a', 'b'],
 'b' : ['a', 'b']
 }) == True # d4

assert is_symmetric_adj({ 'a' : ['a', 'b'],
 'b' : ['b']
 }) == False # d5

assert is_symmetric_adj({ 'a' : ['a'],
 'b' : ['a', 'b']
 }) == False # d6

assert is_symmetric_adj({ 'a' : ['a', 'b'],
 'b' : ['a']
 }) == True # d7

assert is_symmetric_adj({ 'a' : ['b'],
 'b' : ['a', 'b']
 }) == True # d8

assert is_symmetric_adj({ 'a' : ['b'],
 'b' : ['a']
 }) == True # d9

```

(continues on next page)

(continued from previous page)

```
assert is_symmetric_adj({ 'a' : [],
 'b' : ['a']
 }) == False # d10

assert is_symmetric_adj({ 'a' : ['b', 'c'],
 'b' : ['a'],
 'c' : ['a', 'b', 'c']
 }) == False # d11

assert is_symmetric_adj({ 'a' : ['b', 'c'],
 'b' : ['a', 'c'],
 'c' : ['a', 'b', 'c']
 }) == True # d12
```

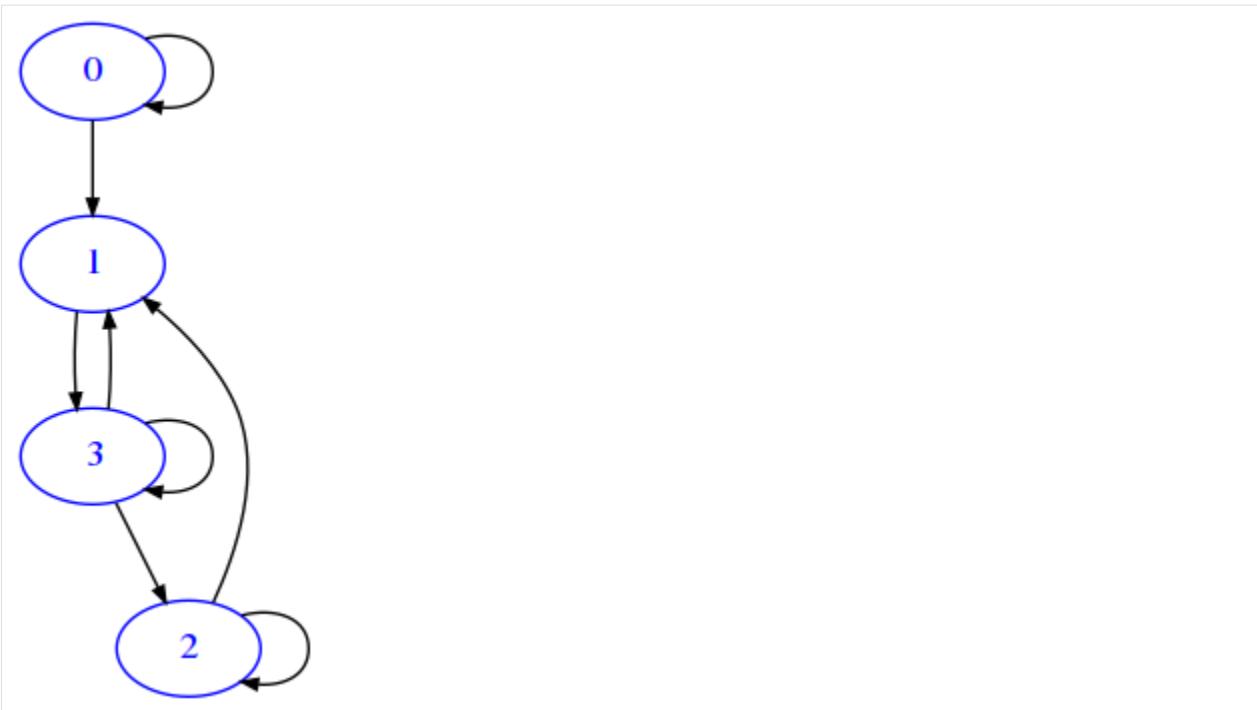
## Surjective relations

If we consider a graph as a nxn binary relation where the domain is the same as the codomain, such relation is called *surjective* if every node is reached by *at least* one edge.

For example, G1 here is surjective, because there is at least one edge reaching into each node (self-loops as in 0 node also count as incoming edges)

```
[11]: G1 = [
 [True, True, False, False],
 [False, False, False, True],
 [False, True, True, False],
 [False, True, True, True],
```

```
[12]: draw_mat(G1)
```



G2 down here instead does not represent a surjective relation, as there is *at least* one node ( 2 in our case) which does not have any incoming edge:

```
[13]: G2 = [
 [True, True, False, False],
 [False, False, False, True],
 [False, True, False, False],
 [False, True, False, False],
]
```

```
[14]: draw_mat(G2)
```



### Exercise - surjective

⊗⊗ RETURN True if provided graph mat as list of boolean lists is an nxn surjective binary relation, otherwise return False

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[15]: def surjective(mat):

 n = len(mat)
 c = 0 # number of incoming edges found
 for j in range(len(mat)): # go column by column
 for i in range(len(mat)): # go row by row
 if mat[i][j]:
 c += 1
 break # as you find first incoming edge, increment c and stop
 ↪search for that column
 return c == n

m1 = [[False]]
assert surjective(m1) == False

m2 = [[True]]
assert surjective(m2) == True

m3 = [[True, False],
 [False, False]]
assert surjective(m3) == False

m4 = [[False, True],
 [False, False]]
assert surjective(m4) == False

m5 = [[False, False],
 [True, False]]
assert surjective(m5) == False

m6 = [[False, False],
 [False, True]]
assert surjective(m6) == False

m7 = [[True, False],
 [True, False]]
assert surjective(m7) == False

m8 = [[True, False],
 [False, True]]
assert surjective(m8) == True

m9 = [[True, True],
```

(continues on next page)

(continued from previous page)

```
[False, True]]
assert surjective(m9) == True

m10 = [[True, True, False, False],
 [False, False, False, True],
 [False, True, False, False],
 [False, True, False, False]]
assert surjective(m10) == False

m11 = [[True, True, False, False],
 [False, False, False, True],
 [False, True, True, False],
 [False, True, True, True]]
assert surjective(m11) == True
```

&lt;/div&gt;

```
[15]: def surjective(mat):
 raise Exception('TODO IMPLEMENT ME !')
```

```
m1 = [[False]]
assert surjective(m1) == False
```

```
m2 = [[True]]
assert surjective(m2) == True
```

```
m3 = [[True, False],
 [False, False]]
assert surjective(m3) == False
```

```
m4 = [[False, True],
 [False, False]]
assert surjective(m4) == False
```

```
m5 = [[False, False],
 [True, False]]
assert surjective(m5) == False
```

```
m6 = [[False, False],
 [False, True]]
assert surjective(m6) == False
```

```
m7 = [[True, False],
 [True, False]]
assert surjective(m7) == False
```

```
m8 = [[True, False],
 [False, True]]
assert surjective(m8) == True
```

(continues on next page)

(continued from previous page)

```
m9 = [[True, True],
 [False, True]]
assert surjective(m9) == True

m10 = [[True, True, False, False],
 [False, False, False, True],
 [False, True, False, False],
 [False, True, False, False]]
assert surjective(m10) == False

m11 = [[True, True, False, False],
 [False, False, False, True],
 [False, True, True, False],
 [False, True, True, True]]
assert surjective(m11) == True
```

## Further resources

- Rule based design<sup>411</sup> by Lex Wedemeijer, Stef Joosten, Jaap van der woude: a very readable text on how to represent information using only binary relations with boolean matrices. This a theoretical book with no python exercise so it is not a mandatory read, it only gives context and practical applications for some of the material on graphs presented during the course

## Continue

Go on with simple statistics<sup>412</sup>

### 8.4.3 Relational data 3 - simple statistics

#### Download exercises zip

Browse files online<sup>413</sup>

We will now compute simple statistics about graphs (they don't require node discovery algorithms).

#### What to do

- unzip exercises in a folder, you should get something like this:

```
relational
 relational1-intro.ipynb
 relational1-intro-sol.ipynb
 relational2-binrel.ipynb
 relational2-binrel-sol.ipynb
 relational3-simple-stats.ipynb
 relational3-simple-stats-sol.ipynb
```

(continues on next page)

<sup>411</sup> [https://www.researchgate.net/profile/Stef\\_Joosten/publication/327022933\\_Rule\\_Based\\_Design/links/5b7321be45851546c903234a/Rule-Based-Design.pdf](https://www.researchgate.net/profile/Stef_Joosten/publication/327022933_Rule_Based_Design/links/5b7321be45851546c903234a/Rule-Based-Design.pdf)

<sup>412</sup> <https://en.softpython.org/relational/relational3-simple-stats-sol.html>

<sup>413</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/relational>

(continued from previous page)

```
relational4-chal.ipynb
jupman.py
soft.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook relational/relational3-simple-stats.ipynb

**WARNING 2:** DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

- Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### Outdegrees and indegrees

The *out-degree*  $\deg^+(v)$  of a node  $v$  is the number of edges going out from it, while the *in-degree*  $\deg^-(v)$  is the number of edges going into it.

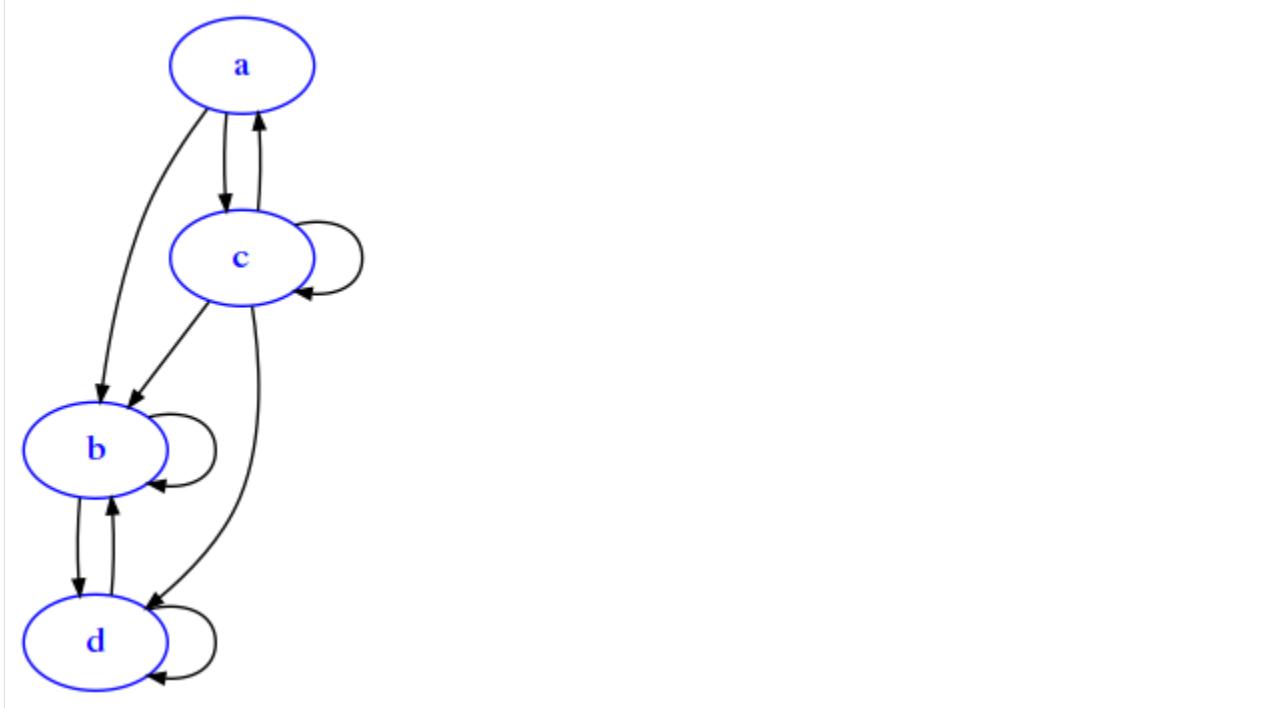
*NOTE:* the out-degree and in-degree are *not* the sum of weights ! They just count presence or absence of edges.

For example, consider this graph:

```
[2]: from soft import draw_adj

d = {
 'a' : ['b', 'c'],
 'b' : ['b', 'd'],
 'c' : ['a', 'b', 'c', 'd'],
 'd' : ['b', 'd']
}

draw_adj(d)
```



The out-degree of d is 2, because it has one outgoing edge to b but also an outgoing edge to itself. The indegree of d is 3, because it has an edge coming from b, one from c and one self-loop from d itself.

### Exercise - outdegree\_adj

⊕ RETURN the outdegree of a node from graph d represented as a dictionary of adjacency lists

- If v is not a vertex of d, raise ValueError

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: def outdegree_adj(d, v):

 if v not in d:
 raise ValueError("Vertex %s is not in %s" % (v, d))

 return len(d[v])

try:
 outdegree_adj({'a':[], 'b'})
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

d1 = { 'a':[] }
assert outdegree_adj(d1, 'a') == 0

d2 = { 'a':['a'] }
assert outdegree_adj(d2, 'a') == 1
```

(continues on next page)

(continued from previous page)

```
d3 = { 'a':['a','b'],
 'b':[] }
assert outdegree_adj(d3, 'a') == 2

d4 = { 'a':['a','b'],
 'b':['a','b','c'],
 'c':[] }
assert outdegree_adj(d4, 'b') == 3
```

&lt;/div&gt;

```
[3]: def outdegree_adj(d, v):
 raise Exception('TODO IMPLEMENT ME !')

 try:
 outdegree_adj({ 'a':[] }, 'b')
 raise Exception("SHOULD HAVE FAILED !")
 except ValueError:
 "passed test"

d1 = { 'a':[] }
assert outdegree_adj(d1, 'a') == 0

d2 = { 'a':['a'] }
assert outdegree_adj(d2, 'a') == 1

d3 = { 'a':['a','b'],
 'b':[] }
assert outdegree_adj(d3, 'a') == 2

d4 = { 'a':['a','b'],
 'b':['a','b','c'],
 'c':[] }
assert outdegree_adj(d4, 'b') == 3
```

### Exercise - outdegree\_mat

⊕ RETURN the outdegree of a node  $i$  from a graph boolean matrix  $n \times n$  represented as a list of lists

- If  $i$  is not a node of the graph, raise `ValueError`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[4]: def outdegree_mat(mat, i):

 n = len(mat)
 if i < 0 or i > n:
 raise ValueError("i %s is not a row of matrix %s" % (i, mat))
 ret = 0
 for j in range(n):
 if mat[i][j]:
 ret += 1
 return ret
```

(continues on next page)

(continued from previous page)

```

try:
 outdegree_mat([[False]], 7)
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

try:
 outdegree_mat([[False]], -1)
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

m1 = [[False]]
assert outdegree_mat(m1, 0) == 0

m2 = [[True]]
assert outdegree_mat(m2, 0) == 1

m3 = [[True, True],
 [False, False]]
assert outdegree_mat(m3, 0) == 2

m4 = [[True, True, False],
 [True, True, True],
 [False, False, False]]
assert outdegree_mat(m4,1) == 3

```

&lt;/div&gt;

```

[4]: def outdegree_mat(mat, i):
 raise Exception('TODO IMPLEMENT ME !')

try:
 outdegree_mat([[False]], 7)
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

try:
 outdegree_mat([[False]], -1)
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

m1 = [[False]]
assert outdegree_mat(m1, 0) == 0

m2 = [[True]]
assert outdegree_mat(m2, 0) == 1

m3 = [[True, True],
 [False, False]]
assert outdegree_mat(m3, 0) == 2

m4 = [[True, True, False],
 [True, True, True],
 [False, False, False]]

```

(continues on next page)

(continued from previous page)

```
[False, False, False]]
assert outdegree_mat(m4, 1) == 3
```

### Exercise - outdegree\_avg

⊕⊕ RETURN the average outdegree of nodes in graph d, represented as dictionary of adjacency lists.

- Assume all nodes are in the keys.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[5]: def outdegree_avg(d):

 s = 0
 for k in d:
 s += len(d[k])
 return s / len(d)

d1 = { 'a':[] }
assert outdegree_avg(d1) == 0

d2 = { 'a':['a'] }
assert round(outdegree_avg(d2), 2) == 1.00 / 1.00

d3 = { 'a':['a','b'],
 'b':[] }
assert round(outdegree_avg(d3), 2) == (2 + 0) / 2

d4 = { 'a':['a','b'],
 'b':['a','b','c'],
 'c':[] }
assert round(outdegree_avg(d4), 2) == round((2 + 3) / 3 , 2)
```

</div>

```
[5]: def outdegree_avg(d):
 raise Exception('TODO IMPLEMENT ME !')

d1 = { 'a':[] }
assert outdegree_avg(d1) == 0

d2 = { 'a':['a'] }
assert round(outdegree_avg(d2), 2) == 1.00 / 1.00

d3 = { 'a':['a','b'],
 'b':[] }
assert round(outdegree_avg(d3), 2) == (2 + 0) / 2

d4 = { 'a':['a','b'],
 'b':['a','b','c'],
 'c':[] }
assert round(outdegree_avg(d4), 2) == round((2 + 3) / 3 , 2)
```

### Exercise - indegree\_adj

The indegree of a node  $v$  is the number of edges going into it.

⊕⊕ RETURN the indegree of node  $v$  in graph  $d$ , represented as a dictionary of adjacency lists

- If  $v$  is not a node of the graph, raise `ValueError`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[6]: def indegree_adj(d, v):

 if v not in d:
 raise ValueError("Vertex %s is not in %s" % (v, d))
 ret = 0
 for k in d:
 if v in d[k]:
 ret += 1
 return ret

try:
 indegree_adj({'a':[],'b'})
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

d1 = { 'a':[] }
assert indegree_adj(d1, 'a') == 0

d2 = {'a':['a']}
assert indegree_adj(d2, 'a') == 1

d3 = { 'a':['a','b'],
 'b':[]}
assert indegree_adj(d3, 'a') == 1

d4 = { 'a':['a','b'],
 'b':['a','b','c'],
 'c':[]}
assert indegree_adj(d4, 'b') == 2
```

</div>

```
[6]: def indegree_adj(d, v):
 raise Exception('TODO IMPLEMENT ME !')

try:
 indegree_adj({'a':[],'b'})
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

d1 = { 'a':[] }
assert indegree_adj(d1, 'a') == 0

d2 = {'a':['a']}
```

(continues on next page)

(continued from previous page)

```

assert indegree_adj(d2, 'a') == 1

d3 = { 'a':['a','b'],
 'b':[]}
assert indegree_adj(d3, 'a') == 1

d4 = { 'a':['a','b'],
 'b':['a','b','c'],
 'c':[]}
assert indegree_adj(d4, 'b') == 2

```

**Exercise - indegree\_mat**

⊕ RETURN the indegree of a node  $i$  from a graph boolean matrix  $nxn$  represented as a list of lists

- If  $i$  is not a node of the graph, raise ValueError

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[7]: def indegree_mat(mat, i):

 n = len(mat)
 if i < 0 or i > n:
 raise ValueError("i %s is not a row of matrix %s" % (i, mat))
 ret = 0
 for k in range(n):
 if mat[k][i]:
 ret += 1
 return ret

try:
 indegree_mat([[False]], 7)
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

assert indegree_mat(
 [
 [False]
],
 0) == 0

m1 = [[True]]
assert indegree_mat(m1, 0) == 1

m2 = [[True, True],
 [False, False]]
assert indegree_mat(m2, 0) == 1

m3 = [[True, True, False],
 [True, True, True],
 [False, False, False]]
assert indegree_mat(m3, 1) == 2
```

&lt;/div&gt;

```
[7]: def indegree_mat(mat, i):
 raise Exception('TODO IMPLEMENT ME !')

try:
 indegree_mat([[False]], 7)
 raise Exception("SHOULD HAVE FAILED !")
except ValueError:
 "passed test"

assert indegree_mat(
 [
 [False]
],
 0) == 0

m1 = [[True]]
assert indegree_mat(m1, 0) == 1

m2 = [[True, True],
 [False, False]]
assert indegree_mat(m2, 0) == 1

m3 = [[True, True, False],
 [True, True, True],
 [False, False, False]]
assert indegree_mat(m3, 1) == 2
```

### Exercise - indegree\_avg

⊕⊕ RETURN the average indegree of nodes in graph d, represented as dictionary of adjacency lists.

- Assume all nodes are in the keys

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[8]: def indegree_avg(d):

 s = 0
 for k in d:
 s += len(d[k])
 return s / len(d)

d1 = { 'a':[] }
assert indegree_avg(d1) == 0

d2 = { 'a':['a'] }
assert round(indegree_avg(d2), 2) == 1.00 / 1.00

d3 = { 'a':['a','b'],
 'b':[] }
assert round(indegree_avg(d3), 2) == (1 + 1) / 2

d4 = { 'a':['a','b'],
```

(continues on next page)

(continued from previous page)

```
'b':['a','b','c'],
'c':[]}
assert round(indegree_avg(d4), 2) == round((2 + 2 + 1) / 3 , 2)
```

&lt;/div&gt;

```
[8]: def indegree_avg(d):
 raise Exception('TODO IMPLEMENT ME !')

d1 = { 'a':[] }
assert indegree_avg(d1) == 0

d2 = { 'a':['a'] }
assert round(indegree_avg(d2), 2) == 1.00 / 1.00

d3 = { 'a':['a','b'],
 'b':[]}
assert round(indegree_avg(d3), 2) == (1 + 1) / 2

d4 = { 'a':['a','b'],
 'b':['a','b','c'],
 'c':[]}
assert round(indegree_avg(d4), 2) == round((2 + 2 + 1) / 3 , 2)
```

## Was it worth it?

**QUESTION:** Is there any difference between the results of `indegree_avg` and `outdegree_avg` ?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** They give the same result. Think about what you did: for `outdegree_avg` you summed over all rows and then divided by n. For `indegree_avg` you summed over all columns, and then divided by n.

More formally, we have that the so-called *degree sum formula* holds (see [Wikipedia<sup>414</sup>](#) for more info):

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |A|$$

&lt;/div&gt;

## networkx Indegrees and outdegrees

With Networkx we can easily calculate indegrees and outdegrees of a node:

```
[9]: import networkx as nx

notice with networkx if nodes are already referenced to in an adjacency list
you do not need to put them as keys:

G=nx.DiGraph({
 'a':['b','c'], # node a links to b and c
 'b':['b','c', 'd'] # node b links to b itself, c and d
```

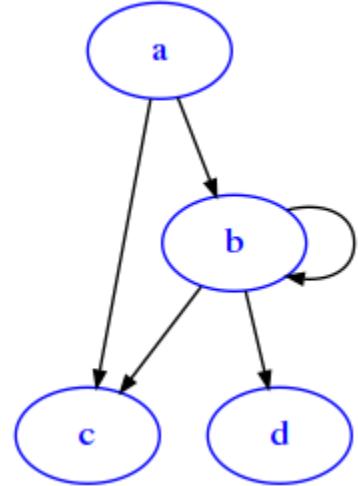
(continues on next page)

<sup>414</sup> [https://en.wikipedia.org/wiki/Directed\\_graph#Indegree\\_and\\_outdegree](https://en.wikipedia.org/wiki/Directed_graph#Indegree_and_outdegree)

(continued from previous page)

```
}
```

```
draw_nx(G)
```



```
[10]: G.out_degree('a')
```

```
[10]: 2
```

**QUESTION:** What is the outdegree of 'b'? Try to think about it and then confirm your thoughts with networkx:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[11]: # write here
#print("indegree b: %s" % G.in_degree('b'))
#print("outdegree b: %s" % G.out_degree('b'))
```

</div>

```
[11]: # write here
```

**QUESTION:** We defined *indegree* and *outdegree*. Can you guess what the *degree* might be? In particular, for a self pointing node like 'b', what could it be? Try to use `G.degree('b')` methods to validate your thoughts.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[12]: # write here
#print("degree b: %s" % G.degree('b'))
```

</div>

```
[12]: # write here
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

**ANSWER:** it is the sum of indegree and outdegree. In presence of a self-loop like for 'b', we count the self-loop twice, once as outgoing edge and one as incident edge

```
</div>
```

```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

```
[13]: # write here
#G.degree('b')
```

```
</div>
```

```
[13]: # write here
```

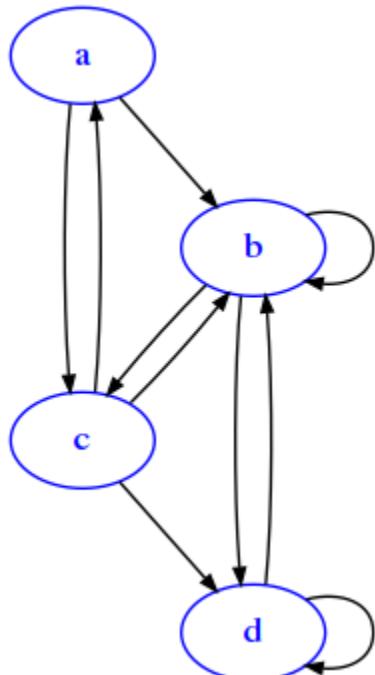
## Visualizing distributions

We will try to study the distributions visually. Let's take an example networkx DiGraph:

```
[14]: import networkx as nx

G1=nx.DiGraph({
 'a':['b','c'],
 'b':['b','c', 'd'],
 'c':['a','b','d'],
 'd':['b', 'd']
})

draw_nx(G1)
```



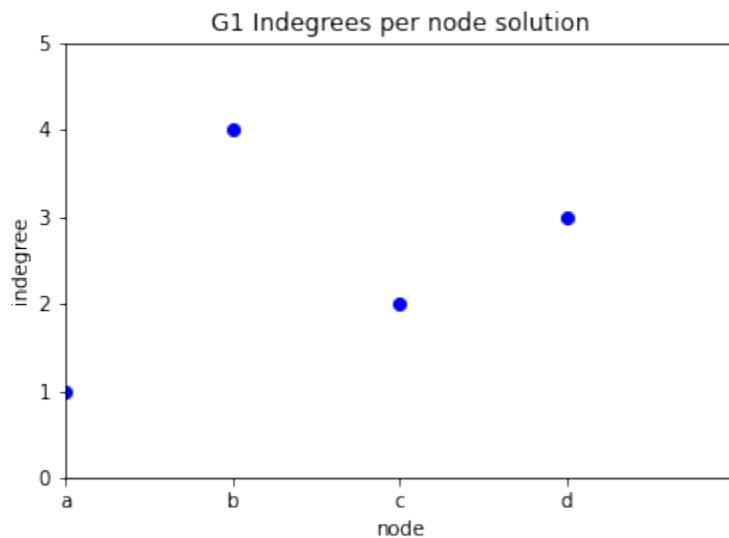
## indegree per node

⊕⊕ Display a plot for graph G where the xtick labels are the nodes, and the y is the indegree of those nodes.

**Note:** instead of `xticks` you might directly use categorical variables<sup>415</sup> IF you have matplotlib >= 2.1.0

Here we use `xticks` as sometimes you might need to fiddle with them anyway

Expected result:



To get the nodes, you can use the `G1.nodes()` function:

```
[15]: G1.nodes()
[15]: NodeView(('a', 'b', 'c', 'd'))
```

It gives back a `NodeView` which is not a list, but still you can iterate through it with a `for` in cycle:

```
[16]: for n in G1.nodes():
 print(n)
a
b
c
d
```

Also, you can get the indegree of a node with

```
[17]: G1.in_degree('b')
[17]: 4
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); " data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[18]: # write here
```

(continues on next page)

<sup>415</sup> [https://matplotlib.org/gallery/lines\\_bars\\_and\\_markers/categorical\\_variables.html](https://matplotlib.org/gallery/lines_bars_and_markers/categorical_variables.html)

(continued from previous page)

```
import numpy as np
import matplotlib.pyplot as plt

xs = np.arange(G1.number_of_nodes())
ys_in = [G1.in_degree(n) for n in G1.nodes()]

plt.plot(xs, ys_in, 'bo')

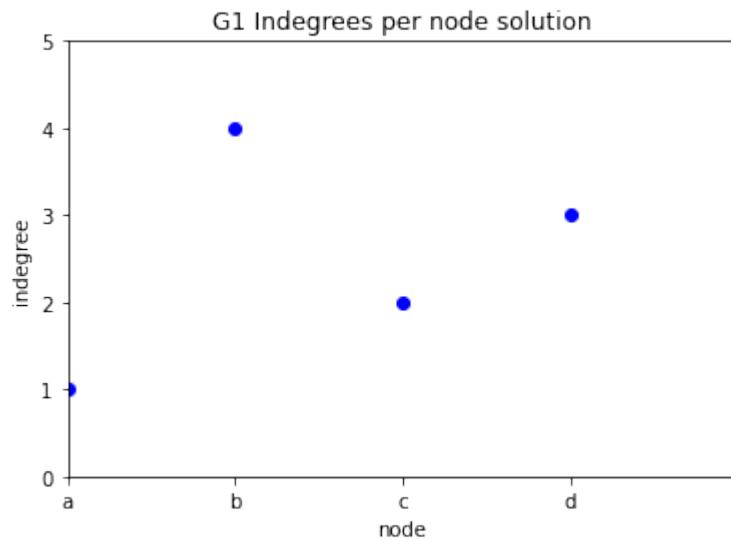
plt.ylim(0,max(ys_in) + 1)
plt.xlim(0,max(xs) + 1)

plt.title("G1 Indegrees per node solution")

plt.xticks(xs, G1.nodes())

plt.xlabel('node')
plt.ylabel('indegree')

plt.show()
```



&lt;/div&gt;

[18]:

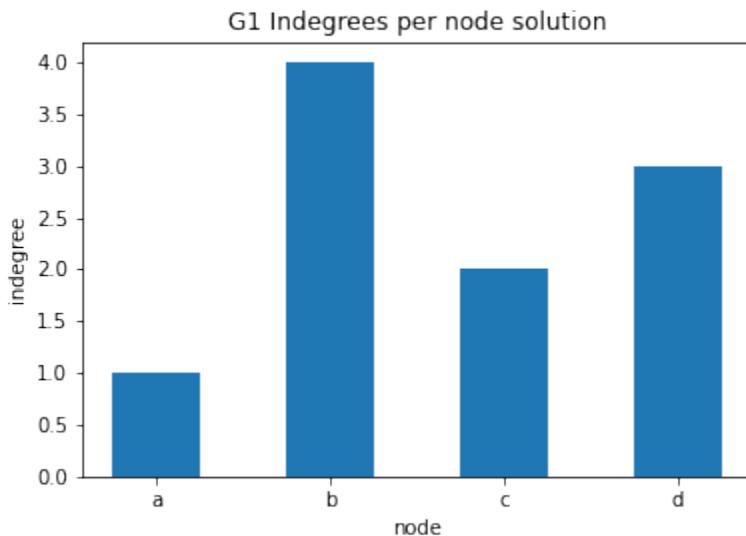
# write here

## indegree per node bar plot

The previous plot with dots doesn't look so good - we might try to use instead a bar plot. First look at [this example<sup>416</sup>](#), then proceed with the exercise

⊕⊕ Display a bar plot<sup>417</sup> for graph G1 where the xtick labels are the nodes, and the y is the indegree of those nodes.

Expected result:



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[19]:

```
write here

import numpy as np
import matplotlib.pyplot as plt

xs = np.arange(G1.number_of_nodes())
ys_in = [G1.in_degree(n) for n in G1.nodes()]

plt.bar(xs, ys_in, 0.5, align='center')

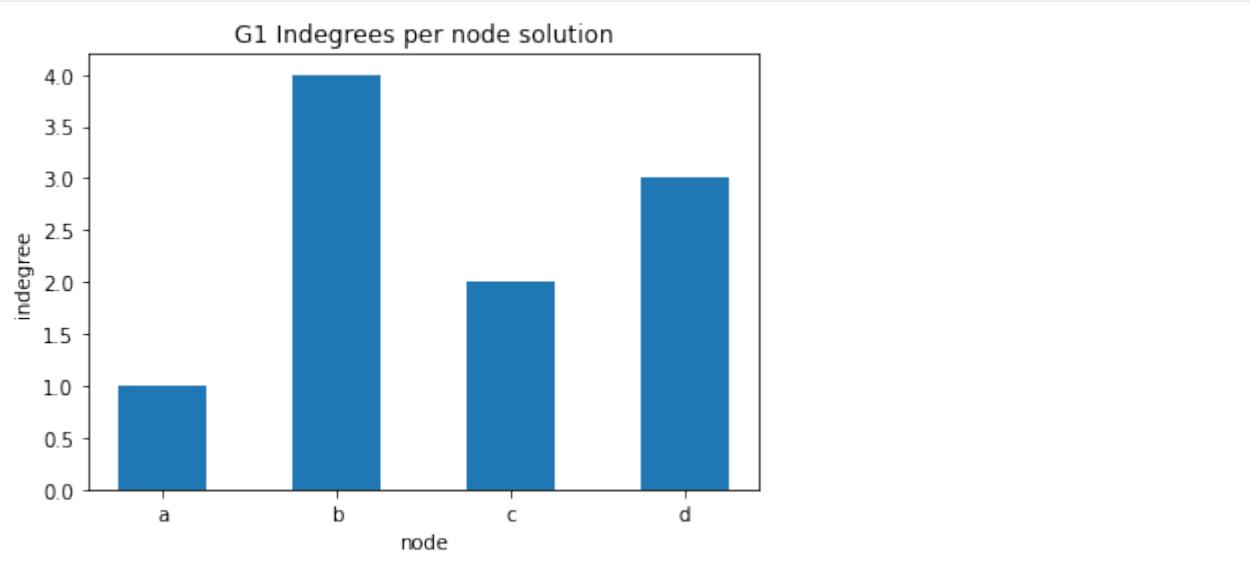
plt.title("G1 Indegrees per node solution")
plt.xticks(xs, G1.nodes())

plt.xlabel('node')
plt.ylabel('indegree')

plt.show()
```

<sup>416</sup> <https://en.softpython.org/visualization/visualization1-sol.html#Bar-plots>

<sup>417</sup> [https://matplotlib.org/api/\\_as\\_gen/matplotlib.pyplot.bar.html](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.bar.html)



</div>

[19]:

```
write here
```

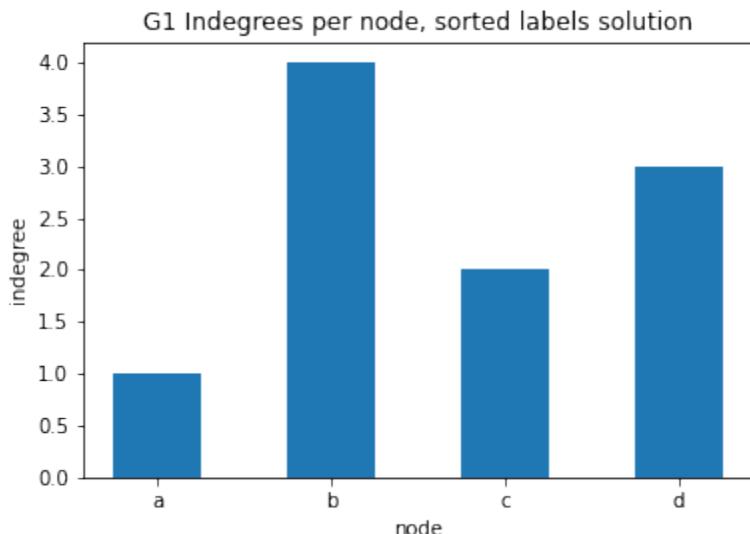
[ ]:

### indegree per node sorted alphabetically

⊕⊕ Display the same bar plot as before, but now sort nodes alphabetically.

**NOTE:** you cannot run `.sort()` method on the result given by `G1.nodes()`, because nodes in network by default have no inherent order. To use `.sort()` you need first to convert the result to a `list` object.

You should get something like this:



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[20]:

```
write here

import numpy as np
import matplotlib.pyplot as plt

xs = np.arange(G1.number_of_nodes())

xs_labels = list(G1.nodes())

xs_labels.sort()

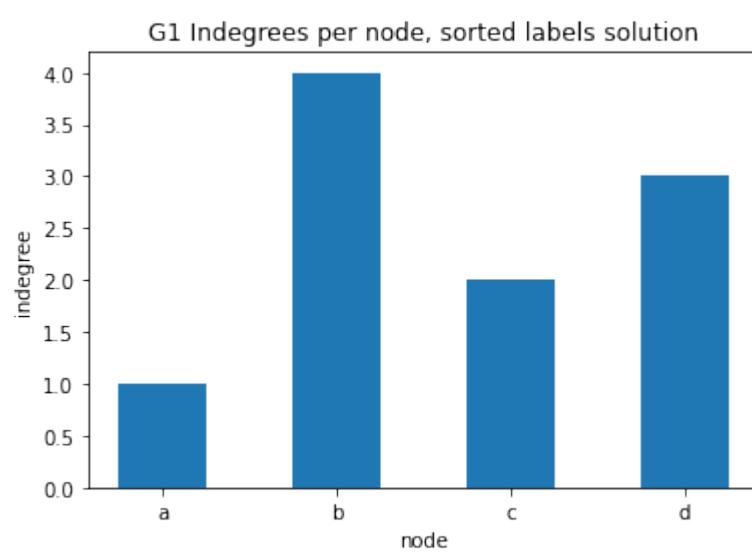
ys_in = [G1.in_degree(n) for n in xs_labels]

plt.bar(xs, ys_in, 0.5, align='center')

plt.title("G1 Indegrees per node, sorted labels solution")
plt.xticks(xs, xs_labels)

plt.xlabel('node')
plt.ylabel('indegree')

plt.show()
```



</div>

[20]:

```
write here
```

[21]:

```
write here
```

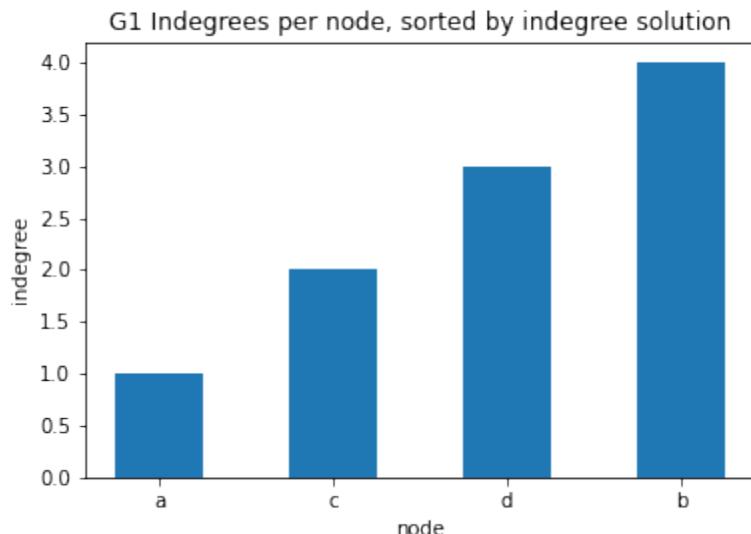
### indegree per node sorted

⊕⊕⊕ Display the same bar plot as before, but now sort nodes according to their indegree. This is more challenging, to do it you need to use some sort trick.

- **HINT:** first read the [Python documentation](#)<sup>418</sup>

Expected result:

<sup>418</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[22]:

```
write here

import numpy as np
import matplotlib.pyplot as plt

xs = np.arange(G1.number_of_nodes())

coords = [(v, G1.in_degree(v)) for v in G1.nodes()]

coords.sort(key=lambda c: c[1])

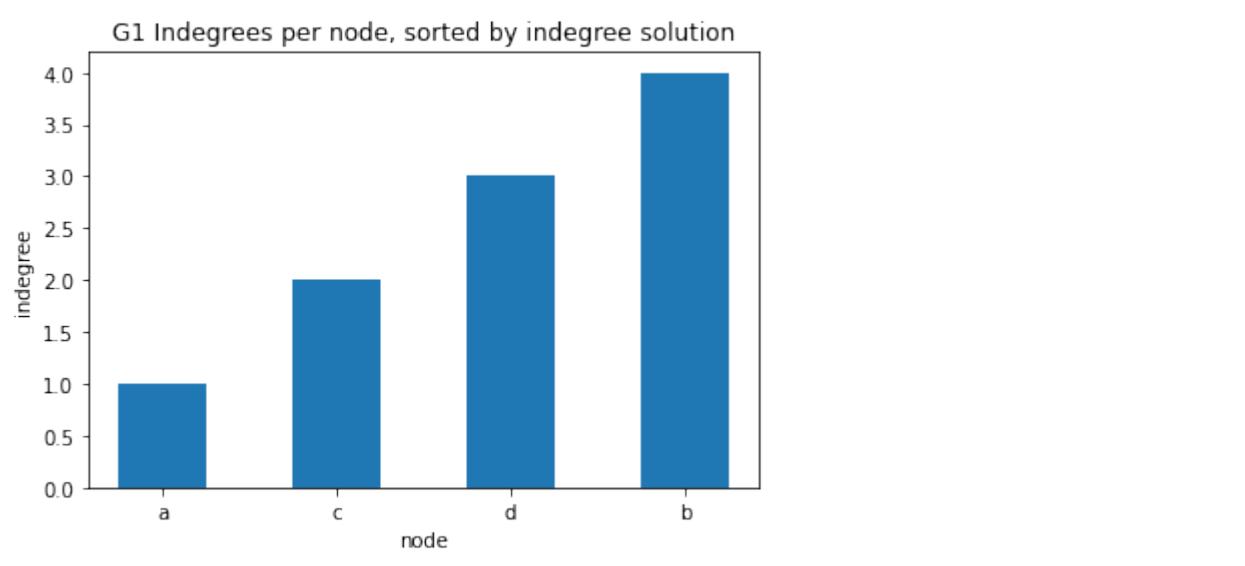
ys_in = [c[1] for c in coords]

plt.bar(xs, ys_in, 0.5, align='center')

plt.title("G1 Indegrees per node, sorted by indegree solution")
plt.xticks(xs, [c[0] for c in coords])

plt.xlabel('node')
plt.ylabel('indegree')

plt.show()
```



</div>

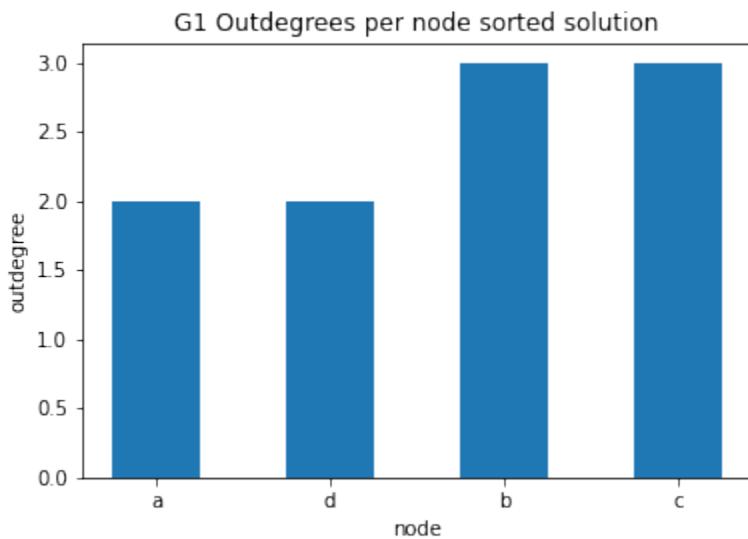
[22]:

```
write here
```

### out degrees per node sorted

⊕⊕⊕ Do the same graph as before for the outdegrees.

Expected result:



You can get the outdegree of a node with:

```
[23]: G1.out_degree('b')
```

```
[23]: 3
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[24]:
```

```
write here
import numpy as np
import matplotlib.pyplot as plt

xs = np.arange(G1.number_of_nodes())

coords = [(v, G1.out_degree(v)) for v in G1.nodes()]

coords.sort(key=lambda c: c[1])

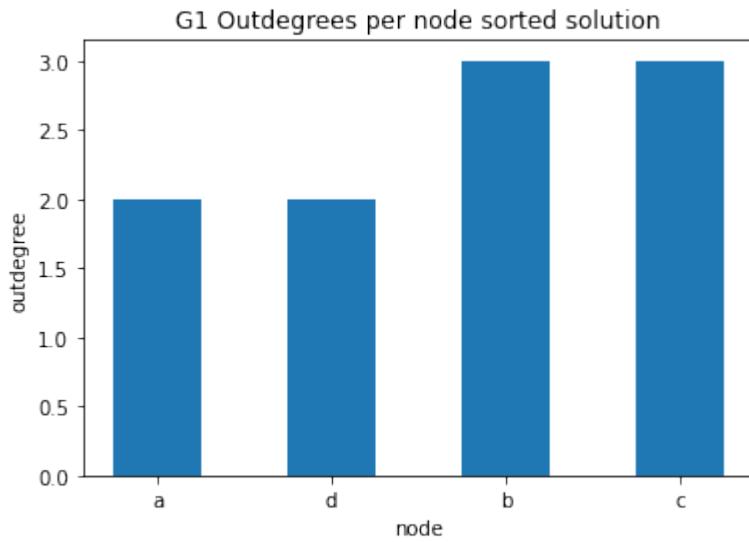
ys_out = [c[1] for c in coords]

plt.bar(xs, ys_out, 0.5, align='center')

plt.title("G1 Outdegrees per node sorted solution")
plt.xticks(xs, [c[0] for c in coords])

plt.xlabel('node')
plt.ylabel('outdegree')

plt.show()
```



</div>

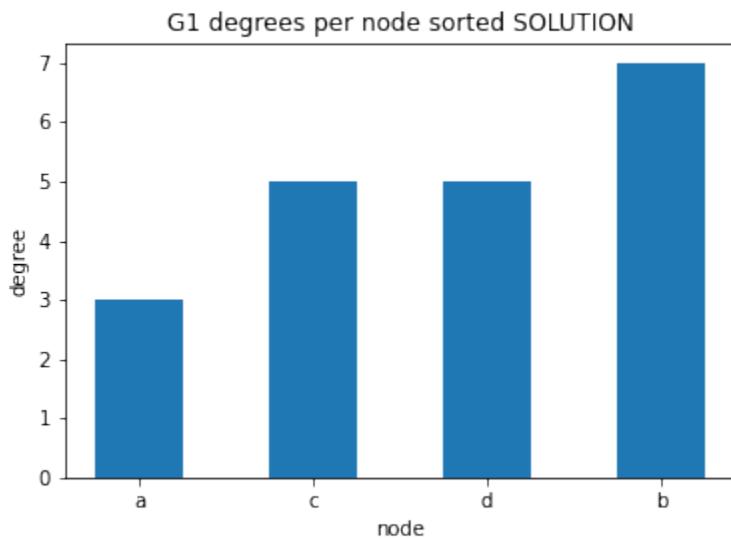
```
[24]:
```

```
write here
```

### degrees per node

⊕⊕⊕ We might check as well the sorted degrees per node, intended as the sum of in\_degree and out\_degree. To get the sum, use G1.degree(node) function.

Expected result:



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[25]:

```
write here

import numpy as np
import matplotlib.pyplot as plt

xs = np.arange(G1.number_of_nodes())

coords = [(v, G1.degree(v)) for v in G1.nodes()]

coords.sort(key=lambda c: c[1])

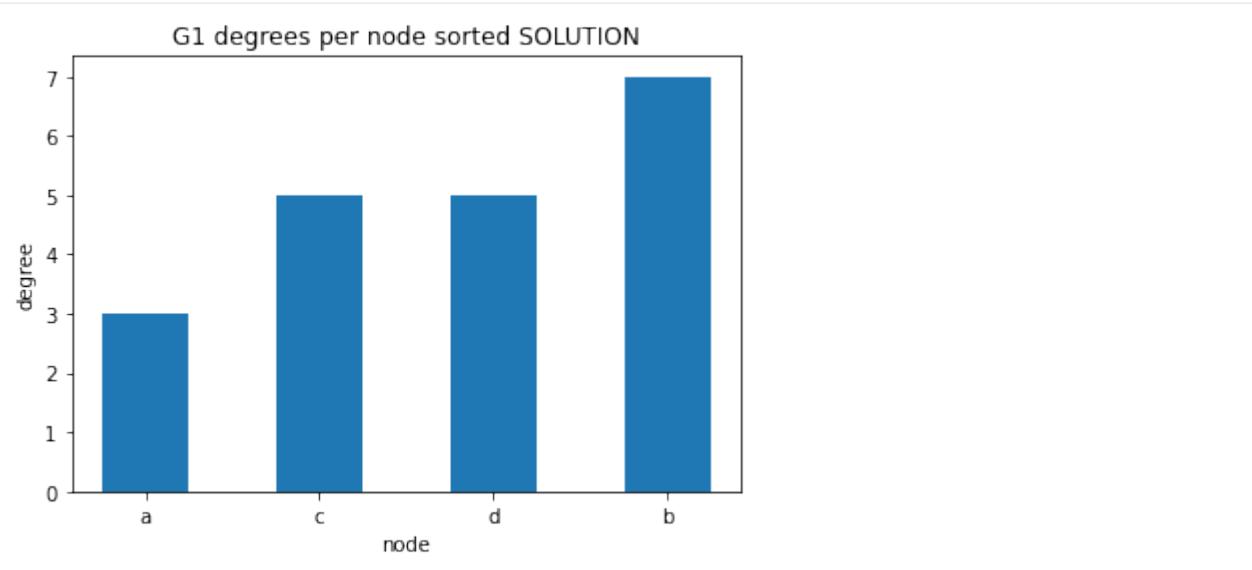
ys_deg = [c[1] for c in coords]

plt.bar(xs, ys_deg, 0.5, align='center')

plt.title("G1 degrees per node sorted SOLUTION")
plt.xticks(xs, [c[0] for c in coords])

plt.xlabel('node')
plt.ylabel('degree')

plt.show()
```



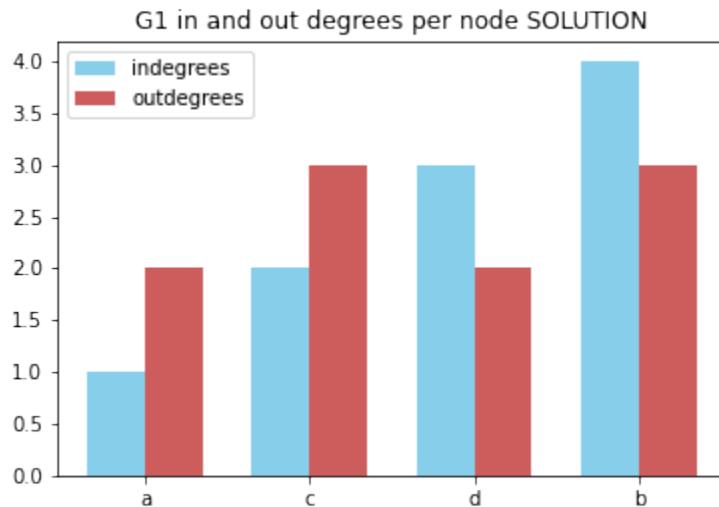
</div>

[25] :

```
write here
```

### In out degrees per node

⊕⊕⊕ Look at [this example<sup>419</sup>](#), and make a double bar chart sorting nodes by their *total* degree. To do so, in the tuples you will need vertex, in\_degree, out\_degree and also degree.



Show solutionHide>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

<sup>419</sup> [https://matplotlib.org/gallery/lines\\_bars\\_and\\_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py](https://matplotlib.org/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py)

```
[26]: # write here

import numpy as np
import matplotlib.pyplot as plt

xs = np.arange(G1.number_of_nodes())

coords = [(v, G1.degree(v), G1.in_degree(v), G1.out_degree(v)) for v in G1.nodes()]

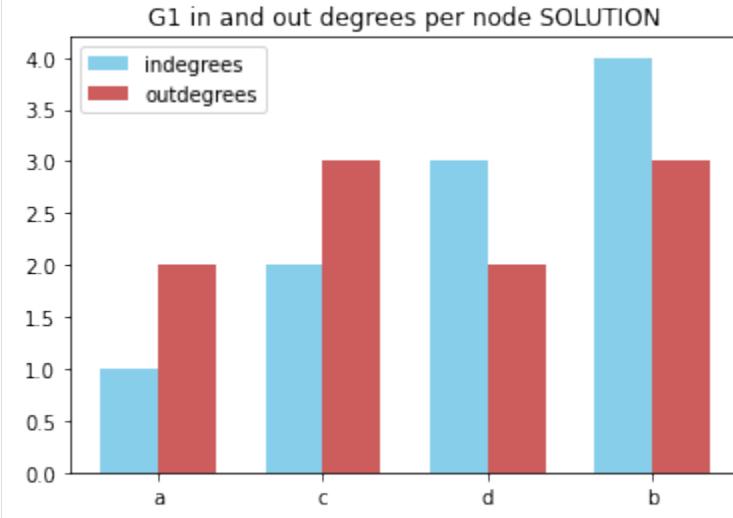
coords.sort(key=lambda c: c[1])

ys_deg = [c[1] for c in coords]
ys_in = [c[2] for c in coords]
ys_out = [c[3] for c in coords]

width = 0.35
fig, ax = plt.subplots()
rects1 = ax.bar(xs - width/2, ys_in, width,
 color='SkyBlue', label='indegrees')
rects2 = ax.bar(xs + width/2, ys_out, width,
 color='IndianRed', label='outdegrees')

Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_title('G1 in and out degrees per node SOLUTION')
ax.set_xticks(xs)
ax.set_xticklabels([c[0] for c in coords])
ax.legend()

plt.show()
```



&lt;/div&gt;

```
[26]: # write here
```

(continues on next page)

(continued from previous page)

## Frequency histogram

Now let's try to draw degree frequencies, that is, for each degree present in the graph we want to display a bar as high as the number of times that particular degree appears.

For doing so, we will need a matplotlib histogram, see [documentation](#)<sup>420</sup>

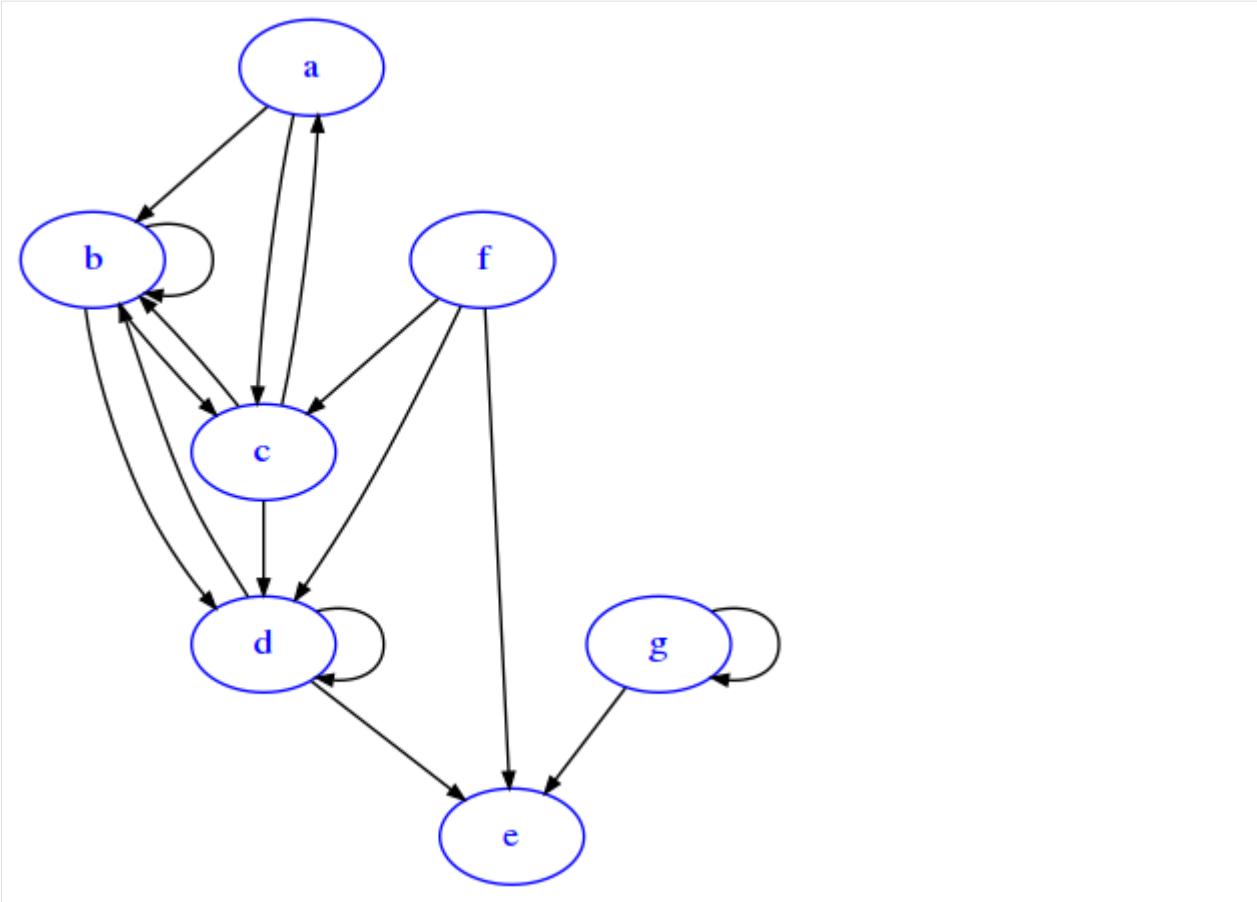
We will need to tell matplotlib how many columns we want, which in histogram terms are called *bins*. We also need to give the histogram a series of numbers so it can count how many times each number occurs. Let's consider this graph G2:

```
[27]: import networkx as nx

G2=nx.DiGraph({
 'a':['b','c'],
 'b':['b','c','d'],
 'c':['a','b','d'],
 'd':['b','d','e'],
 'e':[],
 'f':['c','d','e'],
 'g':['e','g']
})

draw_nx(G2)
```

<sup>420</sup> [https://matplotlib.org/api/\\_as\\_gen/matplotlib.pyplot.hist.html](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html)



If we take the the degree sequence of G2 we get this:

```
[28]: degrees_G2 = [G2.degree(n) for n in G2.nodes()]
degrees_G2
[28]: [3, 7, 6, 7, 3, 3, 3]
```

We see 3 appears four times, 6 once, and seven twice.

Let's try to determine a good number for the bins. First we can check the boundaries our x axis should have:

```
[29]: min(degrees_G2)
```

```
[29]: 3
```

```
[30]: max(degrees_G2)
```

```
[30]: 7
```

So our histogram on the x axis must go at least from 3 and at least to 7. If we want integer columns (bins), we will need at least ticks for going from 3 included to 7 included, so at least ticks for 3,4,5,6,7. For getting precise display, wen we have integer x it is best to also manually provide the sequence of bin edges, remembering it should start at least from the minimum *included* (in our case, 3) and arrive to the maximum + 1 *included* (in our case, 7 + 1 = 8)

**NOTE:** precise histogram drawing can be quite tricky, please do read [this StackOverflow post<sup>421</sup>](https://stackoverflow.com/a/27084005) for more details about it.

<sup>421</sup> <https://stackoverflow.com/a/27084005>

[31]:

```

import matplotlib.pyplot as plt
import numpy as np

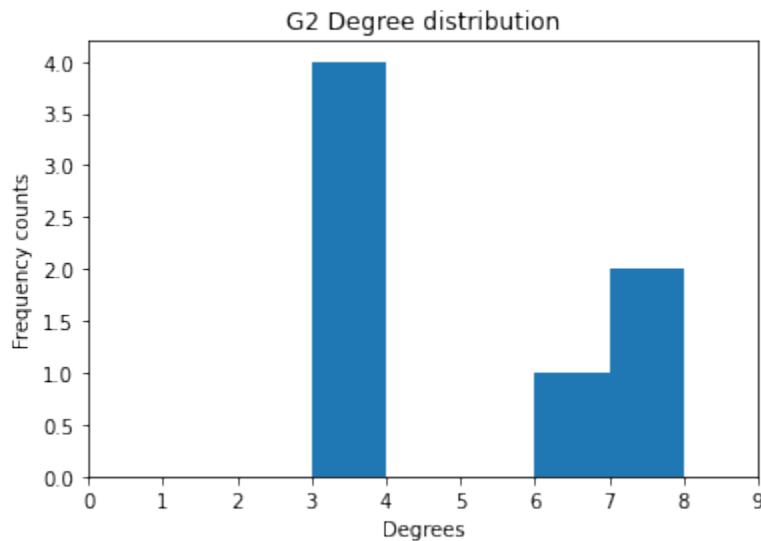
degrees = [G2.degree(n) for n in G2.nodes()]

add histogram

in this case hist returns a tuple of three values
we put in three variables
n, bins, columns = plt.hist(degrees_G2,
 bins=range(3,9), # 3 *included* , 4, 5, 6, 7, 8
 width=1.0) # graphical width of the bars

plt.xlabel('Degrees')
plt.ylabel('Frequency counts')
plt.title('G2 Degree distribution')
plt.xlim(0, max(degrees) + 2)
plt.show()

```



As expected we see 3 is counted four times, 6 once, and seven twice.

### Exercise - better histogram display

⊕⊕⊕ Still, it would be visually better to align the x ticks to the middle of the bars with `xticks`, and also to make the graph more tight by setting the `xlim` appropriately. This is not always easy to do.

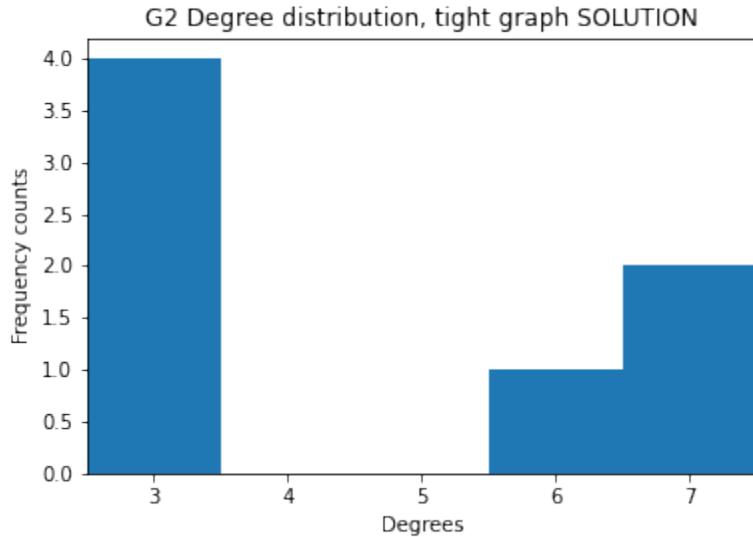
Read carefully this StackOverflow post<sup>422</sup> and try do it by yourself.

**NOTE:** set *one thing at a time* and try if it works(i.e. first `xticks` and then `xlim`), doing everything at once might get quite confusing

Expected result:

---

<sup>422</sup> <https://stackoverflow.com/a/27084005>



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[32]: # write here

import matplotlib.pyplot as plt
import numpy as np

degrees = [G2.degree(n) for n in G2.nodes()]

add histogram

min_x = min(degrees) # 3
max_x = max(degrees) # 7
bar_width = 1.0

in this case hist returns a tuple of three values
we put in three variables
n, bins, columns = plt.hist(degrees_G2,
 bins= range(3,9), # 3 *included* to 9 *excluded*
 # it is like the xs, but with one_
 ↪number more !!
 # to understand why read this
 # https://stackoverflow.com/questions/
 ↪27083051/matplotlib-xticks-not-lining-up-with-histogram/27084005#27084005
 width=bar_width) # graphical width of the bars

plt.xlabel('Degrees')
plt.ylabel('Frequency counts')
plt.title('G2 Degree distribution, tight graph SOLUTION')

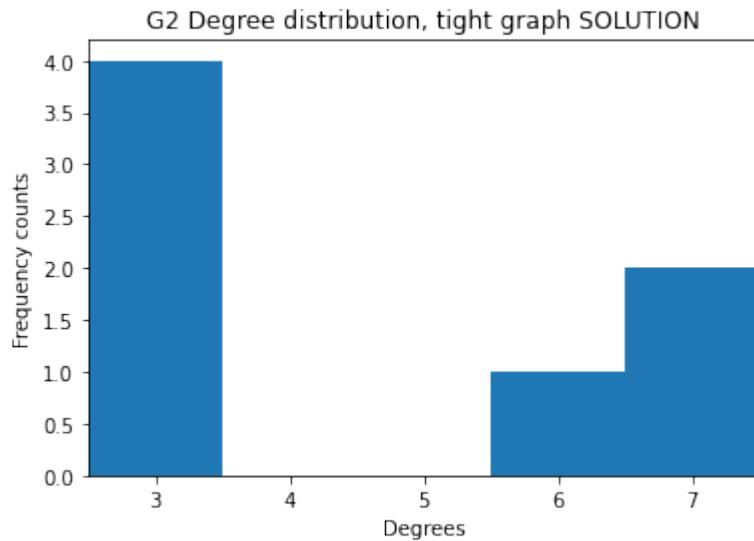
xs = np.arange(min_x,max_x + 1) # 3 *included* to 8 *excluded*
 # used numpy so we can later reuse it for float_
 ↪vector operations

plt.xticks(xs + bar_width / 2, # position of ticks
```

(continues on next page)

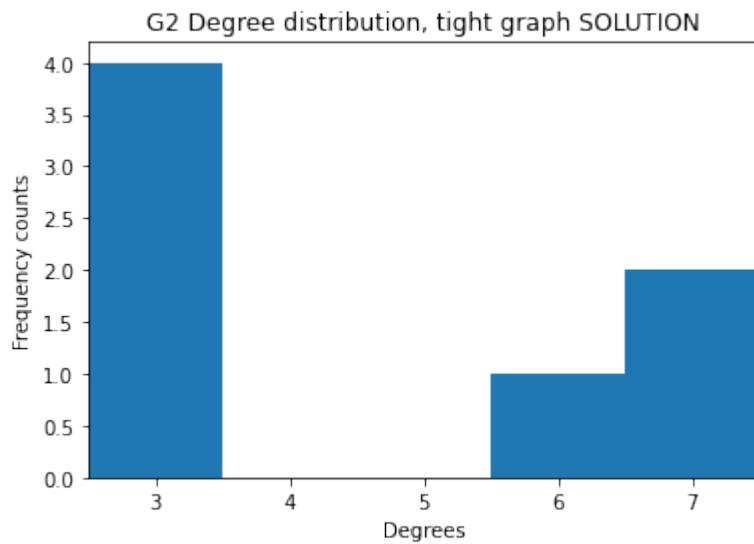
(continued from previous page)

```
 xs) # labels of ticks
plt.xlim(min_x, max_x + 1) # 3 *included* to 8 *excluded*
plt.show()
```



&lt;/div&gt;

[32]: # write here



## Graph models

Let's study frequencies of some known network types.

### Exercise - Erdős–Rényi model

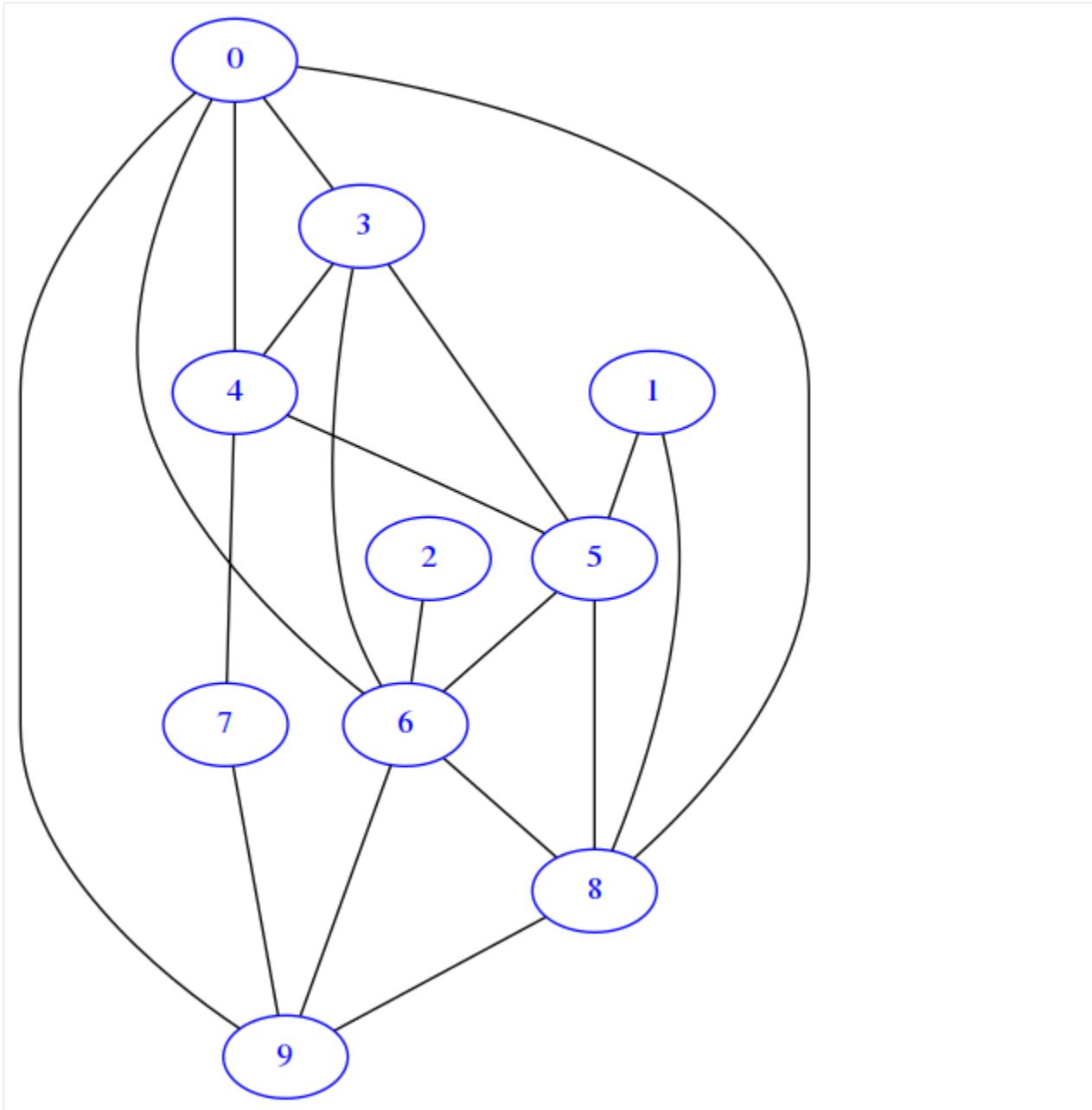
⊕⊕ A simple graph model we can think of is the so-called Erdős–Rényi model<sup>423</sup>: it is an *undirected* graph where have n nodes, and each node is connected to each other with probability p. In networkx, we can generate a random one by issuing this command:

```
[33]: G = nx.erdos_renyi_graph(10, 0.5)
```

In the drawing, by looking the absence of arrows confirms it is undirected:

```
[34]: draw_nx(G)
```

<sup>423</sup> [https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi\\_model](https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model)

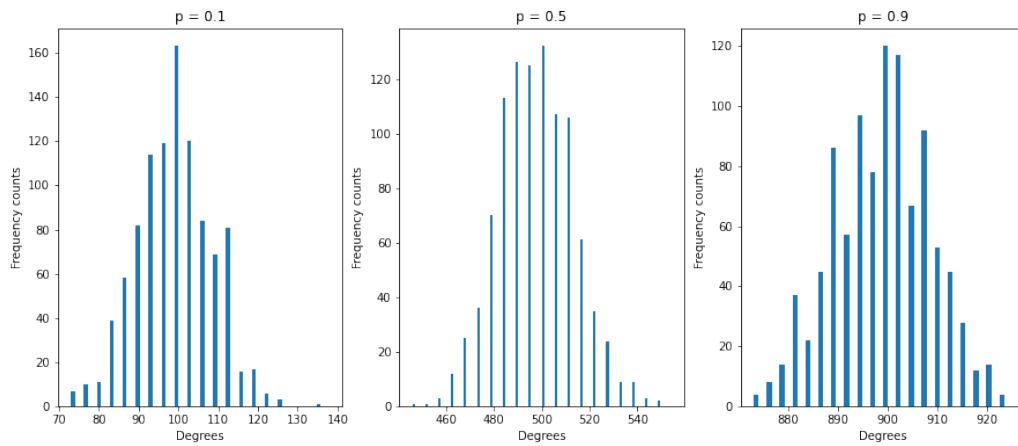


Try plotting degree distribution for different values of  $p$  (0.1, 0.5, 0.9) with a fixed  $n=1000$ , putting them side by side on the same row. What does their distribution look like ? Where are they centered ?

- to put them side by side, look at this example<sup>424</sup>
- to avoid rewriting the same code again and again, define a `plot_erdos(n, p, j)` function to be called three times.

Expected result:

<sup>424</sup> <https://en.softpython.org/visualization/visualization1-sol.html#Showing-plots-side-by-side>



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[35]:

```
write here

import matplotlib.pyplot as plt
import numpy as np

def plot_erdos(n, p, j):
 G = nx.erdos_renyi_graph(n, p)

 plt.subplot(1, 3, j) # 1 row, 3 columns, jth cell

 degrees = [G.degree(n) for n in G.nodes()]
 num_bins = 20

 n, bins, columns = plt.hist(degrees, num_bins, width=1.0)

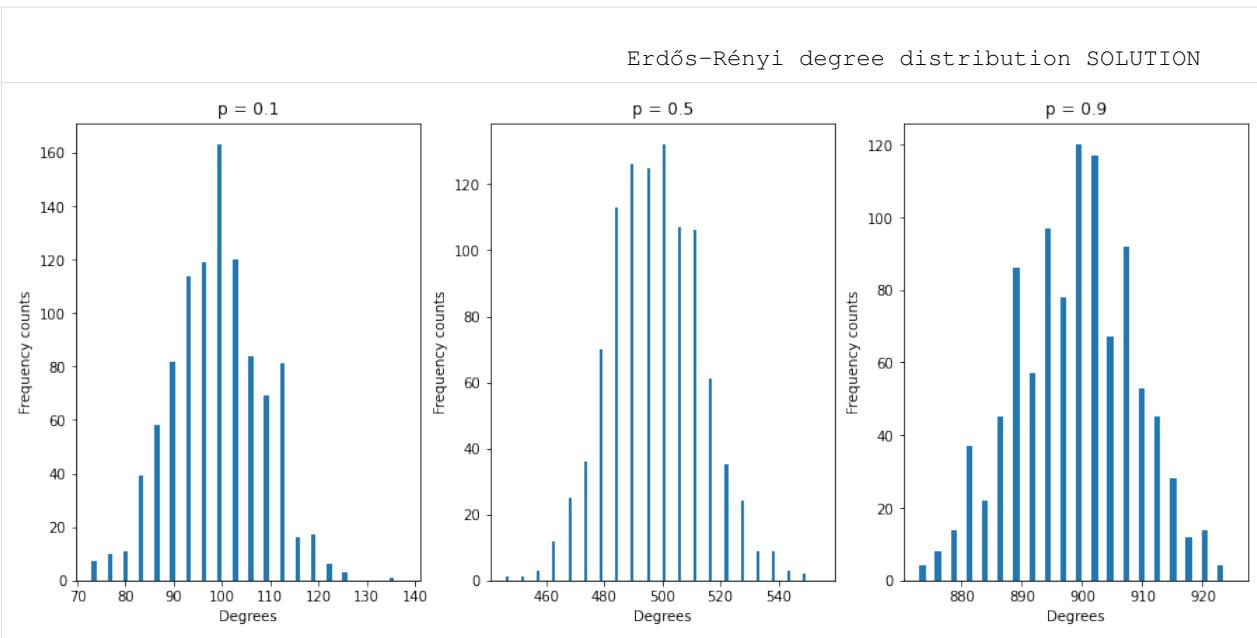
 plt.xlabel('Degrees')
 plt.ylabel('Frequency counts')
 plt.title('p = %s' % p)

n = 1000

fig = plt.figure(figsize=(15,6)) # width: 10 inches, height 3 inches

plot_erdos(n, 0.1, 1)
plot_erdos(n, 0.5, 2)
plot_erdos(n, 0.9, 3)

print() Erdős-Rényi degree distribution
print("→SOLUTION")
```



</div>

[35] :

```
write here
```

## Continue

Go on with the challenges<sup>425</sup>

### 8.4.4 Relational data 4 - Challenges

[Download exercises zip](#)

[Browse online files](#)<sup>426</sup>

## Matrices

**HOW TO DISPLAY:** In these exercises you never need to display the chain exactly as in the examples, the important thing is having correct node numbers and links among them. Still, for optimal display we will sometimes suggest some options parameters.

First off, import `draw_mat`:

[1] : `from soft import draw_mat`

<sup>425</sup> <https://en.softpython.org/relational/relational4-chal.html>

<sup>426</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/relational>

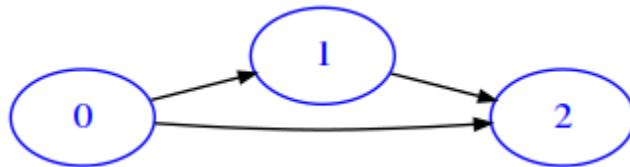
### Challenge - trichain

Write a function which given an odd number  $n$ , displays a graph represented as matrix of lists of lists of booleans as in the examples.

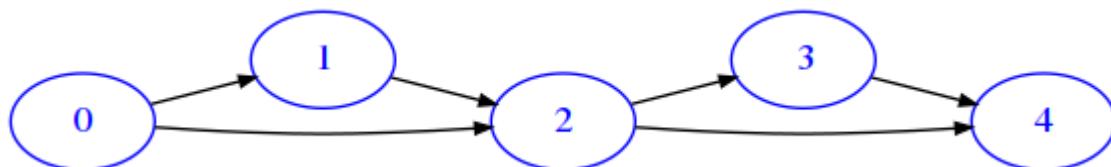
- if  $n$  is negative or even, raise ValueError
- For optimal display call `draw_mat` like this:  
`draw_mat( mat , options={'graph':{'rankdir':'LR'}} )`

Examples:

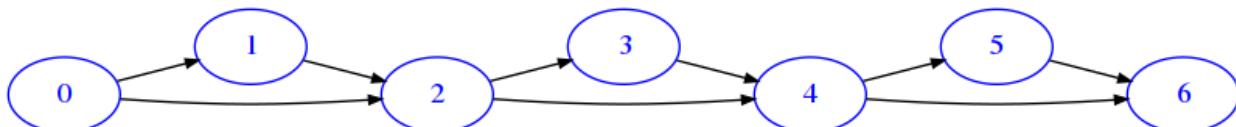
```
>>> trichain(3)
```



```
>>> trichain(5)
```



```
>>> trichain(7)
```



[2]:

```
from soft import draw_mat

def trichain(n):
 raise Exception('TODO IMPLEMENT ME !')

trichain(3)
trichain(5)
trichain(7)
```

### Challenge - Bipartite

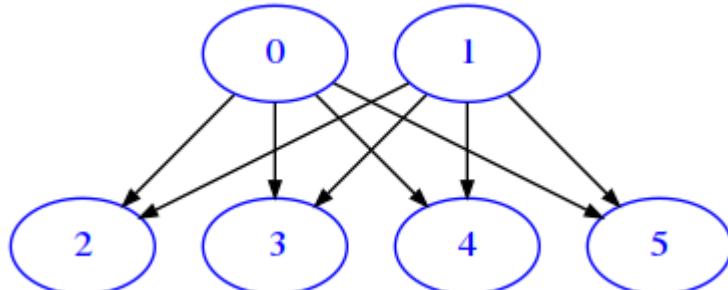
⊕⊕ Write a function which given two numbers  $n$  and  $m$ , displays a boolean matrix as list of lists, representing a graph in which first  $n$  nodes are linked to all successive  $m$  nodes.

- for optimal drawing, add `options` parameter like this:

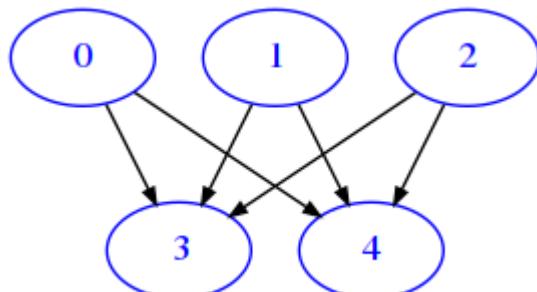
```
draw_mat(mat , options= {'graph': {'ordering': 'out'}})
```

**Examples:**

```
>>> bipartite(2,4)
```



```
>>> bipartite(3,2)
```



[3]:

```
from soft import draw_mat

def bipartite(n, m):
 raise Exception('TODO IMPLEMENT ME !')

bipartite(2,4)
bipartite(3,2)

#bipartite(1,1)
#bipartite(1,2)
#bipartite(2,1)
```

**Challenge - Luna park**

⊕⊕ A luna park receives hordes of tourists who all want to have a ride on *The Spinning Head*. The attraction has many queues to get tickets, and each queue holds tourists identified by an id. The attraction operators take 5 minutes to service each tourist.

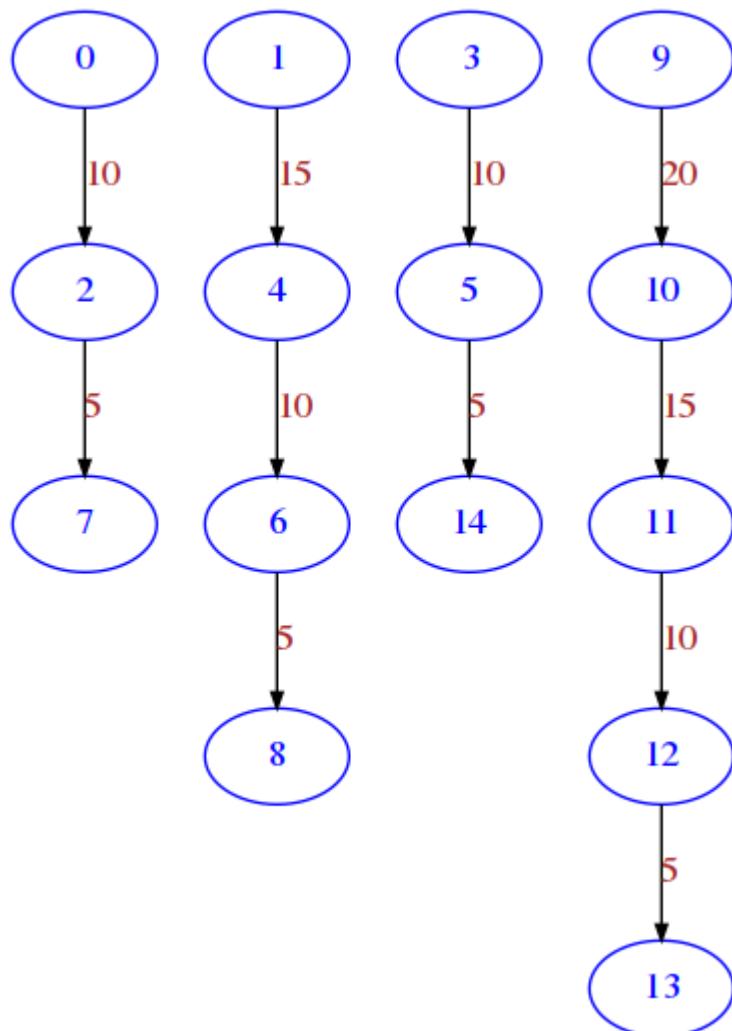
Implement a function `lunapark` which takes a list of queues (each queue is a list of ids), and displays a graph represented as matrix as a list of lists of integers, with the times tourists have to wait until they get served.

- assume there are always  $n$  distinct tourists, with ids starting from 0 until  $n$  excluded

**Example:**

We have 4 queues, and in the first queue tourist 0 will have to wait 10 minutes and tourist 2 will have to wait 5 minutes. Tourist 7 is being serviced so she has zero minutes to wait.

```
tourists = [[0, 2, 7],
 [1, 4, 6, 8],
 [3, 5, 14],
 [9, 10, 11, 12, 13]]
>>> lunapark(tourists)
```



```
[4]:
from soft import draw_mat

def lunapark(queues):
 raise Exception('TODO IMPLEMENT ME !')

tourists = [[0,2,7],
 [1,4,6,8],
 [3,5,14],
 [9,10,11,12,13]]

lunapark(tourists)

#lunapark([[0]])
#lunapark([[0,1]])
#lunapark([[0,1],
[2]])
```

## Challenge - Factory

A factory has several conveyor belts to process raw materials. Each conveyor has a number of machines which refine the material. Each machine takes a different time to work, except the last one of each line which is just a collector.

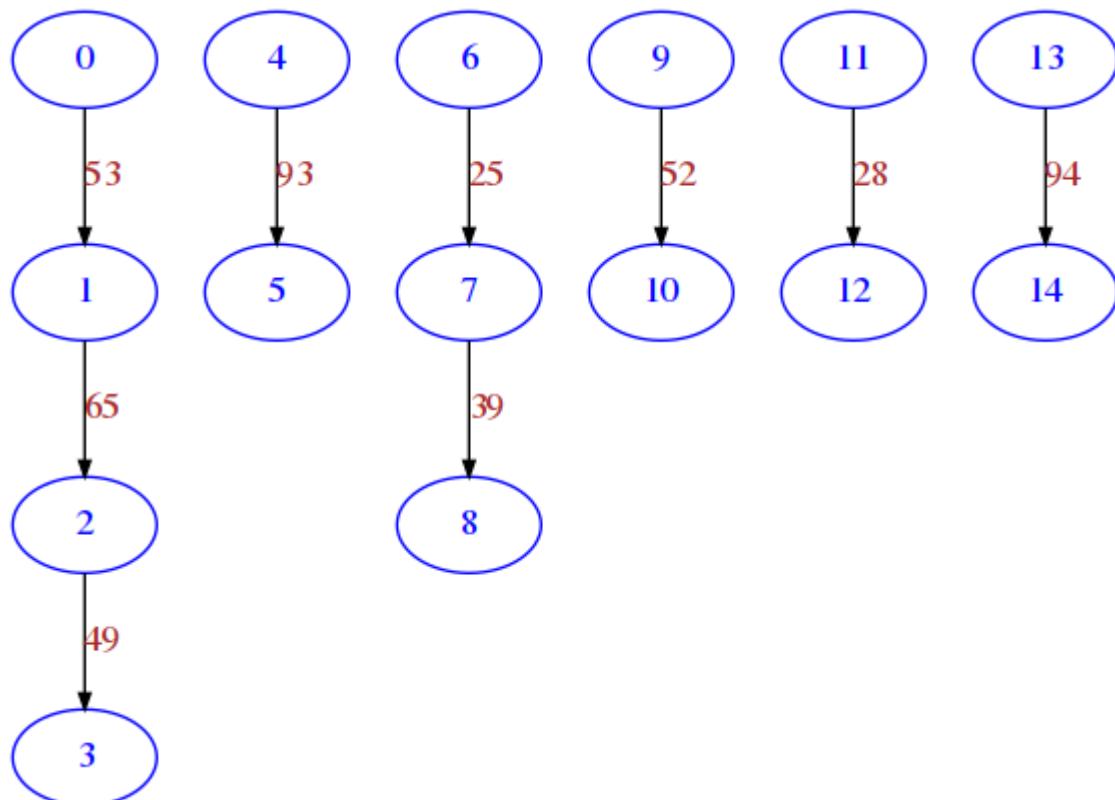
### Factory process

⊕⊕ Write a function `process` which takes a list of machine lines and displays a graph as a matrix of list of lists of integers holding the timings between each machine. A machine line is a list with the timings of each machine.

- this time **also RETURN** the matrix (we will reuse it later)
- Note node ids are implicit, you have to derive them

**Example:** the first line has 4 machines: the first takes 53 minutes, the second 65 and the third one 49 minutes and the last one is implicitly supposed to take zero minutes as it's just a collector.

```
station ids
>>> process([[53,65,49], # 0 1 2 3 conveyor line 0
 [93], # 4 5 conveyor line 1
 [25,39], # 6 7 8 conveyor line 2
 [52], # 9 10 conveyor line 3
 [28], # 11 12 conveyor line 4
 [94]]) # 13 14 conveyor line 5
```



[5]:

```

from soft import draw_mat

def process(lines):
 raise Exception('TODO IMPLEMENT ME !')

 # station ids
processing_lines = [[53, 65, 49], # 0 1 2 3 conveyor line 0
 [93], # 4 5 conveyor line 1
 [25, 39], # 6 7 8 conveyor line 2
 [52], # 9 10 conveyor line 3
 [28], # 11 12 conveyor line 4
 [94], # 13 14 conveyor line 5
]
]

process(processing_lines)

#process([[34]])
#process([[51, 83]])
#process([[55],
[92, 41]])

```

## Factory assemble

⊕⊕⊕ After processing, each material part is assembled into intermediate products until the final product is made. Multiple conveyor lines can join into another ones for assembly. The joining time is always fixed to 5 minutes. Write a function assemble which given conveyor lines as before and a joins list displays a graph matrix as a list of lists of integers

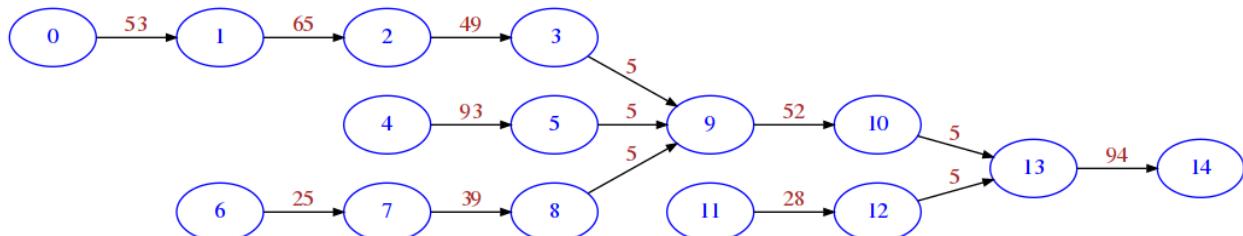
- **HINT 1:** you can call previous process(lines) function and then connect the joins
- to draw the graph horizontally, add the parameter options={'graph':{'rankdir':'LR'}} to draw\_mat

**Example:**

```
processing_lines = [
 [53, 65, 49], # station ids conveyor line
 [93], # 0 1 2 3 0
 [25, 39], # 4 5
 [52], # 6 7 8 2
 [28], # 9 10
 [94]] # 11 12 4
 # 13 14 5

conveyor line 0,1 and 2 outputs must go into conveyor line 3 input in 5 minutes
conveyor line 3 and 5 outputs must go into conveyor line 5 input in 5 minutes
joins = [(0,3), (1,3), (2,3), (3,5), (4,5)]

>>> draw_mat(assemble(processing_lines, joins), options={'graph':{'rankdir':'LR'}})
```



[6]:

```
from soft import draw_mat

def assemble(lines, joins):
 raise Exception('TODO IMPLEMENT ME !')

processing_lines = [
 [53, 65, 49], # station ids conveyor line
 [93], # 0 1 2 3 0
 [25, 39], # 4 5
 [52], # 6 7 8 2
 [28], # 9 10
 [94]] # 11 12 4
 # 13 14 5

conveyor line 0,1 and 2 outputs must go into conveyor line 3 input in 5 minutes
conveyor line 3 and 5 outputs must go into conveyor line 5 input in 5 minutes
joins = [(0,3), (1,3), (2,3), (3,5), (4,5)]

assemble(processing_lines, joins)

#assemble([[23]], [])
```

(continues on next page)

(continued from previous page)

```
#assemble([[23, 81], []])
#t = [[23],
#[84, 12]
#assemble(t, [(1, 0)])
```

**Challenge - Sharing is caring**

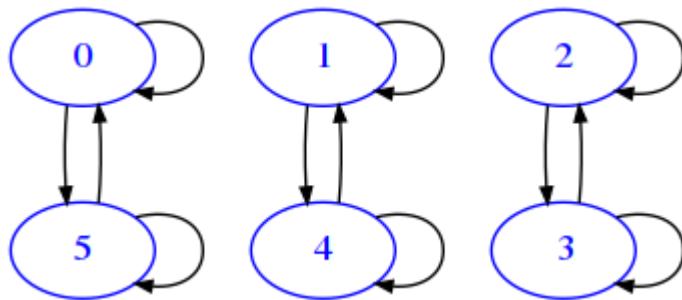
⊕⊕ Given a number  $n$  of couples husband / wife, write a function `sharing` which displays a graph which models the relation *can access property of*. As output format use a matrix list of lists of booleans.

- if  $n$  is zero or negative, raise `ValueError`

**Example:**

- mister 0 can access his car and miss 5's car
- miss 5 can access her car and mister 0's car

```
>>> sharing(3)
```



[7]:

```
from soft import draw_mat

def sharing(n):
 raise Exception('TODO IMPLEMENT ME !')

sharing(3)

#sharing(1)
#sharing(0) # ValueError
```

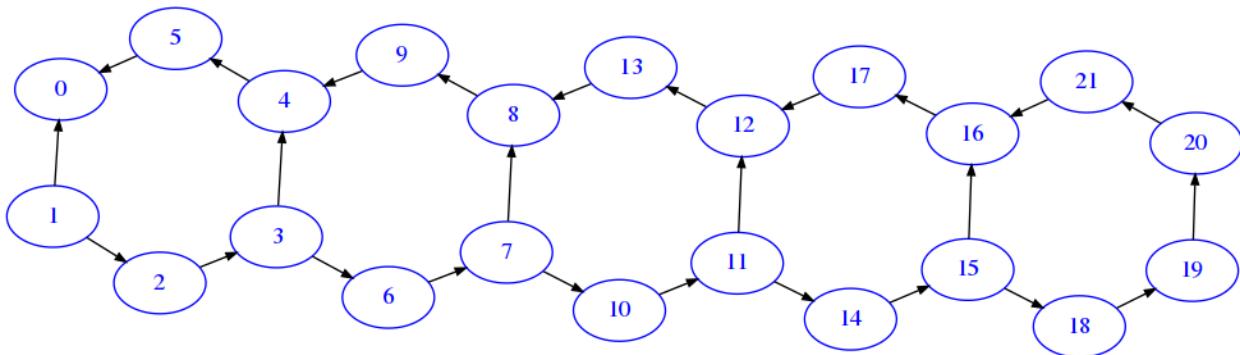
## Challenge - Hexagons

⊕⊕⊕ Write a function `hexagons (h)` which displays a graph with `h` hexagons as in the example. As output format, use a matrix as list of lists of booleans.

- **WARNING:** node 1 is special, it doesn't obey any pattern so treat it separately
- Use these options for optimal display:  
`draw_mat (mat, options={'graph': {'layout': 'neato', 'scale': '0.9', 'start': 'random6'}})`
- **NOTE:** sometimes the layout is displayed garbled, in particular the first hexagon tends to be messy. If this happen, try to changing the number in 'random6' parameter with other numbers, like 'random3', 'random4', etc.

### Example:

```
>>> draw_mat (hexagons (5), options={'graph': {'layout': 'neato', 'scale': '0.9', 'start': 'random6'}})
```



[8]:

```
from soft import draw_mat

def hexagons (h):
 raise Exception ('TODO IMPLEMENT ME !')

hexagons (5)

#hexagons (1)
#hexagons (2)
```

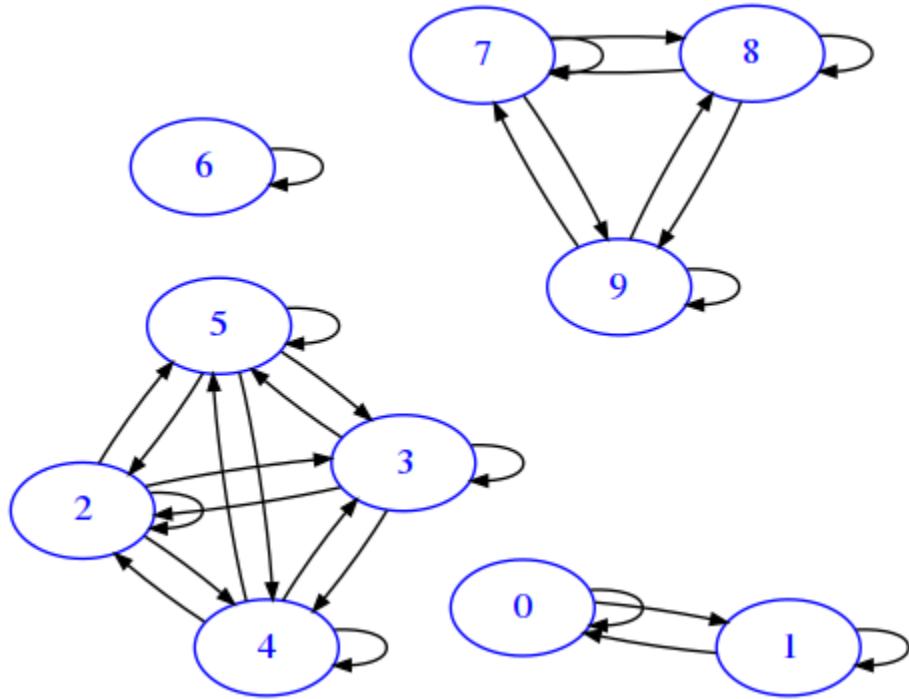
## Challenge - Trust your tribe

⊕⊕⊕ There are `n` people subdivided in `m` political parties they vote for. It's a well known fact that each voter trusts only people who vote for their same party, and of course everybody also trusts him/herself. Given a list `groups` containing the number of people each group contains, write a function `trust (groups)` which displays a graph as a list of lists of booleans which shows the trust network among people. For example, `trust ([2, 4, 1, 3])` must generate a graph where the first party has two voters, the second party four voters, etc.

- if any party has negative people, raise `ValueError`
- zero member parties are allowed
- For optimal display, use following options:

```
draw_mat(mat, options={'graph': {'layout': 'neato', 'scale': '1.4'}})
```

Example:



```
[9]:
from soft import draw_mat

def trust(groups):
 raise Exception('TODO IMPLEMENT ME !')

trust([2,4,1,3])

#trust([1])
#trust([3,2])
#trust([5,0,3]) # should work
#trust([-1]) # ValueError
```

## Adjacency lists

### Challenge - From matrices to adjacency lists

Redo the previous exercises this time using adjacency lists. To draw them, use:

```
[10]: from soft import draw_adj
```

## Challenge - Friends

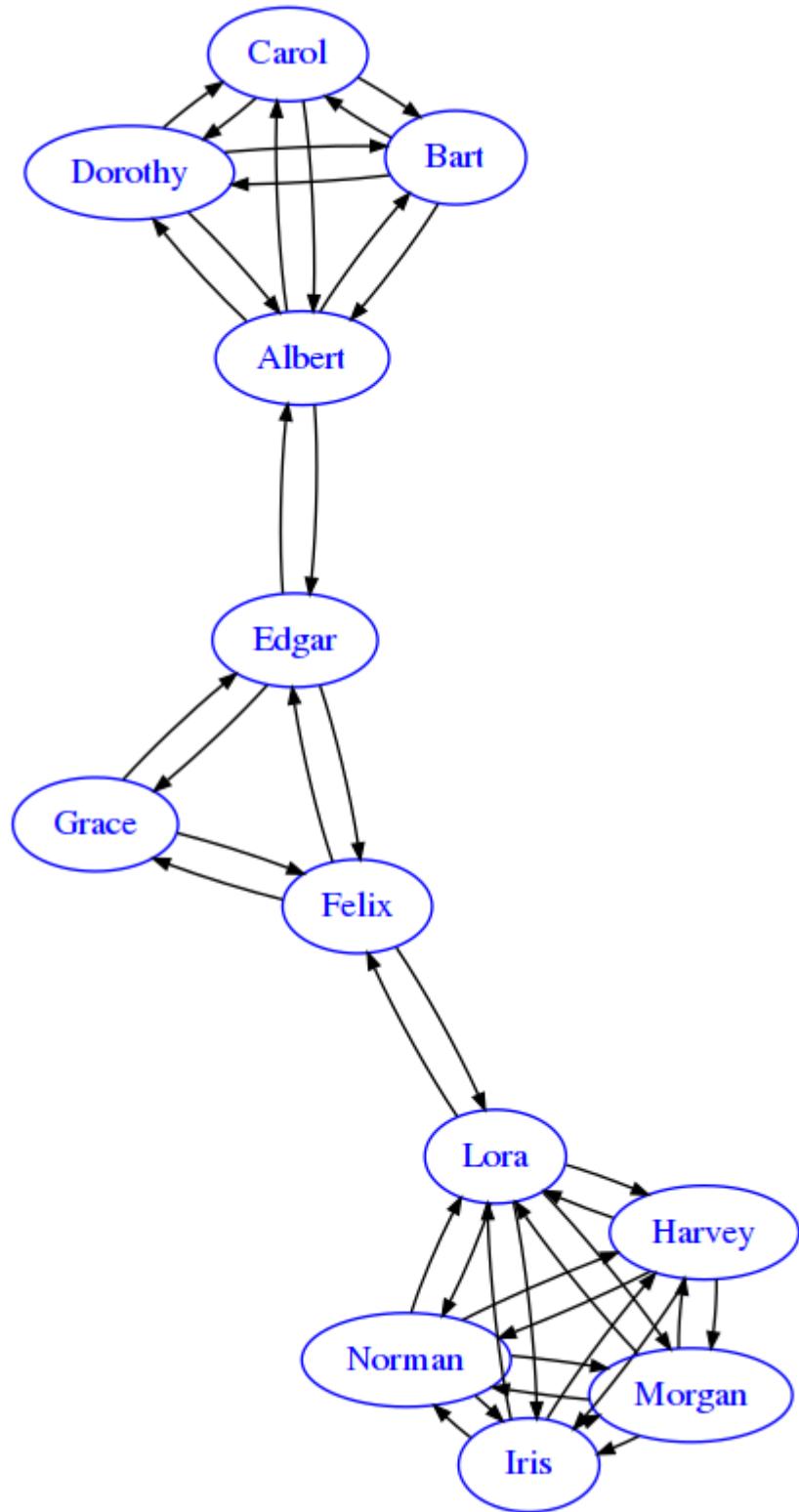
⊕⊕ In groups of friends, everybody is friend with each other, and often somebody from each group is friend with somebody from another group, forming thus a connection among different communities. Write a function friends which given a list of groups and connections among them, creates a NEW graph and displays it as a dictionary of adjacency lists where each person in a group is linked to all other persons in the group except itself, and groups are connected according to the couple of names specified in connections

- **DO NOT MODIFY THE INPUT !!!**
- **NOTE:** *isFriendOf* is a symmetrical relation, so you will need two arrows for each couple in connections
- for optimal display, use options like this:

```
draw_adj(d, options={'graph': {'layout': 'neato', 'scale': '1.4'}})
```

### Example:

```
>>> groups = [['Albert', 'Bart', 'Carol', 'Dorothy'],
 ... ['Edgar', 'Felix', 'Grace'],
 ... ['Harvey', 'Iris', 'Lora', 'Morgan', 'Norman']]
>>> connections = [('Albert', 'Edgar'), ('Felix', 'Lora')]
>>> friends(groups, connections)
```



```
[11]:
from soft import draw_adj

def friends(groups, connections):
```

(continues on next page)

(continued from previous page)

```

raise Exception('TODO IMPLEMENT ME !')

groups = [['Albert', 'Bart', 'Carol', 'Dorothy'],
 ['Edgar', 'Felix', 'Grace'],
 ['Harvey', 'Iris', 'Lora', 'Morgan', 'Norman']]
connections = [('Albert', 'Edgar'), ('Felix', 'Lora')]

#friends([['A', 'B', 'C'], ['D']], [(B', 'D')])
```

## Challenge - Counter Intelligence

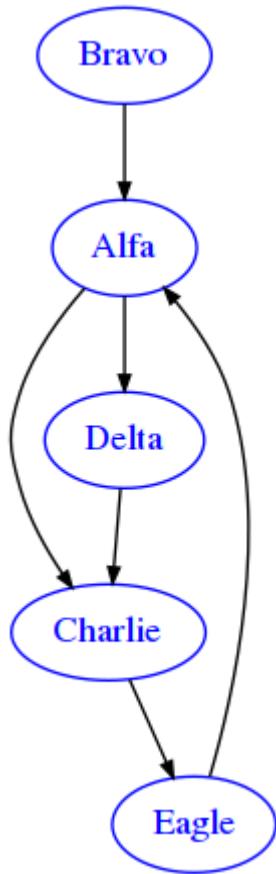
⊕⊕⊕ The secret service has just raided the house of a suspected spy. The foreign agent had left the building few hours before, but while searching the house a microfilm is found with instructions for the spy about the next moves he should do. On the paper, locations are anonymized with words like ‘Alpha’, ‘Bravo’, etc. Supposing names in the text follow a temporal order, the secret service wants you to derive a map of the performed trips, so to show the connections among the various locations. Write a function decode which given the text and a list of locations names, displays a graph as a dictionary of adjacency lists

- **DO NOT** put self-loops in the map
- Some words are followed by punctuation, clean them using the provided list

Example:

```

>>> punctuation = ['.', ',', '']
>>> locations = ['Alfa', 'Bravo', 'Charlie', 'Delta', 'Eagle']
>>> text = """
Go to Bravo, then take a bus to Alfa. Afterwards, go to Charlie
and meet your partner in Eagle. Next day go back to Alfa and later
take a train to Delta. Remain for a day in Delta, then rent a car and reach Charlie.
"""
>>> decode(text, locations)
```



[3]:

```
from soft import draw_adj

def decode(text, locations):
 raise Exception('TODO IMPLEMENT ME !')

punctuation = ['.', ',', '']
locations = ['Alfa', 'Bravo', 'Charlie', 'Delta', 'Eagle']
text = """
Go to Bravo, then take a bus to Alfa. Afterwards, go to Charlie
and meet your partner in Eagle. Next day go back to Alfa and later
take a train to Delta. Remain for a day in Delta, then rent a car and reach Charlie.
"""

decode(text, locations)

#punctuation, locations = [';', '.', '.'], ['Tango', 'Whiskey', 'Yankee']
#text = "Find Yankee; dance Tango; drink all the Whiskey. Then go find another Yankee
↪with some Whiskey."
#decode(text, locations)
```

## Networkx

We will now see some exercises with NetworkX. To do them, you will need to import the library and relative drawing function:

```
[13]: import networkx as nx
from soft import draw_nx
```

### Challenge - Offshore

⊕⊕ Your government's Revenue Service has launched an investigation into big tech firms: there is clear evidence they manage to pay almost zero taxes by having a complex network of transactions between a myriad of accounts in offshore tax havens. In order to get a clear picture of what is going on, the Revenue Service asks you to draw a graph showing suspect transactions occurred in a fiscal year among accounts of different countries.

A transaction is a list with four values: the date, the origin and the destination country, and the amount of the transaction (we assume billion dollars as units).

Write a function `offshore` which takes a list of transactions and displays a Networkx graph showing each of the occurred transactions for a given fiscal year.

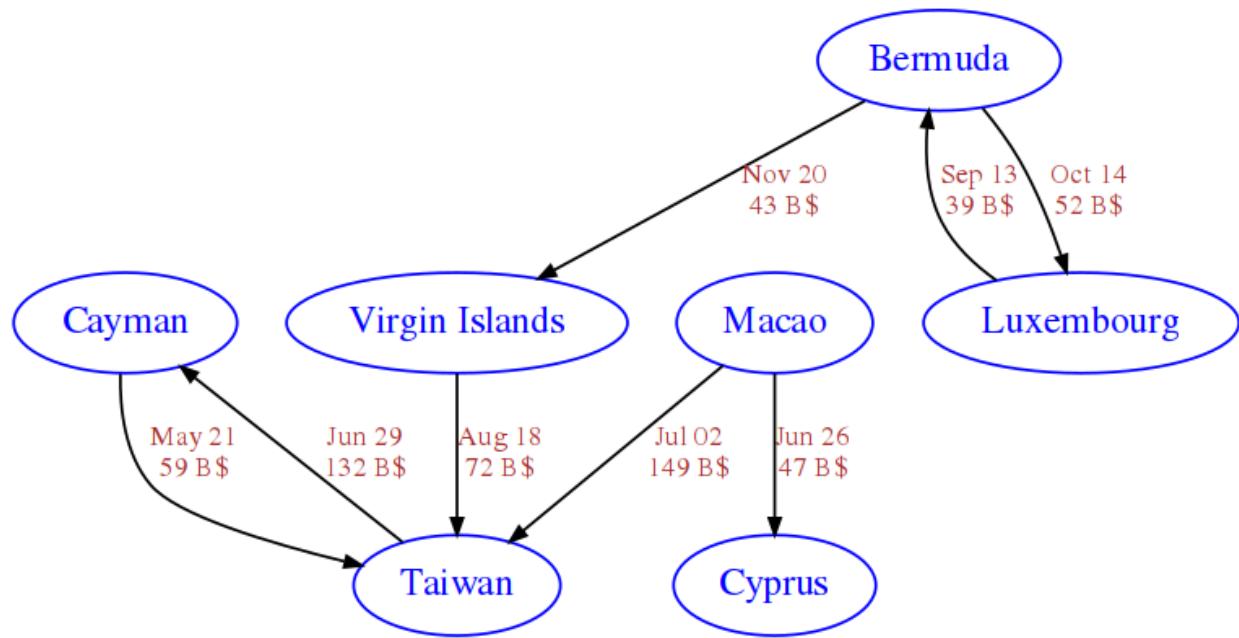
- this time **also RETURN** the graph data structure, so to allow some tests with assert
- label the transaction without the year and use a month name, like Jun 26, and place the amount 47 B\$ in a new line. There are many ways to convert to month, you can use a crude 'do-it-yourself' solution or using python string formatting
- **REMEMBER** to set both '`weight`' and '`label`', which are different things !
- For optimal display, use options like this:

```
draw_nx(G, options={'graph':{'size':'8,5!'}, 'edge':{'fontsize':'10'}})
```

#### Example:

```
>>> transactions = [
 ['2018-02-10', 'Taiwan', 'Cyprus', 30],
 ['2018-02-12', 'Taiwan', 'Virgin Islands', 83],
 ['2018-02-14', 'Cayman', 'Cyprus', 34],
 ['2018-03-25', 'Cayman', 'Bermuda', 143],
 ['2018-03-28', 'Cyprus', 'Macao', 72],
 ['2018-04-17', 'Cayman', 'Bermuda', 28],
 ['2019-05-21', 'Cayman', 'Taiwan', 59],
 ['2019-06-26', 'Macao', 'Cyprus', 47],
 ['2019-06-29', 'Taiwan', 'Cayman', 132],
 ['2019-07-02', 'Macao', 'Taiwan', 149],
 ['2019-08-18', 'Virgin Islands', 'Taiwan', 72],
 ['2019-09-13', 'Luxembourg', 'Bermuda', 39],
 ['2019-10-14', 'Bermuda', 'Luxembourg', 52],
 ['2019-11-20', 'Bermuda', 'Virgin Islands', 43],
 ['2020-05-20', 'Virgin Islands', 'Luxembourg', 18],
 ['2020-11-20', 'Singapore', 'Taiwan', 86],
 ['2020-12-21', 'Cyprus', 'Luxembourg', 43],
 ['2020-12-22', 'Bermuda', 'Luxembourg', 13],
]

>>> res = offshore(transactions, 2019)
```



[14]:

```

import networkx as nx
from soft import draw_nx

def offshore(transactions, year):
 raise Exception('TODO IMPLEMENT ME !')

transactions = [
 ['2018-02-10', 'Taiwan', 'Cyprus', 30],
 ['2018-02-12', 'Taiwan', 'Virgin Islands', 83],
 ['2018-02-14', 'Cayman', 'Cyprus', 34],
 ['2018-03-25', 'Cayman', 'Bermuda', 143],
 ['2018-03-28', 'Cyprus', 'Macao', 72],
 ['2018-04-17', 'Cayman', 'Bermuda', 28],
 ['2019-05-21', 'Cayman', 'Taiwan', 59],
 ['2019-06-26', 'Macao', 'Cyprus', 47],
 ['2019-06-29', 'Taiwan', 'Cayman', 132],
 ['2019-07-02', 'Macao', 'Taiwan', 149],
 ['2019-08-18', 'Virgin Islands', 'Taiwan', 72],
 ['2019-09-13', 'Luxembourg', 'Bermuda', 39],
 ['2019-10-14', 'Bermuda', 'Luxembourg', 52],
 ['2019-11-20', 'Bermuda', 'Virgin Islands', 43],
 ['2020-05-20', 'Virgin Islands', 'Luxembourg', 18],
 ['2020-11-20', 'Singapore', 'Taiwan', 86],
 ['2020-12-21', 'Cyprus', 'Luxembourg', 43],
 ['2020-12-22', 'Bermuda', 'Luxembourg', 13],
]
]

res1 = offshore(transactions, 2019)

weight is different from label !
assert res1['Bermuda']['Luxembourg']['weight'] == 52

```

(continues on next page)

(continued from previous page)

```
#t2 = [['2013-01-27', 'A', 'B', 30],
['2013-12-31', 'C', 'B', 83],
['2014-12-31', 'D', 'A', 24]]
#offshore(t2, 2013)
```

## Challenge - Cashflow

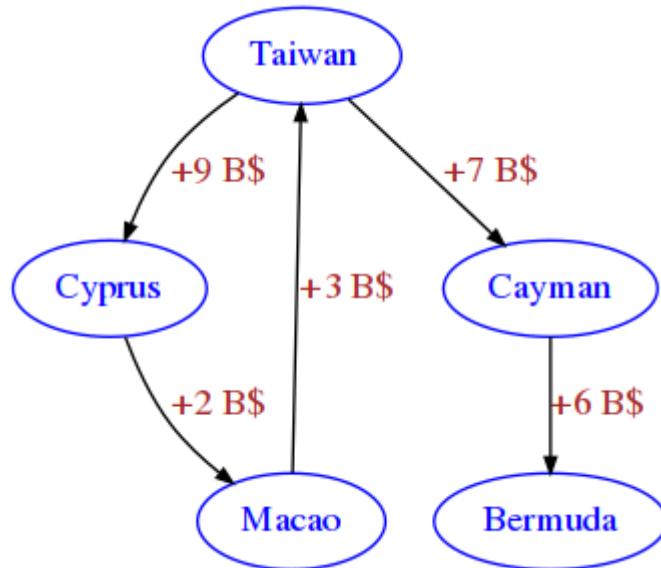
⊕⊕⊕ The Revenue Service is interested in knowing the total positive cash flow which went from a country to another country. For example, if there is a transaction of 7 from Cyprus to Macao and another of 5 from Macao to Cyprus, in the resulting graph you should only show one arrow from Cyprus to Macao with the positive difference 2. The order in which you find transactions in the table is not important, the only important thing is to show at most one arrow. For example, if you first encounter Cayman->Taiwan with 1 and then the reverse Taiwan->Cayman with 8, you will show one arrow Taiwan->Cayman with label '+7 B\$'

Write a function `cash_flow` to display a NetworkX graph

- this time **also RETURN** the graph to allow testing with assert
- **ASSUME** all transactions happen in the same fiscal year - here we removed dates as they are not important
- **DO NOT** show zero valued cash flows
- **HINT:** to solve the exercise, you can first start by summing all transactions which went from one place to another one (like Cayman->Bermuda), without caring about inverse transactions. Having done this, go on and try showing only the positive cash flow

### Example:

```
>>> transactions = [
 ['Taiwan', 'Cyprus', 9], # only one transaction, shows Taiwan->Cyprus 9
 ['Cayman', 'Bermuda', 4], # Cayman->Bermuda 4 sums with Cayman->Bermuda 2
 ↵total Cayman->Bermuda 6
 ['Cyprus', 'Macao', 7], # Cyprus->Macao 7 minus Macao->Cyprus 5 total Cyprus-
 ↵->Macao 2
 ['Cayman', 'Bermuda', 2],
 ['Cayman', 'Taiwan', 1], # Taiwan->Cayman 8 minus Cayman->Taiwan 1 total
 ↵Taiwan->Cayman 7
 ['Macao', 'Cyprus', 5],
 ['Taiwan', 'Cayman', 8],
 ['Macao', 'Taiwan', 3],
 ['Virgin Islands', 'Curacao', 0], # zero, don't show
 ['Luxembourg', 'Singapore', 2], # don't show, total sum with inverse
 ↵transactions is zero
 ['Singapore', 'Luxembourg', 1],
 ['Singapore', 'Luxembourg', 1]
]
>>> cash_flow(transactions)
```



[15]:

```

import networkx as nx
from soft import draw_nx

def cash_flow(transactions):
 raise Exception('TODO IMPLEMENT ME !')

transactions = [
 ['Taiwan', 'Cyprus', 9], # only one transaction, shows Taiwan->Cyprus 9
 ['Cayman', 'Bermuda', 4], # Cayman->Bermuda 4 sums with Cayman->Bermuda 2
 #total Cayman->Bermuda 6
 ['Cyprus', 'Macao', 7], # Cyprus->Macao 7 minus Macao->Cyprus 5 total Cyprus-
 #>Macao 2
 ['Cayman', 'Bermuda', 2],
 ['Cayman', 'Taiwan', 1], # Taiwan->Cayman 8 minus Cayman->Taiwan 1 total
 #>Taiwan->Cayman 7
 ['Macao', 'Cyprus', 5],
 ['Taiwan', 'Cayman', 8],
 ['Macao', 'Taiwan', 3],
 ['Virgin Islands', 'Curacao', 0], # zero, don't show
 ['Luxembourg', 'Singapore', 2], # don't show, total sum with inverse
 #transactions is zero
 ['Singapore', 'Luxembourg', 1],
 ['Singapore', 'Luxembourg', 1]
]

res = cash_flow(transactions)

assert res['Cayman']['Bermuda']['weight'] == 6
assert res['Cyprus']['Macao']['weight'] == 2
assert res['Taiwan']['Cayman']['weight'] == 7
assert not res.has_edge('Virgin Islands','Curacao')
assert not res.has_edge('Singapore','Luxembourg')
assert not res.has_edge('Luxembourg','Singapore')

```

## C - APPLICATIONS

### 9.1 Database

#### 9.1.1 Download exercises zip

Browse files online<sup>427</sup>

In this tutorial we will give a simple overview of connecting to databases with Python, focusing on:

- using SQLStudio to connect to a SQLite database
- simple SQL queries from Python
- examples using pandas module

#### 9.1.2 What to do

1. Unzip `exercises.zip` in a folder, you should obtain something like this:

```
database
database.ipynb
database-sol.ipynb
jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook. Two things should open, first a console and then a browser. The browser should show a file list: navigate the folders and open the notebook `database.ipynb`
3. Go on reading the exercises file, sometimes you will find paragraphs marked **Exercises** which will ask to write Python commands in the following cells.

Shortcut keys:

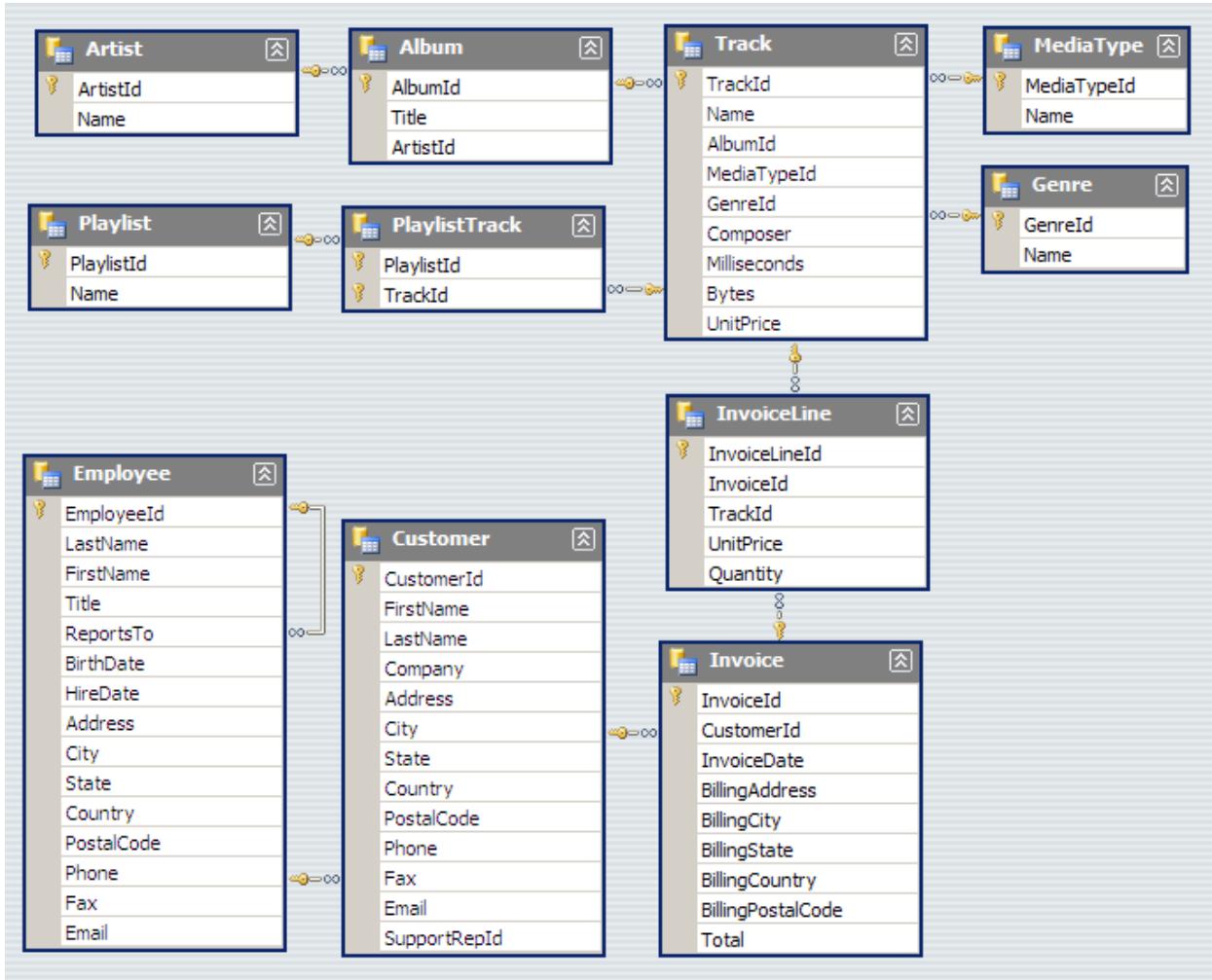
- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

<sup>427</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/database>

### 9.1.3 A first look to the database

We will try accessing the Chihook database by both SQLiteStudio app and Python.

The Chinook data model represents an online store of songs, and includes tables Artist, Album, Track, Invoice and Customer:



Data comes from various sources:

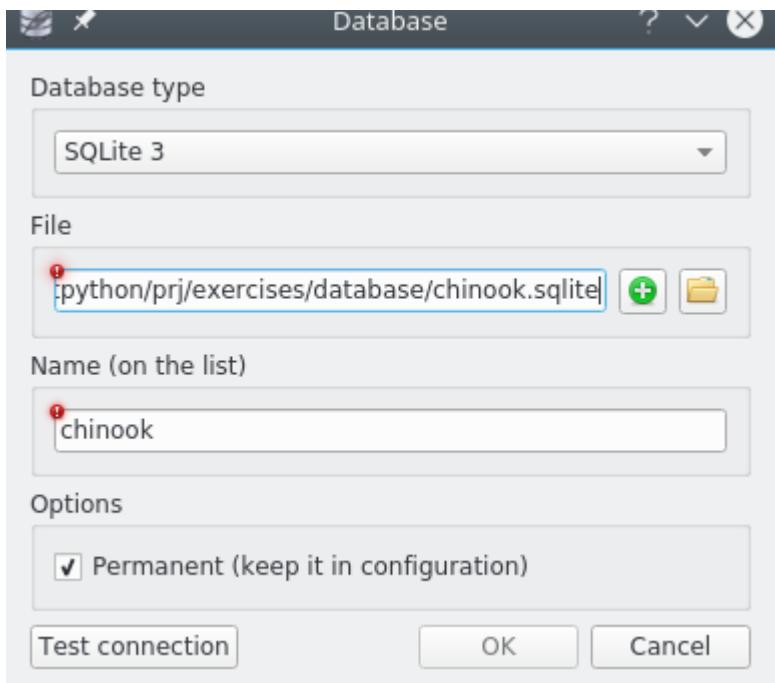
- song data were created using real data from iTunes catalog
- clients data was manually created by using fake names
- addresses are georeferenciable on Google Maps, and other data is well formatted (phone, fax, email, etc.)
- sales data was auto-generated by using random data for a long 4 years period

### 9.1.4 SQLStudio connection

Download<sup>428</sup> and try running SQLite Studio (no admin privileges are needed). If it gives you troubles, as an alternative you might try SQLite browser<sup>429</sup>.

Once SQLStudio is downloaded and unzipped, execute it and then:

1. From the top menu, click Database->Add Database and connect to database chinook.sqlite:



2. Click on Test connection to verify the connection is working, then hit OK.

Let's see a simple table like Album.

**EXERCISE:** Before going on, in SQLiteStudio find the top left menu under the node Tables and double-click on the Album table

Now, in the main panel on the right select Data tab:

<sup>428</sup> <https://sqlitestudio.pl>

<sup>429</sup> <http://sqlitebrowser.org/>

AlbumId	Title	ArtistId
1	For Those About To Rock We Salute You	1
2	Balls to the Wall	2
3	Restless and Wild	2
4	Let There Be Rock	1
5	Big Ones	3
6	Jagged Little Pill	4
7	Facelift	5
8	Warner 25 Anos	6
9	Plays Metallica By Four Cellos	7

We see 3 columns, a couple with numbers `AlbumId` and `ArtistId`, and one of strings called `Title`

**NOTE:** column names in SQL may be arbitrarily given by the database creators. So it is no strictly necessary for column names to end with `Id`.

### 9.1.5 Python connection

Let's try now to retrieve in Python the same data from `Album` table. SQLite is so popular that the module to access it is directly provided with Python, so we don't need to install anything in particular and we can directly dive into coding:

```
[1]: import sqlite3

conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)
```

The operation above creates a connection object and assigns it to the `conn` variable.

But what are we connecting to? The database seems located by the `uri` file:`chinook.sqlite?mode=rw`. But what's an URI? It's a string which denotes a location somewhere, like a database accessible as a service over the internet, or a file on our disk: in this case we want to point to a database we have on disk, so we will use the protocol `file`:

SQLite will then go looking searching the disk for the file `chinook.sqlite`, in the same folder where we are executing Jupyter. If the file were in some subdirectory, we could write something like `some/folder/chinook.sqlite`

**NOTE 1:** we are connecting to the database in binary format `.sqlite`, NOT to the text file `.sql` !

**NOTE 2:** we are specifying we want to open it in `mode=rw`, which means read + write. IF the database doesn't exist, this function will raise an error.

**NOTE 3:** if we wanted to create a new database, we should use the mode read + write + creation, specifying as parameter `mode=rwc` (note the final c )

**NOTE 4:** in many database systems (SQLite included), when we connect to a non-existing database, by default a new one is created. This is cause of many curses, because if by mistake you write a wrong database name no errors appear, and you will find yourself connected to an empty database - wondering where the data is gone. Worse, you will also find your disk filled with wrong database names!

By means of the connection object `conn` we can create a so called `cursor`, which will allow us to execute queries on the database. By using a connection to perform a query, we are telling Python to ask a resource to the system. Good etiquette tells us that whenever we borrow something, after using it we should give it back. In Python the 'giving back' would mean *closing* the opened resource. But while we are using the resource errors might happen, which would prevent

us from properly closing the resource. To ensure Python will properly close the resource automatically on error, we can use the command `with` as we've already done for files<sup>430</sup>:

```
[2]: import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

with conn: # 'with' shields ourselves from unpredictable errors
 cursor = conn.cursor() # we obtain the cursor
 cursor.execute("SELECT * FROM Album LIMIT 5") # execute a query to database
 # in SQL language
 # note 'execute' call does not
 # return values

 for row in cursor.fetchall(): # cursor fetchall() generates a sequence
 # of rows as query result
 # in sequence, the rows are assigned to
 # 'row' object one at a time
 # we print the obtained row
 print(row)

(1, 'For Those About To Rock We Salute You', 1)
(2, 'Balls to the Wall', 2)
(3, 'Restless and Wild', 2)
(4, 'Let There Be Rock', 1)
(5, 'Big Ones', 3)
```

Finally we obtained the list of first 5 database rows from the `Album` table.

**EXERCISE:** try writing down here the instructions to directly print the whole result from `cursor.fetchall()`

- What type of object do you obtain?
- Furthermore, what's the type of the single rows (note they are represented in round parenthesis)?

## 9.1.6 Performance

Databases are specifically designed to handle great amount of data to be stored in hard-disks. Let's briefly review the various types of memory available in a computer, and how they are used in databases:

Memory	Velocity*	Quantity	Notes
RAM	1x	4-16 gigabytes	erased when computer turns off
SSD Disk	2x-10x	hundreds of gigabytes	persistent, but too many writes may ruin it
Hard disk	100x	hundreds of gigabytes, terabytes	persistent, can support numerous write cycles

\* slowness with respect to RAM

If we perform complex queries which potentially deal with a lot of data, we can't always store everything into the RAM. Suppose we're asking the db to calculate the average of all song sales (let's imagine we have a terabyte of songs). Luckily enough, very often the database is smart enough to create a plan to optimize resource usage.

When thinking about the sold songs, it could autonomously perform all these operations:

1. load from hard-disk to RAM 4 gigabytes of songs
2. calculate average sales of these songs in the current RAM block
3. unload the RAM
4. load other 4 gigabytes of songs from hard-disk to RAM

<sup>430</sup> <https://en.softpython.org/formats/formatstl-lines-sol.html#with-block>

5. calculate average sales of second songs block in RAM block, and aggregate with the previously calculated average
6. unload the RAM
7. etc ....

In an ideal scenario, we can write complex SQL queries and hope the database rapidly gives to Python all the results we needed, thus saving us a lot of work. Alas, sometimes this is not possible: if the database takes forever to perform computations, we could be forced to manually optimize the SQL query, or the way we load and elaborate data in Python. For brevity, in this tutorial we will only deal with the latter case (in a simplified way).

### Taking data a bit at a time

In the first Python commands above, we've seen how to take a bit of rows from the DB by using the SQL option `LIMIT`, and how to load all these rows into a Python list in one shot with `fetchall`. What if we wanted to print to screen *all* the rows from a 1 terabyte table? If we tried to load all of them into a list, Python would saturate all the RAM memory for sure. As an alternative to `fetchall`, we can use the command `fetchmany`, which takes a bit of rows each time:

```
[4]: import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

with conn:
 cursor = conn.cursor()
 cursor.execute("SELECT * FROM Album")
 while True: # as long as True is .. true, that is, the cycle never ends ...
 rows = cursor.fetchmany(5) # takes 5 rows
 if len(rows) > 0: # if we have rows, prints them
 for row in rows:
 print(row)
 else: # otherwise the while cycle
 break # gets interrupted

(1, 'For Those About To Rock We Salute You', 1)
(2, 'Balls to the Wall', 2)
(3, 'Restless and Wild', 2)
(4, 'Let There Be Rock', 1)
(5, 'Big Ones', 3)
(6, 'Jagged Little Pill', 4)
(7, 'Facelift', 5)
(8, 'Warner 25 Anos', 6)
(9, 'Plays Metallica By Four Cellos', 7)
(10, 'Audioslave', 8)
(11, 'Out Of Exile', 8)
(12, 'BackBeat Soundtrack', 9)
(13, 'The Best Of Billy Cobham', 10)
(14, 'Alcohol Fueled Brewtality Live! [Disc 1]', 11)
(15, 'Alcohol Fueled Brewtality Live! [Disc 2]', 11)
(16, 'Black Sabbath', 12)
(17, 'Black Sabbath Vol. 4 (Remaster)', 12)
(18, 'Body Count', 13)
(19, 'Chemical Wedding', 14)
(20, 'The Best Of Buddy Guy - The Millenium Collection', 15)
(21, 'Prenda Minha', 16)
(22, 'Sozinho Remix Ao Vivo', 16)
(23, 'Minha Historia', 17)
(24, 'Afrociberdelia', 18)
(25, 'Da Lama Ao Caos', 18)
(26, 'Acústico MTV [Live]', 19)
(27, 'Cidade Negra - Hits', 19)
```

(continues on next page)

(continued from previous page)

```
(28, 'Na Pista', 20)
(29, 'Axé Bahia 2001', 21)
(30, 'BBC Sessions [Disc 1] [Live]', 22)
(31, 'Bongo Fury', 23)
(32, 'Carnaval 2001', 21)
(33, 'Chill: Brazil (Disc 1)', 24)
(34, 'Chill: Brazil (Disc 2)', 6)
(35, 'Garage Inc. (Disc 1)', 50)
(36, 'Greatest Hits II', 51)
(37, 'Greatest Kiss', 52)
(38, 'Heart of the Night', 53)
(39, 'International Superhits', 54)
(40, 'Into The Light', 55)
(41, 'Meus Momentos', 56)
(42, 'Minha História', 57)
(43, 'MK III The Final Concerts [Disc 1]', 58)
(44, 'Physical Graffiti [Disc 1]', 22)
(45, 'Sambas De Enredo 2001', 21)
(46, 'Supernatural', 59)
(47, 'The Best of Ed Motta', 37)
(48, 'The Essential Miles Davis [Disc 1]', 68)
(49, 'The Essential Miles Davis [Disc 2]', 68)
(50, 'The Final Concerts (Disc 2)', 58)
(51, "Up An' Atom", 69)
(52, 'Vinícius De Moraes - Sem Limite', 70)
(53, 'Vozes do MPB', 21)
(54, 'Chronicle, Vol. 1', 76)
(55, 'Chronicle, Vol. 2', 76)
(56, 'Cássia Eller - Coleção Sem Limite [Disc 2]', 77)
(57, 'Cássia Eller - Sem Limite [Disc 1]', 77)
(58, 'Come Taste The Band', 58)
(59, 'Deep Purple In Rock', 58)
(60, 'Fireball', 58)
(61, "Knocking at Your Back Door: The Best Of Deep Purple in the 80's", 58)
(62, 'Machine Head', 58)
(63, 'Purpendicular', 58)
(64, 'Slaves And Masters', 58)
(65, 'Stormbringer', 58)
(66, 'The Battle Rages On', 58)
(67, "Vault: Def Leppard's Greatest Hits", 78)
(68, 'Outbreak', 79)
(69, 'Djavan Ao Vivo - Vol. 02', 80)
(70, 'Djavan Ao Vivo - Vol. 1', 80)
(71, 'Elis Regina-Minha História', 41)
(72, 'The Cream Of Clapton', 81)
(73, 'Unplugged', 81)
(74, 'Album Of The Year', 82)
(75, 'Angel Dust', 82)
(76, 'King For A Day Fool For A Lifetime', 82)
(77, 'The Real Thing', 82)
(78, 'Deixa Entrar', 83)
(79, 'In Your Honor [Disc 1]', 84)
(80, 'In Your Honor [Disc 2]', 84)
(81, 'One By One', 84)
(82, 'The Colour And The Shape', 84)
(83, 'My Way: The Best Of Frank Sinatra [Disc 1]', 85)
(84, 'Roda De Funk', 86)
```

(continues on next page)

(continued from previous page)

(85, 'As Canções de Eu Tu Eles', 27)  
(86, 'Quanta Gente Veio Ver (Live)', 27)  
(87, 'Quanta Gente Veio ver--Bônus De Carnaval', 27)  
(88, 'Faceless', 87)  
(89, 'American Idiot', 54)  
(90, 'Appetite for Destruction', 88)  
(91, 'Use Your Illusion I', 88)  
(92, 'Use Your Illusion II', 88)  
(93, 'Blue Moods', 89)  
(94, 'A Matter of Life and Death', 90)  
(95, 'A Real Dead One', 90)  
(96, 'A Real Live One', 90)  
(97, 'Brave New World', 90)  
(98, 'Dance Of Death', 90)  
(99, 'Fear Of The Dark', 90)  
(100, 'Iron Maiden', 90)  
(101, 'Killers', 90)  
(102, 'Live After Death', 90)  
(103, 'Live At Donington 1992 (Disc 1)', 90)  
(104, 'Live At Donington 1992 (Disc 2)', 90)  
(105, 'No Prayer For The Dying', 90)  
(106, 'Piece Of Mind', 90)  
(107, 'Powerslave', 90)  
(108, 'Rock In Rio [CD1]', 90)  
(109, 'Rock In Rio [CD2]', 90)  
(110, 'Seventh Son of a Seventh Son', 90)  
(111, 'Somewhere in Time', 90)  
(112, 'The Number of The Beast', 90)  
(113, 'The X Factor', 90)  
(114, 'Virtual XI', 90)  
(115, 'Sex Machine', 91)  
(116, 'Emergency On Planet Earth', 92)  
(117, 'Synkronized', 92)  
(118, 'The Return Of The Space Cowboy', 92)  
(119, 'Get Born', 93)  
(120, 'Are You Experienced?', 94)  
(121, 'Surfing with the Alien (Remastered)', 95)  
(122, 'Jorge Ben Jor 25 Anos', 46)  
(123, 'Jota Quest-1995', 96)  
(124, 'Cafezinho', 97)  
(125, 'Living After Midnight', 98)  
(126, 'Unplugged [Live]', 52)  
(127, 'BBC Sessions [Disc 2] [Live]', 22)  
(128, 'Coda', 22)  
(129, 'Houses Of The Holy', 22)  
(130, 'In Through The Out Door', 22)  
(131, 'IV', 22)  
(132, 'Led Zeppelin I', 22)  
(133, 'Led Zeppelin II', 22)  
(134, 'Led Zeppelin III', 22)  
(135, 'Physical Graffiti [Disc 2]', 22)  
(136, 'Presence', 22)  
(137, 'The Song Remains The Same (Disc 1)', 22)  
(138, 'The Song Remains The Same (Disc 2)', 22)  
(139, 'A Tempestade Ou O Livro Dos Dias', 99)  
(140, 'Mais Do Mesmo', 99)  
(141, 'Greatest Hits', 100)

(continues on next page)

(continued from previous page)

- (142, 'Lulu Santos - RCA 100 Anos De Música - Álbum 01', 101)
- (143, 'Lulu Santos - RCA 100 Anos De Música - Álbum 02', 101)
- (144, 'Misplaced Childhood', 102)
- (145, 'Barulhinho Bom', 103)
- (146, 'Seek And Shall Find: More Of The Best (1963-1981)', 104)
- (147, 'The Best Of Men At Work', 105)
- (148, 'Black Album', 50)
- (149, 'Garage Inc. (Disc 2)', 50)
- (150, "Kill 'Em All", 50)
- (151, 'Load', 50)
- (152, 'Master Of Puppets', 50)
- (153, 'ReLoad', 50)
- (154, 'Ride The Lightning', 50)
- (155, 'St. Anger', 50)
- (156, '...And Justice For All', 50)
- (157, 'Miles Ahead', 68)
- (158, 'Milton Nascimento Ao Vivo', 42)
- (159, 'Minas', 42)
- (160, 'Ace Of Spades', 106)
- (161, 'Demorou...', 108)
- (162, 'Motley Crue Greatest Hits', 109)
- (163, 'From The Muddy Banks Of The Wishkah [Live]', 110)
- (164, 'Nevermind', 110)
- (165, 'Compositores', 111)
- (166, 'Olodum', 112)
- (167, 'Acústico MTV', 113)
- (168, 'Arquivo II', 113)
- (169, 'Arquivo Os Paralamas Do Sucesso', 113)
- (170, 'Bark at the Moon (Remastered)', 114)
- (171, 'Blizzard of Ozz', 114)
- (172, 'Diary of a Madman (Remastered)', 114)
- (173, 'No More Tears (Remastered)', 114)
- (174, 'Tribute', 114)
- (175, 'Walking Into Clarksdale', 115)
- (176, 'Original Soundtracks 1', 116)
- (177, 'The Beast Live', 117)
- (178, 'Live On Two Legs [Live]', 118)
- (179, 'Pearl Jam', 118)
- (180, 'Riot Act', 118)
- (181, 'Ten', 118)
- (182, 'Vs.', 118)
- (183, 'Dark Side Of The Moon', 120)
- (184, 'Os Cães Ladram Mas A Caravana Não Pára', 121)
- (185, 'Greatest Hits I', 51)
- (186, 'News Of The World', 51)
- (187, 'Out Of Time', 122)
- (188, 'Green', 124)
- (189, 'New Adventures In Hi-Fi', 124)
- (190, 'The Best Of R.E.M.: The IRS Years', 124)
- (191, 'Cesta Básica', 125)
- (192, 'Raul Seixas', 126)
- (193, 'Blood Sugar Sex Magik', 127)
- (194, 'By The Way', 127)
- (195, 'Californication', 127)
- (196, 'Retrospective I (1974-1980)', 128)
- (197, 'Santana - As Years Go By', 59)
- (198, 'Santana Live', 59)

(continues on next page)

(continued from previous page)

(199, 'Maquinarama', 130)  
(200, 'O Samba Poconé', 130)  
(201, 'Judas 0: B-Sides and Rarities', 131)  
(202, 'Rotten Apples: Greatest Hits', 131)  
(203, 'A-Sides', 132)  
(204, 'Morning Dance', 53)  
(205, 'In Step', 133)  
(206, 'Core', 134)  
(207, 'Mezmerize', 135)  
(208, '[1997] Black Light Syndrome', 136)  
(209, 'Live [Disc 1]', 137)  
(210, 'Live [Disc 2]', 137)  
(211, 'The Singles', 138)  
(212, 'Beyond Good And Evil', 139)  
(213, 'Pure Cult: The Best Of The Cult (For Rockers, Ravers, Lovers & Sinners) [UK]',  
→139)  
(214, 'The Doors', 140)  
(215, 'The Police Greatest Hits', 141)  
(216, 'Hot Rocks, 1964-1971 (Disc 1)', 142)  
(217, 'No Security', 142)  
(218, 'Voodoo Lounge', 142)  
(219, 'Tangents', 143)  
(220, 'Transmission', 143)  
(221, 'My Generation - The Very Best Of The Who', 144)  
(222, 'Serie Sem Limite (Disc 1)', 145)  
(223, 'Serie Sem Limite (Disc 2)', 145)  
(224, 'Acústico', 146)  
(225, 'Volume Dois', 146)  
(226, 'Battlestar Galactica: The Story So Far', 147)  
(227, 'Battlestar Galactica, Season 3', 147)  
(228, 'Heroes, Season 1', 148)  
(229, 'Lost, Season 3', 149)  
(230, 'Lost, Season 1', 149)  
(231, 'Lost, Season 2', 149)  
(232, 'Achtung Baby', 150)  
(233, "All That You Can't Leave Behind", 150)  
(234, 'B-Sides 1980-1990', 150)  
(235, 'How To Dismantle An Atomic Bomb', 150)  
(236, 'Pop', 150)  
(237, 'Rattle And Hum', 150)  
(238, 'The Best Of 1980-1990', 150)  
(239, 'War', 150)  
(240, 'Zooropa', 150)  
(241, 'UB40 The Best Of - Volume Two [UK]', 151)  
(242, 'Diver Down', 152)  
(243, 'The Best Of Van Halen, Vol. I', 152)  
(244, 'Van Halen', 152)  
(245, 'Van Halen III', 152)  
(246, 'Contraband', 153)  
(247, 'Vinicius De Moraes', 72)  
(248, 'Ao Vivo [IMPORT]', 155)  
(249, 'The Office, Season 1', 156)  
(250, 'The Office, Season 2', 156)  
(251, 'The Office, Season 3', 156)  
(252, 'Un-Led-Ed', 157)  
(253, 'Battlestar Galactica (Classic), Season 1', 158)  
(254, 'Aquaman', 159)

(continues on next page)

(continued from previous page)

- (255, 'Instant Karma: The Amnesty International Campaign to Save Darfur', 150)  
 (256, 'Speak of the Devil', 114)  
 (257, '20th Century Masters - The Millennium Collection: The Best of Scorpions', 179)  
 (258, 'House of Pain', 180)  
 (259, 'Radio Brasil (O Som da Jovem Vanguarda) - Seleccao de Henrique Amaro', 36)  
 (260, 'Cake: B-Sides and Rarities', 196)  
 (261, 'LOST, Season 4', 149)  
 (262, 'Quiet Songs', 197)  
 (263, 'Muso Ko', 198)  
 (264, 'Realize', 199)  
 (265, 'Every Kind of Light', 200)  
 (266, 'Duos II', 201)  
 (267, 'Worlds', 202)  
 (268, 'The Best of Beethoven', 203)  
 (269, 'Temple of the Dog', 204)  
 (270, 'Carry On', 205)  
 (271, 'Revelations', 8)  
 (272, 'Adorate Deum: Gregorian Chant from the Proper of the Mass', 206)  
 (273, 'Allegri: Miserere', 207)  
 (274, 'Pachelbel: Canon & Gigue', 208)  
 (275, 'Vivaldi: The Four Seasons', 209)  
 (276, 'Bach: Violin Concertos', 210)  
 (277, 'Bach: Goldberg Variations', 211)  
 (278, 'Bach: The Cello Suites', 212)  
 (279, 'Handel: The Messiah (Highlights)', 213)  
 (280, 'The World of Classical Favourites', 214)  
 (281, 'Sir Neville Marriner: A Celebration', 215)  
 (282, 'Mozart: Wind Concertos', 216)  
 (283, 'Haydn: Symphonies 99 - 104', 217)  
 (284, 'Beethoven: Symphonies Nos. 5 & 6', 218)  
 (285, 'A Soprano Inspired', 219)  
 (286, 'Great Opera Choruses', 220)  
 (287, 'Wagner: Favourite Overtures', 221)  
 (288, 'Fauré: Requiem, Ravel: Pavane & Others', 222)  
 (289, 'Tchaikovsky: The Nutcracker', 223)  
 (290, 'The Last Night of the Proms', 224)  
 (291, 'Puccini: Madama Butterfly - Highlights', 225)  
 (292, 'Holst: The Planets, Op. 32 & Vaughan Williams: Fantasies', 226)  
 (293, "Pavarotti's Opera Made Easy", 227)  
 (294, "Great Performances - Barber's Adagio and Other Romantic Favorites for Strings",  
 ↪ 228)  
 (295, 'Carmina Burana', 229)  
 (296, 'A Copland Celebration, Vol. I', 230)  
 (297, 'Bach: Toccata & Fugue in D Minor', 231)  
 (298, 'Prokofiev: Symphony No.1', 232)  
 (299, 'Scheherazade', 233)  
 (300, 'Bach: The Brandenburg Concertos', 234)  
 (301, 'Chopin: Piano Concertos Nos. 1 & 2', 235)  
 (302, 'Mascagni: Cavalleria Rusticana', 236)  
 (303, 'Sibelius: Finlandia', 237)  
 (304, 'Beethoven Piano Sonatas: Moonlight & Pastorale', 238)  
 (305, 'Great Recordings of the Century - Mahler: Das Lied von der Erde', 240)  
 (306, 'Elgar: Cello Concerto & Vaughan Williams: Fantasias', 241)  
 (307, 'Adams, John: The Chairman Dances', 242)  
 (308, "Tchaikovsky: 1812 Festival Overture, Op.49, Capriccio Italien & Beethoven: Wellington's Victory", 243)  
 (309, 'Palestrina: Missa Papae Marcelli & Allegri: Miserere', 244)

(continues on next page)

(continued from previous page)

```
(310, 'Prokofiev: Romeo & Juliet', 245)
(311, 'Strauss: Waltzes', 226)
(312, 'Berlioz: Symphonie Fantastique', 245)
(313, 'Bizet: Carmen Highlights', 246)
(314, 'English Renaissance', 247)
(315, 'Handel: Music for the Royal Fireworks (Original Version 1749)', 208)
(316, 'Grieg: Peer Gynt Suites & Sibelius: Pelléas et Mélisande', 248)
(317, 'Mozart Gala: Famous Arias', 249)
(318, 'SCRIABIN: Vers la flamme', 250)
(319, 'Armada: Music from the Courts of England and Spain', 251)
(320, 'Mozart: Symphonies Nos. 40 & 41', 248)
(321, 'Back to Black', 252)
(322, 'Frank', 252)
(323, 'Carried to Dust (Bonus Track Version)', 253)
(324, 'Beethoven: Symphony No. 6 "Pastoral" Etc.', 254)
(325, 'Bartok: Violin & Viola Concertos', 255)
(326, "Mendelssohn: A Midsummer Night's Dream", 256)
(327, 'Bach: Orchestral Suites Nos. 1 - 4', 257)
(328, 'Charpentier: Divertissements, Airs & Concerts', 258)
(329, 'South American Getaway', 259)
(330, 'Górecki: Symphony No. 3', 260)
(331, 'Purcell: The Fairy Queen', 261)
(332, 'The Ultimate Relaxation Album', 262)
(333, 'Purcell: Music for the Queen Mary', 263)
(334, 'Weill: The Seven Deadly Sins', 264)
(335, 'J.S. Bach: Chaconne, Suite in E Minor, Partita in E Major & Prelude, Fugue and Allegro', 265)
(336, 'Prokofiev: Symphony No.5 & Stravinsky: Le Sacre Du Printemps', 248)
(337, 'Szymanowski: Piano Works, Vol. 1', 266)
(338, 'Nielsen: The Six Symphonies', 267)
(339, "Great Recordings of the Century: Paganini's 24 Caprices", 268)
(340, "Liszt - 12 Études D'Execution Transcendante", 269)
(341, 'Great Recordings of the Century - Schubert: Schwanengesang, 4 Lieder', 270)
(342, 'Locatelli: Concertos for Violin, Strings and Continuo, Vol. 3', 271)
(343, 'Respighi: Pines of Rome', 226)
(344, "Schubert: The Late String Quartets & String Quintet (3 CD's)", 272)
(345, "Monteverdi: L'Orfeo", 273)
(346, 'Mozart: Chamber Music', 274)
(347, 'Koyaanisqatsi (Soundtrack from the Motion Picture)', 275)
```

## 9.1.7 Passing parameters to the query

What if we wanted an easy way to pass parameters to the query, like for example the number of results to fetch? To this end, we can use so-called *placeholders*, which are question mark characters ? marking where we want to put the variables into. In this case we will substitute the 5 with a question mark ?, and pass 5 in a separate parameter list:

```
[5]: import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

with conn: # 'with' block takes care of unexpected errors
 cursor = conn.cursor() # obtain the cursor

 # we execute a query to the db in SQL language
 # note 'execute' call doesn't return stuff
```

(continues on next page)

(continued from previous page)

```

cursor.execute("SELECT * FROM Album LIMIT ?", [5])

for riga in cursor.fetchall(): # cursor.fetchall() generates a sequence of
 # rows holding the query results. One at a
 # time, rows are assigned to the 'row' object
 print(riga) # print the obtained row

(1, 'For Those About To Rock We Salute You', 1)
(2, 'Balls to the Wall', 2)
(3, 'Restless and Wild', 2)
(4, 'Let There Be Rock', 1)
(5, 'Big Ones', 3)

```

We can also put several question marks, and then for each simply pass the corresponding parameter in the list:

```

[6]: import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

with conn: # 'with' block takes care of unexpected errors
 cursores = conn.cursor() # obtain the cursor
 cursores.execute("SELECT * FROM Album WHERE AlbumId < ? AND ArtistId < ?", [30,5])

 for riga in cursores.fetchall(): # cursor.fetchall() generates a sequence of
 # rows holding the query results. One at a
 # time, rows are assigned to the 'row' object
 print(riga) # print the obtained row

(1, 'For Those About To Rock We Salute You', 1)
(2, 'Balls to the Wall', 2)
(3, 'Restless and Wild', 2)
(4, 'Let There Be Rock', 1)
(5, 'Big Ones', 3)
(6, 'Jagged Little Pill', 4)

```

## 9.1.8 Execute query function

To ease further operations, we define a function `exec_query` which runs the desired query and returns a list of fetched rows:

**IMPORTANT:** Hit Ctrl+Enter in the following cell so Python will later recognize the function:

```

[7]: def exec_query(conn, query, params=()):
 """
 Executes a query by using the connection conn, and then returns a list with the
 obtained results.

 In params we can put a list of parameters for our query
 """
 with conn:
 cur = conn.cursor()
 cur.execute(query, params)
 return cur.fetchall()

```

Let's try:

```
[8]: import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

exec_query(conn, "SELECT * FROM Album LIMIT 5")

[8]: [(1, 'For Those About To Rock We Salute You', 1),
 (2, 'Balls to the Wall', 2),
 (3, 'Restless and Wild', 2),
 (4, 'Let There Be Rock', 1),
 (5, 'Big Ones', 3)]
```

Even better, for extra clarity we can rewrite the query by using a string on many lines with enclosing triple double quotes:

```
[9]: import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)

exec_query(conn, """
SELECT *
FROM Album
LIMIT 5
""")

[9]: [(1, 'For Those About To Rock We Salute You', 1),
 (2, 'Balls to the Wall', 2),
 (3, 'Restless and Wild', 2),
 (4, 'Let There Be Rock', 1),
 (5, 'Big Ones', 3)]
```

Let's try passing some parameters:

```
[10]: exec_query(conn, """
SELECT *
FROM Album
WHERE AlbumId < ? AND ArtistId < ?
""", [30, 5])

[10]: [(1, 'For Those About To Rock We Salute You', 1),
 (2, 'Balls to the Wall', 2),
 (3, 'Restless and Wild', 2),
 (4, 'Let There Be Rock', 1),
 (5, 'Big Ones', 3),
 (6, 'Jagged Little Pill', 4)]
```

**EXERCISE:** Try creating a query in SQLStudio to select albums with id between 3 and 5 included:

1. open the query editor with Alt+E
2. write the query
3. execute it by hitting F9

**EXERCISE:** call `exec_query` function with the same query, using parameters

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[11]: # write here the command

exec_query(conn,
```

(continues on next page)

(continued from previous page)

```
"""
SELECT * FROM Album
WHERE AlbumId >= ? AND AlbumId <= ?
"""", (3, 5))

[11]: [(3, 'Restless and Wild', 2), (4, 'Let There Be Rock', 1), (5, 'Big Ones', 3)]
```

</div>

```
[11]: # write here the command

[11]: [(3, 'Restless and Wild', 2), (4, 'Let There Be Rock', 1), (5, 'Big Ones', 3)]
```

## Table structure

**EXERCISE:** Have a better look at the tab Structure of Album:

The screenshot shows the SQLiteStudio interface with the 'Structure' tab selected for the 'Album' table. The left sidebar displays the database structure with 'chinook' database expanded, showing 'Tables (11)' and 'Album' selected. The main window shows the table structure with three columns: AlbumId (INTEGER, Primary Key, Not NULL), Title (NVARCHAR(160)), and ArtistId (INTEGER). Below the table definition, the 'Details' section shows a primary key constraint (PK\_Album) on the AlbumId column and a foreign key constraint (ArtistId) referencing the Artist table's ArtistId column.

Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Default value
1 AlbumId	INTEGER	key				no		NULL
2 Title	NVARCHAR (160)					no		NULL
3 ArtistId	INTEGER		key			no		NULL

Type	Name	Details
1 PRIMARY KEY	PK_Album	(AlbumId)
2 FOREIGN KEY		(ArtistId) REFERENCES Artist (ArtistId) ON DELETE NO ACTION ON UP...

## DDL

Compare above stuff with the tab DDL (Data Definition Language), which contains SQL instructions to create the table in the database:

The screenshot shows the SQLiteStudio interface. On the left, the Database browser displays a tree structure of the 'chinook' database, including tables like 'Album' and 'Artist'. On the right, the main window has tabs for Structure, Data, Constraints, Indexes, Triggers, and DDL. The DDL tab is active, showing the SQL code for creating the 'Album' table:

```

CREATE TABLE Album (
 AlbumId INTEGER NOT NULL,
 Title NVARCHAR (160) NOT NULL,
 ArtistId INTEGER NOT NULL,
 CONSTRAINT PK_Album PRIMARY KEY (
 AlbumId
),
 FOREIGN KEY (
 ArtistId
)
 REFERENCES Artist (ArtistId) ON DELETE NO ACTION
 ON UPDATE NO ACTION
);

```

A feature of databases is the possibility to declare constraints on the inserted data. For example, here we note that:

- the table `Album` has a `PRIMARY KEY`, asserting there cannot be two rows with the same `AlbumId`
- the table `Album` defines the column `ArtistId` as a `FOREIGN KEY`, asserting that for each value in that column, there must always be a corresponding existing id in the column `ArtistId` **from Artist table**. Thus, it will be impossible to refer to a non-existing artist.

**EXERCISE:** Go to tab Data and try changing an `ArtistId` by placing a non-existing number (like 1000). Apparently the database won't complain, but only because we haven't recorded the change on disk yet, in other words, we haven't still performed a *commit* operation. Commits allow us to execute many operations in an atomic way, meaning that either *all* changes are recorded to disk or *none* of the changes are performed.

Try executing a commit by clicking the green button with the tick (or by hitting `Ctrl-Return`). What happens? To recover from the damage just inflicted to the database, click the red button *rollback* with the x (or hit `Ctrl-Backspace`).

## Query to metadata

An interesting and sometimes useful feature of many SQL databases is the presence of metadata describing the table structure, and the metadata itself may be stored in tabular format. For example, with SQLite you can execute a query like this (we don't explain it in detail and just show some example):

```
[12]: def query_schema(conn, table):
 """ Return a string with the SQL instructions to create a table
 (without the data)
 """
 return exec_query(conn, """
SELECT sql FROM sqlite_master
WHERE name = ?
""", (table,))[0][0]
```

```
[13]: import sqlite3
conn = sqlite3.connect('file:chinook.sqlite?mode=rw', uri=True)
```

(continues on next page)

(continued from previous page)

```
print(query_schema(conn, 'Album'))

CREATE TABLE [Album]
(
 [AlbumId] INTEGER NOT NULL,
 [Title] NVARCHAR(160) NOT NULL,
 [ArtistId] INTEGER NOT NULL,
 CONSTRAINT [PK_Album] PRIMARY KEY ([AlbumId]),
 FOREIGN KEY ([ArtistId]) REFERENCES [Artist] ([ArtistId])
 ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

## 9.1.9 ORDER BY

Very often we will want to sort the result according to some column: to do so we can add the ORDER BY clause.

**NOTE:** if we add LIMIT, it is going to be applied AFTER the sorting has been performed:

```
[14]: exec_query(conn, """
SELECT *
FROM Album
ORDER BY Album.Title
LIMIT 10
""")

[14]: [(156, '...And Justice For All', 50),
(257,
 '20th Century Masters - The Millennium Collection: The Best of Scorpions',
 179),
(296, 'A Copland Celebration, Vol. I', 230),
(94, 'A Matter of Life and Death', 90),
(95, 'A Real Dead One', 90),
(96, 'A Real Live One', 90),
(285, 'A Soprano Inspired', 219),
(139, 'A TempestadeTempestade Ou O Livro Dos Dias', 99),
(203, 'A-Sides', 132),
(160, 'Ace Of Spades', 106)]
```

To sort in descending order we can add DESC:

```
[15]: exec_query(conn, """
SELECT *
FROM Album
ORDER BY Album.Title DESC
LIMIT 10
""")

[15]: [(208, '[1997] Black Light Syndrome', 136),
(240, 'Zooropa', 150),
(267, 'Worlds', 202),
(334, 'Weill: The Seven Deadly Sins', 264),
(8, 'Warner 25 Anos', 6),
(239, 'War', 150),
(175, 'Walking Into Clarksdale', 115),
```

(continues on next page)

(continued from previous page)

```
(287, 'Wagner: Favourite Overtures', 221),
(182, 'Vs.', 118),
(53, 'Vozes do MPB', 21)]
```

### 9.1.10 JOIN

In the Album table for artists we only see some numbers. How can we perform a query to also see the artist names? We can try the SQL command JOIN.

**EXERCISE:** To understand what happens, execute the query in SQLStudio

```
[16]: exec_query(conn, """
SELECT *
FROM Album JOIN Artist
WHERE Album.ArtistId = Artist.ArtistId
LIMIT 5
""")

[16]: [(1, 'For Those About To Rock We Salute You', 1, 1, 'AC/DC'),
(2, 'Balls to the Wall', 2, 2, 'Accept'),
(3, 'Restless and Wild', 2, 2, 'Accept'),
(4, 'Let There Be Rock', 1, 1, 'AC/DC'),
(5, 'Big Ones', 3, 3, 'Aerosmith')]
```

Instead of the JOIN, we can use a comma , :

```
[17]: exec_query(conn, """
SELECT * FROM Album, Artist
WHERE Album.ArtistId = Artist.ArtistId
LIMIT 5
""")

[17]: [(1, 'For Those About To Rock We Salute You', 1, 1, 'AC/DC'),
(2, 'Balls to the Wall', 2, 2, 'Accept'),
(3, 'Restless and Wild', 2, 2, 'Accept'),
(4, 'Let There Be Rock', 1, 1, 'AC/DC'),
(5, 'Big Ones', 3, 3, 'Aerosmith')]
```

Even better, since in this case we have the same column name in both tables, we can try the USING clause which also eliminates the duplicated column.

**NOTE:** For obscure reasons, in SQLiteStudio the column ArtistId appears duplicated anyway with the name ArtistId:1

```
[18]: exec_query(conn, """
SELECT *
FROM Album, Artist USING(ArtistId)
LIMIT 5
""")

[18]: [(1, 'For Those About To Rock We Salute You', 1, 'AC/DC'),
(2, 'Balls to the Wall', 2, 'Accept'),
(3, 'Restless and Wild', 2, 'Accept'),
(4, 'Let There Be Rock', 1, 'AC/DC'),
(5, 'Big Ones', 3, 'Aerosmith')]
```

Finally, we can select only the column we're interested in: album Title and artist Name. For added clarity, we can identify the tables with variables we assign in FROM clause - here we use the names ALB and ART but they could be any of your choice:

```
[19]: exec_query(conn, """
SELECT ALB.Title, ART.Name
FROM Album ALB, Artist ART USING(ArtistId)
LIMIT 5
""")

[19]: [('For Those About To Rock We Salute You', 'AC/DC'),
 ('Balls to the Wall', 'Accept'),
 ('Restless and Wild', 'Accept'),
 ('Let There Be Rock', 'AC/DC'),
 ('Big Ones', 'Aerosmith')]
```

### 9.1.11 Track Table

Let's now switch to a more complex table like Track, which contains songs listened by iTunes users:

```
[20]: exec_query(conn, "SELECT * FROM Track LIMIT 5")

[20]: [(1,
 'For Those About To Rock (We Salute You)',
 1,
 1,
 1,
 'Angus Young, Malcolm Young, Brian Johnson',
 343719,
 11170334,
 0.99),
 (2, 'Balls to the Wall', 2, 2, 1, None, 342562, 5510424, 0.99),
 (3,
 'Fast As a Shark',
 3,
 2,
 1,
 'F. Baltes, S. Kaufman, U. Dirksneider & W. Hoffman',
 230619,
 3990994,
 0.99),
 (4,
 'Restless and Wild',
 3,
 2,
 1,
 'F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirksneider & W. Hoffman',
 252051,
 4331779,
 0.99),
 (5,
 'Princess of the Dawn',
 3,
 2,
 1,
 'Deaffy & R.A. Smith-Diesel',
 375418,
```

(continues on next page)

(continued from previous page)

```
6290521,
0.99)]
```

```
[21]: query_schema(conn, "Track")
```

```
[21]: 'CREATE TABLE [Track]\n(\n [TrackId] INTEGER NOT NULL, \n [Name] NVARCHAR(200) ↵
 ↵NOT NULL, \n [AlbumId] INTEGER, \n [MediaTypeId] INTEGER NOT NULL, \n ↵
 ↵[GenreId] INTEGER, \n [Composer] NVARCHAR(220), \n [Milliseconds] INTEGER NOT ↵
 ↵NULL, \n [Bytes] INTEGER, \n [UnitPrice] NUMERIC(10,2) NOT NULL, \n ↵
 ↵CONSTRAINT [PK_Track] PRIMARY KEY ([TrackId]), \n FOREIGN KEY ([AlbumId]) ↵
 ↵REFERENCES [Album] ([AlbumId]) \n ON DELETE NO ACTION ON UPDATE NO ACTION, \n ↵
 ↵FOREIGN KEY ([GenreId]) REFERENCES [Genre] ([GenreId]) \n ON DELETE NO ACTION ON ↵
 ↵UPDATE NO ACTION, \n FOREIGN KEY ([MediaTypeId]) REFERENCES [MediaTypes] ↵
 ↵([MediaTypeId]) \n ON DELETE NO ACTION ON UPDATE NO ACTION\n)'
```

```
[22]: print(query_schema(conn, "Track"))
```

```
CREATE TABLE [Track]
(
 [TrackId] INTEGER NOT NULL,
 [Name] NVARCHAR(200) NOT NULL,
 [AlbumId] INTEGER,
 [MediaTypeId] INTEGER NOT NULL,
 [GenreId] INTEGER,
 [Composer] NVARCHAR(220),
 [Milliseconds] INTEGER NOT NULL,
 [Bytes] INTEGER,
 [UnitPrice] NUMERIC(10,2) NOT NULL,
 CONSTRAINT [PK_Track] PRIMARY KEY ([TrackId]),
 FOREIGN KEY ([AlbumId]) REFERENCES [Album] ([AlbumId])
 ON DELETE NO ACTION ON UPDATE NO ACTION,
 FOREIGN KEY ([GenreId]) REFERENCES [Genre] ([GenreId])
 ON DELETE NO ACTION ON UPDATE NO ACTION,
 FOREIGN KEY ([MediaTypeId]) REFERENCES [MediaTypes] ([MediaTypeId])
 ON DELETE NO ACTION ON UPDATE NO ACTION
)
```

```
[23]: exec_query(conn, """
SELECT Name, Composer
FROM Track
LIMIT 5
""")
```

```
[23]: [('For Those About To Rock (We Salute You)', 'Angus Young, Malcolm Young, Brian Johnson'),
('Balls to the Wall', None),
('Fast As a Shark', 'F. Baltes, S. Kaufman, U. Dirksneider & W. Hoffman'),
('Restless and Wild',
 'F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirksneider & W. Hoffman'),
('Princess of the Dawn', 'Deaffy & R.A. Smith-Diesel')]
```

```
[24]: exec_query(conn, """
SELECT Name, Composer
FROM Track
LIMIT 5
""") [0]
```

```
[24]: ('For Those About To Rock (We Salute You)',
 'Angus Young, Malcolm Young, Brian Johnson')
```

Let's have a look at the second row:

```
[25]: exec_query(conn, """
SELECT Name, Composer
FROM Track
LIMIT 5
""") [1]

[25]: ('Balls to the Wall', None)
```

In this case we note the composer is missing. How is the missing composer represented in the original SQL table?

**EXERCISE:** Using SQLiteStudio, in the left menu double click on the Track table and then select the Data table on the right. Scroll the rows until you find the box with the column Composer.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"  
data-jupman-show="Show answer"  
data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

#### ANSWER:

We note in SQL the empty boxes are denoted with NULL. Since NULL is not a Python type, the NULL SQL object gets converted to the pythonic None.

</div>

Let's try selecting some numerical values in our query, like for example the Milliseconds:

```
[26]: exec_query(conn, """
SELECT Name, Milliseconds
FROM Track
LIMIT 5
""")

[26]: [('For Those About To Rock (We Salute You)', 343719),
 ('Balls to the Wall', 342562),
 ('Fast As a Shark', 230619),
 ('Restless and Wild', 252051),
 ('Princess of the Dawn', 375418)]
```

```
[27]: exec_query(conn, """
SELECT Name, Milliseconds
FROM Track
LIMIT 5
""") [0]

[27]: ('For Those About To Rock (We Salute You)', 343719)
```

```
[28]: exec_query(conn, """
SELECT Name, Milliseconds
FROM Track
LIMIT 5
""") [0] [0]

[28]: 'For Those About To Rock (We Salute You)'
```

```
[29]: exec_query(conn, """
SELECT Name, Milliseconds
```

(continues on next page)

(continued from previous page)

```
FROM Track
LIMIT 5
""") [0] [1]
```

[29]: 343719

```
[30]: exec_query(conn, """
SELECT Name, Milliseconds
FROM Track
ORDER BY Milliseconds DESC
LIMIT 5
""")
```

```
[30]: [('Occupation / Precipice', 5286953),
('Through a Looking Glass', 5088838),
('Greetings from Earth, Pt. 1', 2960293),
('The Man With Nine Lives', 2956998),
('Battlestar Galactica, Pt. 2', 2956081)]
```

**EXERCISE:** Try using ASC instead of DESC

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[31]: # write here the query

exec_query(conn, """
SELECT Name, Composer, Milliseconds
FROM Track
ORDER BY Milliseconds ASC
LIMIT 5
""")
```

```
[31]: [('É Uma Partida De Futebol', 'Samuel Rosa', 1071),
('Now Sports', None, 4884),
('A Statistic', None, 6373),
('Oprah', None, 6635),
('Commercial 1', 'L. Muggerud', 7941)]
```

</div>

```
[31]: # write here the query
```

```
[31]: [('É Uma Partida De Futebol', 'Samuel Rosa', 1071),
('Now Sports', None, 4884),
('A Statistic', None, 6373),
('Oprah', None, 6635),
('Commercial 1', 'L. Muggerud', 7941)]
```

## 9.1.12 Aggregating data

### COUNT

To count the table rows, we can use the keyword COUNT (\*) in a SELECT. For example, to see how many tracks there are, we can do like this:

```
[32]: exec_query(conn, """
SELECT COUNT(*)
FROM Track
""")
```

```
[32]: [(3503,)]
```

**QUESTION:** the method above is way better than importing all the rows with Python and then performing a `len`. Why?

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a><div class="jupman-sol jupman-sol-question" style="display:none">

### ANSWER:

By counting directly in SQL, the database will try to perform all the needed calculations on its own, and will only send to Python a single number. This is much better than sending many rows (which could potentially be a lot) and thus could end up clogging computer memory.

</div>

### GROUP BY and COUNT

Each Track has associated a MediaTypeId. We might ask ourselves how many tracks are present for each media type

- To count, we will need the keyword COUNT (\*) AS Quantity in the SELECT
- to aggregate we need GROUP BY after the FROM line
- to sort the counts in a decreasing way we will also use ORDER BY Quantity DESC

**Note:** in this case COUNT (\*) will count how many elements there are in each group, not in the whole table

```
[33]: exec_query(conn, """
SELECT T.MediaTypeId, COUNT(*) AS Quantity
FROM Track T
GROUP BY T.MediaTypeId
ORDER BY Quantity DESC
""")
```

```
[33]: [(1, 3034), (2, 237), (3, 214), (5, 11), (4, 7)]
```

**EXERCISE:** The MediaTypeId isn't very descriptive. Write down a query to obtain couples with the MediaType name with the respective count. Try also executing the query in SQLStudio:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[34]: # write here
```

```
exec_query(conn, """
SELECT MT.Name, COUNT(*) AS Quantity
```

(continues on next page)

(continued from previous page)

```
FROM Track T, MediaType MT USING (MediaTypeId)
GROUP BY MT.MediaTypeId
ORDER BY Quantity DESC
""")
```

```
[34]: [('MPEG audio file', 3034),
('Protected AAC audio file', 237),
('Protected MPEG-4 video file', 214),
('AAC audio file', 11),
('Purchased AAC audio file', 7)]
```

</div>

```
[34]: # write here
```

```
[34]: [('MPEG audio file', 3034),
('Protected AAC audio file', 237),
('Protected MPEG-4 video file', 214),
('AAC audio file', 11),
('Purchased AAC audio file', 7)]
```

**EXERCISE:** Write down here a query to create a table of two columns: the first should hold musical genre names, and the second the corresponding number of tracks for that genre.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[35]: # write here
```

```
exec_query(conn, """
SELECT G.Name, COUNT(*) AS Quantity
FROM Track T, Genre G USING (GenreId)
GROUP BY G.GenreId
ORDER BY Quantity DESC
""")
```

```
[35]: [('Rock', 1297),
('Latin', 579),
('Metal', 374),
('Alternative & Punk', 332),
('Jazz', 130),
('TV Shows', 93),
('Blues', 81),
('Classical', 74),
('Drama', 64),
('R&B/Soul', 61),
('Reggae', 58),
('Pop', 48),
('Soundtrack', 43),
('Alternative', 40),
('Hip Hop/Rap', 35),
('Electronica/Dance', 30),
```

(continues on next page)

(continued from previous page)

```
('Heavy Metal', 28),
('World', 28),
('Sci Fi & Fantasy', 26),
('Easy Listening', 24),
('Comedy', 17),
('Bossa Nova', 15),
('Science Fiction', 13),
('Rock And Roll', 12),
('Opera', 1)]
```

&lt;/div&gt;

[35]: # write here

```
[35]: [('Rock', 1297),
('Latin', 579),
('Metal', 374),
('Alternative & Punk', 332),
('Jazz', 130),
('TV Shows', 93),
('Blues', 81),
('Classical', 74),
('Drama', 64),
('R&B/Soul', 61),
('Reggae', 58),
('Pop', 48),
('Soundtrack', 43),
('Alternative', 40),
('Hip Hop/Rap', 35),
('Electronica/Dance', 30),
('Heavy Metal', 28),
('World', 28),
('Sci Fi & Fantasy', 26),
('Easy Listening', 24),
('Comedy', 17),
('Bossa Nova', 15),
('Science Fiction', 13),
('Rock And Roll', 12),
('Opera', 1)]
```

**EXERCISE:** Try now to find the average duration in milliseconds of each genre

- USE the function AVG(Track.Milliseconds) instead of COUNT(\*):

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[36]: # write here

```
exec_query(conn, """
SELECT G.Name, AVG(T.Milliseconds) AS Duration
FROM Track T, Genre G USING (GenreId)
GROUP BY G.GenreId
ORDER BY Duration DESC
""")
```

```
[36]: [('Sci Fi & Fantasy', 2911783.0384615385),
 ('Science Fiction', 2625549.076923077),
 ('Drama', 2575283.78125),
 ('TV Shows', 2145041.0215053763),
 ('Comedy', 1585263.705882353),
 ('Metal', 309749.4438502674),
 ('Electronica/Dance', 302985.8),
 ('Heavy Metal', 297452.9285714286),
 ('Classical', 293867.5675675676),
 ('Jazz', 291755.3769230769),
 ('Rock', 283910.0431765613),
 ('Blues', 270359.7777777775),
 ('Alternative', 264058.525),
 ('Reggae', 247177.75862068965),
 ('Soundtrack', 244370.88372093023),
 ('Alternative & Punk', 234353.84939759035),
 ('Latin', 232859.26252158894),
 ('Pop', 229034.10416666666),
 ('World', 224923.82142857142),
 ('R&B/Soul', 220066.8524590164),
 ('Bossa Nova', 219590.0),
 ('Easy Listening', 189164.2083333334),
 ('Hip Hop/Rap', 178176.2857142857),
 ('Opera', 174813.0),
 ('Rock And Roll', 134643.5)]
```

&lt;/div&gt;

```
[36]: # write here
```

```
[36]: [('Sci Fi & Fantasy', 2911783.0384615385),
 ('Science Fiction', 2625549.076923077),
 ('Drama', 2575283.78125),
 ('TV Shows', 2145041.0215053763),
 ('Comedy', 1585263.705882353),
 ('Metal', 309749.4438502674),
 ('Electronica/Dance', 302985.8),
 ('Heavy Metal', 297452.9285714286),
 ('Classical', 293867.5675675676),
 ('Jazz', 291755.3769230769),
 ('Rock', 283910.0431765613),
 ('Blues', 270359.7777777775),
 ('Alternative', 264058.525),
 ('Reggae', 247177.75862068965),
 ('Soundtrack', 244370.88372093023),
 ('Alternative & Punk', 234353.84939759035),
 ('Latin', 232859.26252158894),
 ('Pop', 229034.10416666666),
 ('World', 224923.82142857142),
 ('R&B/Soul', 220066.8524590164),
 ('Bossa Nova', 219590.0),
 ('Easy Listening', 189164.2083333334),
 ('Hip Hop/Rap', 178176.2857142857),
 ('Opera', 174813.0),
 ('Rock And Roll', 134643.5)]
```

### 9.1.13 Pandas

So far we used Python basic methods, but obviously processing everything in Pandas is way easier.

For more info about Pandas, have a look at its [tutorial](#)<sup>431</sup>

```
[37]: import pandas

df = pandas.read_sql_query("SELECT Name, Composer, Milliseconds from Track", conn)

[38]: df
```

	Name	Composer	Milliseconds
0	For Those About To Rock (We Salute You)	Angus Young, Malcolm Young, Brian Johnson	343719
1	Balls to the Wall	None	342562
2	Fast As a Shark	F. Baltes, S. Kaufman, U. Dirksneider & W. Ho...	230619
3	Restless and Wild	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D...	252051
4	Princess of the Dawn	Deaffy & R.A. Smith-Diesel	375418
...	...	...	...
3498	Pini Di Roma (Pinien Von Rom) \ I Pini Della V...	None	286741
3499	String Quartet No. 12 in C Minor, D. 703 "Quar...	Franz Schubert	139200
3500	L'orfeo, Act 3, Sinfonia (Orchestra)	Claudio Monteverdi	66639
3501	Quintet for Horn, Violin, 2 Violas, and Cello ...	Wolfgang Amadeus Mozart	221331
3502	Koyaanisqatsi	Philip Glass	206005

[3503 rows x 3 columns]

#### BEWARE of big databases !

Pandas is very handy, but as [already explained](#)<sup>432</sup> Pandas loads everything in RAM which in a typical 2022 laptop goes from 4 to 16 gigabytes. If you have a big database you might incur into the problems exposed in section [Performance](#)

**EXERCISE:** Milliseconds and occupied bytes should reasonably be linearly dependent. Show it with Pandas.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[39]: # write here

df = pandas.read_sql_query("SELECT Name, Composer, Milliseconds, Bytes from Track", conn)
```

(continues on next page)

<sup>431</sup> <https://en.softpython.org/pandas/pandas1-sol.html>

<sup>432</sup> <http://en.softpython.org/pandas/pandas1-sol.html>

(continued from previous page)

```
df.corr()

the linear correlation between milliseconds and bytes
is close to the maximum of 1.0

[39]:
 Milliseconds Bytes
Milliseconds 1.000000 0.960181
Bytes 0.960181 1.000000
```

</div>

```
[39]:
write here
```

```
[]:
```

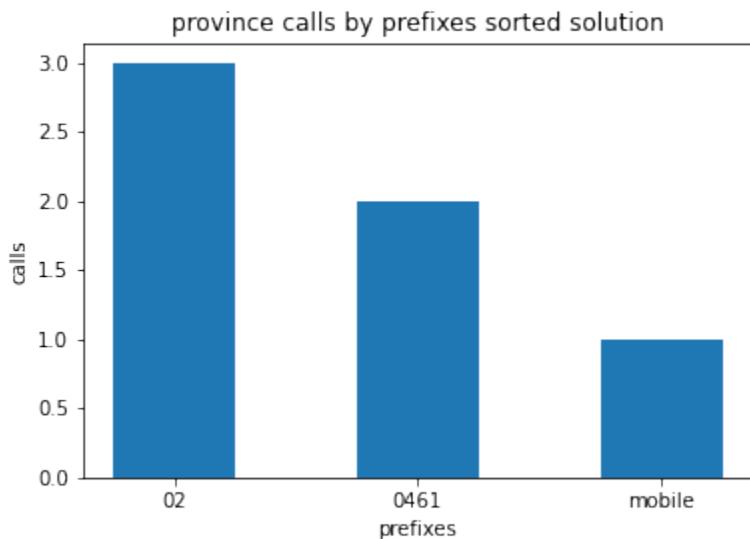
## D - PROJECTS

### 10.1 Text data

#### 10.1.1 Phone calls

**Download worked project**

Browse files online<sup>433</sup>



A radio station gathered calls from listeners, recording just the name of the caller and the phone number, as seen on the phone display. For marketing purposes, the station owner now wants to better understand the locations from where listeners were calling. He then hires you as Algorithmic Market Strategist and asks you to show statistics about the provinces of the calling sites. There is a problem, though. Numbers were written down by hand and sometimes they are not uniform, so it would be better to find a canonical representation.

**NOTE:** Phone prefixes can be a very tricky subject, if you are ever to deal with them seriously please use proper phone number parsing libraries<sup>434</sup> and do read *Falsehoods Programmers Believe About Phone Numbers*<sup>435</sup>

<sup>433</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/phone-calls>

<sup>434</sup> <https://github.com/daviddrysdale/python-phonenumbers>

<sup>435</sup> <https://github.com/googlei18n/libphonenumber/blob/master/FALSEHOODS.md>

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
phone-calls-prj
 phone-calls.ipynb
 phone-calls-sol.ipynb
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook phone-calls.ipynb
3. Go on reading the notebook, and write in the appropriate cells when asked.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### 1. canonical

⊕ We first want to canonicalize a phone number as a string.

We suppose a canonical phone number:

- contains no spaces
- contains no international prefix, so no +39 nor 0039: we assume all calls where placed from Italy (even if they have international prefix)

For example, all of these are canonicalized to “0461123456”:

```
+39 0461 123456
+390461123456
0039 0461 123456
00390461123456
```

These are canonicalized as the following:

```
328 123 4567 -> 3281234567
0039 328 123 4567 -> 3281234567
0039 3771 1234567 -> 37711234567
```

**REMEMBER: strings are immutable !!!!**

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[1]: def canonical(phone):
 """ RETURN the canonical version of phone as a string.
 """
```

(continues on next page)

(continued from previous page)

```

p = phone.replace(' ', '')
if p.startswith('0039'):
 p = p[4:]
if p.startswith('+39'):
 p = p[3:]
return p

assert canonical('+39 0461 123456') == '0461123456'
assert canonical('+390461123456') == '0461123456'
assert canonical('0039 0461 123456') == '0461123456'
assert canonical('00390461123456') == '0461123456'
assert canonical('003902123456') == '02123456'
assert canonical('003902120039') == '02120039'
assert canonical('0039021239') == '021239'

```

&lt;/div&gt;

```
[1]: def canonical(phone):
 """ RETURN the canonical version of phone as a string.
 """
 raise Exception('TODO IMPLEMENT ME !!')

assert canonical('+39 0461 123456') == '0461123456'
assert canonical('+390461123456') == '0461123456'
assert canonical('0039 0461 123456') == '0461123456'
assert canonical('00390461123456') == '0461123456'
assert canonical('003902123456') == '02123456'
assert canonical('003902120039') == '02120039'
assert canonical('0039021239') == '021239'
```

## 2. prefix

⊕⊕ We now want to extract the province prefix - the ones we consider as valid are in `province_prefixes` list. Note some numbers are from mobile operators and you can distinguish them by prefixes like 328 - the ones we consider are in an `mobile_prefixes` list. Write a function that given a phone number RETURN the prefix of the phone as a string.

- Remember first to make it canonical !!
- If phone is mobile, RETURN string 'mobile'. If it is not a phone nor a mobile, RETURN the string 'unrecognized'
- To determine if the phone is mobile or from province, use provided `province_prefixes` and `mobile_prefixes` lists
- **USE the already defined function** `canonical(phone)`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[2]: province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']
```

(continues on next page)

(continued from previous page)

```
def prefix(phone):

 c = canonical(phone)
 for m in mobile_prefixes:
 if c.startswith(m):
 return 'mobile'
 for p in province_prefixes:
 if c.startswith(p):
 return p
 return 'unrecognized'

assert prefix('0461123') == '0461'
assert prefix('+39 0461 4321') == '0461'
assert prefix('0039011 432434') == '011'
assert prefix('328 432434') == 'mobile'
assert prefix('+39340 432434') == 'mobile'
assert prefix('00666011 432434') == 'unrecognized'
assert prefix('12345') == 'unrecognized'
assert prefix('+39 123 12345') == 'unrecognized'
```

&lt;/div&gt;

```
[2]: province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']

def prefix(phone):
 raise Exception('TODO IMPLEMENT ME !')

assert prefix('0461123') == '0461'
assert prefix('+39 0461 4321') == '0461'
assert prefix('0039011 432434') == '011'
assert prefix('328 432434') == 'mobile'
assert prefix('+39340 432434') == 'mobile'
assert prefix('00666011 432434') == 'unrecognized'
assert prefix('12345') == 'unrecognized'
assert prefix('+39 123 12345') == 'unrecognized'
```

### 3. hist

⊕⊕⊕ Write a function that given a list of non-canonical phones, RETURN a dictionary where the keys are the prefixes of the canonical phones and the values are the frequencies of the prefixes (keys may also be unrecognized or mobile)

**NOTE** Numbers corresponding to the same phone (so which have the same canonical representation) must be counted ONLY ONCE!

**USE the already defined functions** canonical(phone) **AND** prefix(phone)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']

def hist(phones):

 d = {}
 s = set()

 for phone in phones:
 c = canonical(phone)
 if c not in s:
 s.add(c)
 p = prefix(phone)
 if p in d:
 d[p] += 1
 else:
 d[p] = 1
 return d

assert hist(['0461123']) == {'0461':1}
assert hist(['123']) == {'unrecognized':1}
assert hist(['328 123']) == {'mobile':1}
assert hist(['0461123', '+390461123']) == {'0461':1} # same canonicals, should be_
↪ counted only once
assert hist(['0461123', '+39 0461 4321']) == {'0461':2}
assert hist(['0461123', '+39 0461 4321', '0039011 432434']) == {'0461':2, '011':1}
assert hist(['+39 02 423', '0461123', '02 426', '+39 0461 4321', '0039328 1234567',
↪ '02 423', '02 424']) == {'0461':2, 'mobile':1, '02':3}
```

&lt;/div&gt;

```
[3]: province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']

def hist(phones):
 raise Exception('TODO IMPLEMENT ME !')

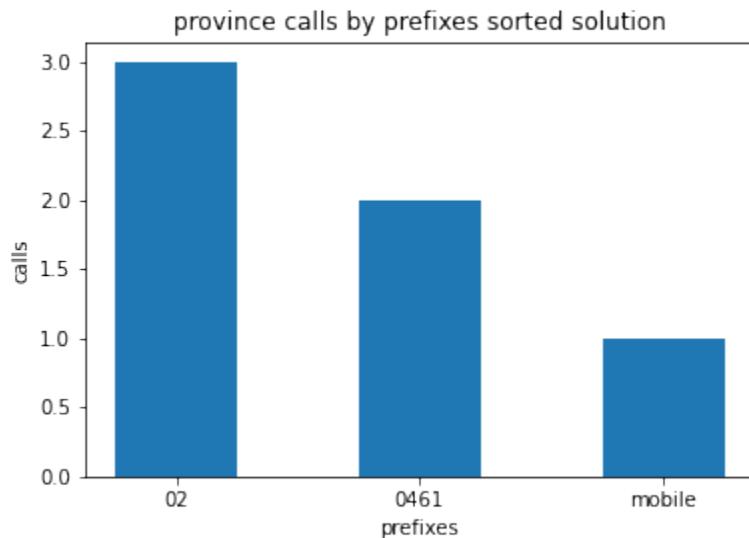
assert hist(['0461123']) == {'0461':1}
assert hist(['123']) == {'unrecognized':1}
assert hist(['328 123']) == {'mobile':1}
assert hist(['0461123', '+390461123']) == {'0461':1} # same canonicals, should be_
↪ counted only once
assert hist(['0461123', '+39 0461 4321']) == {'0461':2}
assert hist(['0461123', '+39 0461 4321', '0039011 432434']) == {'0461':2, '011':1}
assert hist(['+39 02 423', '0461123', '02 426', '+39 0461 4321', '0039328 1234567',
↪ '02 423', '02 424']) == {'0461':2, 'mobile':1, '02':3}
```

#### 4. display calls

⊗⊗ Using matplotlib, display a bar plot of the frequency of calls by prefixes (including mobile and unrecognized), sorting them in reverse order so you first see the province with the higher number of calls. Also, save the plot on disk with plt.savefig('prefixes-count.png') (call it before plt.show())

If you're in trouble you can find plenty of examples in the visualization chapter<sup>436</sup>

You should obtain something like this:



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"  
data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4] :

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']
phones = ['+39 02 423', '0461123', '02 426', '+39 0461 4321', '0039328 1234567',
 ↪'02 423', '02 424']

write here

coords = list(hist(phones).items())

coords.sort(key=lambda x:x[1], reverse=True)

xs = np.arange(len(coords))
ys = [c[1] for c in coords]

plt.bar(xs, ys, 0.5, align='center')

plt.title("province calls by prefixes sorted solution")
plt.xticks(xs, [c[0] for c in coords])
```

(continues on next page)

<sup>436</sup> <https://en.softpython.org/visualization/visualization1-sol.html>

(continued from previous page)

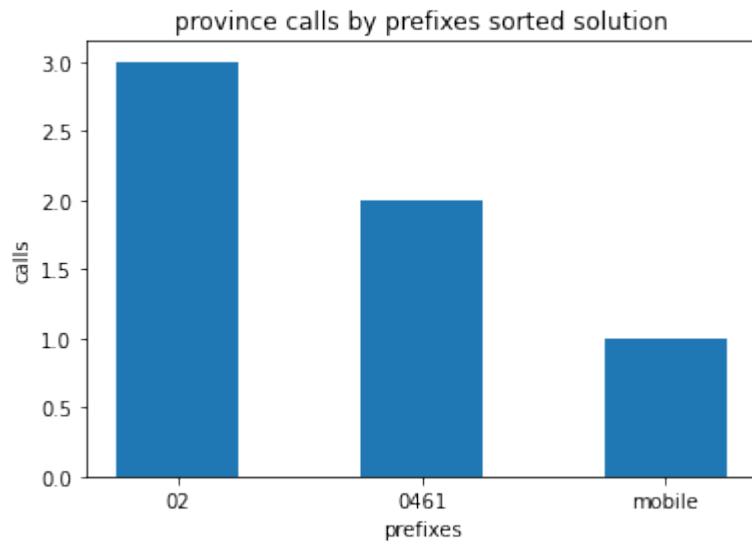
```

plt.xlabel('prefixes')
plt.ylabel('calls')

plt.savefig(
 'prefixes-count.png')

plt.show()

```



&lt;/div&gt;

[4]:

```

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']
phones = ['+39 02 423', '0461123', '02 426', '+39 0461 4321', '0039328 1234567',
 ↪'02 423', '02 424']

write here

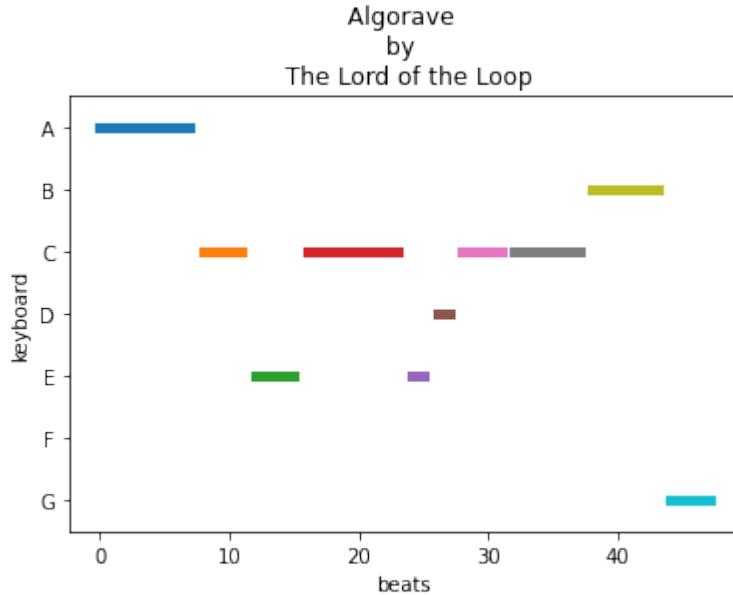
```

[ ]:

### 10.1.2 Music sequencer

#### Download worked project

Browse files online<sup>437</sup>



ABC<sup>438</sup> is a popular format to write music notation in plain text files, you can see an example by opening `tunes1.abc` with a text editor. A music sequencer is an editor software which typically displays notes as a matrix: let's see how to parse simplified abc tunes and display their melodies in such a matrix.

#### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
music-sequencer-prj
 music-sequencer.ipynb
 music-sequencer-sol.ipynb
 tunes1.abc
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `music-sequencer.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter

<sup>437</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/music-sequencer>

<sup>438</sup> <http://abenotation.com/wiki/abc:standard:v2.1#rrhythm>

- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## 1. parse\_melody

Write a function which given a melody as a string of notes translates it to a list of tuples:

```
>>> parse_melody("|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |")
[(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
```

Each melody note is followed by its duration. If no duration number is specified, we assume it is one.

Each tuple first element represents a note as a number from 0 (A) to 6 (G) and the second element is the note length in the sequencer. We assume our sequencer has a resolution of two beats per note, so for us a note A would have length 2, a note A2 a length 4, a note A3 a length 6 and so on.

- **DO NOT** care about spaces nor bars |, they have no meaning at all
- **DO NOT** write a wall of ifs, instead **USE** `ord`<sup>439</sup> python function to get a character position

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[1]:

```
def parse_melody(melody):
 notes = melody.replace(' ', '').split()

 ret = []
 for note in notes:
 n = ord(note[0])-ord('A')
 if len(note) == 1:
 ret.append((n, 2))
 else:
 ret.append((n, 2*int(note[1])))

 return ret

from pprint import pprint
melody1 = "|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |"
pprint(parse_melody(melody1))

assert parse_melody("||") == []
assert parse_melody("|A|") == [(0, 2)]
assert parse_melody("| B|") == [(1, 2)]
assert parse_melody("|C |") == [(2, 2)]
assert parse_melody("|A3|") == [(0, 6)]
assert parse_melody("|A B|") == [(0, 2), (1, 2)]
assert parse_melody(" | G F | ") == [(6, 2), (5, 2)]
assert parse_melody("|D|B|") == [(3, 2), (1, 2)]
assert parse_melody("|D3 E4|") == [(3, 6), (4, 8)]
assert parse_melody("|F|A2 B|") == [(5, 2), (0, 4), (1, 2)]
assert parse_melody("|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |") == \
 [(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
```

(continues on next page)

<sup>439</sup> <https://en.softpython.org/strings/strings2-sol.html#Comparing-characters>

(continued from previous page)

```
[(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
```

&lt;/div&gt;

[1]:

```
def parse_melody(melody):
 raise Exception('TODO IMPLEMENT ME !')

from pprint import pprint
melody1 = "|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |"
pprint(parse_melody(melody1))

assert parse_melody("||") == []
assert parse_melody("|A|") == [(0,2)]
assert parse_melody("| B|") == [(1,2)]
assert parse_melody("|C |") == [(2,2)]
assert parse_melody("|A3|") == [(0,6)]
assert parse_melody("|A B|") == [(0,2), (1,2)]
assert parse_melody(" | G F | ") == [(6,2), (5,2)]
assert parse_melody("|D|B|") == [(3,2), (1,2)]
assert parse_melody("|D3 E4|") == [(3,6), (4,8)]
assert parse_melody("|F|A2 B|") == [(5,2), (0,4), (1,2)]
assert parse_melody("|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |") == \
 [(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
```

## 2. parse\_tunes

An .abc file is a series of key:value fields. Keys are always one character long. Anything after a % is a comment and must be ignored

File tunes1.abc EXCERPT:

```
[2]: with open("tunes1.abc", encoding='utf-8') as f: print(''.join(f.readlines()[0:18]))
```

```
%abc-2.1
H:Tune made in a dark algorithmic night % history and origin in header, so_
→replicated in all tunes!
O:Trento

X:1 % index
T:Algorave % title
C:The Lord of the Loop % composer
M:4/4 % meter
K:C % key
|A4 C2 E2 |C4 E D C2 |C3 B3 G2 | % melodies can also have a comment

X:2
T:Transpose Your Head
C:Matrix Queen
O:Venice % overriding header
M:3/4
K:G
```

(continues on next page)

(continued from previous page)

F2	G4	E4	E	F	A2	B2	D2	D3	E3	C3	C3	
----	----	----	---	---	----	----	----	----	----	----	----	--

First lines (3 in the example) are the file header, separated by tunes with a blank line.

- first line must always be ignored
- fields specified in the file header must be copied in *all* tunes
  - Note a tune **may override** a field (es O:Venice).

After the first blank line, there is the first tune:

- X is the tune index, convert it to integer
- M is the meter, convert it to a tuple of two integers
- K is the last field of metadata
- melody line has no field key, it always follows line with K and it immediately begins with a pipe: convert it to list by calling `parse_melody`

Following tunes are separated by blank lines

Write a function `parse_tunes` which parses the file and outputs a list of dictionaries, one per tune. Use provided `field_names` to obtain dictionary keys. Full expected db is in `expected_db1.py` file.

**DO NOT write hundreds of ifs**

Special keys are listed above, all others should be treated in a generic way

**DO NOT assume header always contains 'origin' and 'history'**

It can contain *any* field, which has to be then copied in all the tunes, see `tunes2.abc` for extra examples.

**Example:**

```
>>> tunes_db1 = parse_tunes('tunes1.abc')
>>> pprint(tunes_db1[:2], width=150)
[
 {'composer': 'The Lord of the Loop',
 'history': 'Tune made in a dark algorithmic night',
 'index': 1,
 'key': 'C',
 'melody': [(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6),
 ↪ (6, 4)],
 'meter': (4, 4),
 'origin': 'Trento',
 'title': 'Algorave'},
 {'composer': 'Matrix Queen',
 'history': 'Tune made in a dark algorithmic night',
 'index': 2,
 'key': 'G',
 'melody': [(5, 4), (6, 8), (4, 8), (4, 2), (5, 2), (0, 4), (1, 4), (3, 4), (3, 6), (4, 6), (2,
 ↪ 6), (2, 6)],
 'meter': (3, 4),
```

(continues on next page)

(continued from previous page)

```
'origin': 'Venice',
'title': 'Transpose Your Head'
}
]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```
field_names = {
 'C':'composer',
 'D':'discography',
 'H':'history',
 'K':'key',
 'M':'meter',
 'O':'origin',
 'T':'title',
 'X':'index',
}

def parse_tunes(filename):

 with open(filename, encoding='utf-8') as f:
 f.readline() # skips %abc-2.1
 tunes = []
 common = {}
 line = f.readline()
 while line != '':
 clean_line = line.split('%')[0].strip()
 if clean_line == '':
 tune = common.copy()
 tunes.append(tune)
 else:
 # process value
 k,v = clean_line.split(':')
 if k == 'X': # index
 vp = int(v)
 elif k == 'K': # key
 vp = v
 melody_line = f.readline()
 melody_line = melody_line.split('%')[0].strip()
 tune['melody'] = parse_melody(melody_line)
 elif k == 'M': # meter
 s = v.split('/')
 vp = (int(s[0]), int(s[1]))
 else:
 vp = v

 if len(tunes) == 0: # header
 common[field_names[k]] = vp
 else: # tune
 tune[field_names[k]] = vp
 line = f.readline()
```

(continues on next page)

(continued from previous page)

```

 return tunes

tunes_db1 = parse_tunes('tunes1.abc')
pprint(tunes_db1[:3], width=150)

[{'composer': 'The Lord of the Loop',
 'history': 'Tune made in a dark algorithmic night',
 'index': 1,
 'key': 'C',
 'melody': [(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6),
 ↪(6, 4)],
 'meter': (4, 4),
 'origin': 'Trento',
 'title': 'Algorave'},
 {'composer': 'Matrix Queen',
 'history': 'Tune made in a dark algorithmic night',
 'index': 2,
 'key': 'G',
 'melody': [(5, 4), (6, 8), (4, 8), (4, 2), (5, 2), (0, 4), (1, 4), (3, 4), (3, 6),
 ↪(4, 6), (2, 6), (2, 6)],
 'meter': (3, 4),
 'origin': 'Venice',
 'title': 'Transpose Your Head'},
 {'composer': 'anonymous truck driver',
 'history': 'Tune made in a dark algorithmic night',
 'index': 3,
 'key': 'E',
 'melody': [(6, 4), (1, 4), (3, 8), (3, 6), (4, 4), (4, 4), (3, 2), (6, 6), (1, 6),
 ↪(6, 4)],
 'meter': (4, 4),
 'origin': 'Trento',
 'title': 'Pedal to the Metal'}]

```

&lt;/div&gt;

[3]:

```

field_names = {
 'C':'composer',
 'D':'discography',
 'H':'history',
 'K':'key',
 'M':'meter',
 'O':'origin',
 'T':'title',
 'X':'index',
}

def parse_tunes(filename):
 raise Exception('TODO IMPLEMENT ME !')

tunes_db1 = parse_tunes('tunes1.abc')
pprint(tunes_db1[:3], width=150)

```

```
[4]:
assert tunes_db1[0]['history']=='Tune made in a dark algorithmic night'
assert tunes_db1[0]['origin']=='Trento'
assert tunes_db1[0]['index']==1
assert tunes_db1[0]['title']=='Algorave'
assert tunes_db1[0]['composer']=='The Lord of the Loop'
assert tunes_db1[0]['meter']==(4,4)
assert tunes_db1[0]['key']=='C'
assert tunes_db1[0]['melody']==\n[(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
assert tunes_db1[1]['history']=='Tune made in a dark algorithmic night'
assert tunes_db1[1]['origin']=='Venice' # tests override
assert tunes_db1[1]['index']==2
assert tunes_db1[1]['title']=='Transpose Your Head'
assert tunes_db1[1]['composer']=='Matrix Queen'
assert tunes_db1[1]['meter']==(3,4)
assert tunes_db1[1]['key']=='G'
assert tunes_db1[1]['melody']==\n[(5, 4), (6, 8), (4, 8), (4, 2), (5, 2), (0, 4), (1, 4), (3, 4), (3, 6), (4, 6), (2, 6), (2, 6)]
from expected_db1 import expected_db1
assert len(tunes_db1) == len(expected_db1)
assert tunes_db1 == expected_db1

tunes_db2 = parse_tunes('tunes2.abc')
pprint(tunes_db2)

from expected_db2 import expected_db2
assert tunes_db2 == expected_db2
```

### 3. sequencer

Write a function `sequencer` which takes a melody in text format and outputs a matrix of note events, as a list of strings.

The rows are all the notes on keyboard (we assume 7 notes without black keys) and the columns represent the duration of a note.

- a note start is marked with < character, a sustain with = character and end with >
- **HINT 1:** call `parse_melody` to obtain notes as a list of tuples (if you didn't manage to implement it copy expected list from `expected_db1.py`)
- **HINT 2:** build first a list of list of characters, and only at the very end convert to a list of strings
- **HINT 3:** try obtaining the note letters for first column by using `ord` and `chr`

#### Example 1:

```
>>> from pprint import pprint
>>> melody1 = "|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |"
>>> res1 = sequencer(melody1)
>>> print(' ' + melody1)
|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |
>>> pprint(res1)
['A<=====>
 'B <=====> ',
 'C <==> <=====> <==><=====> ',
 'D <> ',
```

(continues on next page)

(continued from previous page)

```
'E <==> <> ',
'F <==> <> ',
'G <==> <> ',
 ',
 ',
 '>]
```

**Example 2:**

```
>>> melody2 = "|F2 G4 |E4 E F|A2 B2 D2 |D3 E3 |C3 C3 |"
>>> res2 = sequencer(melody2)
>> print(' ' + melody2)
|F2 G4 |E4 E F|A2 B2 D2 |D3 E3 |C3 C3 |
>>> pprint(res2)
['A <==> ',
 'B <==> ',
 'C <=====><=====> ',
 'D <=====><====> ',
 'E <=====><> ',
 'F<==> <> ',
 'G <=====> ',
]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5]:

```
def sequencer(melody):

 notes = parse_melody(melody)
 length = 0
 for note in notes:
 length += note[1]
 ret = []

 work = [[' ']*(length+1) for i in range(7)]

 for i in range(7):
 work[i][0] = chr(ord('A')+i)

 j = 0
 for n, d in notes:
 work[n][j+1] = '<'
 eqlen = d-2
 work[n][j+2 : j+eqlen+2] = '=' * eqlen # cool slice writing
 work[n][j+d] = '>'
 j += d
 return [''.join(w) for w in work]

from pprint import pprint
melody1 = "|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |"
exp1 = [
 'A<=====>',
 'B <=====> <=====> ',
 'C <==> <=====> <==><=====> ',
 'D <=====> <> ',
 'E <==> <> ',
 'F <=====> ',
]
```

(continues on next page)

(continued from previous page)

```
'G' <==> ']'

res1 = sequencer(melody1)
print(' ' + melody1)
print()
pprint(res1)
assert res1 == exp1

|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |

['A<=====> ',
 'B ' <=====> ' ',
 'C <==> <=====> <==><=====> ',
 'D <> ',
 'E <==> <> ',
 'F ',
 'G <==> ']
```

</div>

[5] :

```
def sequencer(melody):
 raise Exception('TODO IMPLEMENT ME !')

from pprint import pprint
melody1 = "|A4 C2 E2 |C4 E D C2 |C3 B3 G2 |"
exp1 = [
 'A<=====>',
 'B',
 'C <==> <=====> <==><=====>',
 'D',
 'E <==> <>',
 'F',
 'G'
]

res1 = sequencer(melody1)
print(' ' + melody1)
print()
pprint(res1)
assert res1 == exp1
```

[ 6 ] :

```
from pprint import pprint
melody2 = "|F2 G4 |E4 E F|A2 B2 D2 |D3 E3 |C3 C3 |"
exp2 = ['A' <--> 'B' <--> 'C' <--> 'D' <--> 'E' <--> 'F' <--> 'G' <-->
 '<=====|><=====|><=====|><=====|><=====|><=====|><=====|>'
```

(continues on next page)

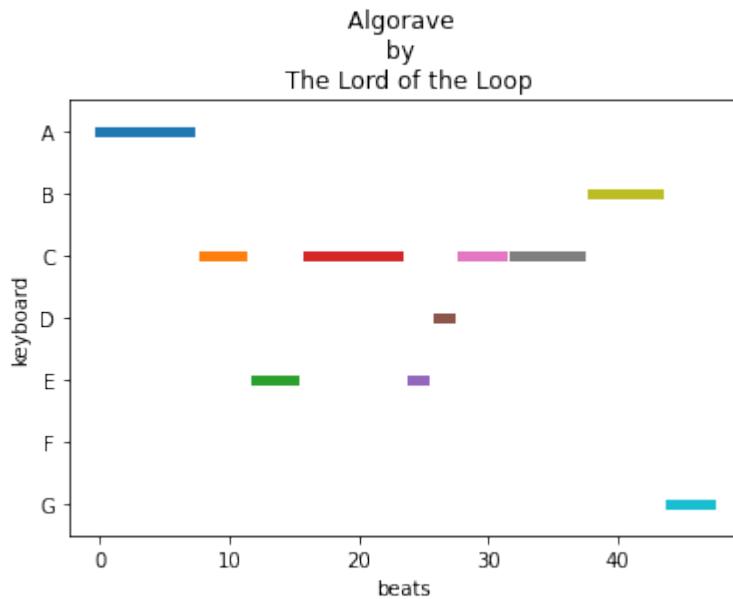
(continued from previous page)

```
assert res2 == exp2
```

#### 4. plot\_tune

Make it fancy: write a function which takes a tune dictionary from the db and outputs a plot

- use beats as xs, remembering the shortest note has two beats
- to increase thickness, use linewidth=5 parameter



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[7]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_tune(tune):

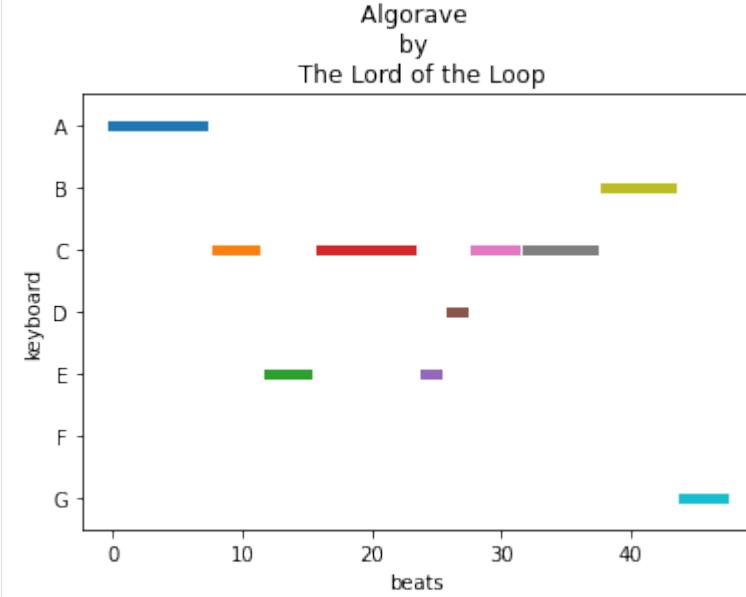
 j = 0
 for n,d in tune['melody']:
 r = 6 - n
 xs = [j, j+d-1]
 ys = [r, r]
 j += d
 plt.plot(xs, ys, linewidth=5)

 plt.title('%s \nby \n %s' %(tune['title'], tune['composer']))
 plt.ylim(-0.5,6.5)
 plt.yticks(range(7), [chr(ord('G')-i) for i in range(7)])
 plt.xlabel('beats')
 plt.ylabel('keyboard')
```

(continues on next page)

(continued from previous page)

```
plot_tune(tunes_db1[0])
```



```
</div>
```

```
[7]:
%matplotlib inline
import matplotlib.pyplot as plt

def plot_tune(tune):
 raise Exception('TODO IMPLEMENT ME !')

plot_tune(tunes_db1[0])
```

```
[]:
```

## 10.2 Tabular data

### 10.2.1 Bus speed

[Download worked project](#)

[Browse files online<sup>440</sup>](#)

In this little project, we will analyze intercity bus velocities in GTFS format.

Data source: [dati.trentino.it<sup>441</sup>](https://dati.trentino.it/), MITT service, released under [Creative Commons Attribution 4.0<sup>442</sup>](#) licence.

<sup>440</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/bus-speed>

<sup>441</sup> <https://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs>

<sup>442</sup> <http://creativecommons.org/licenses/by/4.0/deed.it>

## What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
bus-speed-prj
bus-speed.ipynb
bus-speed-sol.ipynb
network.csv
jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook bus-speed.ipynb
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## The dataset

Original GTFS data was split in several files which we merged into dataset `network.csv` containing the bus stop times of three extra-urban routes. To load it, we provide this function:

```
[1]: def load_stops():
 "Loads file network.csv and RETURN a list of dictionaries with the stop times"

 import csv
 with open('network.csv', newline='', encoding='UTF-8') as csvfile:
 reader = csv.DictReader(csvfile)
 lst = []
 for d in reader:
 lst.append(d)
 return lst
```

```
[2]: stops = load_stops()

stops[0:2]

[2]: [OrderedDict([('1',
 ('route_id', '76'),
 ('agency_id', '12'),
 ('route_short_name', 'B202'),
 ('route_long_name',
 'Trento-Sardagna-Candriai-Vaneze-Vason-Viote'),
 ('route_type', '3'),
 ('service_id', '22018091220190621'),
 ('trip_id', '0002402742018091220190621'),
 ('trip_headsign', 'Trento-Autostaz.')),
```

(continues on next page)

(continued from previous page)

```
('direction_id', '0'),
('arrival_time', '06:25:00'),
('departure_time', '06:25:00'),
('stop_id', '844'),
('stop_sequence', '2'),
('stop_code', '2620'),
('stop_name', 'Sardagna'),
('stop_desc', ''),
('stop_lat', '46.064848'),
('stop_lon', '11.09729'),
('zone_id', '2620.0'))],
OrderedDict([('2',
 ('route_id', '76'),
 ('agency_id', '12'),
 ('route_short_name', 'B202'),
 ('route_long_name',
 'Trento-Sardagna-Candriai-Vaneze-Vason-Viote'),
 ('route_type', '3'),
 ('service_id', '22018091220190621'),
 ('trip_id', '0002402742018091220190621'),
 ('trip_headsign', 'Trento-Autostaz.'),
 ('direction_id', '0'),
 ('arrival_time', '06:26:00'),
 ('departure_time', '06:26:00'),
 ('stop_id', '5203'),
 ('stop_sequence', '3'),
 ('stop_code', '2620VD'),
 ('stop_name', 'Sardagna Civ. 22'),
 ('stop_desc', ''),
 ('stop_lat', '46.069494'),
 ('stop_lon', '11.095252'),
 ('zone_id', '2620.0'))])]
```

Of interest to you are the fields `route_short_name`, `arrival_time`, and `stop_lat` and `stop_lon` which provide the geographical coordinates of the stop. Stops are already sorted in the file from earliest to latest.

Given a `route_short_name`, like B202, we want to plot the graph of bus velocity measured in **km/hours** at each stop. We define velocity at stop n as

$$\text{velocity}_n = \frac{\Delta s_{pace_n}}{\Delta t_{ime_n}}$$

where

$\Delta t_{ime_n} = t_{ime_n} - t_{ime_{n-1}}$  as the time **in hours** the bus takes between stop n and stop  $n - 1$ .

and

$\Delta s_{pace_n} = s_{pace_n} - s_{pace_{n-1}}$  is the distance the bus has moved between stop n and stop  $n - 1$ .

We also set  $\text{velocity}_0 = 0$

**NOTE FOR TIME:** When we say time in **hours**, it means that if you have the time as string `08:27:42`, its number in seconds since midnight is like:

[3]: secs = 8\*60\*60+27\*60+42

and to calculate the time in **float hours** you need to divide `secs` by  $60*60=3600$ :

[4]: hours\_float = secs / (60\*60)
hours\_float

[4]: 8.461666666666666

**NOTE FOR SPACE:** Unfortunately, we could not find the actual distance as road length done by the bus between one stop and the next one. So, for the sake of the exercise, we will take the *geo distance*, that is, we will calculate it using the line distance between the points of the stops, using their geographical coordinates. The function to calculate the geo\_distance is **already implemented**:

```
[5]: def geo_distance(lat1, lon1, lat2, lon2):
 """ Return the geo distance in kilometers
 between the points 1 and 2 at provided geographical coordinates.

 """
 # Shamelessly copied from https://stackoverflow.com/a/19412565

 from math import sin, cos, sqrt, atan2, radians

 # approximate radius of earth in km
 R = 6373.0

 lat1 = radians(lat1)
 lon1 = radians(lon1)
 lat2 = radians(lat2)
 lon2 = radians(lon2)

 dlon = lon2 - lon1
 dlat = lat2 - lat1

 a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
 c = 2 * atan2(sqrt(a), sqrt(1 - a))

 return R * c
```

In the following we see the bus line B102, going from Sardagna to Trento. The graph should show something like the following.

We can see that as long as the bus is taking stops within Sardagna town, velocity (always intended as air-line velocity) is high, but when the bus has to go to Trento, since there are many twists and turns on the road, it takes a while to arrive even if in geo-distance Trento is near, so actual velocity decreases. In such case it would be much more convenient to take the cable car.

These type of graphs might show places in the territory where shortcuts such as cable cars, tunnels or bridges might be helpful for transportation.

### to\_float\_hour

Implement a function to\_float\_hour that takes a time string in the format like 08:27:42 and RETURN the time since midnight in hours as a float (es 8.461666666666666)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[6]: def to_float_hour(time_string):

 hours = int(time_string[0:2])
 mins = int(time_string[3:5])
```

(continues on next page)

(continued from previous page)

```
secs = int(time_string[6:])
return (hours * 60 * 60 + mins * 60 + secs) / (60*60)
```

```
</div>
```

```
[6]: def to_float_hour(time_string):
 raise Exception('TODO IMPLEMENT ME !')
```

### plot

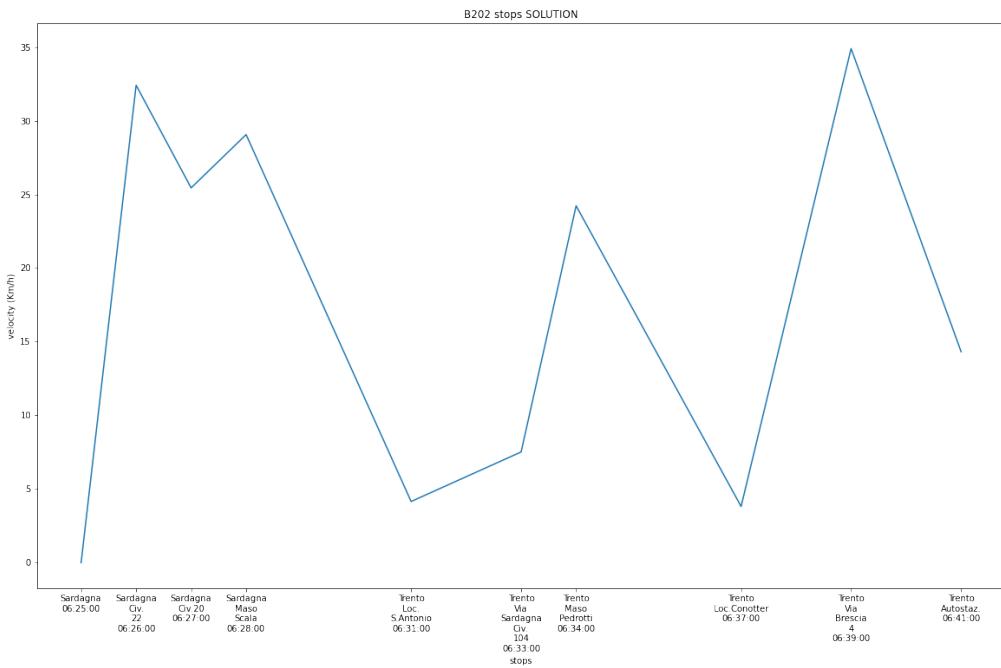
Implement a function `plot` which takes a `route_short_name` and displays with matplotlib a graph of the velocity of the bus trip for that route

- just use matplotlib, you **don't** need pandas and **don't** need numpy
- xs positions **MUST** be in **float hours**, distanced at lengths proportional to the actual time the bus arrives that stop
- xticks **MUST** show:
  - the stop name **NICELY** (with carriage returns)
  - the time in **08:50:12 format**
- ys **MUST** show the velocity of the bus at that time
- assume velocity at stop 0 equals 0
- remember to set the figure width and height
- remember to set axis labels and title

#### Example 1:

```
>>> plot('B202')
```

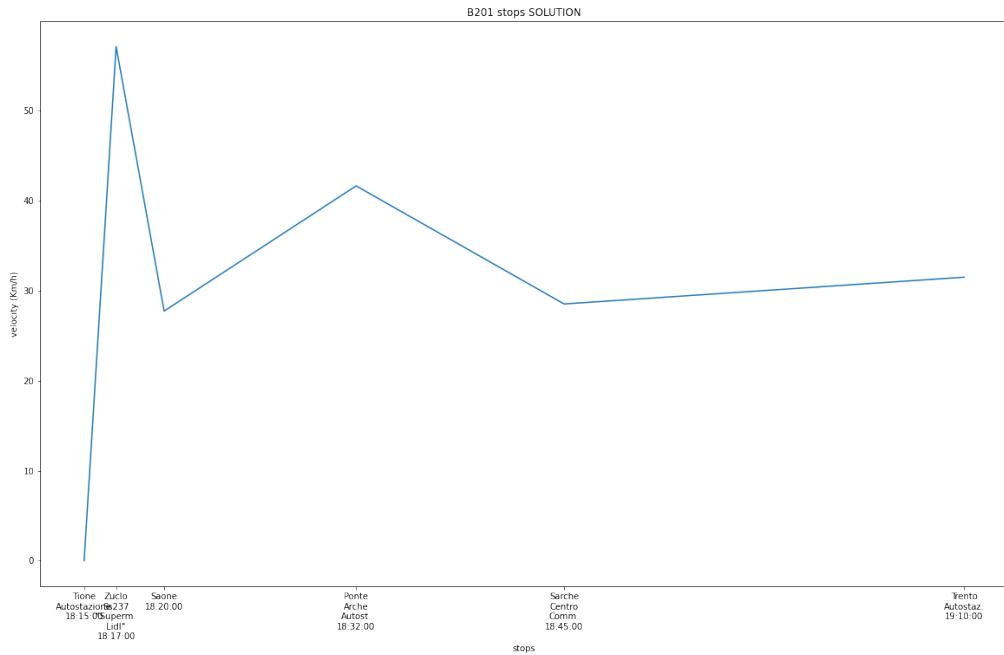
```
xs = [6.416666666666667, 6.433333333333334, 6.45, 6.466666666666667, 6.
˓→516666666666667, 6.55, 6.56666666666666, 6.61666666666666, 6.65, 6.
˓→683333333333334]
ys = [0, 32.410644806589666, 25.440452145453996, 29.058090168277648, 4.
˓→151814096935986, 7.514788081665398, 24.226499833822754, 3.8149164687282586, 34.
˓→89698602693173, 14.321244382769315]
xticks = ['Sardagna\n06:25:00', 'Sardagna\nCiv.\n22\n06:26:00', 'Sardagna\nCiv.20\n06:
˓→27:00', 'Sardagna\nMaso\nScala\n06:28:00', 'Trento\nLoc.\nS.Antonio\n06:31:00',
˓→'Trento\nVia\nSardagna\nCiv.\n104\n06:33:00', 'Trento\nMaso\nPedrotti\n06:34:00',
˓→'Trento\nLoc.Conotter\n06:37:00', 'Trento\nVia\nBrescia\n4\n06:39:00', 'Trento\
˓→Autostaz.\n06:41:00']
```



### Example 2:

```
>>> plot('B201')
```

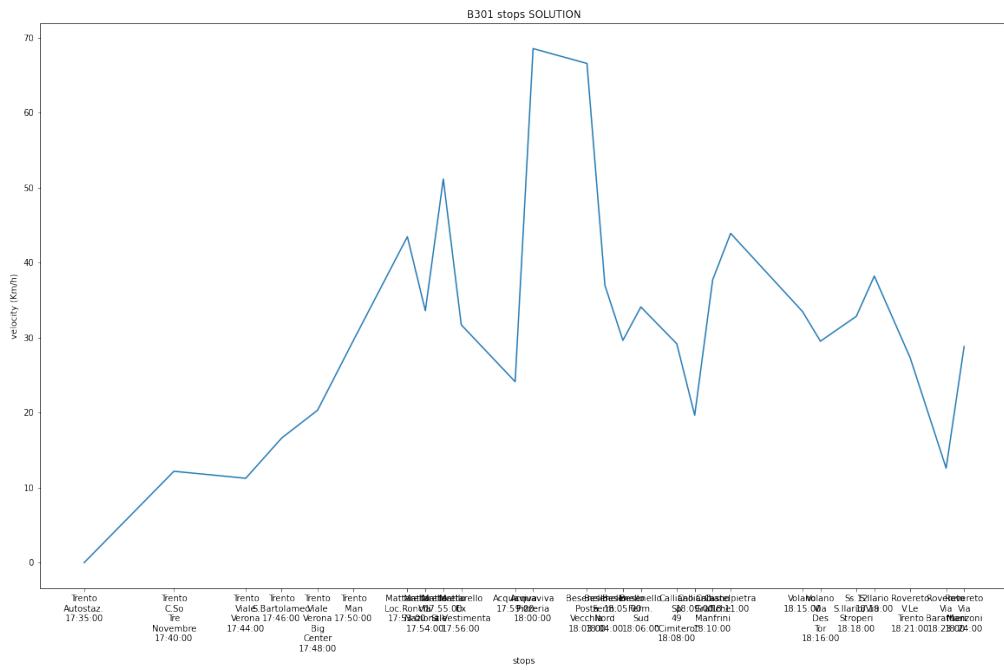
```
xs = [18.25, 18.28333333333335, 18.33333333333332, 18.53333333333335, 18.75, 19.
→1666666666666668]
ys = [0, 57.11513455659372, 27.731105466934423, 41.63842308087865, 28.5197376150513,
→31.49374154105802]
xticks = ['Tione\nAutostazione\n18:15:00', 'Zuclo\nSs237\n"Superm.\nLidl"\n18:17:00',
→'Saone\n18:20:00', 'Ponte\nArche\nAutost.\n18:32:00', 'Sarche\nCentro\nComm.\n18:45:
→00', 'Trento\nAutostaz.\n19:10:00']
```



### Example 3:

```
>>> plot('B301')
```

```
xs = [17.58333333333332, 17.666666666666668, 17.73333333333334, 17.766666666666666, 17.8, 17.8333333333332, 17.88333333333333, 17.9, 17.916666666666668, 17.93333333333334, 17.98333333333334, 18.0, 18.05, 18.066666666666666, 18.08333333333332, 18.1, 18.13333333333333, 18.15, 18.166666666666668, 18.18333333333334, 18.25, 18.266666666666666, 18.3, 18.316666666666666, 18.35, 18.38333333333333, 18.4]
ys = [0, 12.183536596091201, 11.250009180954352, 16.612469697023045, 20.32290877261807, 29.650645502388567, 43.45858933073937, 33.590326783093374, 51.14340770207765, 31.710506116846854, 24.12416002315475, 68.52690370810224, 66.54632979050625, 36.97129817779247, 29.62791050495846, 34.08490909322781, 29.184331044522004, 19.648559840967014, 37.7140096915846, 43.892216115372726, 33.48796397878209, 29.521341752309603, 32.83990219938084, 38.20505182104893, 27.292895333249888, 12.602972475349818, 28.804672730461583]
xticks = ['Trento\nAutostaz.\n17:35:00', 'Trento\nnC.So\nnTre\nnNovembre\n17:40:00', 'Trento\nViale\nVerona\n17:44:00', 'Trento\nnS.Bartolomeo\n17:46:00', 'Trento\nViale\nnVerona\nBig\nCenter\n17:48:00', 'Trento\nMan\n17:50:00', 'Mattarello\nLoc.Ronchi\n17:53:00', 'Mattarello\nVia\nNazionale\n17:54:00', 'Mattarello\n17:55:00', 'Mattarello\nEx\nSt.Vestimenta\n17:56:00', 'Acquaviva\n17:59:00', 'Acquaviva\nnPizzeria\n18:00:00', 'Besenello\nPosta\nVecchia\n18:03:00', 'Besenello\nFerm.\nNord\n18:04:00', 'Besenello\n18:05:00', 'Besenello\nFerm.\nSud\n18:06:00', 'Calliano\nSp\nn49\nCimitero"\n18:08:00', 'Calliano\n18:09:00', 'Calliano\nGrafiche\nManfrini\n18:10:00', 'Castelpietra\n18:11:00', 'Volano\n18:15:00', 'Volano\nVia\nDes\nTor\n18:16:00', 'Ss.12\nnS.Ilario/Via\nnStroperi\n18:18:00', 'S.Ilario\n18:19:00', 'Rovereto\nV.Le\nTrento\n18:21:00', 'Rovereto\nVia\nBarattieri\n18:23:00', 'Rovereto\nVia\nManzoni\n18:24:00']
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[7]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

def plot(route_short_name):

 stops = load_stops()

 xs = []
 ys = []
 ticks = []
 seq = [d for d in stops if d['route_short_name'] == route_short_name]
 d_prev = seq[0]
 n = 0
 for d in seq:
 xs.append(to_float_hour(d['arrival_time']))
 if n == 0:
 v = 0
 else:
 delta_distance = geo_distance(float(d['stop_lat']), float(d['stop_lon']),
 float(d_prev['stop_lat']), float(d_prev['stop_lon']))
 delta_time = (to_float_hour(d['arrival_time']) - to_float_hour(d_prev['arrival_time']))
 v = delta_distance / delta_time
 ys.append(v)
 ticks.append("%s\n%s" % (d['stop_name'].replace(' ', '\n').replace('-', '\n'), d['arrival_time']))
 d_prev = d
```

(continues on next page)

(continued from previous page)

```
n += 1

fig = plt.figure(figsize=(20,12)) # width: 20 inches, height 12 inches
plt.plot(xs, ys)

plt.title("%s stops SOLUTION" % route_short_name)
plt.xlabel('stops')
plt.ylabel('velocity (Km/h)')

FIRST NEEDS A SEQUENCE WITH THE POSITIONS, THEN A SEQUENCE OF SAME LENGTH WITH_
↪LABELS
plt.xticks(xs, ticks)
print('xs = %s' % xs)
print('ys = %s' % ys)
print('xticks = %s' % ticks)

plt.show()
```

&lt;/div&gt;

```
[7]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

def plot(route_short_name):
 raise Exception('TODO IMPLEMENT ME !')
```

```
[8]: plot('B202')
```

```
[9]: plot('B201')
```

```
[10]: plot('B301')
```

## 10.2.2 Town events

### Download worked project

Browse files online<sup>443</sup>

We will work on a dataset of events which occurred in the Municipality of Trento (Italy) during years 2019-20. Each event can be held during a particular day, two days, or many specified as a range. Events are written using natural language, so we will try to extract such dates, taking into account that information sometimes can be partial or absent.

Data source: [Municipality of Trento](#)<sup>444</sup>, released under [Creative Commons Attribution 4.0](#)<sup>445</sup> licence.

<sup>443</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/town-events>

<sup>444</sup> <https://dati.trentino.it/dataset/eventi-del-comune-di-trento>

<sup>445</sup> <http://creativecommons.org/licenses/by/4.0/deed.it>

## What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
town-events-prj
 town-events.ipynb
 town-events-sol.ipynb
 eventi.csv
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook town-events.ipynb
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## The dataset

Let's have a look of the dataset `eventi.csv`, note we used pandas to show some data but it's not actually necessary to solve the exercises.

```
[1]: import pandas as pd
import numpy as np

eventi = pd.read_csv('eventi.csv', encoding='UTF-8') # remember the encoding !
eventi.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253 entries, 0 to 252
Data columns (total 35 columns):
 # Column Non-Null Count Dtype

 0 remoteID 253 non-null object
 1 published 253 non-null object
 2 modified 253 non-null object
 3 Priorità 253 non-null int64
 4 Evento speciale 0 non-null float64
 5 Titolo 253 non-null object
 6 Titolo breve 1 non-null object
 7 Sottotitolo 227 non-null object
 8 Descrizione 224 non-null object
 9 Locandina 16 non-null object
 10 Inizio 253 non-null object
 11 Termine 252 non-null object
 12 Quando 253 non-null object
 13 Orario 251 non-null object
```

(continues on next page)

(continued from previous page)

```

14 Durata 6 non-null object
15 Dove 252 non-null object
16 lat 253 non-null float64
17 lon 253 non-null float64
18 address 241 non-null object
19 Pagina web 201 non-null object
20 Contatto email 196 non-null object
21 Contatto telefonico 196 non-null object
22 Informazioni 62 non-null object
23 Costi 132 non-null object
24 Immagine 252 non-null object
25 Evento - manifestazione 252 non-null object
26 Manifestazione cui fa parte 108 non-null object
27 Tipologia 252 non-null object
28 Materia 252 non-null object
29 Destinatari 24 non-null object
30 Circoscrizione 109 non-null object
31 Struttura ospitante 220 non-null object
32 Associazione 1 non-null object
33 Ente organizzatore 0 non-null float64
34 Identificativo 0 non-null float64
dtypes: float64(5), int64(1), object(29)
memory usage: 69.3+ KB

```

We will focus on Quando (*When*) column:

```
[2]: eventi['Quando']
```

```

[2]: 0 venerdì 5 aprile alle 20:30 in via degli Olmi ...
1 Giovedì 7 novembre 2019
2 Giovedì 14 novembre 2019
3 Giovedì 21 novembre 2019
4 Giovedì 28 novembre 2019
...
248 sabato 9 novembre 2019
249 da venerdì 8 a domenica 10 novembre 2019
250 giovedì 7 novembre 2019
251 giovedì 28 novembre 2019
252 giovedì 21 novembre 2019
Name: Quando, Length: 253, dtype: object

```

## 1. leap\_year

⊕ A leap year has 366 days instead of regular 365. You are given some criteria to detect whether or not a year is a leap year. Implement them in a function which given a year as a number RETURN True if it is a leap year, False otherwise.

**IMPORTANT: in Python there are predefined methods to detect leap years, but here you MUST write your own code!**

1. If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
2. If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
3. If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
4. The year is a leap year (it has 366 days)
5. The year is not a leap year (it has 365 days)

(if you're curious about calendars, see [this link<sup>446</sup>](#))

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: def is_leap(year):

 if year % 4 == 0:
 if year % 100 == 0:
 return year % 400 == 0
 else:
 return True
 else:
 return False

assert is_leap(4) == True
assert is_leap(104) == True
assert is_leap(204) == True
assert is_leap(400) == True
assert is_leap(1600) == True
assert is_leap(2000) == True
assert is_leap(2400) == True
assert is_leap(2000) == True
assert is_leap(2004) == True
assert is_leap(2008) == True
assert is_leap(2012) == True

assert is_leap(1) == False
assert is_leap(5) == False
assert is_leap(100) == False
assert is_leap(200) == False
assert is_leap(1700) == False
assert is_leap(1800) == False
assert is_leap(1900) == False
assert is_leap(2100) == False
assert is_leap(2200) == False
assert is_leap(2300) == False
assert is_leap(2500) == False
assert is_leap(2600) == False
```

</div>

```
[3]: def is_leap(year):
 raise Exception('TODO IMPLEMENT ME !')

assert is_leap(4) == True
assert is_leap(104) == True
assert is_leap(204) == True
assert is_leap(400) == True
assert is_leap(1600) == True
assert is_leap(2000) == True
assert is_leap(2400) == True
assert is_leap(2000) == True
assert is_leap(2004) == True
```

(continues on next page)

<sup>446</sup> <https://docs.microsoft.com/en-us/office/troubleshoot/excel/determine-a-leap-year>

(continued from previous page)

```
assert is_leap(2008) == True
assert is_leap(2012) == True

assert is_leap(1) == False
assert is_leap(5) == False
assert is_leap(100) == False
assert is_leap(200) == False
assert is_leap(1700) == False
assert is_leap(1800) == False
assert is_leap(1900) == False
assert is_leap(2100) == False
assert is_leap(2200) == False
assert is_leap(2300) == False
assert is_leap(2500) == False
assert is_leap(2600) == False
```

## 2. full\_date

### WARNING: avoid constants in function bodies !!

In the exercises data you will find many names and connectives such as 'Giovedì', 'Novembre', 'e', 'a', etc. DO NOT put such constant names inside your code and use instead the provided lists (DAYS, MONTHS...) !!  
You have to write generic code which works with any input.

⊕⊕ Write function `full_date` which takes some natural language text representing a complete date and outputs a string in the format `yyyy-mm-dd` like `2019-03-25`.

- Dates will be expressed in Italian, so we report here the corresponding translations
- your function should work regardless of capitalization of input
- we assume the date to be always well formed

#### Examples:

At the beginning you always have day name (Mercoledì means *Wednesday*):

```
>>> full_date("Mercoledì 13 Novembre 2019")
"2019-11-13"
```

Right after day name, you *may* also find a day phase, like mattina for morning:

```
>>> full_date("Mercoledì mattina 13 Novembre 2019")
"2019-11-13"
```

Remember you can have lowercases and single digits which must be prepended by zero:

```
>>> full_date("domenica 4 dicembre 1923")
"1923-12-04"
```

For more examples, see assertions.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[4]: DAYS = ['lunedì', 'martedì', 'mercoledì', 'giovedì', 'venerdì', 'sabato', 'domenica']

MONTHS = ['gennaio', 'febbraio', 'marzo' , 'aprile' , 'maggio' , 'giugno',
 'luglio' , 'agosto' , 'settembre', 'ottobre', 'novembre', 'dicembre']

morning, afternoon, evening, night
DAY_PHASES = ['mattina', 'pomeriggio', 'sera', 'notte']

def full_date(text):

 ntext = text.lower()
 words = ntext.split()
 i = 1
 if words[i] in DAY_PHASES:
 i += 1
 day = int(words[i])
 i += 1

 month = int(MONTHS.index(words[i])) + 1
 i += 1

 year = int(words[i])

 return "{:04d}-{:02d}-{:02d}".format(year, month, day)

assert full_date("Giovedì 14 novembre 2019") == "2019-11-14"
assert full_date("Giovedì 7 novembre 2019") == "2019-11-07"
assert full_date("Giovedì pomeriggio 14 novembre 2019") == "2019-11-14"
assert full_date("sabato mattina 25 marzo 2017") == "2017-03-25"
assert full_date("Mercoledì 13 Novembre 2019") == "2019-11-13"
assert full_date("domenica 4 dicembre 1923") == "1923-12-04"
```

&lt;/div&gt;

```
[4]: DAYS = ['lunedì', 'martedì', 'mercoledì', 'giovedì', 'venerdì', 'sabato', 'domenica']

MONTHS = ['gennaio', 'febbraio', 'marzo' , 'aprile' , 'maggio' , 'giugno',
 'luglio' , 'agosto' , 'settembre', 'ottobre', 'novembre', 'dicembre']

morning, afternoon, evening, night
DAY_PHASES = ['mattina', 'pomeriggio', 'sera', 'notte']

def full_date(text):
 raise Exception('TODO IMPLEMENT ME !')

assert full_date("Giovedì 14 novembre 2019") == "2019-11-14"
assert full_date("Giovedì 7 novembre 2019") == "2019-11-07"
assert full_date("Giovedì pomeriggio 14 novembre 2019") == "2019-11-14"
assert full_date("sabato mattina 25 marzo 2017") == "2017-03-25"
assert full_date("Mercoledì 13 Novembre 2019") == "2019-11-13"
assert full_date("domenica 4 dicembre 1923") == "1923-12-04"
```

### 3. partial\_date

⊕⊕⊕ Write a function `partial_date` which takes a natural language text representing one or more dates, and RETURN only the FIRST date found, in the format `yyyy-mm-dd`. If the FIRST date contains insufficient information to form a complete date, in the returned date leave the characters '`yyyy`' for unknown year, '`mm`' for unknown months and '`dd`' for unknown day.

**NOTE:** Here we only care about FIRST date, **DO NOT** attempt to fetch eventual missing information from the second date, we will deal with that in a later exercise.

**Examples:**

```
>>> partial_date("Giovedì 7 novembre 2019")
"2019-11-07"

>>> partial_date("venerdì 15 novembre")
"yyyy-11-15"

>>> partial_date("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019")
"yyyy-mm-15"
```

For more examples, see asserts.

Show solution

</a><div class="jupman-sol-jupman-sol-code" style="display:none">

```
[5]: CONNECTIVE_AND = 'e'

CONNECTIVE_FROM = 'da'
CONNECTIVE_TO = 'a'

DAYS = ['lunedì', 'martedì', 'mercoledì', 'giovedì', 'venerdì', 'sabato', 'domenica']
MONTHS = ['gennaio', 'febbraio', 'marzo', 'aprile', 'maggio', 'giugno',
 'luglio', 'agosto', 'settembre', 'ottobre', 'novembre', 'dicembre']

 # morning, afternoon, evening, night
DAY_PHASES = ['mattina', 'pomeriggio', 'sera', 'notte']

def partial_date(text):

 if type(text) != str:
 return 'yyyy-mm-dd'

 year = 'yyyy'
 month = 'mm'
 day = 'dd'

 ntext = text.lower()
 ret = []
 words = ntext.split()

 if len(words) > 0:
 if words[0] == CONNECTIVE_FROM:
 i = 1
 else:
 i = 0
 if words[i] in DAYS:
 i = i + 1
```

(continues on next page)

(continued from previous page)

```

if words[i] in DAY_PHASES:
 i += 1
day = "{:02d}".format(int(words[i]))
i += 1
if i < len(words):
 # 'e' case with double date
 if words[i] in MONTHS:
 month = "{:02d}".format(MONTHS.index(words[i]) + 1)
 i += 1
 if i < len(words):
 if words[i].isdigit():
 year = "{:04d}".format(int(words[i]))

return "%s-%s-%s" % (year, month, day)

complete, uppercase day
assert partial_date("Giovedì 7 novembre 2019") == "2019-11-07"
assert partial_date("Giovedì 14 novembre 2019") == "2019-11-14"
lowercase day
assert partial_date("mercoledì 13 novembre 2019") == "2019-11-13"
lowercase, dayphase, missing month and year
assert partial_date("venerdì pomeriggio 15") == "yyyy-mm-15"
single day, lowercase, no year
assert partial_date("venerdì 15 novembre") == "yyyy-11-15"

no year, hour / location to be discarded
assert partial_date("venerdì 5 aprile alle 20:30 in via degli Olmi 26 (Trento sud)") \
 == "yyyy-04-05"

two dates, 'and' connective ('e'), day phase morning/afternoon ('mattina'/
'pomeriggio')
assert partial_date("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019") \
 == "yyyy-mm-15"

two dates, begins with connective 'Da'
assert partial_date("Da lunedì 25 novembre a domenica 01 dicembre 2019") == "yyyy-11-\
 25"
assert partial_date("da giovedì 12 a domenica 15 dicembre 2019") == "yyyy-mm-12"
assert partial_date("da giovedì 9 a domenica 12 gennaio 2020") == "yyyy-mm-09"
assert partial_date("Da lunedì 04 a domenica 10 novembre 2019") == "yyyy-mm-04"

```

&lt;/div&gt;

```

[5]: CONNECTIVE_AND = 'e'

CONNECTIVE_FROM = 'da'
CONNECTIVE_TO = 'a'

DAYS = ['lunedì', 'martedì', 'mercoledì', 'giovedì', 'venerdì', 'sabato', 'domenica']
MONTHS = ['gennaio', 'febbraio', 'marzo', 'aprile', 'maggio', 'giugno',
 'luglio', 'agosto', 'settembre', 'ottobre', 'novembre', 'dicembre']

morning, afternoon, evening, night
DAY_PHASES = ['mattina', 'pomeriggio', 'sera', 'notte']

def partial_date(text):

```

(continues on next page)

(continued from previous page)

```
raise Exception('TODO IMPLEMENT ME !')

complete, uppercase day
assert partial_date("Giovedì 7 novembre 2019") == "2019-11-07"
assert partial_date("Giovedì 14 novembre 2019") == "2019-11-14"
lowercase day
assert partial_date("mercoledì 13 novembre 2019") == "2019-11-13"
lowercase, dayphase, missing month and year
assert partial_date("venerdì pomeriggio 15") == "yyyy-mm-15"
single day, lowercase, no year
assert partial_date("venerdì 15 novembre") == "yyyy-11-15"

no year, hour / location to be discarded
assert partial_date("venerdì 5 aprile alle 20:30 in via degli Olmi 26 (Trento sud)") \
 == "yyyy-04-05"

two dates, 'and' connective ('e'), day phase morning/afternoon ('mattina'/
→ 'pomeriggio')
assert partial_date("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019") \
 == "yyyy-mm-15"

two dates, begins with connective 'Da'
assert partial_date("Da lunedì 25 novembre a domenica 01 dicembre 2019") == "yyyy-11-\
 ↪ 25"
assert partial_date("da giovedì 12 a domenica 15 dicembre 2019") == "yyyy-mm-12"
assert partial_date("da giovedì 9 a domenica 12 gennaio 2020") == "yyyy-mm-09"
assert partial_date("Da lunedì 04 a domenica 10 novembre 2019") == "yyyy-mm-04"
```

#### 4. parse\_dates\_and

⊕⊕⊕ Write a function which, given a string representing two possibly partial dates separated by the e connective (*and*), RETURN a tuple holding the two extracted dates each in the format yyyy-mm-dd.

- **IMPORTANT:** Notice that the year or month of the first date might actually be indicated in the second date ! In this exercise we want missing information in the first date to be filled in with year and/or month taken from second date.
- **HINT:** implement this function calling previously defined functions. If you do so, it will be fairly easy.

Examples:

```
>>> parse_dates_and("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019")
("2019-11-15", "2019-11-16")

>>> parse_dates_and("lunedì 4 e domenica 10 novembre")
("yyyy-11-04", "yyyy-11-10")
```

For more examples, see asserts.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```
def parse_dates_and(text):

 ntext = text.lower()
```

(continues on next page)

(continued from previous page)

```

strings = ntext.split(' ' + CONNECTIVE_AND + ' ')
date_left = partial_date(strings[0])
date_right = partial_date(strings[1])
if 'yyyy' in date_left:
 date_left = date_left.replace('yyyy', date_right[0:4])
if 'mm' in date_left:
 date_left = date_left.replace('mm', date_right[5:7])
return (date_left, date_right)

complete dates
assert parse_dates_and("lunedì 25 aprile 2018 e domenica 01 dicembre 2019") == ("2018-04-25", "2019-12-01")

exactly two dates, day phase morning/afternoon ('mattina'/'pomeriggio')
assert parse_dates_and("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019") == ("2019-11-15", "2019-11-16")

first date missing year
assert parse_dates_and("lunedì 13 settembre e sabato 25 dicembre 2019") == ("2019-09-13", "2019-12-25")

first date missing month and year
assert parse_dates_and("Giovedì 12 e domenica 15 dicembre 2019") == ("2019-12-12", "2019-12-15")

assert parse_dates_and("giovedì 9 e domenica 12 gennaio 2020") == ("2020-01-09", "2020-01-12")

assert parse_dates_and("lunedì 4 e domenica 10 novembre 2019") == ("2019-11-04", "2019-11-10")

first missing month and year, second missing year
assert parse_dates_and("lunedì 4 e domenica 10 novembre") == ("yyyy-11-04", "yyyy-11-10")

first missing month and year, second missing month and year
assert parse_dates_and("lunedì 4 e domenica 10") == ("yyyy-mm-04", "yyyy-mm-10")

```

&lt;/div&gt;

[6]:

```

def parse_dates_and(text):
 raise Exception('TODO IMPLEMENT ME !')

complete dates
assert parse_dates_and("lunedì 25 aprile 2018 e domenica 01 dicembre 2019") == ("2018-04-25", "2019-12-01")

exactly two dates, day phase morning/afternoon ('mattina'/'pomeriggio')
assert parse_dates_and("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019") == ("2019-11-15", "2019-11-16")

```

(continues on next page)

(continued from previous page)

```
first date missing year
assert parse_dates_and("lunedì 13 settembre e sabato 25 dicembre 2019") == ("2019-09-13", "2019-12-25")

first date missing month and year
assert parse_dates_and("Giovedì 12 e domenica 15 dicembre 2019") == ("2019-12-12", "2019-12-15")

assert parse_dates_and("giovedì 9 e domenica 12 gennaio 2020") == ("2020-01-09", "2020-01-12")

assert parse_dates_and("lunedì 4 e domenica 10 novembre 2019") == ("2019-11-04", "2019-11-10")

first missing month and year, second missing year
assert parse_dates_and("lunedì 4 e domenica 10 novembre") == ("yyyy-11-04", "yyyy-11-10")

first missing month and year, second missing month and year
assert parse_dates_and("lunedì 4 e domenica 10") == ("yyyy-mm-04", "yyyy-mm-10")
```

### 10.2.3 What's your business?

#### Download worked project

Browse files online<sup>447</sup>

So you want to be a data scientist. Good, plenty of opportunities ahead!

After graduating, you might discover though that many companies require you to actually work as a freelancer: you will just need to declare to the state which type of economic activity you are going to perform, they say. Seems easy, but you will soon encounter a pretty bureaucratic problem: do public institutions even *know* what a data scientist is? If not, what is the closest category they recognize? Is there any specific *exclusion* that would bar you from entering that category?

If you are in Europe, you will be presented with a catalog of economic activites you can choose from called **NACE**<sup>448</sup>, which is then further specialized by various states (for example Italy's catalog is called **ATECO**<sup>449</sup>)

#### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
nace-prj
 nace.ipynb
 nace-sol.ipynb
 NACE_REV2_20200628_213139.csv
 jupman.py
```

<sup>447</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/nace>

<sup>448</sup> [https://ec.europa.eu/eurostat/ramon/nomenclatures/index.cfm?TargetUrl=LST\\_NOM\\_DTL&StrNom=NACE\\_REV2&StrLanguageCode=EN&IntPcKey=&StrLayoutCode=HIERARCHIC](https://ec.europa.eu/eurostat/ramon/nomenclatures/index.cfm?TargetUrl=LST_NOM_DTL&StrNom=NACE_REV2&StrLanguageCode=EN&IntPcKey=&StrLayoutCode=HIERARCHIC)

<sup>449</sup> <https://www.istat.it/it/archivio/17888>

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook nace.ipynb
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## NACE Sections

A NACE code is subdivided in a hierarchical, four-level structure. The categories at the highest level are called *sections*, here they are:

<b>Detail</b>	
+ A	AGRICULTURE, FORESTRY AND FISHING <a href="#">Detail</a>
+ B	MINING AND QUARRYING <a href="#">Detail</a>
+ C	MANUFACTURING <a href="#">Detail</a>
+ D	ELECTRICITY, GAS, STEAM AND AIR CONDITIONING SUPPLY <a href="#">Detail</a>
+ E	WATER SUPPLY; SEWERAGE, WASTE MANAGEMENT AND REMEDIATION ACTIVITIES <a href="#">Detail</a>
+ F	CONSTRUCTION <a href="#">Detail</a>
+ G	WHOLESALE AND RETAIL TRADE; REPAIR OF MOTOR VEHICLES AND MOTORCYCLES <a href="#">Detail</a>
+ H	TRANSPORTATION AND STORAGE <a href="#">Detail</a>
+ I	ACCOMMODATION AND FOOD SERVICE ACTIVITIES <a href="#">Detail</a>
+ J	INFORMATION AND COMMUNICATION <a href="#">Detail</a>
+ K	FINANCIAL AND INSURANCE ACTIVITIES <a href="#">Detail</a>
+ L	REAL ESTATE ACTIVITIES <a href="#">Detail</a>
+ M	PROFESSIONAL, SCIENTIFIC AND TECHNICAL ACTIVITIES <a href="#">Detail</a>
+ N	ADMINISTRATIVE AND SUPPORT SERVICE ACTIVITIES <a href="#">Detail</a>
+ O	PUBLIC ADMINISTRATION AND DEFENCE; COMPULSORY SOCIAL SECURITY <a href="#">Detail</a>
+ P	EDUCATION <a href="#">Detail</a>
+ Q	HUMAN HEALTH AND SOCIAL WORK ACTIVITIES <a href="#">Detail</a>
+ R	ARTS, ENTERTAINMENT AND RECREATION <a href="#">Detail</a>
+ S	OTHER SERVICE ACTIVITIES <a href="#">Detail</a>
+ T	ACTIVITIES OF HOUSEHOLDS AS EMPLOYERS; UNDIFFERENTIATED GOODS- AND SERVICES-PRODUCING ACTIVITIES OF HOUSEHOLDS FOR OWN USE <a href="#">Detail</a>
+ U	ACTIVITIES OF EXTRATERRITORIAL ORGANISATIONS AND BODIES <a href="#">Detail</a>

## Section detail

If you drill down in say, section M, you will find something like this:

The first two digits of the code identify the *division*, the third digit identifies the *group*, and the fourth digit identifies the *class*:

**M PROFESSIONAL, SCIENTIFIC AND TECHNICAL ACTIVITIES**

69 Legal and accounting activities

69.1 Legal activities

69.10 Legal activities

69.2 Accounting, bookkeeping and auditing activities; tax consultancy

69.20 Accounting, bookkeeping and auditing activities; tax consultancy

70 Activities of head offices; management consultancy activities

70.1 Activities of head offices

70.10 Activities of head offices

70.2 Management consultancy activities

70.21 Public relations and communication activities

70.22 Business and other management consultancy activities

71 Architectural and engineering activities; technical testing and analysis

71.1 Architectural and engineering activities and related technical consultancy

71.11 Architectural activities

71.12 Engineering activities and related technical consultancy

71.2 Technical testing and analysis

71.20 Technical testing and analysis

72 Scientific research and development

72.1 Research and experimental development on natural sciences and engineering

72.11 Research and experimental development on biotechnology

72.19 Other research and experimental development on natural sciences and engineering

72.2 Research and experimental development on social sciences and humanities

72.20 Research and experimental development on social sciences and humanities

73 Advertising and market research

73.1 Advertising

73.11 Advertising agencies

73.12 Media representation

---

Let's pick for example *Advertising agencies*, which has code 73.11:

Level		Code	Spec	Description
1	<b>Sec-tion</b>	<b>M</b>	a single alphabetic char	PROFESSIONAL, SCIENTIFIC AND TECHNICAL ACTIVITIES
2	<b>Divi-sion</b>	<b>73</b>	two-digits	Advertising and market research
3	<b>Group</b>	<b>73.1</b>	three-digits, with dot after first two	Advertising
4	<b>Class</b>	<b>73.12</b>	four-digits, with dot after first two	Advertising agencies

## Specifications

**WARNING: CODES MAY CONTAIN ZEROES!**

**IF YOU LOAD THE CSV IN LIBREOFFICE CALC OR EXCEL, MAKE SURE IT IMPORTS EVERYTHING AS STRING!**

**WATCH OUT FOR CHOPPED ZEROES !**

### Zero examples:

- *Veterinary activities* contains a double zero *at the end* : 75.00
- group *Manufacture of beverages* contains a single zero at the end: 11.0
- *Manufacture of beer* contains zero *inside* : 11.05
- *Support services to forestry* contains a zero *at the beginning* : 02.4 which is different from 02.40 even if they have the same description !

**The section level code is not integrated in the NACE code:** For example, the activity *Manufacture of glues* is identified by the code 20.52, where 20 is the code for the division, 20.5 is the code for the group and 20.52 is the code of the class; section C, to which this class belongs, does not appear in the code itself.

**There may be gaps** (not very important for us): The divisions are coded consecutively. However, some “gaps” have been provided to allow the introduction of additional divisions without a complete change of the NACE coding.

## NACE CSV

We provide you with a CSV `NACE_REV2_20200628_213139.csv` that contains all the codes. Try to explore it with LibreOffice Calc or pandas

Here we show some relevant parts (**NOTE:** you **DON'T** need to use pandas)

```
[1]: import pandas as pd # we import pandas and for ease we rename it to 'pd'
import numpy as np # we import numpy and for ease we rename it to 'np'

pd.set_option('display.max_colwidth', None)
df = pd.read_csv('NACE_REV2_20200628_213139.csv', encoding='UTF-8')
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 996 entries, 0 to 995
Data columns (total 10 columns):
 # Column Non-Null Count Dtype
 --- --
 0 Order 996 non-null int64
 1 Level 996 non-null int64
 2 Code 996 non-null object
 3 Parent 975 non-null object
 4 Description 996 non-null object
 5 This item includes 778 non-null object
 6 This item also includes 202 non-null object
 7 Rulings 134 non-null object
 8 This item excludes 507 non-null object
 9 Reference to ISIC Rev. 4 996 non-null object
dtypes: int64(2), object(8)
memory usage: 77.9+ KB
```



(continued from previous page)

4  
This class includes:  
(including organic farming and the growing of genetically modified rice)

This item also includes \

0  
NaN

1 This division also includes service activities incidental to agriculture, as well  
as hunting, trapping and related activities.

2  
NaN

3  
NaN

4  
NaN

Rulings \

0 NaN

1 NaN

2 NaN

3 NaN

4 NaN

This item excludes \

0

NaN

1 Agricultural activities exclude any subsequent processing of the agricultural products (classified under divisions 10 and 11 (Manufacture of food products and beverages) and division 12 (Manufacture of tobacco products)), beyond that needed to prepare them for the primary markets. The preparation of products for the primary markets is included here.  
The division excludes field construction (e.g. agricultural land terracing, drainage, preparing rice paddies etc.) classified in section F (Construction) and buyers and cooperative associations engaged in the marketing of farm products classified in section G. Also excluded is the landscape care and maintenance, which is classified in class 81.30.

(continues on next page)

(continued from previous page)

We can focus on just these columns:

```
[3]: selection = [398482, 398488, 398530, 398608, 398482, 398518, 398521, 398567]

from IPython.display import display

example_df = df[['Order', 'Level', 'Code', 'Parent', 'Description', 'This item excludes']]
Assuming the variable df contains the relevant DataFrame
example_df = example_df[example_df['Order'].isin(selection)]
display(example_df.style.set_properties(**{'white-space': 'pre-wrap',}))
```

## 1. Extracting codes

Let's say European Commission wants to review the catalog to simplify it. One way to do it, could be to look for codes that have lots of exclusions, the reasoning being that trying to explain somebody something by stating what it is *not* often results in confusion.

### 1.1 is\_nace

Implement following function. NOTE: it was not explicitly required in the original exam but could help detecting words.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4]:

```
def is_nace(word):
 """Given a word, RETURN True if the word is a NACE code, else otherwise"""

 # we could implement it also with regexes, here we use explicit methods:
 if len(word) == 1:
 return word.isalpha() and word.isupper()
 elif len(word) == 2:
 return word.isdigit()
 elif len(word) == 4:
 return word[:2].isdigit() and word[2] == '.' and word[3].isdigit()
 elif len(word) == 5:
 return word[:2].isdigit() and word[2] == '.' and word[3:4].isdigit()
 else:
 return False

assert is_nace('0') == False
assert is_nace('01') == True
assert is_nace('A') == True # this is a Section
assert is_nace('AA') == False
assert is_nace('a') == False
assert is_nace('01.2') == True
assert is_nace('01.20') == True
assert is_nace('03.25') == True
assert is_nace('02.753') == False
assert is_nace('300') == False
assert is_nace('5012') == False
```

</div>

[4]:

```
def is_nace(word):
 """Given a word, RETURN True if the word is a NACE code, else otherwise"""
 raise Exception('TODO IMPLEMENT ME !')

assert is_nace('0') == False
assert is_nace('01') == True
assert is_nace('A') == True # this is a Section
assert is_nace('AA') == False
assert is_nace('a') == False
assert is_nace('01.2') == True
assert is_nace('01.20') == True
assert is_nace('03.25') == True
```

(continues on next page)

(continued from previous page)

```
assert is_nace('02.753') == False
assert is_nace('300') == False
assert is_nace('5012') == False
```

## 1.2 extract\_codes

Implement following function which extracts codes from This item excludes column cells. For examples, see asserts.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[5]: def extract_codes(text):
 """Extracts all the NACE codes from given text (a single string),
 and RETURN a list of the codes

 - also extracts section letters
 - list must have *no* duplicates
 """
 ret = []

 words = [word.strip(';,.:\()' for word in text.replace('-', ' ').split())
 for i in range(len(words)):

 if i < len(words) - 1 \
 and words[i].lower() == 'section' \
 and len(words[i+1]) == 1 \
 and words[i+1][0].isalpha():

 if words[i+1] not in ret:
 ret.append(words[i+1])
 else:
 if is_nace(words[i]) and words[i] not in ret:
 ret.append(words[i])

 return ret

assert extract_codes('group 02.4') == ['02.4']
assert extract_codes('class 02.40') == ['02.40']
assert extract_codes('.') == []
assert extract_codes('exceeding 300 litres') == []
assert extract_codes('see 46.34') == ['46.34']
assert extract_codes('divisions 10 and 11') == ['10', '11']
assert extract_codes('(10.20)') == ['10.20']
assert extract_codes('(30.1, 33.15)') == ['30.1', '33.15']
assert extract_codes('as outlined in groups 85.1-85.4, i.e.') == ['85.1', '85.4']
assert extract_codes('see 25.99 see 25.99') == ['25.99'] # no duplicates
assert extract_codes('section A') == ['A']
assert extract_codes('in section G. Also') == ['G']
assert extract_codes('section F (Construction)') == ['F']
assert extract_codes('section A, section A') == ['A']
```

</div>

```
[5]: def extract_codes(text):
 """Extracts all the NACE codes from given text (a single string),
 and RETURN a list of the codes

 - also extracts section letters
 - list must have *no* duplicates
 """
 raise Exception('TODO IMPLEMENT ME !')

assert extract_codes('group 02.4') == ['02.4']
assert extract_codes('class 02.40') == ['02.40']
assert extract_codes('.') == []
assert extract_codes('exceeding 300 litres') == []
assert extract_codes('see 46.34') == ['46.34']
assert extract_codes('divisions 10 and 11') == ['10', '11']
assert extract_codes('(10.20)') == ['10.20']
assert extract_codes('(30.1, 33.15)') == ['30.1', '33.15']
assert extract_codes('as outlined in groups 85.1-85.4, i.e.') == ['85.1', '85.4']
assert extract_codes('see 25.99 see 25.99') == ['25.99'] # no duplicates
assert extract_codes('section A') == ['A']
assert extract_codes('in section G. Also') == ['G']
assert extract_codes('section F (Construction)') == ['F']
assert extract_codes('section A, section A') == ['A']
```

```
[6]: # MORE REALISTIC asserts:

t01 = """Agricultural activities exclude any subsequent processing of the agricultural products (classified under divisions 10 and 11 (Manufacture of food products and beverages) and division 12 (Manufacture of tobacco products)), beyond that needed to prepare them for the primary markets. The preparation of products for the primary markets is included here.

The division excludes field construction (e.g. agricultural land terracing, drainage, preparing rice paddies etc.) classified in section F (Construction) and ↴buyers and cooperative associations engaged in the marketing of farm products classified in section G. Also excluded is the landscape care and maintenance, which is classified in class 81.30.
"""
assert extract_codes(t01) == ['10', '11', '12', 'F', 'G', '81.30']

t01_15 = """This class excludes:
- manufacture of tobacco products, see 12.00
"""
assert extract_codes(t01_15) == ['12.00']

t03 = """This division does not include building and repairing of ships and boats (30.1, 33.15) and sport or recreational fishing activities (93.19). Processing of fish, crustaceans or molluscs is excluded, whether at land-based plants or on factory ships (10.20).
"""
assert extract_codes(t03) == ['30.1', '33.15', '93.19', '10.20']

t11_03 = """This class excludes:
- merely bottling and labelling, see 46.34 (if performed as part of wholesale) and 82.92 (if performed on a fee or contract basis)
"""

```

(continues on next page)

(continued from previous page)

```
"""
assert extract_codes(t11_03) == ['46.34', '82.92']

t01_64 = """This class excludes:
- growing of seeds, see groups 01.1 and 01.2
- processing of seeds to obtain oil, see 10.41
- research to develop or modify new forms of seeds, see 72.11
"""
assert extract_codes(t01_64) == ['01.1','01.2','10.41','72.11']

t02 = """Excluded is further processing of wood beginning with sawmilling and planing
of wood,
see division 16.
"""
assert extract_codes(t02) == ['16']

t09_90 = """This class excludes:
- operating mines or quarries on a contract or fee basis, see division 05, 07 or 08
- specialised repair of mining machinery, see 33.12
- geophysical surveying services, on a contract or fee basis, see 71.12
"""
assert extract_codes(t09_90) == ['05','07','08','33.12','71.12']
```

## 2. build\_db

Given a filepath pointing to a NACE CSV, reads the CSV and RETURN a dictionary mapping codes to dictionaries which hold the code descriptionn and a field with the list of excluded codes, for example:

```
{'01': {'description': 'Crop and animal production, hunting and related service',
 'activities': [
 'exclusions': ['10', '11', '12', 'F', 'G', '81.30']],
 '01.1': {'description': 'Growing of non-perennial crops', 'exclusions': []},
 '01.11': {'description': 'Growing of cereals (except rice), leguminous crops and oil
seeds',
 'exclusions': ['01.12', '01.13', '01.19', '01.26']},
 '01.12': {'description': 'Growing of rice', 'exclusions': []},
 '01.13': {'description': 'Growing of vegetables and melons, roots and tubers',
 'exclusions': ['01.28', '01.30']},
 ...
 ...
}
```

The complete desired output is in file `expected_db.py`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[7]: def build_db(filepath):

 ret = {}
 import csv
 with open(filepath, encoding='utf-8', newline='') as f:
 my_reader = csv.DictReader(f, delimiter=',')
 for d in my_reader:
```

(continues on next page)

(continued from previous page)

```

diz = {'description' : d['Description'],
 'exclusions' : extract_codes(d['This item excludes'])}
ret[d['Code']] = diz
return ret

```

activities\_db = build\_db('NACE\_REV2\_20200628\_213139.csv')  
#activities\_db

&lt;/div&gt;

```
[7]: def build_db(filepath):
 raise Exception('TODO IMPLEMENT ME !')

activities_db = build_db('NACE_REV2_20200628_213139.csv')
#activities_db
```

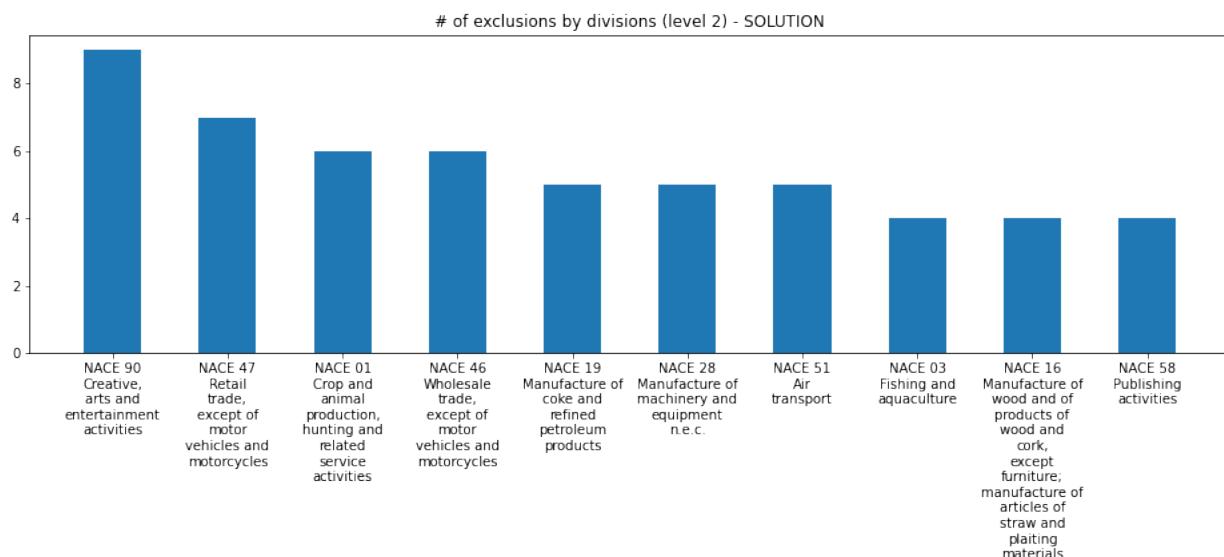
### 3. plot

Implement function `plot` which given a `db` as created at previous point and a code `level` among 1,2,3,4, plots the number of exclusions for all codes of that `exact` level (so do not include sublevels in the sum), sorted in reversed order.

- remember to plot title, notice it should shows the type of level (could be Section, Division, Group, or Class)
- try to display labels nicely as in the example output

(if you look at the graph, apparently European Union has a hard time defining what an artist is :-)

**IMPORTANT: IF you couldn't implement the function `build_db`, you will still find the complete desired output in `expected_db.py`, to import it write: `from expected_db import activities_db`**



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8]:

```
%matplotlib inline
def plot(db, level):

 import matplotlib.pyplot as plt

 coords = [(code, len(db[code]['exclusions'])) for code in db if len(code.replace(
 '.', '')) == level]
 coords.sort(key=lambda c: c[1], reverse=True)

 coords = coords[:10]

 xs = [c[0] for c in coords]
 ys = [c[1] for c in coords]

 fig = plt.figure(figsize=(13, 6)) # width: 10 inches, height 3 inches

 plt.bar(xs, ys, 0.5, align='center')

 def fix_label(label):
 # coding horror, sorry
 return label.replace(' ', '\n').replace('\nand\n', ' and\n').replace('\nof\n', '\n
 ↪of\n')

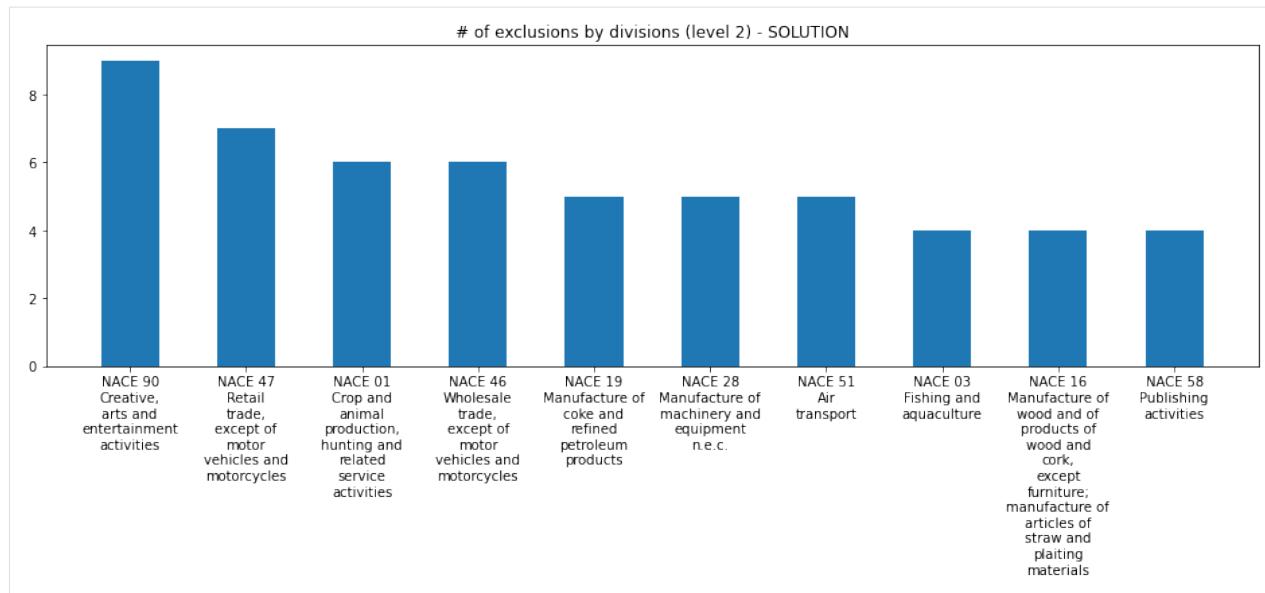
 plt.xticks(xs, ['NACE ' + c[0] + '\n' + fix_label(db[c[0]]['description']) for c
 ↪in coords])

 level_names = {
 1:'Section',
 2:'division',
 3:'Group',
 4:'Class'
 }
 plt.title("# of exclusions by %ss (level %s) - SOLUTION" % (level_names[level],_
 ↪level))
 #plt.xlabel('level_names[level]')
 #plt.ylabel('y')
 fig.tight_layout()

 plt.show()

#Uncomment *only* if you had problems with build_db
#from expected_db import activities_db

#1 Section
#2 Division
#3 Group
#4 Class
plot(activities_db, 2)
```



</div>

```
[8] :
%matplotlib inline
def plot(db, level):

 import matplotlib.pyplot as plt
 raise Exception('TODO IMPLEMENT ME !')

#Uncomment *only* if you had problems with build_db
#from expected_db import activities_db

#1 Section
#2 Division
#3 Group
#4 Class
plot(activities_db, 2)
```

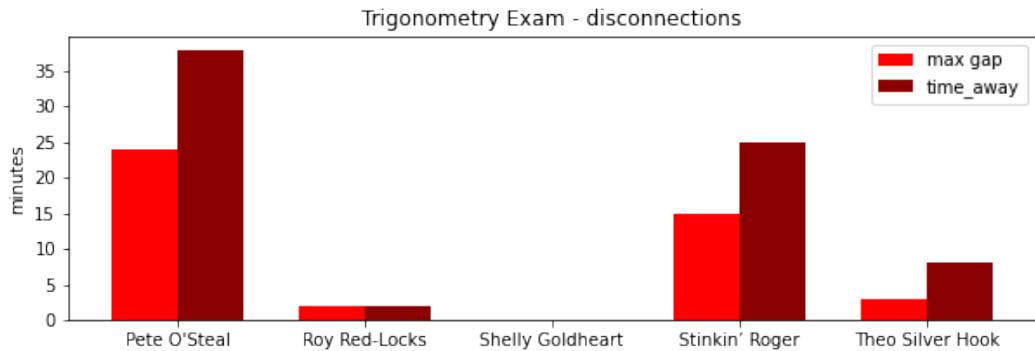
[ ]:

## 10.2.4 Zoom surveillance

[Download worked project](#)

Browse files online<sup>450</sup>

<sup>450</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/zoom-surveillance>



The Academy for Pirate Studies holds online courses with [Zoom software<sup>451</sup>](#). During exams short disconnections may happen due to network problems: for some reason, teachers don't trust much their students and if gaps get too long they may invalidate the exam. Zoom allows to save a meeting log in a sort of CSV format which holds the sessions of each student as join and leave time. You will clean the file content and show relevant data in charts.

If you're a student, you are basically going to build a surveillance system to monitor YOU. Welcome to digital age.

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
zoom-prj
 zoom.ipynb
 zoom-sol.ipynb
 UserQos_12345678901.csv
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `zoom.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press **Control + Enter**
- to execute Python code inside a Jupyter cell AND select next cell, press **Shift + Enter**
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press **Alt + Enter**
- If the notebooks look stuck, try to select **Kernel -> Restart**

<sup>451</sup> <https://zoom.us/>

## CSV format

You are provided with the file `UserQos_12345678901.csv`. Unfortunately, it is a weird CSV which actually looks like two completely different CSVs were merged together, one after the other. It contains the following:

- 1st line: general meeting header
- 2nd line: general meeting data
- 3rd line: empty
- 4th line completely different header for participant sessions for that meeting. Each session contains a join time and a leave time, and each participant can have multiple sessions in a meeting.
- 5th line and following: sessions data

The file has lots of useless fields, try to explore it and understand the format (use LibreOffice Calc to help yourself)

Here we only show the few fields we are actually interested in, and examples of transformations you should apply:

From general meeting information section:

- Meeting ID: 123 4567 8901
- Topic: Trigonometry Exam
- Start Time: "Apr 17, 2020 02:00 PM" should become Apr 17, 2020

From participant sessions section:

- Participant: Roy Red-Locks
- Join Time: 01:54 PM should become 13:54
- Leave Time: 03:10 PM (Roy Red-Locks got disconnected from the meeting. Reason: Network connection error.) should be split into two fields, one for actual leave time in 15:10 format and another one for disconnection reason.

There are 3 possible disconnection reasons (try to come up with a general way to parse them - **notice that there is no dot at the end of transformed string**):

- (Roy Red-Locks got disconnected from the meeting. Reason: Network connection error.) should become Network connection error
- (Pete O'Steal left the meeting. Reason: Host closed the meeting.) should become Host closed the meeting
- (Shelly Goldheart left the meeting. Reason: left the meeting.) should become left the meeting

Your first goal will be to load the dataset and restructure the data so it looks like this:

```
[1]: [[{"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Roy Red-Locks", "join_time": "13:54", "leave_time": "15:10", "reason": "Network connection error"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Roy Red-Locks", "join_time": "15:12", "leave_time": "15:54", "reason": "left the meeting"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Theo Silver Hook", "join_time": "14:02", "leave_time": "14:16", "reason": "Network connection error"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Theo Silver Hook", "join_time": "14:19", "leave_time": "15:02", "reason": "Network connection error"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Theo Silver Hook", "join_time": "15:04", "leave_time": "15:50", "reason": "Network connection error"}]]
```

(continues on next page)

(continued from previous page)

```
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Theo Silver Hook', '15:52', '15:55
↳', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Theo Silver Hook', '15:56', '16:00
↳', 'Host closed the meeting'],
...
]
```

## 1. time24

To fix the times, you will first need to implement the following function.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[2]: def time24(t):
 """ Takes a time string like '06:27 PM' and outputs a string like 18:27
 """

 if t.endswith('AM'):
 if t.startswith('12:00'):
 return '00:00'
 else:
 return t.replace(' AM', '')
 else:
 if t.startswith('12:00'):
 return '12:00'

 h = '%0.d' % (int(t.split(':')[0]) + 12)

 return h + ':' + t.split(':')[1].replace(' PM', '')

assert time24('12:00 AM') == '00:00' # midnight
assert time24('01:06 AM') == '01:06'
assert time24('09:45 AM') == '09:45'
assert time24('12:00 PM') == '12:00' # special case, it's actually midday
assert time24('01:27 PM') == '13:27'
assert time24('06:27 PM') == '18:27'
assert time24('10:03 PM') == '22:03'
```

</div>

```
[2]: def time24(t):
 """ Takes a time string like '06:27 PM' and outputs a string like 18:27
 """
 raise Exception('TODO IMPLEMENT ME !')

assert time24('12:00 AM') == '00:00' # midnight
assert time24('01:06 AM') == '01:06'
assert time24('09:45 AM') == '09:45'
assert time24('12:00 PM') == '12:00' # special case, it's actually midday
assert time24('01:27 PM') == '13:27'
assert time24('06:27 PM') == '18:27'
assert time24('10:03 PM') == '22:03'
```

## 2. load

Implement a function which loads the file `UserQos_12345678901.csv` and RETURN a list of lists, see the format in `EXPECTED_MEETING_LOG` provided below.

To parse the file, you can use simple CSV reader<sup>452</sup> (there is no need to use pandas)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```
import csv

def load(filepath):

 ret = []
 with open(filepath, encoding='utf-8', newline='') as f:

 my_reader = csv.reader(f, delimiter=',')
 next(my_reader)
 row_meeting = next(my_reader)
 meeting_id = row_meeting[0]
 topic = row_meeting[1]
 meeting_date = row_meeting[7]
 next(my_reader) # empty row
 next(my_reader) # second header
 ret.append(['meeting_id', 'topic', 'date', 'participant', 'join_time', 'leave_
 ↪time', 'reason'])
 for row in my_reader:
 session = {}
 if len(row) > 0:
 ret.append([meeting_id,
 topic,
 meeting_date[:12],
 row[0],
 time24(row[10]),
 time24(row[11].split('(')[0]),
 row[11].split('Reason: ')[1].split('.')[0]])
 return ret

meeting_log = load('UserQos_12345678901.csv')

from pprint import pprint
pprint(meeting_log, width=150)

[['meeting_id', 'topic', 'date', 'participant', 'join_time', 'leave_time', 'reason'],
 ['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Roy Red-Locks', '13:54', '15:
 ↪10', 'Network connection error'],
 ['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Roy Red-Locks', '15:12', '15:
 ↪54', 'left the meeting'],
 ['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Theo Silver Hook', '14:02',
 ↪'14:16', 'Network connection error'],
 ['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Theo Silver Hook', '14:19',
 ↪'15:02', 'Network connection error'],
 ['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Theo Silver Hook', '15:04',
 ↪'15:50', 'Network connection error']]
```

(continues on next page)

<sup>452</sup> <https://en.softpython.org/formats/format2-csv-sol.html>

(continued from previous page)

```
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Theo Silver Hook', '15:52',
 ↪'15:55', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Theo Silver Hook', '15:56',
 ↪'16:00', 'Host closed the meeting'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', "Pete O'Steal", '14:15', '14:
 ↪30', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', "Pete O'Steal", '14:54', '15:
 ↪03', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', "Pete O'Steal", '15:12', '15:
 ↪40', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', "Pete O'Steal", '15:45', '16:
 ↪00', 'Host closed the meeting'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Shelly Goldheart', '13:56',
 ↪'15:33', 'left the meeting'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '14:05',
 ↪'14:10', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '14:15',
 ↪'14:29', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '14:33',
 ↪'15:10', 'left the meeting'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '15:25',
 ↪'15:54', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '15:55',
 ↪'16:00', 'Host closed the meeting']]
```

&lt;/div&gt;

[3]:

```
import csv

def load(filepath):
 raise Exception('TODO IMPLEMENT ME !')

meeting_log = load('UserQos_12345678901.csv')

from pprint import pprint
pprint(meeting_log, width=150)
```

[4]:

```
EXPECTED_MEETING_LOG = \
[[{"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Roy Red-Locks", "join_time": "13:54", "leave_time": "15:10", "reason": "Network connection error"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Roy Red-Locks", "join_time": "15:12", "leave_time": "15:54", "reason": "left the meeting"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Theo Silver Hook", "join_time": "14:02", "leave_time": "14:16", "reason": "Network connection error"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Theo Silver Hook", "join_time": "14:19", "leave_time": "15:02", "reason": "Network connection error"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Theo Silver Hook", "join_time": "15:04", "leave_time": "15:50", "reason": "Network connection error"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Theo Silver Hook", "join_time": "15:52", "leave_time": "15:55", "reason": "Network connection error"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Theo Silver Hook", "join_time": "15:56", "leave_time": "16:00", "reason": "Host closed the meeting"}, {"meeting_id": "123 4567 8901", "topic": "Trigonometry Exam", "date": "Apr 17, 2020", "participant": "Pete O'Steal", "join_time": "14:15", "leave_time": "14:30", "reason": "Network connection error"}]]
```

(continues on next page)

(continued from previous page)

```
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', "Pete O'Steal", '14:54', '15:
→03', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', "Pete O'Steal", '15:12', '15:
→40', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', "Pete O'Steal", '15:45', '16:
→00', 'Host closed the meeting'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Shelly Goldheart', '13:56',
→'15:33', 'left the meeting'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '14:05',
→'14:10', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '14:15',
→'14:29', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '14:33',
→'15:10', 'left the meeting'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '15:25',
→'15:54', 'Network connection error'],
['123 4567 8901', 'Trigonometry Exam', 'Apr 17, 2020', 'Stinkin' Roger', '15:55',
→'16:00', 'Host closed the meeting']]
```

```
assert meeting_log[0] == EXPECTED_MEETING_LOG[0] # header
assert meeting_log[1] == EXPECTED_MEETING_LOG[1] # first Roy Red-Locks row
assert meeting_log[1:3] == EXPECTED_MEETING_LOG[1:3] # Roy Red-Locks rows
assert meeting_log[:4] == EXPECTED_MEETING_LOG[:4] # until first Theo Silver Hook
→row included
assert meeting_log == EXPECTED_MEETING_LOG # all table
```

### 3.1 duration

Given two times as strings **a** and **b** in format like 17:34, RETURN the duration in minutes between them as an integer.

To calculate gap durations, we assume a meeting NEVER ends after midnight

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

```
[5]: def duration(a, b):

 asp = a.split(':')
 ta = int(asp[0])*60+int(asp[1])
 bsp = b.split(':')
 tb = int(bsp[0])*60 + int(bsp[1])
 return tb - ta

assert duration('15:00','15:34') == 34
assert duration('15:00','17:34') == 120 + 34
assert duration('15:50','16:12') == 22
assert duration('09:55','11:06') == 5 + 60 + 6
assert duration('00:00','00:01') == 1
#assert duration('11:58','00:01') == 3 # no need to support this case !!
```

</div>

```
[5]: def duration(a, b):
 raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
assert duration('15:00','15:34') == 34
assert duration('15:00','17:34') == 120 + 34
assert duration('15:50','16:12') == 22
assert duration('09:55','11:06') == 5 + 60 + 6
assert duration('00:00','00:01') == 1
#assert duration('11:58','00:01') == 3 # no need to support this case !!
```

### 3.2 calc\_stats

We want to know something about the time each participant has been disconnected from the exam. We call such intervals gaps, which are the difference between a session leave time and successive session join time.

Implement the function `calc_stats` that given a cleaned log produced by `load`, RETURN a dictionary mapping each participant to a dictionary with these statistics:

- `max_gap` : the longest time in minutes in which the participant has been disconnected
- `gaps` : the number of disconnections happened to the participant during the meeting
- `time_away` : the total time in minutes during which the participant has been disconnected during the meeting

To calculate gap durations, we assume a meeting NEVER ends after midnight

For the data format details, see EXPECTED\_STATS below.

**To test the function, you DON'T NEED to have correctly implemented previous functions**

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[6] :

```
def calc_stats(log):
 ret = {}

 last_sessions = {}

 first = True
 for session in log:
 if first:
 first = False
 continue
 date = session[2]
 participant = session[3]
 join_time = session[4]
 leave_time = session[5]
 reason = session[6]

 if participant not in ret:
 ret[participant] = {'max_gap': 0,
 'gaps': 0,
 'time_away': 0}
```

(continues on next page)

(continued from previous page)

```

if participant in last_sessions:
 last_leave_time = last_sessions[participant][5]
 gap = duration(last_leave_time, join_time)
 ret[participant]['max_gap'] = max(gap, ret[participant]['max_gap'])
 ret[participant]['gaps'] += 1
 ret[participant]['time_away'] += gap

last_sessions[participant] = session
return ret

```

stats = calc\_stats(meeting\_log)

# in case you had trouble implementing load function, use this:  
`#stats = calc_stats(EXPECTED_MEETING_LOG)`

stats

[6]: { 'Roy Red-Locks': {'max\_gap': 2, 'gaps': 1, 'time\_away': 2},  
 'Theo Silver Hook': {'max\_gap': 3, 'gaps': 4, 'time\_away': 8},  
 'Pete O'Steal': {'max\_gap': 24, 'gaps': 3, 'time\_away': 38},  
 'Shelly Goldheart': {'max\_gap': 0, 'gaps': 0, 'time\_away': 0},  
 'Stinkin' Roger': {'max\_gap': 15, 'gaps': 4, 'time\_away': 25} }

&lt;/div&gt;

[6]:

```

def calc_stats(log):
 raise Exception('TODO IMPLEMENT ME !')

stats = calc_stats(meeting_log)

in case you had trouble implementing load function, use this:
#stats = calc_stats(EXPECTED_MEETING_LOG)

stats

```

[7]: EXPECTED\_STATS = {"Pete O'Steal" : {'gaps': 3, 'max\_gap': 24, 'time\_away': 38},  
 "Roy Red-Locks" : {'gaps': 1, 'max\_gap': 2, 'time\_away': 2},  
 "Theo Silver Hook": {'gaps': 4, 'max\_gap': 3, 'time\_away': 8},  
 "Shelly Goldheart": {'gaps': 0, 'max\_gap': 0, 'time\_away': 0},  
 "Stinkin' Roger" : {'gaps': 4, 'max\_gap': 15, 'time\_away': 25} }

```

assert stats == EXPECTED_STATS

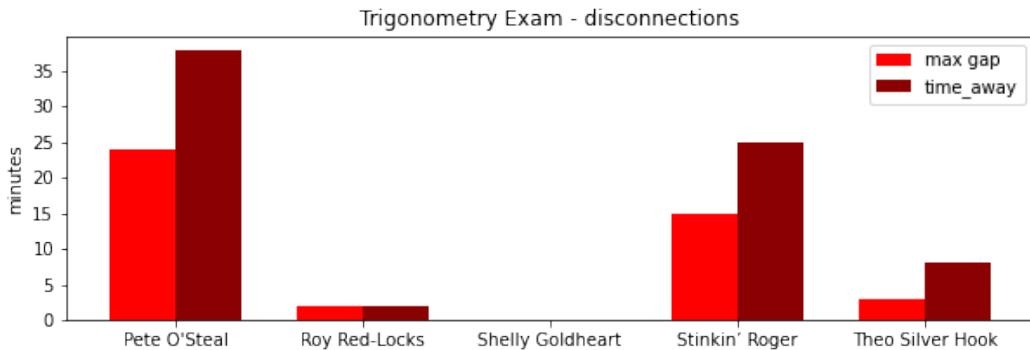
```

#### 4. viz

Produce a bar chart of the statistics you calculated before. For how to do it, see example [here](#)<sup>453</sup>

- participant names MUST be sorted in alphabetical order
- remember to put title, legend and axis labels

To test the function, you DON'T NEED to have correctly implemented previous functions



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8] :

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def viz(exam_name, stats):

 xs = np.arange(len(stats))
 ys_max_gap = []
 ys_time_away = []

 labels = list(sorted(stats.keys()))

 for participant in sorted(stats):
 pstats = stats[participant]
 ys_max_gap.append(pstats['max_gap'])
 ys_time_away.append(pstats['time_away'])

 width = 0.35
 fig, ax = plt.subplots(figsize=(10, 3))
 rects1 = ax.bar(xs - width/2, ys_max_gap, width,
 color='red', label='max gap')
 rects2 = ax.bar(xs + width/2, ys_time_away, width,
 color='darkred', label='time_away')

 plt.xticks(xs, labels)
```

(continues on next page)

<sup>453</sup> <https://en.softpython.org/visualization/visualization1-sol.html#Exercise---superheroes>

(continued from previous page)

```

ax.set_title('%s - disconnections' % exam_name)
ax.legend()

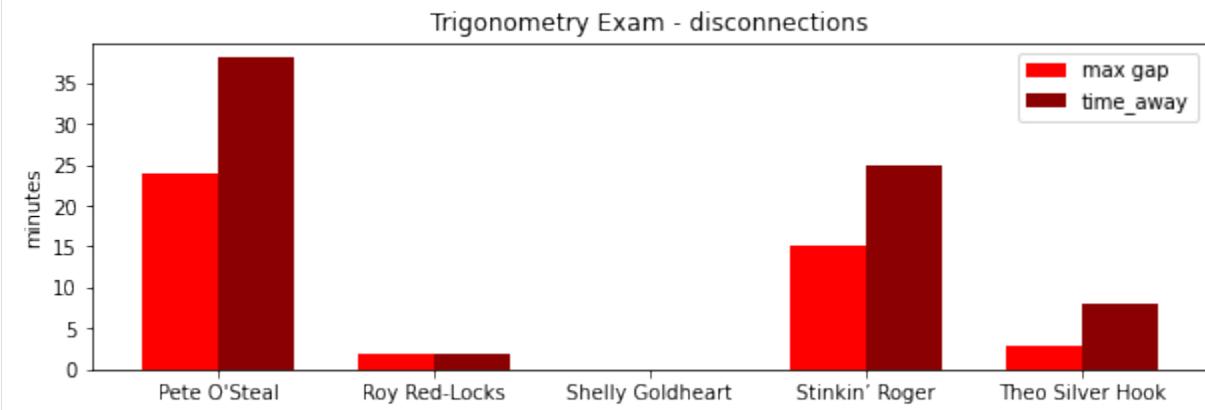
plt.ylabel('minutes')

plt.show()

viz(meeting_log[1][1], stats)

in case you had trouble implementing calc_stats, use this:
#viz(meeting_log[1][1], EXPECTED_STATS)

```



&lt;/div&gt;

```
[8]:
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def viz(exam_name, stats):
 raise Exception('TODO IMPLEMENT ME !')

viz(meeting_log[1][1], stats)

in case you had trouble implementing calc_stats, use this:
#viz(meeting_log[1][1], EXPECTED_STATS)

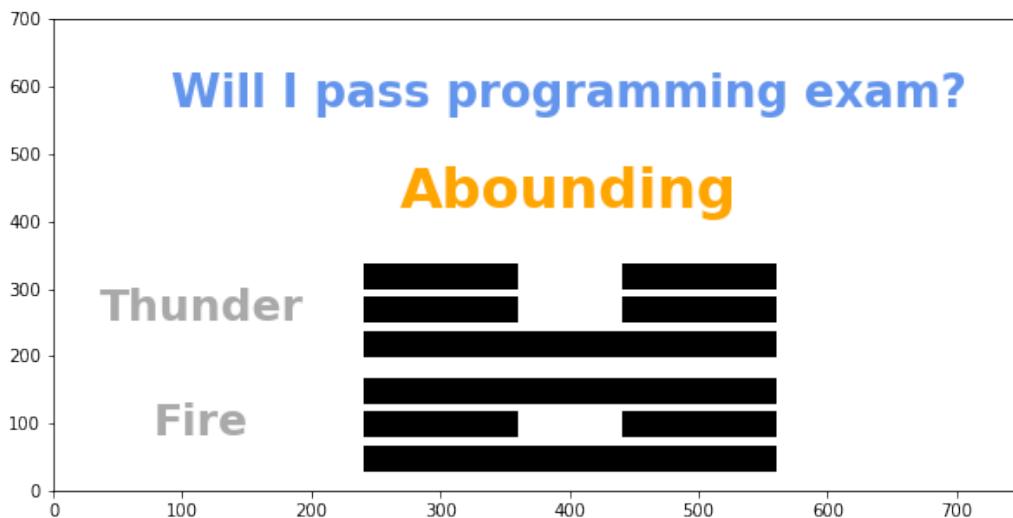
```

[ ]:

## 10.2.5 I CHING Divination

[Download worked project](#)

Browse files online<sup>454</sup>



The I Ching, or Book of Changes, is a Chinese divination manual and philosophical text which is believed to be one of the world's oldest books, dating from over 3,000 years ago.

The great mathematician Gottfried Wilhelm Leibniz (1646 - 1716) is considered the first information theorist, and extensively documented the binary numeral system. Leibniz was also interested in Chinese culture, and saw in the I Ching<sup>455</sup> diagrams showing solid and broken lines called yin and yang, which progressed in a sequence: that was unmistakably a binary encoding.

You will parse a dataset of hexagrams and develop a divinator software which will predict the outcome of your exams.

Data source: [Wikipedia](#), July 2021, Bagua page<sup>456</sup>

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
iching-prj
 iching.ipynb
 iching-sol.ipynb
 iching.csv
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

<sup>454</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/iching>

<sup>455</sup> [https://en.wikipedia.org/wiki/Gottfried\\_Wilhelm\\_Leibniz#Sinophile](https://en.wikipedia.org/wiki/Gottfried_Wilhelm_Leibniz#Sinophile)

<sup>456</sup> <https://en.wikipedia.org/wiki/Bagua>

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `iching.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

## The dataset

**Yin and yang:** Yin and yang are represented by lines:

name	line	bit
yin	--	0
yang	---	1

**Trigrams:** Different constructions of three yin and yang lines lead to 8 trigrams. We can express a trigram as a sequence of bits, reading lines from bottom to top. For example *Fire* is 101, *Thunder* is 100.

乾 Qián ☰ Heaven	坤 Kūn ☷ Earth	震 Zhèn ☳ Thunder	坎 Kǎn ☵ Water	艮 Gèn ☶ Mountain	巽 Xùn ☴ Air	離 Lí ☲ Fire	兌 Dui ☱ Lake
-----------------------	---------------------	------------------------	---------------------	------------------------	-------------------	-------------------	--------------------

**Hexagrams:** Combining a lower trigram with an upper trigram leads to 64 hexagrams. Each hexagram can be represented as a sequence of bits and the outcome of a divination. For example trigrams *Fire* (lower) and *Thunder* (upper) gives outcome hexagram *Abounding*: 101100

Upper → Lower ↓	乾 Qián  Heaven	坤 Kūn  Earth	震 Zhèn  Thunder	坎 Kǎn  Water	艮 Gèn  Mountain	巽 Xùn  Air	離 Lí  Fire	兌 Dui  Lake
乾 Qián  Heaven	01  Force	11  Pervading	34  Great Invigorating	05  Attending	26  Great Accumulating	09  Small Harvest	14  Great Possessing	43  Displacement
坤 Kūn  Earth	12  Obstruction	02  Field	16  Providing-For	08  Grouping	23  Stripping	20  Viewing	35  Prospering	45  Clustering
震 Zhèn  Thunder	25  Innocence	24  Returning	51  Shake	03  Sprouting	27  Swallowing	42  Augmenting	21  Gnawing Bite	17  Following
坎 Kǎn  Water	06  Arguing	07  Leading	40  Deliverance	29  Gorge	04  Enveloping	59  Dispersing	64  Before Completion	47  Confining
艮 Gèn  Mountain	33  Retiring	15  Humbling	62  Small Exceeding	39  Limping	52  Bound	53  Infiltrating	56  Sojourning	31  Conjoining
巽 Xùn  Air	44  Coupling	46  Ascending	32  Persevering	48  Welling	18  Correcting	57  Ground	50  Holding	28  Great Exceeding
離 Lí  Fire	13  Concording People	36  Intelligence Hidden	55  Abounding	63  Already Fording	22  Adorning	37  Dwelling People	30  Radiance	49  Skinning
兌 Dui  Lake	10  Treading	19  Nearing	54  Converting the Maiden	60  Articulating	41  Diminishing	61  Inner Truth	38  Polarising	58  Open

## 1. load\_db

Parse `iching.csv` and output a dictionary mapping each sequence to a dictionary with all the information you can extract. Use CSV reader.

- in headers and first column you will find a bit sequence like 011
- in body cells, you will **not** find a bit sequence: you will have to determine it according to the corresponding tri-sequences from the header and first column
- note for hexagrams you must extract **only** name-en, ignore the decimal numbers

Example (complete output is in file `expected_icing_db.py`):

```
>>> load_db('iching.csv')
{
 '111': {'name-en': 'Heaven', 'name-ch': '乾', 'spelling': 'Qián'}
```

(continues on next page)

(continued from previous page)

```
'000': {'name-en': 'Earth', 'name-ch': '䷂', 'spelling': 'Kūn'}
'100': {'name-en': 'Thunder', 'name-ch': '䷁', 'spelling': 'Zhèn'}
'010': {'name-en': 'Water', 'name-ch': '䷀', 'spelling': 'Kǎn'}
'001': {'name-en': 'Mountain', 'name-ch': '䷃', 'spelling': 'Gèn'}
'011': {'name-en': 'Air', 'name-ch': '䷄', 'spelling': 'Xùn'}
'101': {'name-en': 'Fire', 'name-ch': '䷅', 'spelling': 'Lí'}
'110': {'name-en': 'Lake', 'name-ch': '䷆', 'spelling': 'Dui'}
'111111': {'name-en': 'Force'}
'111000': {'name-en': 'Pervading'}
'111100': {'name-en': 'Great Invigorating'}
'111010': {'name-en': 'Attending'}
'111001': {'name-en': 'Great Accumulating'}
'111011': {'name-en': 'Small Harvest'}
'111101': {'name-en': 'Great Possessing'}
.
.
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[1]: import csv

def load_db(filepath):

 with open(filepath, encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 header = next(my_reader)
 ret = {}

 linear = []

 for s in header[1:]:
 diz = {}
 tokens = s.split('\n')
 tks = tokens[0].split('\xa0')
 diz['name-en'] = tokens[2]
 diz['name-ch'] = tks[0]
 diz['spelling'] = tks[1]
 code = tokens[1]
 ret[code] = diz
 linear.append(code)

 i = 1
 for row in my_reader:
 for j in range(1, len(row)):
 tokens = row[j].replace('\n', '').split()
 num = int(tokens[0])
 bottom = linear[i-1]
 upper = linear[j-1]
 ret[bottom + upper] = {
 'name-en': ' '.join(tokens[1:])
 }
 i += 1
 return ret
```

(continues on next page)

(continued from previous page)

```
iching_db = load_db('iching.csv')
iching_db

</div>

[1]: import csv

def load_db(filepath):
 raise Exception('TODO IMPLEMENT ME !')

iching_db = load_db('iching.csv')
iching_db
```

```
[3]: # EXECUTE FOR TESTING
from pprint import pformat; from expected_iching_db import expected_iching_db
for seq in expected_iching_db.keys():
 if seq not in iching_db: print('\nERROR: MISSING sequence', seq); break
 for k in expected_iching_db[seq]:
 if k not in iching_db[seq]:
 print('\nERROR at sequence', seq, '\n\n MISSING key:', k); break
 if expected_iching_db[seq][k] != iching_db[seq][k]:
 print('\nERROR at sequence', seq, 'key:', k)
 print(' ACTUAL:\n', pformat(iching_db[seq][k]))
 print(' EXPECTED:\n', pformat(expected_iching_db[seq][k]))
 break
```

## 2. divine

A divination is done by flipping 3 coins to determine the bottom trigram (**bottom up order**), flipping other three coins for the upper trigram (again **bottom up order**), and then the union gives the resulting hexagram. Write a function that PRINTS the process as in the example and RETURNS a string of bits representing the resulting hexagram.

- try to avoid writing duplicated code
- **HINT:** to flip coins use `random.randint(0,1)`

**WARNING: DOUBLE CHECK THE ORDER IN WHICH LINES ARE VISUALIZED!**

### Example:

```
>>> divination = divine(iching_db, "Will I pass the exam?")
>>> print("\nRETURNED:", divination)
```

```
Dear stranger, welcome to SOFTPYTHON I CHING ☰ DIVINATOR

Tell me your question...

 Will I pass the exam?

The coin says 'heads' : we get a yang ---
```

(continues on next page)

(continued from previous page)

```
The coin says 'tails' : we get a yin --
The coin says 'heads' : we get a yang ---
```

The sacred bottom trigram is:

Fire

```

- -

```

```
The coin says 'heads' : we get a yang ---
The coin says 'tails' : we get a yin --
The coin says 'tails' : we get a yin --
```

The sacred upper trigram is:

Thunder

```
--
--

```

The final response hexagram is...

Abounding

```
--
--

--

```

RETURNED: 101100

<a class="jupman-sol" jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol" jupman-sol-code" style="display:none">

[4] :

```
import random

def divine(iching, question):

 #THE SEED DETERMINES FOLLOWING randint RESULTS
 random.seed(109) # Abounding
 # Thunder
 # Fire

 #IMPORTANT: try also this seed to check lines visualization order
 #random.seed(1)
 #
 # Infiltrating 001011
 # Air ---
 # ----
 # -- -
```

(continues on next page)

(continued from previous page)

```
Mountain ---
- -
- -

print()
print("Dear stranger, welcome to SOFTPYTHON I CHING ☰ DIVINATOR")
print()
print("Tell me your question...")
print()
print(' ', question)
print()

def get_trigram(part):
 lst = []
 stack = []
 for i in range(3):
 r = random.randint(0,1)
 kind = 'yang' if r else 'yin'
 line = '---' if r else '- -'
 coin = "'heads'" if r else "'tails'"
 print('The coin says', coin, ': we get a', kind, line)
 stack.append(line)
 lst.append(str(r))
 stack.reverse()
 digits = ''.join(lst)
 print()
 print("The sacred", part, "trigram is:",)
 print()

 print(iching[digits]['name-en'])
 print()
 print(' ' + '\n '.join(stack))

 return (stack, ''.join(lst))

bottom = get_trigram('bottom')
print()
upper = get_trigram('upper')

print()
print('The final response hexagram is...')
print()
print(iching[bottom[1] + upper[1]]['name-en'])
print()
print(' ' + '\n '.join(upper[0] + bottom[0]))

return bottom[1] + upper[1]

divination = divine(iching_db, "Will I pass the exam?")
print("\nRETURNED:", divination)
```

Dear stranger, welcome to SOFTPYTHON I CHING ☰ DIVINATOR

---

(continues on next page)

(continued from previous page)

Tell me your question...

Will I pass the exam?

The coin says 'heads' : we get a yang ---  
 The coin says 'tails' : we get a yin --  
 The coin says 'heads' : we get a yang ---

The sacred bottom trigram is:

Fire

---  
 - -  
 ---

The coin says 'heads' : we get a yang ---  
 The coin says 'tails' : we get a yin --  
 The coin says 'tails' : we get a yin --

The sacred upper trigram is:

Thunder

- -  
 - -  
 ---

The final response hexagram is...

Abounding

- -  
 - -  
 ---  
 ---  
 - -  
 ---

RETURNED: 101100

</div>

[4]:

```
import random

def divine(iching, question):

 #THE SEED DETERMINES FOLLOWING randint RESULTS
 random.seed(109) # Abounding
 # Thunder
 # Fire

 #IMPORTANT: try also this seed to check lines visualization order
 #random.seed(1)
 #
 # Infiltrating 001011
```

(continues on next page)

(continued from previous page)

```

Air ---

- -
Mountain ---
- -
- -
- -

raise Exception('TODO IMPLEMENT ME !')

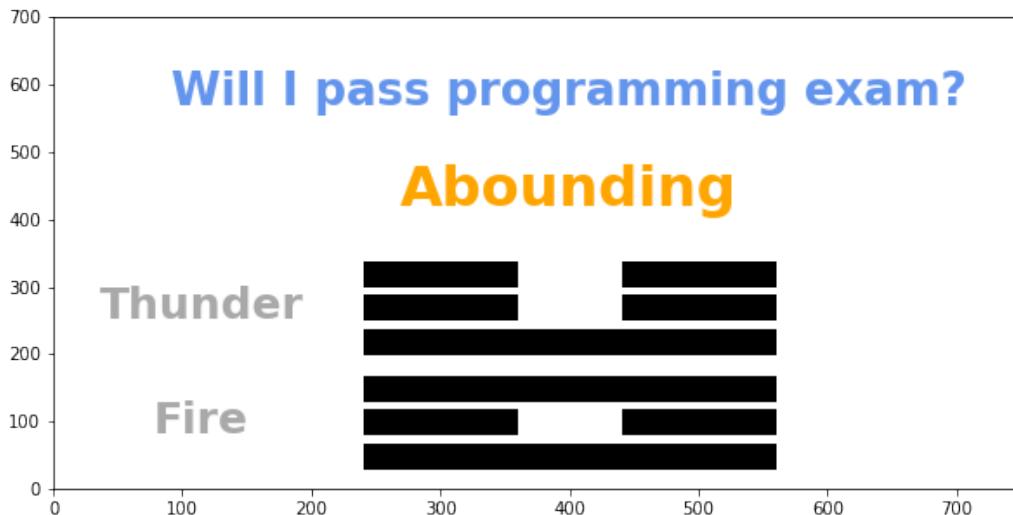
divination = divine(iching_db, "Will I pass the exam?")
print("\nRETURNED:", divination)

```

### 3. plot\_divination

Given a divination as a string of bits, plot the divination.

- first draw the lines, then the rest if you have time.
- make it fancy with these examples<sup>457</sup>
- to center text you can use these parameters: ha='center', va='center'



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5] :

```

%matplotlib inline
import matplotlib.pyplot as plt

def plot_divination(iching, question, divination):

```

(continues on next page)

<sup>457</sup> <https://en.softpython.org/visualization/visualization-sol.html#Fancy-plots>

(continued from previous page)

```

fig = plt.figure(figsize=(10,5))

plt.xlim(0,750)
plt.ylim(0,700)

xl = 150
yd = 50
segw = 100
midx = 400

def plot_trigram(seq, yl):
 plt.text(xl-35,
 yl + yd*2,
 iching[seq]['name-en'],
 fontsize=25,
 fontweight='bold',
 color="darkgray",
 ha='center',
 va='center')

 lw = 15
 for i in range(3):
 h = yl + yd*(i+1)
 if seq[i] == '0':
 plt.plot([xl + segw,xl + segw*2], [h,h],
 color='black',
 linewidth=lw)

 plt.plot([xl + segw*3, xl + segw*4], [h,h],
 color='black',
 linewidth=lw)
 else:
 plt.plot([xl + segw,xl + segw*4], [h,h],
 color='black',
 linewidth=lw)

 plt.text(midx,
 570,
 question,
 fontsize=26,
 fontweight='bold',
 color="CornflowerBlue",
 ha='center')

plot_trigram(divination[:3], 0)
plot_trigram(divination[3:], 170)

plt.text(midx,
 420,
 iching[divination]['name-en'],
 fontsize=32,
 fontweight='bold',
 color="orange",
 ha='center')

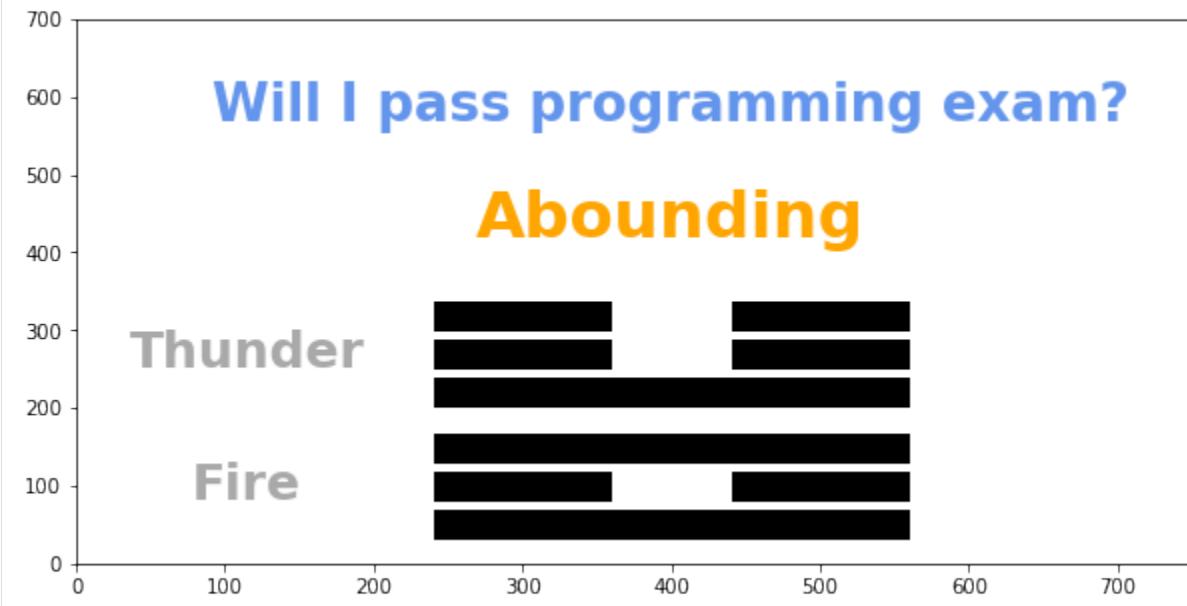
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

```
plot_divination(iching_db, "Will I pass programming exam?", '101100') # Abounding
#plot_divination(iching_db, "Will I pass programming exam?", '111011') # Small Harvest
#plot_divination(iching_db, "Will I pass programming exam?", '001011') # Infiltrating
```



```
</div>
```

[5]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_divination(iching, question, divination):
 raise Exception('TODO IMPLEMENT ME !')

plot_divination(iching_db, "Will I pass programming exam?", '101100') # Abounding
#plot_divination(iching_db, "Will I pass programming exam?", '111011') # Small Harvest
#plot_divination(iching_db, "Will I pass programming exam?", '001011') # Infiltrating
```

## 10.2.6 Witchcraft

### Download worked project

Browse files online<sup>458</sup>

The early sixteenth century saw a dramatic rise in awareness and terror of witchcraft in the troubled lands of [early modern Scotland](#)<sup>459</sup>: thousands of people were executed, imprisoned, tortured, banished, and had lands and possessions confiscated. Persecution took place in courts of law: you shall analyze the evidence gathered during those dark days.

Data source: Julian Goodare, Lauren Martin, Joyce Miller and Louise Yeoman, 'The Survey of Scottish Witchcraft' <http://www.shca.ed.ac.uk/witches/> (archived January 2003, accessed 11/1/2016).

<sup>458</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/witchcraft>

<sup>459</sup> <https://www.youtube.com/watch?v=4s9Hd8onAKQ>

## What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
witchcraft-prj
 witchcraft.ipynb
 witchcraft-sol.ipynb
 WDB_Case.csv
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `witchcraft.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

## The dataset

Among the various tables, we took `WDB_Case.csv` as published on [data.world](#)<sup>460</sup>, which contains cases brought against suspected witches, along with annotations by researchers, mostly as boolean fields.

The dataset has lots of columns, we show here only the relevant ones `Case_date`, `CaseCommonName`, `Suspects_text` and an excerpt of the many boolean columns:

```
[1]: import pandas as pd
case_df = pd.read_csv('WDB_Case.csv', encoding='UTF-8')
sel_case_df = case_df[['Case_date', 'CaseCommonName', 'Suspects_text', 'Demonic_p',
 'Demonic_s', 'Maleficium_p', 'Maleficium_s', 'WitchesMeeting']]
sel_case_df[1986:1994]
```

	Case_date	CaseCommonName	Suspects_text	Demonic_p	Demonic_s	\
1986	29/6/1649	3 unnamed witches	3.0	0	0	
1987	19/8/1590	Leslie, William	NaN	0	0	
1988	1679	McGuffock, Margaret	NaN	0	0	
1989	1679	Rae, Grissell	NaN	0	0	
1990	1679	Howat, Jonet	NaN	0	0	
1991	15/10/1673	McNicol, Janet	NaN	1	1	
1992	4/6/1674	Clerk, Margaret	NaN	0	0	
1993	29/7/1675	Hendrie, Agnes	NaN	0	1	
			Maleficium_p	Maleficium_s	WitchesMeeting	
1986			0	0	0	
1987			0	1	0	
1988			0	1	0	
1989			0	0	0	

(continues on next page)

<sup>460</sup> <https://data.world/history/scottish-witchcraft>

(continued from previous page)

1990	0	0	0
1991	0	1	1
1992	0	0	0
1993	0	0	1

## 1. parse\_bool\_cols

Since boolean columns are so many, as a first step you will build a recognizer for them.

- Consider a column as boolean if **ALL** of its values are either 0 or 1
- Parse with CSV DictReader<sup>461</sup>

### WARNING: Have you carefully read the text above?

Most students don't, and write bad algorithms which declare a column as boolean as soon as a single 0 or a 1 are found, and manually discard columns which don't fit such flawed logic (like NamedIndividual).

To prevent messing up simple exercises, always ask yourself:

1. Am I sure about the results of my algorithm *without* looking at the expected solution ? In this case, it should be obvious that if you don't scan **all** cells in a column and you still declare it's boolean you are basically resorting to being lucky.
2. Am I putting *constants* in the code (like 'NamedIndividual')? Whenever you have such urge, please ask first for permission to your instructor
3. Is the exercise open to interpretation, maybe because it has so many possible weird cases and relative assertions, or is the text pretty clear? In this case the scope is quite definite, so you are expected to find a generic solution which could work with *any* dataset.

**Example** (for full output see `expected_bool_cols.py`):

```
>>> bool_cols = get_bool_cols('WDB_Case.csv')
>>> print('Found', len(bool_cols), 'cols. EXCERPT:', ' '.join(bool_cols[:17]), '...')
Found 77 cols. EXCERPT:
AdmitLesserCharge AggravatingDisease AnimalDeath AnimalIllness ClaimedBewitched_
↪ClaimedNaturalCauses ClaimedPossessed CommunalSex Consulting_p Consulting_s Cursing_
↪Dancing DemonicPact Demonic_p Demonic_posess_p Demonic_posess_s Demonic_s ...
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]:

```
def get_bool_cols(filename):
 """RETURN a sorted list of all the names of boolean columns"""

 import csv
 with open(filename, encoding='utf-8', newline='') as f:

 cols = set(next(csv.DictReader(f, delimiter=',')).keys())
```

(continues on next page)

<sup>461</sup> <https://en.softpython.org/formats/format-sol.html#Reading-as-dictionaries>

(continued from previous page)

```

with open(filename, encoding='utf-8', newline='') as f:
 my_reader = csv.DictReader(f, delimiter=',') # Notice we now used DictReader
 for diz in my_reader:
 for k in diz:

 if not (diz[k] == '0' or diz[k] == '1'):
 if k in cols:
 # Note: for an optimal algorithm we could also stop scanning
 # the rows for this column
 cols.remove(k)

 return sorted(cols)

bool_cols = get_bool_cols('WDB_Case.csv')
print('Found', len(bool_cols), 'cols.', 'EXCERPT: ',)
print(' '.join(bool_cols[:17]), '...')

from expected_bool_cols import expected_bool_cols
if len(bool_cols) != len(expected_bool_cols):
 print('ERROR! different lengths: bools_cols: %s expected_bools_cols: %s' %
 (len(bool_cols), len(expected_bool_cols)))
else:
 for i in range(len(expected_bool_cols)):
 if bool_cols[i] != expected_bool_cols[i]:
 print('ERROR at index', i, ':')
 print(' ACTUAL:', repr(bool_cols[i]))
 print(' EXPECTED:', repr(expected_bool_cols[i]))

```

Found 77 cols. EXCERPT:  
AdmitLesserCharge AggravatingDisease AnimalDeath AnimalIllness ClaimedBewitched  
ClaimedNaturalCauses ClaimedPossessed CommunalSex Consulting\_p Consulting\_s Cursing  
Dancing DemonicPact Demonic\_p Demonic\_posess\_p Demonic\_posess\_s Demonic\_s ...

&lt;/div&gt;

[2]:

```

def get_bool_cols(filename):
 """RETURN a sorted list of all the names of boolean columns"""
 raise Exception('TODO IMPLEMENT ME !')

bool_cols = get_bool_cols('WDB_Case.csv')
print('Found', len(bool_cols), 'cols.', 'EXCERPT: ',)
print(' '.join(bool_cols[:17]), '...')

from expected_bool_cols import expected_bool_cols
if len(bool_cols) != len(expected_bool_cols):
 print('ERROR! different lengths: bools_cols: %s expected_bools_cols: %s' %
 (len(bool_cols), len(expected_bool_cols)))
else:
 for i in range(len(expected_bool_cols)):
 if bool_cols[i] != expected_bool_cols[i]:
 print('ERROR at index', i, ':')
 print(' ACTUAL:', repr(bool_cols[i]))

```

(continues on next page)

(continued from previous page)

```
 print(' EXPECTED:', repr(expected_bool_cols[i]))
```

## 2. fix\_date

Implement `fix_date`, which takes a possibly partial date as a string `d/m/yyyy` and RETURN a string formatted as `mm/dd/yyyy`. If data is missing, omits it in the output as well, see examples.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: def fix_date(d):

 if '/' in d:
 case_date = d.split('/')
 if len(case_date) == 3:
 d = case_date[1]
 m = case_date[0]
 y = case_date[2]
 return "%02d/%02d/%04d" % (int(d), int(m), int(y))
 elif len(case_date) == 2:
 m = case_date[0]
 y = case_date[1]
 return "%02d/%04d" % (int(m), int(y))
 return d

assert fix_date('2/8/1649') == '08/02/1649'
assert fix_date('25/4/1627') == '04/25/1627'
assert fix_date('6/11/1629') == '11/06/1629'
assert fix_date('12/1649') == '12/1649'
assert fix_date('7/1652') == '07/1652'
assert fix_date('1560') == '1560'
assert fix_date('') == ''
Note there is a damned extra space in the dataset (Oswald, Katharine)
assert fix_date('13/11/ 1629') == '11/13/1629'
```

</div>

```
[3]: def fix_date(d):
 raise Exception('TODO IMPLEMENT ME !')

assert fix_date('2/8/1649') == '08/02/1649'
assert fix_date('25/4/1627') == '04/25/1627'
assert fix_date('6/11/1629') == '11/06/1629'
assert fix_date('12/1649') == '12/1649'
assert fix_date('7/1652') == '07/1652'
assert fix_date('1560') == '1560'
assert fix_date('') == ''
Note there is a damned extra space in the dataset (Oswald, Katharine)
assert fix_date('13/11/ 1629') == '11/13/1629'
```

### 3. parse\_db

Given a CSV of cases, outputs a list of dictionaries (parse with [CSV DictReader<sup>462</sup>](#)), each representing a case with these fields:

- name: the isolated name of the witch taken from CaseCommonName column if parseable, otherwise the full cell content
- surname: the isolated surname of the witch taken from CaseCommonName column if parseable, otherwise empty string
- case\_date: Case\_date column corrected with fix\_date
- suspects: number of suspects as **integer**, to be taken from the column Suspects\_text. If column is empty, use 1

primary, secondary and tags fields are to be filled with names of **boolean** columns for which the corresponding cell is marked with '1' according to these criteria:

- primary: a **single string** as follows: if a column ending with \_p is marked '1', this field contains that column name without the '\_p'. If column name is 'NotEnoughInfo\_p' or in other cases, use None.
- secondary: **sorted** column names ending with \_s. If col name is 'NotEnoughInfo\_s' or it's already present as primary, it's discarded. Remove trailing \_s from values in the list.
- tags: **sorted** column names which are not primary nor secondary

**Example** (full output is in `expected_cases_db.py`):

```
>>> cases_db = parse_db('WDB_Case.csv')
>>> cases_db[1991:1994]
[{'primary': 'Demonic',
 'secondary': ['ImplicatedByAnother', 'Maleficium', 'UNorthodoxRelPract'],
 'tags': ['DevilPresent', 'UnorthodoxReligiousPractice', 'WitchesMeeting'],
 'name': 'Janet',
 'surname': 'McNicol',
 'suspects': 1,
 'case_date': '10/15/1673'},
 {'primary': None,
 'secondary': [],
 'tags': [],
 'name': 'Margaret',
 'surname': 'Clerk',
 'suspects': 1,
 'case_date': '06/04/1674'},
 {'primary': None,
 'secondary': ['Demonic'],
 'tags': ['Dancing', 'DevilPresent', 'Singing', 'WitchesMeeting'],
 'name': 'Agnes',
 'surname': 'Hendrie',
 'suspects': 1,
 'case_date': '07/29/1675'}]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4]:

```
import csv
```

(continues on next page)

<sup>462</sup> <https://en.softpython.org/formats/format2-csv-sol.html#Reading-as-dictionaries>

(continued from previous page)

```
def parse_db(filename):

 characterisations = [c[:-2] for c in bool_cols if c.endswith('_p')]
 other_bool_fields = [c for c in bool_cols if not (c.endswith('_p') or c.endswith(
 '_s'))]

 with open(filename, encoding='utf-8', newline='') as f:
 my_reader = csv.DictReader(f, delimiter=',')

 ret = []
 found = {}
 for diz in my_reader:
 work = {'primary': None,
 'secondary': [],
 'tags': []}
 for c in characterisations:
 if diz[c + '_p'] == '1':
 if c != 'NotEnoughInfo':
 work['primary'] = c
 if diz[c + '_s'] == '1':
 if c != 'NotEnoughInfo':
 if c != work['primary']:
 work['secondary'].append(c)
 cn = diz['CaseCommonName']
 if ',' in cn:
 spli = cn.split(',')
 work['name'] = spli[-1].strip()
 if len(spli) == 2:
 work['surname'] = spli[0].strip()
 else:
 work['surname'] = ','.join(spli[:-1]).strip()
 else:
 work['name'] = cn.strip()
 work['surname'] = ''

 sus = diz['Suspects_text']
 if sus:
 work['suspects'] = int(sus)
 else:
 work['suspects'] = 1

 for c in other_bool_fields:
 if diz[c] == '1':
 work['tags'].append(c)

 work['case_date'] = fix_date(diz['Case_date'])
 ret.append(work)

 return ret

cases_db = parse_db('WDB_Case.csv')
```

(continues on next page)

(continued from previous page)

```

cases_db[1991:1994]

[4]: [{"primary": "Demonic",
 "secondary": ['ImplicatedByAnother', 'Maleficium', 'UNorthodoxRelPract'],
 "tags": ['DevilPresent', 'UnorthodoxReligiousPractice', 'WitchesMeeting'],
 "name": 'Janet',
 "surname": 'McNicol',
 "suspects": 1,
 "case_date": '10/15/1673'},
 {"primary": None,
 "secondary": [],
 "tags": [],
 "name": 'Margaret',
 "surname": 'Clerk',
 "suspects": 1,
 "case_date": '06/04/1674'},
 {"primary": None,
 "secondary": ['Demonic'],
 "tags": ['Dancing', 'DevilPresent', 'Singing', 'WitchesMeeting'],
 "name": 'Agnes',
 "surname": 'Hendrie',
 "suspects": 1,
 "case_date": '07/29/1675'}]
```

&lt;/div&gt;

```

[4]:
import csv

def parse_db(filename):
 raise Exception('TODO IMPLEMENT ME !')

cases_db = parse_db('WDB_Case.csv')

cases_db[1991:1994]
```

```

[5]: # TESTS
assert cases_db[0]['primary'] == None
assert cases_db[0]['secondary'] == []
assert cases_db[0]['name'] == '3 unnamed witches'
assert cases_db[0]['surname'] == ''
assert cases_db[0]['suspects'] == 3 # int !
assert cases_db[0]['case_date'] == '08/02/1649'

assert cases_db[1]['primary'] == None
assert cases_db[1]['secondary'] == ['ImplicatedByAnother']
assert cases_db[1]['tags'] == []
assert cases_db[1]['name'] == 'Cristine'
assert cases_db[1]['surname'] == 'Kerington'
assert cases_db[1]['suspects'] == 1 # Suspects_text is '', we put 1
assert cases_db[1]['case_date'] == '05/08/1591'

assert cases_db[1991]['primary'] == 'Demonic'
#NOTE: since 'Demonic' is already 'primary', we removed it from 'secondary'
assert cases_db[1991]['secondary'] == ['ImplicatedByAnother', 'Maleficium',
 ↵ 'UNorthodoxRelPract']
assert cases_db[1991]['tags'] == ['DevilPresent', 'UnorthodoxReligiousPractice',
 ↵ 'WitchesMeeting']
```

(continues on next page)

(continued from previous page)

```

assert cases_db[1991]['name'] == 'Janet'
assert cases_db[1991]['surname'] == 'McNicol'
assert cases_db[1991]['suspects'] == 1 # Suspects_text is '', we put 1
assert cases_db[1991]['case_date'] == '10/15/1673'

assert cases_db[0]['case_date'] == '08/02/1649' # 2/8/1649
assert cases_db[1143]['case_date'] == '1560' # 1560
assert cases_db[924]['case_date'] == '07/1652' # 7/1652
assert cases_db[491]['suspects'] == 15 # 15

#composite name
assert cases_db[249]['name'] == 'Francis' # "Stewart, Earl of Bothwell,Francis"
assert cases_db[249]['surname'] == 'Stewart, Earl of Bothwell'

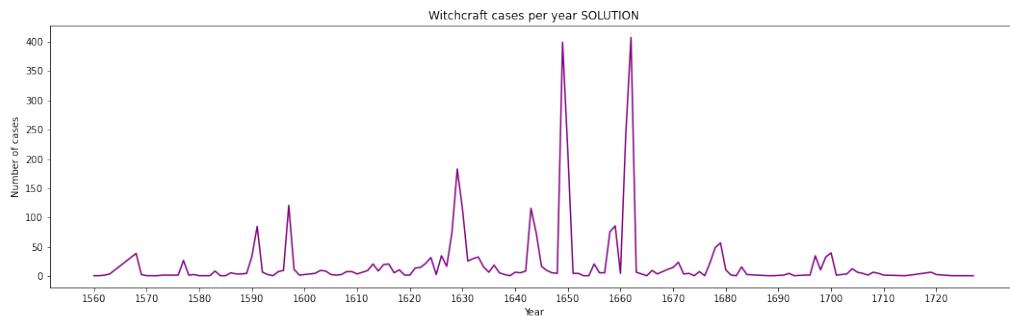
from expected_cases_db import expected_cases_db
from pprint import pprint
for i in range(len(expected_cases_db)):
 if cases_db[i] != expected_cases_db[i]:
 print('ERROR at index %s!' % i)
 print('ACTUAL:')
 pprint(cases_db[i])
 print('EXPECTED:',)
 pprint(expected_cases_db[i])
if len(cases_db) != len(expected_cases_db):
 print('ERROR! different lengths: cases_db: %s expected_cases_db: %s' %_
 (len(cases_db), len(expected_cases_db)))
assert cases_db == expected_cases_db

```

#### 4. plot\_cases

Given the previously computed db, plot the number of cases per year.

- plot the ticks with 10 years intervals, according to the actual data (**DO NOT use constants like 1560 !!**)
- careful some cases have no year



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);'' data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```

def plot_cases(db):

 import numpy as np

```

(continues on next page)

(continued from previous page)

```

import matplotlib.pyplot as plt
from collections import Counter

years = [int(diz['case_date'].split('/')[-1]) for diz in db if diz['case_date']]
hist = Counter(years)
xs = sorted(hist.keys())
ys = [hist[x] for x in xs]

fig = plt.figure(figsize=(18,5))
plt.plot(xs, ys, color='purple')

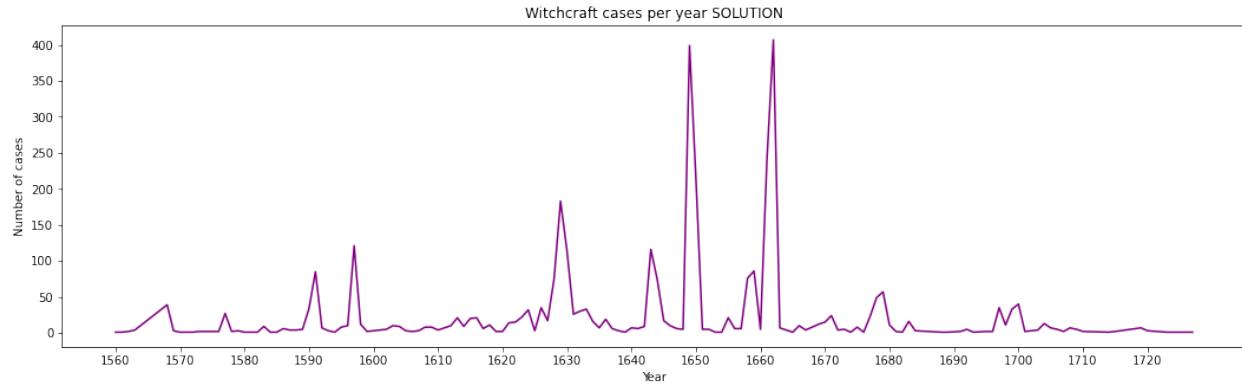
plt.title("Witchcraft cases per year SOLUTION")
plt.xlabel('Year')
plt.ylabel('Number of cases')

tks = np.arange(min(xs), max(xs), 10)
plt.xticks(tks, tks)

plt.show()

```

plot\_cases(cases\_db)



&lt;/div&gt;

[6]:

```

def plot_cases(db):
 raise Exception('TODO IMPLEMENT ME !')

plot_cases(cases_db)

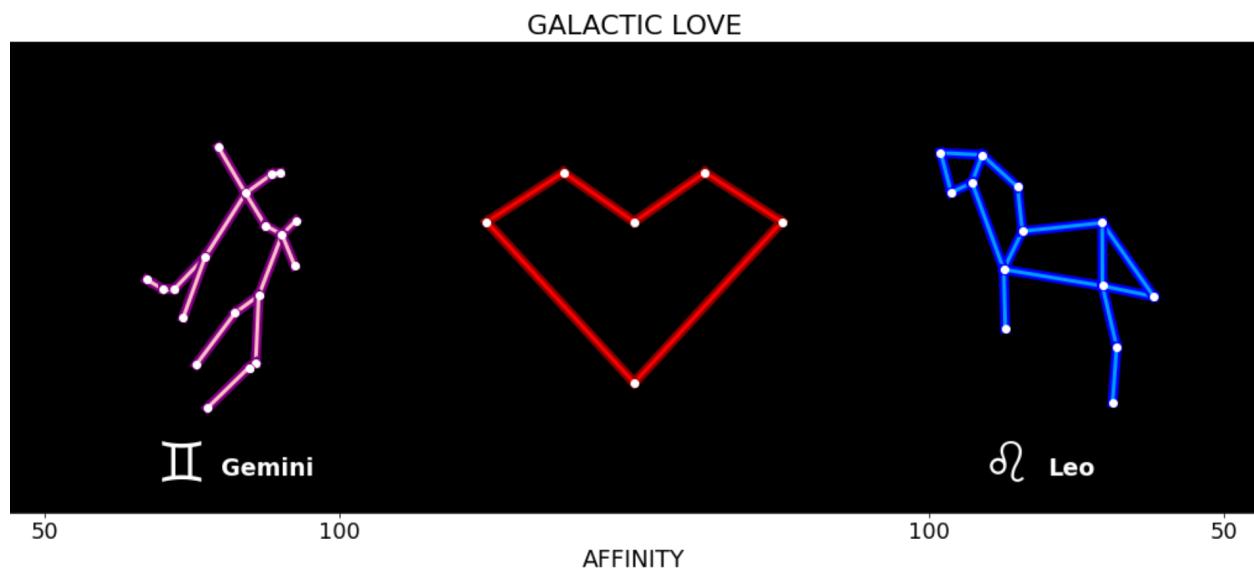
```

[ ]:

### 10.2.7 Galactic love

[Download worked project](#)

Browse files online<sup>463</sup>



The company Astro Logic provides horoscopes to thousands of loyal customers, who each day require a number of divinations. The most requested is whether or not they should engage in love affairs with a potential partner, who is chosen according to rigorous criteria like his/her astrological sign. You are then hired to devise a fancy visualization which given two astrological signs and their love compatibility, displays the constellations of their signs close when the compatibility is high and far away when compatibility is low.

Astrology has been dated to at least the 2nd millennium BCE, and has its roots in calendrical systems used to predict seasonal shifts and to interpret celestial cycles as signs of divine communications. Even if considered a pseudo-science by today standards, it can still provide us with some light-hearted fun while we develop fancy visualizations and matrix manipulations.

#### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
galactic-love-prj
 galactic-love.ipynb
 galactic-love-sol.ipynb
 stars.csv
 zodiac.csv
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook galactic-love.ipynb

<sup>463</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/galactic-love>

3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## parse\_stars

Let's start with real astronomical data. You are given a database of constellations called `stars.csv` (we slightly tweaked it for this occasion - original data source: [Space Telescope Science Institute](#)<sup>464</sup>)

```
[1]: import pandas as pd

stars_df = pd.read_csv('stars.csv', encoding='UTF-8')
stars_df[0:32]
```

	constellation	type	ra	dec	description
0	Andromeda	0	3717	2539	move gamma 1
1	Andromeda	1	2091	2137	draw beta
2	Andromeda	1	1179	1851	draw delta
3	Andromeda	1	251	1745	draw alpha
4	Andromeda	0	1716	1405	move eta
5	Andromeda	1	1420	1456	draw zeta
6	Andromeda	1	1156	1758	draw epsilon
7	Andromeda	1	1179	1851	draw delta
8	Andromeda	1	1106	2023	draw pi
9	Andromeda	1	512	2320	draw theta
10	Andromeda	1	42544	2596	draw iota
11	Andromeda	1	42612	2660	draw kappa
12	Andromeda	1	42526	2787	draw lambda
13	Andromeda	0	42544	2596	move iota
14	Andromeda	1	41457	2539	draw omicron
15	Andromeda	0	1106	2023	move pi
16	Andromeda	1	2091	2137	draw beta
17	Andromeda	1	1702	2309	draw mu
18	Andromeda	1	1494	2464	draw nu
19	Andromeda	1	2085	2834	draw phi
20	Andromeda	1	2939	2917	draw 51
21	Andromeda	-1	0	0	NaN
22	Antlia	0	17077	-2157	move epsilon
23	Antlia	2	18814	-1864	dotted alpha
24	Antlia	2	19701	-2228	dotted iota
25	Antlia	-1	0	0	NaN
26	Apus	0	26635	-4742	move alpha
27	Apus	2	29803	-4733	dotted gamma
28	Apus	2	30092	-4651	dotted beta
29	Apus	2	29410	-4721	dotted delta 1
30	Apus	2	29803	-4733	dotted gamma
31	Apus	-1	0	0	NaN

You will have to parse it so to obtain a dictionary which maps each constellation to its stars, expressed as a list of lists of points type and coordinates.

<sup>464</sup> [https://github.com/mperrin/misc\\_astro](https://github.com/mperrin/misc_astro)

Since later we will need to show points in a 2d chart, you will have to transform the coordinates obtained from the data (right ascension and declination in degrees) as follows:

$$x = \frac{15}{1800}ra$$

$$y = \frac{dec}{60}$$

You can find the complete output in `expected_stars_db.py`

Excerpt:

```
{ 'Andromeda': [
 [0, 30.974999999999998, 42.31666666666666],
 [1, 17.425, 35.61666666666667],
 [1, 9.825000000000001, 30.84999999999998],
 [1, 2.0916666666666667, 29.08333333333332],
 [0, 14.3, 23.416666666666668],
 [1, 11.83333333333332, 24.26666666666666],
 [1, 9.63333333333333, 29.3],
 [1, 9.825000000000001, 30.84999999999998],
 [1, 9.216666666666667, 33.71666666666667],
 [1, 4.266666666666667, 38.66666666666664],
 [1, 354.533333333333, 43.26666666666666],
 [1, 355.0999999999997, 44.3333333333336],
 [1, 354.383333333333, 46.45],
 [0, 354.533333333333, 43.26666666666666],
 [1, 345.475, 42.31666666666666],
 [0, 9.216666666666667, 33.71666666666667],
 [1, 17.425, 35.61666666666667],
 [1, 14.18333333333334, 38.48333333333334],
 [1, 12.45, 41.06666666666666],
 [1, 17.375, 47.23333333333334],
 [1, 24.49166666666667, 48.61666666666667],
 [-1, 0.0, 0.0]
],
'Antlia': [
 [0, 142.3083333333334, -35.95],
 [2, 156.7833333333333, -31.06666666666666],
 [2, 164.175, -37.13333333333333],
 [-1, 0.0, 0.0]
],
.
.
.
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]:

```
import csv

def parse_stars(filename):

 ret = {}
 with open(filename, encoding='utf-8', newline='') as f:
```

(continues on next page)

(continued from previous page)

```

my_reader = csv.reader(f, delimiter=',')
next(my_reader) # skips header
constellation = ''
d = None
for row in my_reader:
 if row[0] != constellation:
 constellation = row[0]
 stars = []
 ret[constellation] = stars
 coords = [0]* 3
 coords[0] = int(row[1])
 coords[1] = int(row[2]) * (1.0 / 1800) * 15
 coords[2] = int(row[3]) * (1.0 / 60)
 stars.append(coords)

return ret

stars_db = parse_stars('stars.csv')

stars_db['Antlia']
#stars_db['Andromeda']

[2]: [[0, 142.30833333333334, -35.95],
 [2, 156.78333333333333, -31.066666666666666],
 [2, 164.175, -37.13333333333333],
 [-1, 0.0, 0.0]]

```

&lt;/div&gt;

```

[2]:
import csv

def parse_stars(filename):
 raise Exception('TODO IMPLEMENT ME !')

stars_db = parse_stars('stars.csv')

stars_db['Antlia']
#stars_db['Andromeda']

```

## plot\_stars 1

Write a function `plot_stars` to plot constellations.

### WARNING: DO NOT use GraphViz!

Even if we are making plots which look like networks, for these visualizations you just need basic matplotlib (and some creativity ;-)

### WARNING: for now, ignore the `new_center` parameter

A point type can either be:

- 0: start a new line not connected with the previous one
- 1: connect previous point with a straight segment
- 2: connect previous point with a dotted segment (draw it with `linestyle=':'` parameter)
- -1: last point, ignore

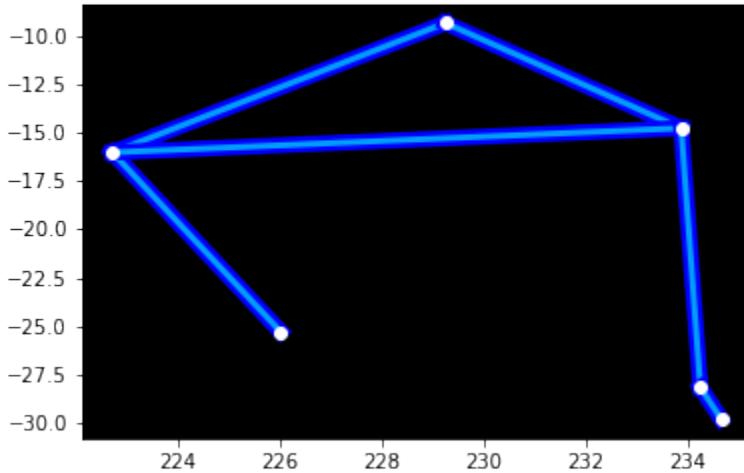
Available colorschemes are 'M', 'F', or 'R' (red)

- to set a black background, set `plt.rcParams['axes.facecolor'] = 'black'`
- draw stars as white dots, setting `markersize=6`
- **to get a nice glowing effect for the lines, draw twice:** once with a thick line and dark color, and once with a thin line with a bright color. You can find the colors in `color_schemes`. To set them in `plt.plot` call, use `linewidth` (sets width in pixels) and `color` parameter, note `color` takes a **single** parameter

### Examples:

```
>>> stars_db['Libra']
[[0, 226.01666666666665, -25.266666666666666],
 [1, 222.71666666666667, -16.03333333333333],
 [1, 229.25, -9.36666666666667],
 [1, 233.875, -14.78333333333333],
 [1, 222.71666666666667, -16.03333333333333],
 [0, 233.875, -14.78333333333333],
 [1, 234.25, -28.13333333333333],
 [1, 234.65833333333333, -29.76666666666666],
 [-1, 0.0, 0.0]]

>>> plot_stars('Libra', 'M', stars_db)
```



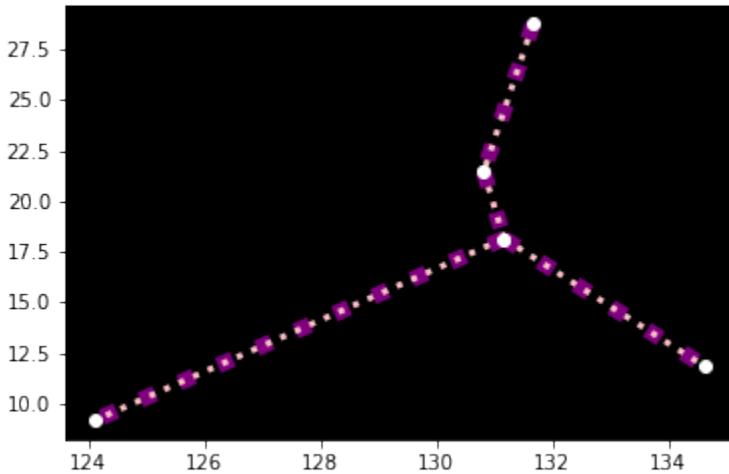
```
>>> stars_db['Cancer'] # has type-2 dotted points
[[0, 131.66666666666669, 28.75],
 [2, 130.81666666666667, 21.466666666666665],
 [2, 131.16666666666666, 18.15],
 [2, 134.61666666666667, 11.85],
 [0, 131.16666666666666, 18.15],
 [2, 124.125, 9.183333333333334],
```

(continues on next page)

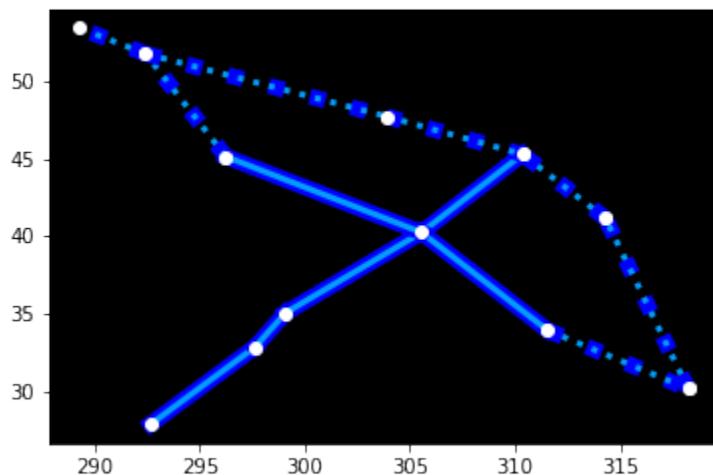
(continued from previous page)

```
[-1, 0.0, 0.0]

>>> plot_stars("Cancer", 'F', stars_db)
```



```
>>> plot_stars("Cygnus", 'M', stars_db) # mixed segment types
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```

color_schemes = {
 'M': ('blue', '#039dfc'),
 'F': ('purple', 'pink'),
 'R': ('darkred', 'red')
}

def plot_stars(constellation_name, color_scheme, stars, new_center=None):

 plt.rcParams['axes.facecolor'] = 'black'

 color1, color2 = color_schemes[color_scheme]

 point_list = stars[constellation_name]
 points = np.asarray(point_list)
 drawtype = points[:,0]
 ra_degrees = points[:-1,1]
 dec_degrees = points[:-1,2]

 if new_center:
 xbounds = (np.min(ra_degrees), np.max(ra_degrees))
 ybounds = (np.min(dec_degrees), np.max(dec_degrees))
 halfx = (xbounds[1]-xbounds[0])/2
 halfy = (ybounds[1]-ybounds[0])/2

 ra_degrees -= xbounds[0] + halfx - new_center[0]
 dec_degrees -= ybounds[1] - halfy - new_center[1]

 for i in range(0, len(drawtype)-1):
 if drawtype[i] == 0 or drawtype[i] == -1:
 continue

 xs = ra_degrees[i - 1:(i)+1]
 ys = dec_degrees[i - 1:(i)+1]
 plt.plot(xs,ys, linewidth=8, linestyle=':' if drawtype[i] == 2 else "--", ↵
 ↵color=color1)
 plt.plot(xs,ys, linewidth=3, linestyle=':' if drawtype[i] == 2 else "--", ↵
 ↵color=color2)
 plt.plot(xs, ys, 'o', markersize=6, color='white')

from pprint import pprint
pprint(stars_db['Libra'])
plot_stars('Libra', 'M', stars_db)

[[0, 226.01666666666665, -25.266666666666666],

 [1, 222.71666666666667, -16.03333333333333],

 [1, 229.25, -9.366666666666667],

 [1, 233.875, -14.78333333333333],

 [1, 222.71666666666667, -16.03333333333333],

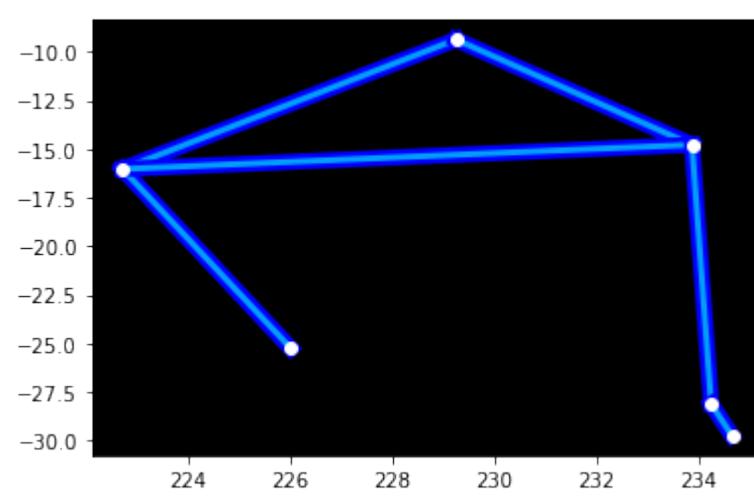
 [0, 233.875, -14.78333333333333],

 [1, 234.25, -28.13333333333333],

 [1, 234.65833333333333, -29.766666666666666],

 [-1, 0.0, 0.0]]

```



&lt;/div&gt;

```
[3]: %matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

color_schemes = {
 'M': ('blue', '#039dfc'),
 'F': ('purple', 'pink'),
 'R': ('darkred', 'red')
}

def plot_stars(constellation_name, color_scheme, stars, new_center=None):
 raise Exception('TODO IMPLEMENT ME !')

from pprint import pprint
pprint(stars_db['Libra'])
plot_stars('Libra', 'M', stars_db)
```

```
[4]: pprint(stars_db['Cancer']) # has type-2 dotted points
plot_stars("Cancer", 'F', stars_db)
```

```
[5]: plot_stars("Cygnus", 'M', stars_db) # mixed segment types
```

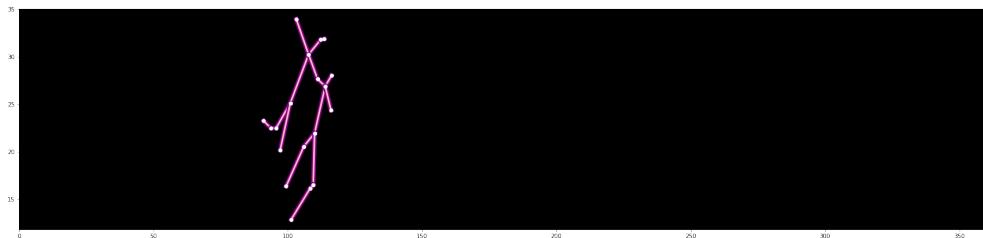
## plot\_stars 2 - new\_center

Change the previous function `plot_stars` so it accepts a new argument `new_center`, which is either `None` or a tuple of coordinates where the constellation should be centered:

- be precise in determining the boundaries of the constellation
- **DO NOT** assume the constallation has a fixed width nor height (so no constants in code!)

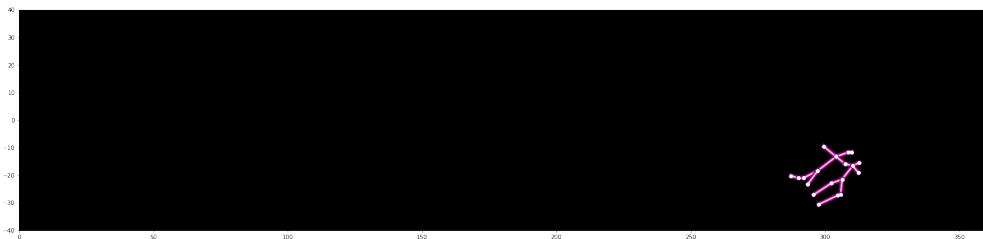
### Example 1:

```
fig = plt.figure(figsize=(30,7))
plt.xlim(0,360)
plt.ylim(-40,40)
plot_stars('Gemini', 'F', stars_db, new_center=None) # no translation
```



### Example 2:

```
fig = plt.figure(figsize=(30,7))
plt.xlim(0,360)
plt.ylim(-40,40)
plot_stars('Gemini', 'F', stars_db, new_center=(300, -20)) # centered in 300, -20
```



## parse\_zodiac

You are given a file `zodiac.csv`. For each sign, the table contains astrological information and affinity with other signs, expressed as a relation matrix:

```
[8]: import pandas as pd
df = pd.read_csv('zodiac.csv', encoding='UTF-8')
df[:4]
```

	Constellation	House	Glyph	Symbol	Dates	Element	\
0	Aries	1	♈	Ram	21 March\n-\n20 April	Fire	
1	Taurus	2	♉	Bull	21 April\n-\n21 May	Earth	
2	Gemini	3	♊	Twins	22 May\n-\n21 June	Air	

(continues on next page)

(continued from previous page)

3	Cancer	4	♋	Crab	22 June\n-\n21 July	Water														
0	Cardinal			Quality Ruling Planet Day/Night	Aries	...	Gemini	Cancer	Leo	Virgo	\									
1	Fixed			Mars Day	NaN	...	4.0	NaN	5.0	NaN										
2	Mutable			Venus Night	NaN	...	NaN	4.0	NaN	4.0	NaN									
3	Cardinal			Mercury Day	4.0	...	NaN	NaN	4.0	NaN										
				Moon Night	NaN	...	NaN	NaN	NaN	NaN										
	Libra	Scorpius	Sagittarius	Capricornus	Aquarius	Pisces														
0	NaN	NaN		5.0	NaN		4.0	NaN												
1	NaN	NaN		NaN	5.0		NaN	4.0												
2	5.0	NaN		NaN	NaN		5.0	NaN												
3	NaN	5.0		NaN	NaN		NaN	NaN												

[4 rows x 21 columns]

Parse the table so to get a dictionary of dictionaries, with some selected data:

- affinities are in the scale 1-5, normalize them to floats 0.0-1.0
- dates contain \n , normalize them so to have dates separated by a dash as in 21 March-20 April

**NOTE:** To parse the file, a `csv.reader` is sufficient, it's not necessary to use pandas - even if data seem to span multiple lines because of the \n in dates, note they are bounded by " so rows will be correctly parsed by `csv.reader`

You can find the complete output in `expected_zodiac_db.py`

```
{
 'Aquarius': {
 'affinities': {
 'Aries': 0.8,
 'Gemini': 1.0,
 'Libra': 1.0,
 'Sagittarius': 0.8
 },
 'dates': '21 January-18 February',
 'glyph': '♒',
 'house': 11
 },
 'Aries': {
 'affinities': {
 'Aquarius': 0.8,
 'Gemini': 0.8,
 'Leo': 1.0,
 'Sagittarius': 1.0
 },
 'dates': '21 March-20 April',
 'glyph': '♈',
 'house': 1
 },
 .
 .
 .
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-hide="Hide">>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[9]: import csv

def parse_zodiac(filename):

 with open(filename, encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter=',')
 header = next(my_reader)

 ret = {}
 for row in my_reader:
 workd = {}
 workd['glyph'] = row[2]
 workd['house'] = int(row[1])
 workd['dates'] = row[4].replace('\n', '')
 ret[row[0]] = workd
 workd['affinities'] = {}
 for j in range(9, len(row)):
 if row[j] != '':
 workd['affinities'][header[j]] = int(row[j])/5
 return ret

zodiac_db = parse_zodiac('zodiac.csv')

from pprint import pprint
#pprint(zodiac_db, width=100)
assert zodiac_db['Aries']['dates'] == '21 March-20 April'
assert zodiac_db['Aries']['affinities'] == {'Aquarius': 0.8, 'Gemini': 0.8, 'Leo': 1.
 ↪0, 'Sagittarius': 1.0}
assert zodiac_db['Aries']['glyph'] == '♈'
assert zodiac_db['Aries']['house'] == 1
assert zodiac_db['Gemini']['dates'] == '22 May-21 June'
assert zodiac_db['Gemini']['affinities'] == {'Aquarius': 1.0, 'Aries': 0.8, 'Leo': 0.
 ↪8, 'Libra': 1.0}
assert zodiac_db['Gemini']['glyph'] == '♊'
assert zodiac_db['Gemini']['house'] == 3
from expected_zodiac_db import expected_zodiac_db
assert zodiac_db == expected_zodiac_db
```

</div>

```
[9]: import csv

def parse_zodiac(filename):
 raise Exception('TODO IMPLEMENT ME !')

zodiac_db = parse_zodiac('zodiac.csv')

from pprint import pprint
#pprint(zodiac_db, width=100)
assert zodiac_db['Aries']['dates'] == '21 March-20 April'
assert zodiac_db['Aries']['affinities'] == {'Aquarius': 0.8, 'Gemini': 0.8, 'Leo': 1.
 ↪0, 'Sagittarius': 1.0}
assert zodiac_db['Aries']['glyph'] == '♈'
```

(continues on next page)

(continued from previous page)

```

assert zodiac_db['Aries']['house'] == 1
assert zodiac_db['Gemini']['dates'] == '22 May-21 June'
assert zodiac_db['Gemini']['affinities'] == {'Aquarius': 1.0, 'Aries': 0.8, 'Leo': 0.
 ↪8, 'Libra': 1.0}
assert zodiac_db['Gemini']['glyph'] == '♊'
assert zodiac_db['Gemini']['house'] == 3
from expected_zodiac_db import expected_zodiac_db
assert zodiac_db == expected_zodiac_db

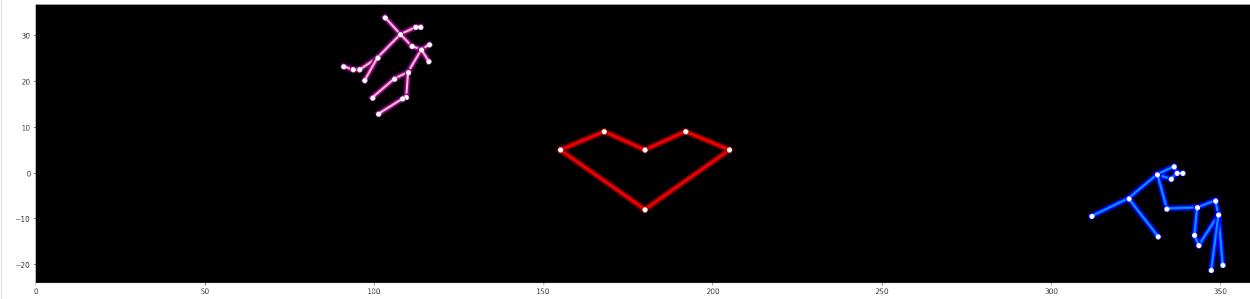
```

## plot\_love

In stars.csv we inserted the special (fake!) constellation of 'Love': given the importance, we placed it at the center of the galaxy, positioned at x=180 degrees and y=0. If you try to plot it now, you should get something like this:

```
[10]: # 'Aries', 'Taurus', 'Gemini', 'Cancer', 'Leo', 'Virgo',
'Libra', 'Scorpius', 'Sagittarius', 'Capricornus', 'Aquarius', 'Pisces'

fig = plt.figure(figsize=(30, 7))
plt.xlim(0, 360)
plot_stars('Gemini', 'F', stars_db)
plot_stars('Aquarius', 'M', stars_db)
plot_stars('Love', 'R', stars_db) # fake!
```



Given two astrological signs, place them on the same y=0 axis as the heart and make them symmetrically closer or farther from it according to their astrological affinity, also displaying their name and astrological glyph:

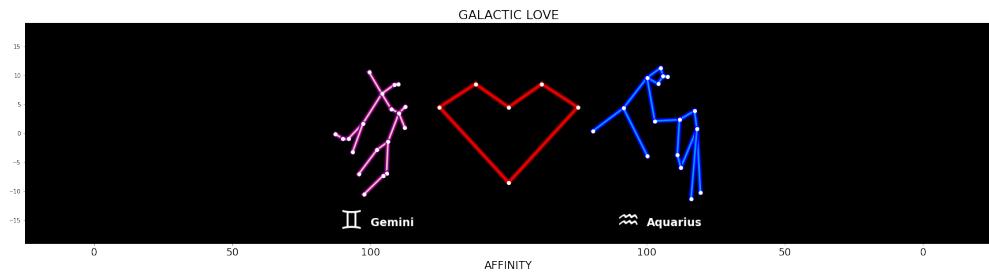
- **REMEMBER** title and xlabel !
- you can reuse previously defined `plot_stars` function
- constellations x centers should go from 50 to 150 degrees (and symmetrically, from -50 to -150)
- **BUT you will have to display reversed ticks:** 100 50 0 for positive (and symmetrically 0 50 100 for negative)

For drawing text:

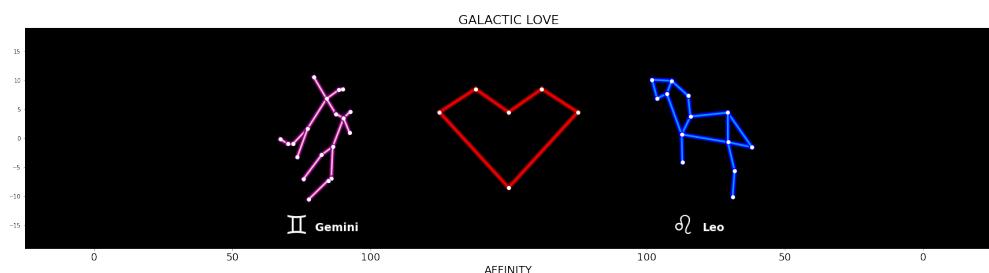
- For increasing text size in `title`, `xticks`, `xlabel`, `text` calls, you can use `fontsize=20` parameter (for glyphs you will need a bigger number)
- for text inside the chart use `plt.text(x, y, "some text")`
- the glyph must be drawn bigger than the sign name, so you will need a separate call to `plt.text`

### Examples:

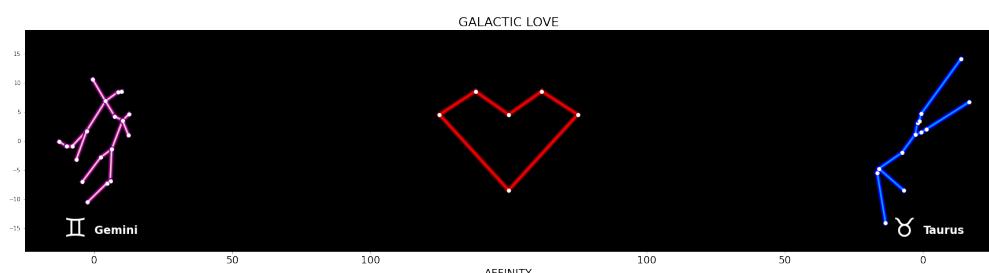
```
>>> plot_love('Gemini', 'Aquarius', stars_db, zodiac_db) # 1.0 affinity
```



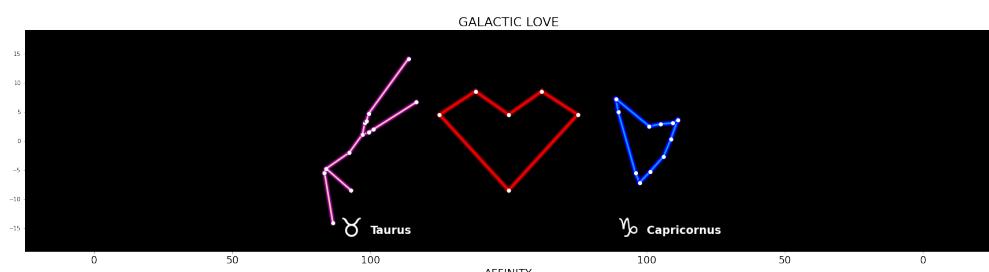
```
>>> plot_love('Gemini','Leo', stars_db, zodiac_db) # 0.8 affinity
```



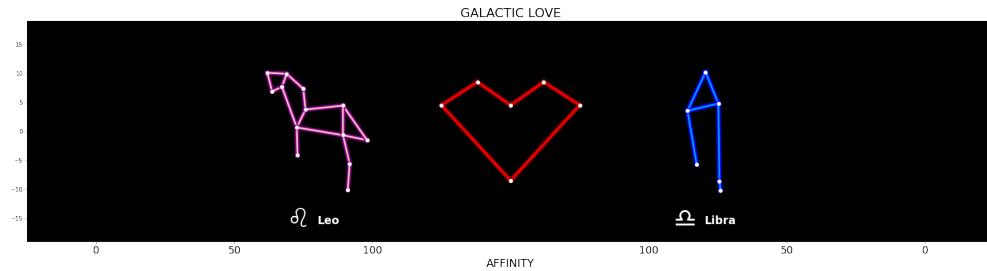
```
>>> plot_love('Gemini','Taurus', stars_db, zodiac_db) # 0.0 affinity
```



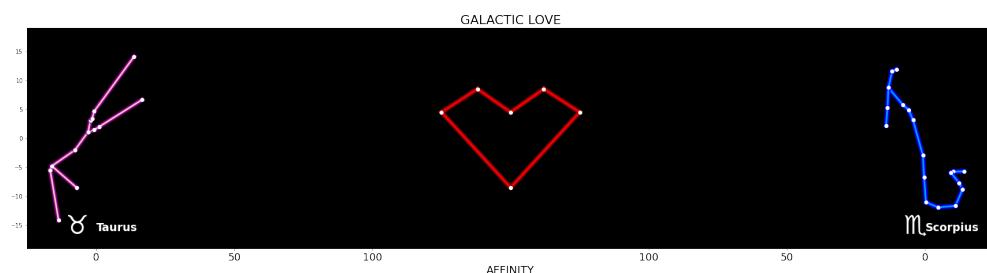
```
>>> plot_love('Taurus','Capricornus', stars_db, zodiac_db) # 1.0 affinity
```



```
>>> plot_love('Leo','Libra', stars_db, zodiac_db) # 0.8 affinity
```



```
>>> plot_love('Taurus', 'Scorpius', stars_db, zodiac_db) # 0.0 affinity
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[11]:

```
def plot_love(f_sign, m_sign, stars, zodiac):

 fig = plt.figure(figsize=(30,7)) # 30 inches large by 7 high
 plt.xlim(-175,175)

 if m_sign in zodiac[f_sign]['affinities']:
 coeff = zodiac[f_sign]['affinities'][m_sign]
 else:
 coeff = 0.0

 plt.title('GALACTIC LOVE', fontsize=22)
 xs = np.array([50,100,150])
 plt.xlabel('AFFINITY', fontsize=19)
 plt.xticks(np.hstack((-xs,xs)), np.hstack((150-np.abs(xs), 150-np.abs(xs))),
fontsize=18)
 plt.ylim(-19,19)
 plot_stars('Love', 'R', stars, new_center=(0,0))
 prox = (1.0 - coeff)*100+25+25
 plot_stars(m_sign, 'M', stars, new_center=(prox,0))
 plot_stars(f_sign, 'F', stars, new_center=(-prox,0))

 plt.text(+prox,-16, m_sign, fontsize=19, fontweight='bold', color='white')
 plt.text(+prox-11,-16.5, zodiac[m_sign]['glyph'], fontsize=45, fontweight='bold',
 color='white')

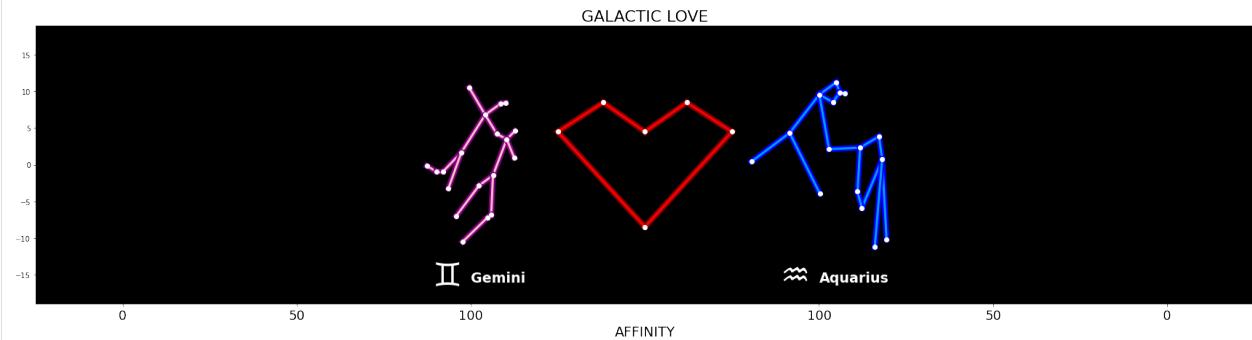
 plt.text(-prox,-16, f_sign, fontsize=19, fontweight='bold', color='white')
```

(continues on next page)

(continued from previous page)

```
plt.text(-prox-11,-16.5, zodiac[f_sign]['glyph'], fontsize=45, fontweight='bold',
 color='white')
```

```
plot_love('Gemini','Aquarius', stars_db, zodiac_db) # 1.0 affinity
```



&lt;/div&gt;

[11]:

```
def plot_love(f_sign, m_sign, stars, zodiac):

 fig = plt.figure(figsize=(30,7)) # 30 inches large by 7 high
 plt.xlim(-175,175)

 raise Exception('TODO IMPLEMENT ME !')

plot_love('Gemini','Aquarius', stars_db, zodiac_db) # 1.0 affinity
```

[12]:

```
plot_love('Gemini','Leo', stars_db, zodiac_db) # 0.8 affinity
```

[13]:

```
plot_love('Gemini','Taurus', stars_db, zodiac_db) # 0.0 affinity
```

[14]:

```
plot_love('Taurus','Capricornus', stars_db, zodiac_db) # 1.0 affinity
```

[15]:

```
plot_love('Leo','Libra', stars_db, zodiac_db) # 0.8 affinity
```

[16]:

```
plot_love('Taurus','Scorpius', stars_db, zodiac_db) # 0.0 affinity
```

[ ]:

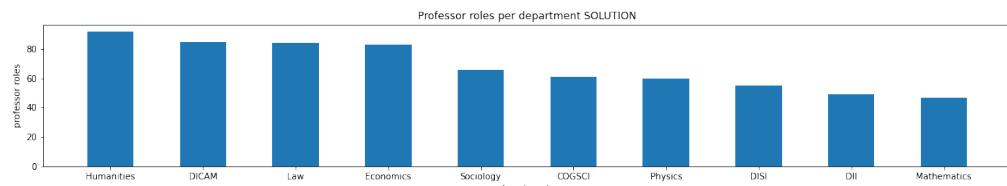
## 10.2.8 University staff

### Download worked project

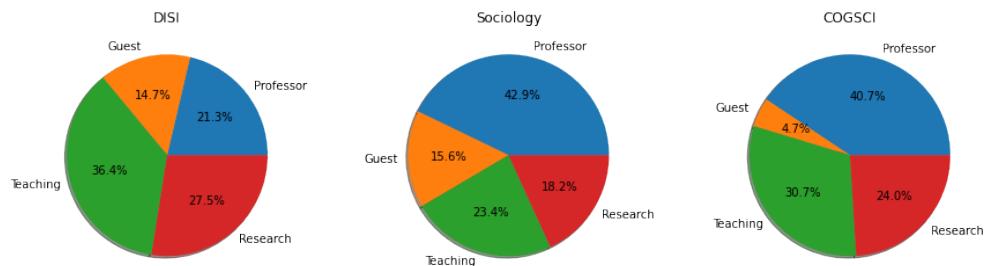
Browse files online<sup>465</sup>

Given the dataset of University of Trento staff (modified so not to contain names or surnames), we want to display:

- how many professors there are in each department:



- given some department, we want to show the roles of its employees as percentages:



Data source: University of Trento<sup>466</sup>, released under Creative Commons Attribution 4.0<sup>467</sup> licence.

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
university-staff-prj
university-staff.ipynb
university-staff-sol.ipynb
2019-06-30-personale-en-striped.json
jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook university-staff.ipynb
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter

<sup>465</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/university-staff>

<sup>466</sup> <https://dati.trentino.it/dataset/personale-academico-e-tecnico-amministrativo-dell-universita-di-trento>

<sup>467</sup> <http://creativecommons.org/licenses/by/4.0/deed.it>

- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### load\_data

A function `load_data` is given to load the dataset `2019-06-30-persone-en-stripped.json` (you don't need to implement it):

```
[1]: import json

def load_data():
 with open('2019-06-30-persone-en-stripped.json', encoding='utf-8') as json_file:
 data = json.load(json_file)
 return data

unitn = load_data()
```

**IMPORTANT:** look at the dataset !

Here we show only first 2 rows, but to get a clear picture of the dataset you should explore it further.

The dataset contains a list of employees, each of whom may have one or more positions, in one or more university units. Each unit is identified by a code like STO0000435:

```
[2]: unitn[:2]

[2]: [{'givenName': 'NAME-1',
 'phone': ['0461 283752'],
 'identifier': 'eb9139509dc40d199b6864399b7e805c',
 'familyName': 'SURNAME-1',
 'positions': [{ 'unitIdentifier': 'STO0008929',
 'role': 'Staff',
 'unitName': 'Student Support Service: Economics, Law and International Studies' }] },
 ↪ { 'givenName': 'NAME-2',
 'phone': ['0461 281521'],
 'identifier': 'b6292ffe77167b31e856d2984544e45b',
 'familyName': 'SURNAME-2',
 'positions': [{ 'unitIdentifier': 'STO0000435',
 'role': 'Associate professor',
 'unitName': 'Doctoral programme - Physics' },
 { 'unitIdentifier': 'STO0000435',
 'role': 'Deputy coordinator',
 'unitName': 'Doctoral programme - Physics' },
 { 'unitIdentifier': 'STO0008627',
 'role': 'Associate professor',
 'unitName': 'Department of Physics' }] }]
```

Department names can be very long, so when you need to display them you can use the function `this.abbreviate`.

**NOTE:** function is already fully implemented, **do not** modify it.

```
[3]: def abbreviate(unitName):

 abbreviations = {

 "Department of Psychology and Cognitive Science": "COGSCI",
 "Center for Mind/Brain Sciences - CIMeC": "CIMeC",
 "Department of Civil, Environmental and Mechanical Engineering": "DICAM",
 "Centre Agriculture Food Environment - C3A": "C3A",
 "School of International Studies - SIS": "SIS",
 "Department of Sociology and social research": "Sociology",
 "Faculty of Law": "Law",
 "Department of Economics and Management": "Economics",
 "Department of Information Engineering and Computer Science": "DISI",
 "Department of Cellular, Computational and Integrative Biology - CIBIO": "CIBIO"
 },
 "Department of Industrial Engineering": "DII"
 }
 if unitName in abbreviations:
 return abbreviations[unitName]
 else:
 return unitName.replace("Department of ", "")
```

**Example:**

```
[4]: abbreviate("Department of Information Engineering and Computer Science")
[4]: 'DISI'
```

**1. calc\_uid\_to\_abbr**

⊕ It will be useful having a map from department ids to their abbreviations, if they are actually present, otherwise to their original name. To implement this, you can use the previously defined function abbreviate.

```
{
.
.
.
'STO0008629': 'DISI',
'STO0008630': 'Sociology',
'STO0008631': 'COGSCI',
.
.
.
'STO0012897': 'Institutional Relations and Strategic Documents',
.
.
.
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this); data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[5]: def calc_uid_to_abbr(db):

 ret = {}
 for person in db:
 for position in person['positions']:
```

(continues on next page)

(continued from previous page)

```
 uid = position['unitIdentifier']
 ret[uid] = abbreviate(position['unitName'])
 return ret

#calc_uid_to_abbr(unitn)
print(calc_uid_to_abbr(unitn) ['ST00008629']) # DISI
print(calc_uid_to_abbr(unitn) ['ST00012897']) # Institutional Relations and Strategic
 ↪Documents
```

DISI  
Institutional Relations and Strategic Documents

&lt;/div&gt;

```
[5]:
```

```
def calc_uid_to_abbr(db):
 raise Exception('TODO IMPLEMENT ME !')

#calc_uid_to_abbr(unitn)
print(calc_uid_to_abbr(unitn) ['ST00008629']) # DISI
print(calc_uid_to_abbr(unitn) ['ST00012897']) # Institutional Relations and Strategic
 ↪Documents
```

## 2.1 calc\_prof\_roles

⊗⊗ For each department, we want to see how many professor roles are covered, sorting them from greatest to lowest. In returned list we will only put the 10 department with most roles.

- **NOTE 1:** we are interested in *roles* covered. Don't care if actual people might be less (one person can cover more professor roles within the same unit)
- **NOTE 2:** there are several professor roles. Please avoid listing all roles in the code ("Senior Professor", "Visiting Professor", ....), and prefer using some smarter way to match them.

**Expected result:**

```
>>> calc_prof_roles(unitn)
[('Humanities', 92),
 ('DICAM', 85),
 ('Law', 84),
 ('Economics', 83),
 ('Sociology', 66),
 ('COGSCI', 61),
 ('Physics', 60),
 ('DISI', 55),
 ('DII', 49),
 ('Mathematics', 47)]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[6]:
```

```
def calc_prof_roles(db):
```

(continues on next page)

(continued from previous page)

```

hist = {}
uid_to_abbr = calc_uid_to_abbr(db)

for person in db:
 for position in person['positions']:

 role = position['role']
 uid = position['unitIdentifier']
 if 'professor'.lower() in role.lower():
 if uid in hist:
 hist[uid] += 1
 else:
 hist[uid] = 1

ret = [(uid_to_abbr[x[0]], x[1]) for x in hist.items()]
ret.sort(key=lambda c: c[1], reverse=True)
return ret[:10]

```

```
calc_prof_roles(unitn)
```

```
[6]: [('Humanities', 92),
 ('DICAM', 85),
 ('Law', 84),
 ('Economics', 83),
 ('Sociology', 66),
 ('COGSCI', 61),
 ('Physics', 60),
 ('DISI', 55),
 ('DII', 49),
 ('Mathematics', 47)]
```

</div>

```
[6]:
```

```

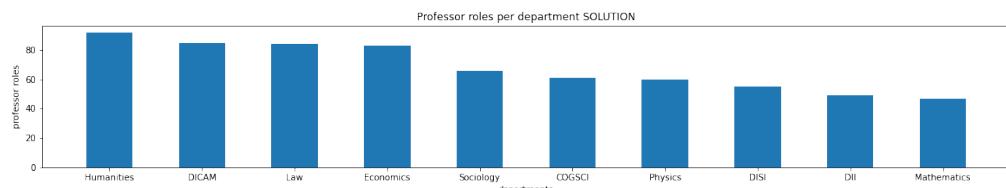
def calc_prof_roles(db):
 raise Exception('TODO IMPLEMENT ME !')

calc_prof_roles(unitn)

```

## 2.2 plot\_profs

⊕ Write a function to plot a bar chart of data calculated above



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[7]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_profs(db):

 prof_roles = calc_prof_roles(db)

 xs = list(range(len(prof_roles)))
 xticks = [p[0] for p in prof_roles]
 ys = [p[1] for p in prof_roles]

 fig = plt.figure(figsize=(20,3))

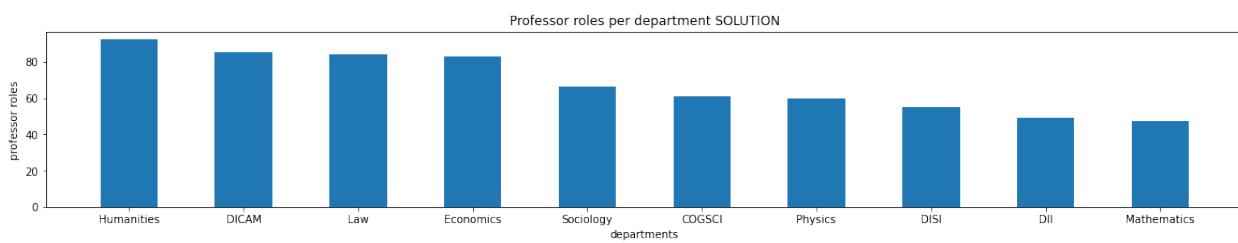
 plt.bar(xs, ys, 0.5, align='center')

 plt.title("Professor roles per department SOLUTION")
 plt.xticks(xs, xticks)

 plt.xlabel('departments')
 plt.ylabel('professor roles')

 plt.show()
```

```
plot_profs(unitn)
```



```
</div>
```

[7]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_profs(db):
 raise Exception('TODO IMPLEMENT ME !')

plot_profs(unitn)
```

### 3.1 calc\_roles

⊕⊕ We want to calculate how many roles are covered for each department.

You will group roles by these macro groups (some already exist, some are new):

- Professor : “Senior Professor”, “Visiting Professor”, ...
- Research : “Senior researcher”, “Research collaborator”, ...
- Teaching : “Teaching assistant”, “Teaching fellow”, ...
- Guest : “Guest”, ...

and discard all the others (there are many, like “Rector”, “Head”, etc ..)

**NOTE:** Please avoid listing all roles in the code (“Senior researcher”, “Research collaborator”, ...), and prefer using some smarter way to match them.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8]:

```
def calc_roles(db):

 ret = {}
 for person in db:
 for position in person['positions']:
 uid = position['unitIdentifier']
 role = position['role']
 grouped_role = None
 if "professor" in role.lower():
 grouped_role = 'Professor'
 elif "research" in role.lower():
 grouped_role = 'Research'
 elif "teaching" in role.lower():
 grouped_role = 'Teaching'
 elif "guest" in role.lower():
 grouped_role = 'Guest'

 if grouped_role:
 if uid in ret:
 if grouped_role in ret[uid]:
 ret[uid][grouped_role] += 1
 else:
 ret[uid][grouped_role] = 1
 else:
 diz = {}
 diz[grouped_role] = 1
 ret[uid] = diz

 return ret

print('ST00000001:', calc_roles(unitn) ['ST00000001'])
print('ST00000006:', calc_roles(unitn) ['ST00000006'])
print('ST00000012:', calc_roles(unitn) ['ST00000012'])
print('ST00008629:', calc_roles(unitn) ['ST00008629'])

ST00000001: {'Teaching': 9, 'Research': 3, 'Professor': 12}
ST00000006: {'Professor': 1}
```

(continues on next page)

(continued from previous page)

```
STO0000012: {'Guest': 3}
STO0008629: {'Professor': 55, 'Teaching': 94, 'Research': 71, 'Guest': 38}
```

</div>

[8]:

```
def calc_roles(db):
 raise Exception('TODO IMPLEMENT ME !')

print('STO0000001:', calc_roles(unitn)['STO0000001'])
print('STO0000006:', calc_roles(unitn)['STO0000006'])
print('STO0000012:', calc_roles(unitn)['STO0000012'])
print('STO0008629:', calc_roles(unitn)['STO0008629'])
```

EXPECTED RESULT - Showing just first ones ...

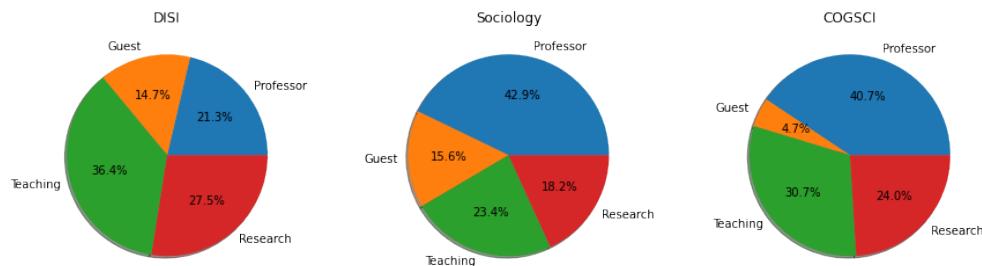
```
>>> calc_roles(unitn)

{
 'STO0000001': {'Teaching': 9, 'Research': 3, 'Professor': 12},
 'STO0000006': {'Professor': 1},
 'STO0000012': {'Guest': 3},
 'STO0008629': {'Teaching': 94, 'Research': 71, 'Professor': 55, 'Guest': 38}
}
```

### 3.2 plot\_roles

⊕⊕ Implement a function `plot_roles` that given, the abbreviations (or long names) of some departments, plots pie charts of their grouped role distribution, all in one row.

- **NOTE 1:** different plots MUST show equal groups with equal colors
- **NOTE 2:** always show all the 4 macro groups defined before, even if they have zero frequency
  - For on example on how to plot the pie charts, see this<sup>468</sup>
  - For on example on plotting side by side, see this<sup>469</sup>



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

<sup>468</sup> <https://en.softpython.org/visualization/visualization1-sol.html#Pie-chart>

<sup>469</sup> <https://en.softpython.org/visualization/visualization1-sol.html#Showing-plots-side-by-side>

[9]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_roles(db, abbrs):

 fig = plt.figure(figsize=(15,4))
 uid_to_abbr = calc_uid_to_abbr(db)

 for i in range(len(abbrs)):

 abbr = abbrs[i]
 roles = calc_roles(db)

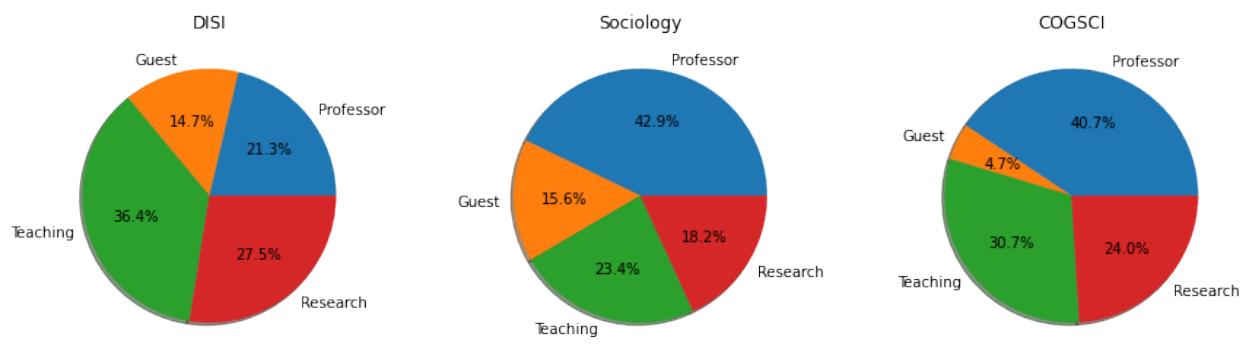
 uid = None

 for key in uid_to_abbr:
 if uid_to_abbr[key] == abbr:
 uid = key

 labels = ['Professor', 'Guest', 'Teaching', 'Research']
 fracs = []
 for role in labels:
 if role in roles[uid]:
 fracs.append(roles[uid][role])
 else:
 fracs.append(0)

 plt.subplot(1, len(abbrs), i+1) # rows
 # columns
 # plotting in first cell
 plt.pie(fracs, labels=labels, autopct='%.1f%%', shadow=True)
 plt.title(abbr)
```

```
plot_roles(unitn, ['DISI', 'Sociology', 'COGSCI'])
```



```
</div>
```

[9]:

```
%matplotlib inline
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt

def plot_roles(db, abbrs):
 raise Exception('TODO IMPLEMENT ME !')

plot_roles(unitn, ['DISI', 'Sociology', 'COGSCI'])
```

## 4.1 calc\_shared

⊕⊕⊕ We want to calculate the 10 *department pairs* that have the greatest number of people working in *both* departments (regardless of role), sorted in decreasing order.

For example, ‘CIMeC’ and ‘COGSCI’ have 23 people working in both departments, meaning each of these 23 people has at least a position at CIMeC and at least a position at COGSCI.

**NOTE:** in this case we are looking at number of actual people, *not* number of roles covered

- **DO NOT** consider Doctoral programmes
- **DO NOT** consider ‘University of Trento’ department (STO0000001)
- if your calculations display with swapped names ('COGSCI', 'CIMeC', 23) instead of ('CIMeC', 'COGSCI', 23) it's not important, as long as they display just once per pair.

**Expected result:**

```
>>> calc_shared(unitn)
[('COGSCI', 'CIMeC', 23),
 ('DICAM', 'C3A', 14),
 ('DISI', 'Economics', 7),
 ('SIS', 'Sociology', 7),
 ('SIS', 'Law', 6),
 ('Economics', 'Sociology', 5),
 ('SIS', 'Humanities', 5),
 ('Economics', 'Law', 4),
 ('DII', 'DISI', 4),
 ('CIBIO', 'C3A', 4)]
```

**HINT:** follow this sketch:

- build a dict which assigns unit codes to a set of *identifiers* of people that work for that unit
- to add elements to a set, use `.add` method
- to find common employees between two units, use `set.intersection` method (NOTE: it generates a *new* set)
- to check for all possible unit couples, you will need a double `for` on a list of departments. To avoid double checking pairs (so not have both ('CIMeC', 'COGSCI', 23) and ('COGSCI', 'CIMeC', 23) in output), you can think like you are visiting the lower of a matrix (for the sake of the example here we put only 4 departments with random numbers).

	0	1	2	3
0	DISI	--	--	--
1	COGSCI	313	--	--
2	CIMeC	231	23	--
3	DICAM	12	13	123

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[10]:

```
def calc_shared(db):

 ret = {}
 uid_to_people = {}

 uid_to_abbr = calc_uid_to_abbr(db)

 for person in db:

 for position in person['positions']:
 uid = position['unitIdentifier']
 if not uid in uid_to_people:
 uid_to_people[uid] = set()
 uid_to_people[uid].add(person['identifier'])

 uids = list(uid_to_people)

 ret = []
 for x in range(len(uids)):
 uidx = uids[x]
 for y in range(x):
 uidy = uids[y]
 num = len(uid_to_people[uidx].intersection(uid_to_people[uidy]))
 if (num > 0) \
 and ("Doctoral programme" not in uid_to_abbr[uidx]) \
 and ("Doctoral programme" not in uid_to_abbr[uidy]) \
 and (uidx != 'STO00000001') \
 and (uidy != 'STO00000001'):
 ret.append((uid_to_abbr[uidx], uid_to_abbr[uidy], num))

 ret.sort(key=lambda c: c[2], reverse=True)
 ret = ret[:10]
 return ret
```

calc\_shared(unitn)

[10]:

```
[('COGSCI', 'CIMeC', 23),
 ('DICAM', 'C3A', 14),
 ('DISI', 'Economics', 7),
 ('SIS', 'Sociology', 7),
 ('SIS', 'Law', 6),
 ('Economics', 'Sociology', 5),
 ('SIS', 'Humanities', 5),
 ('Economics', 'Law', 4),
 ('DII', 'DISI', 4),
 ('CIBIO', 'C3A', 4)]
```

</div>

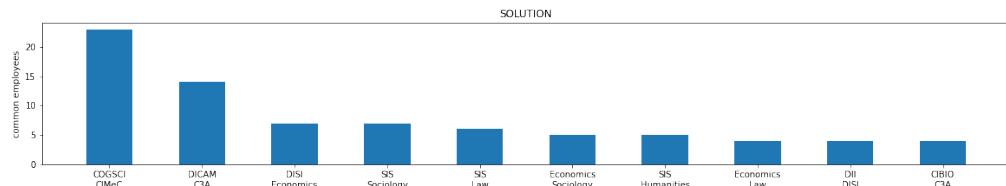
[10]:

```
def calc_shared(db):
 raise Exception('TODO IMPLEMENT ME !')

calc_shared(unitn)
```

### 4.2 plot\_shared

⊕ Plot the above in a bar chart, where on the x axis there are the department pairs and on the y the number of people in common.



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[11]:

```
import matplotlib.pyplot as plt

%matplotlib inline

def plot_shared(db):

 uid_to_abbr = calc_uid_to_abbr(db)

 shared = calc_shared(db)
 xs = range(len(shared))

 xticks = [x[0] + "\n" + x[1] for x in shared]

 ys = [x[2] for x in shared]

 fig = plt.figure(figsize=(20,3))

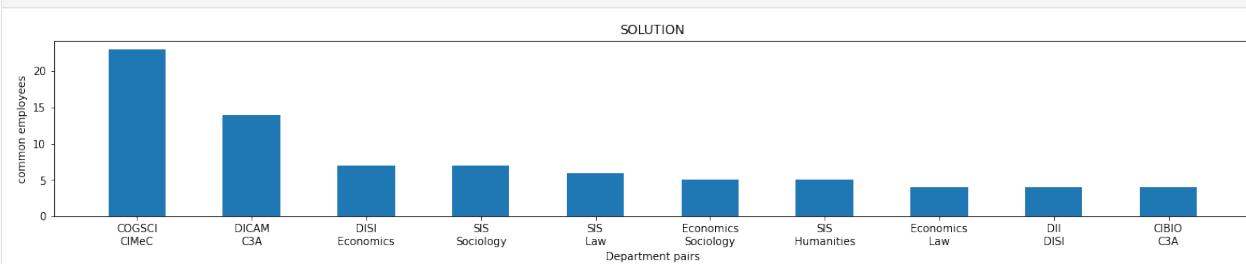
 plt.bar(xs, ys, 0.5, align='center')

 plt.title("SOLUTION")
 plt.xticks(xs, xticks)

 plt.xlabel('Department pairs')
 plt.ylabel('common employees')

 plt.show()

plot_shared(unitn)
```



&lt;/div&gt;

```
[11]: import matplotlib.pyplot as plt
%matplotlib inline

def plot_shared(db):
 raise Exception('TODO IMPLEMENT ME !')

plot_shared(unitn)
```

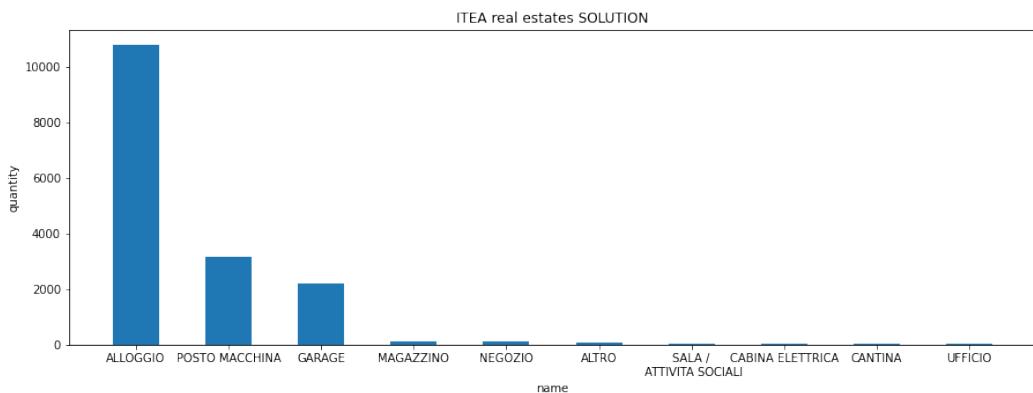
[ ]:

## 10.2.9 ITEA Real Estate

### Download worked project

Browse files online<sup>470</sup>

You will now analyze public real estates which are managed by ITEA agency in Trentino region, Italy. Every real estate has a type, and we will analyze the type distribution.



Data source: ITEA - dati.trentino.it<sup>471</sup>, released under Creative Commons Attribution 4.0<sup>472</sup> license.

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
itea-real-estate-prj
itea-real-estate.ipynb
itea-real-estate-sol.ipynb
itea.csv
jupman.py
```

<sup>470</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/itea-real-estate>

<sup>471</sup> <https://dati.trentino.it/dataset/patrimonio-immobiliare>

<sup>472</sup> <http://creativecommons.org/licenses/by/4.0/deed.it>

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `itea-real-estate.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### load\_itea

A function `load_itea` is given to load the dataset `itea.csv` (you don't need to implement it):

```
[1]: import csv

def load_itea():
 """Loads file data and RETURN a list of dictionaries with the stop times
 """
 with open('itea.csv', newline='', encoding='latin-1') as csvfile:
 reader = csv.DictReader(csvfile, delimiter=';')
 lst = []
 for d in reader:
 lst.append(d)
 return lst

itea = load_itea()
```

**IMPORTANT:** look at the dataset by yourself !

Here we show only first 5 rows, but to get a clear picture of the dataset you need to study it a bit by yourself

```
[2]: itea[:5]

[2]: [OrderedDict([('Tipologia', 'ALTRO'),
 ('Proprietà', 'ITEA'),
 ('Indirizzo', "Codice unita': 30100049"),
 ('Frazione', ''),
 ('Comune', "BASELGA DI PINE")]),
 OrderedDict([('Tipologia', 'ALLOGGIO'),
 ('Proprietà', 'ITEA'),
 ('Indirizzo', "Codice unita': 43100011"),
 ('Frazione', ''),
 ('Comune', 'TRENTO')]),
 OrderedDict([('Tipologia', 'ALLOGGIO'),
 ('Proprietà', 'ITEA'),
 ('Indirizzo', "Codice unita': 43100002"),
 ('Frazione', ''),
 ('Comune', 'TRENTO'))],
```

(continues on next page)

(continued from previous page)

```
OrderedDict([('Tipologia', 'ALLOGGIO'),
 ('Proprietà', 'ITEA'),
 ('Indirizzo', 'VIALE DELLE ROBINIE 26'),
 ('Frazione', ''),
 ('Comune', 'TRENTO')]),
OrderedDict([('Tipologia', 'ALLOGGIO'),
 ('Proprietà', 'ITEA'),
 ('Indirizzo', 'VIALE DELLE ROBINIE 26'),
 ('Frazione', ''),
 ('Comune', 'TRENTO')])]
```

## calc\_types\_hist

Implement function `calc_types_hist` to extract the types ('Tipologia') of ITEA real estate and RETURN a histogram which associates to each type its frequency.

- You will discover there are three types of apartments: 'ALLOGGIO', 'ALLOGGIO DUPLEX' and 'ALLOGGIO MONOLOCALE'. In the resulting histogram you must place only the key 'ALLOGGIO' which will be the sum of all of them.
- Same goes for 'POSTO MACCHINA' (parking lot): there are many of them ('POSTO MACCHINA COMUNE ESTERNO', 'POSTO MACCHINA COMUNE INTERNO', 'POSTO MACCHINA ESTERNO', 'POSTO MACCHINA INTERNO', 'POSTO MACCHINA SOTTO TETTOIA') but we only want to see 'POSTO MACCHINA' as key with the sum of all of them.
- **DO NOT** use 5 ifs, try to come up with some generic code to catch all these cases ..

Expected output:

```
>>> calc_types_hist(itea)
{'ALTRO': 64,
 'ALLOGGIO': 10778,
 'POSTO MACCHINA': 3147,
 'MAGAZZINO': 143,
 'CABINA ELETTRICA': 41,
 'LOCALE COMUNE': 28,
 'NEGOZIO': 139,
 'CANTINA': 40,
 'GARAGE': 2221,
 'CENTRALE TERMICA': 4,
 'UFFICIO': 29,
 'TETTOIA': 2,
 'ARCHIVIO ITEA': 10,
 'SALA / ATTIVITA SOCIALI': 45,
 'AREA URBANA': 6,
 'ASILO': 1,
 'CASERMA': 2,
 'LABORATORIO PER ARTI E MESTIERI': 3,
 'MUSEO': 1,
 'SOFFITTA': 3,
 'AMBULATORIO': 1,
 'LEGNAIA': 3,
 'RUDERE': 1}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```
def calc_types_hist(db):

 tipologie = {}
 for diz in db:
 if diz['Tipologia'].startswith('ALLOGGIO'):
 chiave = 'ALLOGGIO'
 elif diz['Tipologia'].startswith('POSTO MACCHINA'):
 chiave = 'POSTO MACCHINA'
 else:
 chiave = diz['Tipologia']

 if chiave in tipologie:
 tipologie[chiave] += 1
 else:
 tipologie[chiave] = 1

 return tipologie

calc_types_hist(itea)
```

[3]:

```
{'ALTRO': 64,
 'ALLOGGIO': 10778,
 'POSTO MACCHINA': 3147,
 'MAGAZZINO': 143,
 'CABINA ELETTRICA': 41,
 'LOCALE COMUNE': 28,
 'NEGOZIO': 139,
 'CANTINA': 40,
 'GARAGE': 2221,
 'CENTRALE TERMICA': 4,
 'UFFICIO': 29,
 'TETTOIA': 2,
 'ARCHIVIO ITEA': 10,
 'SALA / ATTIVITA SOCIALI': 45,
 'AREA URBANA': 6,
 'ASILO': 1,
 'CASERMA': 2,
 'LABORATORIO PER ARTI E MESTIERI': 3,
 'MUSEO': 1,
 'SOFFITTA': 3,
 'AMBULATORIO': 1,
 'LEGNAIA': 3,
 'RUDERE': 1}
```

```
</div>
```

[3]:

```
def calc_types_hist(db):
 raise Exception('TODO IMPLEMENT ME !')

calc_types_hist(itea)
```

## calc\_types\_series

Implement a function to take a dictionary histogram and RETURN a list of tuples containing key/value pairs, sorted from most frequent to least frequent items.

**HINT:** if you don't remember how to sort by an element of a tuple, look at [this example<sup>473</sup>](#) in python documentation.

Expected output:

```
>>> calc_types_series(calc_types_hist(itea))
[('ALLOGGIO', 10778),
 ('POSTO MACCHINA', 3147),
 ('GARAGE', 2221),
 ('MAGAZZINO', 143),
 ('NEGOZIO', 139),
 ('ALTRO', 64),
 ('SALA / ATTIVITA SOCIALI', 45),
 ('CABINA ELETTRICA', 41),
 ('CANTINA', 40),
 ('UFFICIO', 29)]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4]:

```
def calc_types_series(hist):
 ret = []
 for key in hist:
 ret.append((key, hist[key]))
 ret.sort(key=lambda c: c[1], reverse=True)
 return ret[:10]
```

```
types = calc_types_series(calc_types_hist(itea))
```

```
types
```

[4]:

```
[('ALLOGGIO', 10778),
 ('POSTO MACCHINA', 3147),
 ('GARAGE', 2221),
 ('MAGAZZINO', 143),
 ('NEGOZIO', 139),
 ('ALTRO', 64),
 ('SALA / ATTIVITA SOCIALI', 45),
 ('CABINA ELETTRICA', 41),
 ('CANTINA', 40),
 ('UFFICIO', 29)]
```

```
</div>
```

[4]:

```
def calc_types_series(hist):
 raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

<sup>473</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>

(continued from previous page)

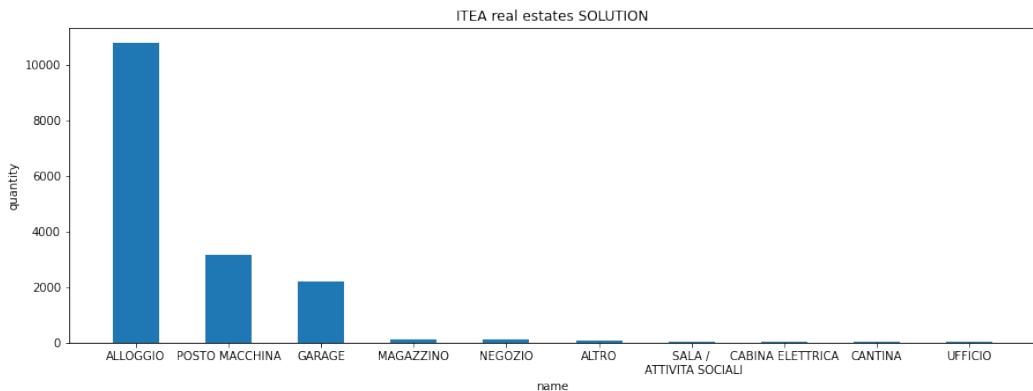
```
types = calc_types_series(calc_types_hist(itea))

types
```

### Real estates plot

Once you obtained the series as above, plot the first 10 most frequent items, in decreasing order.

- pay attention to plot title, width and height, axis labels. Everything MUST display in a readable way.
- try also to print nice the labels, if they are too long / overlap like for 'SALA / ATTIVITA SOCIALI' put carriage returns in a generic way.



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

write here

xs = np.arange(len(types))

xs_labels = [t[0].replace('/', '/\n') for t in types]

ys = [t[1] for t in types]

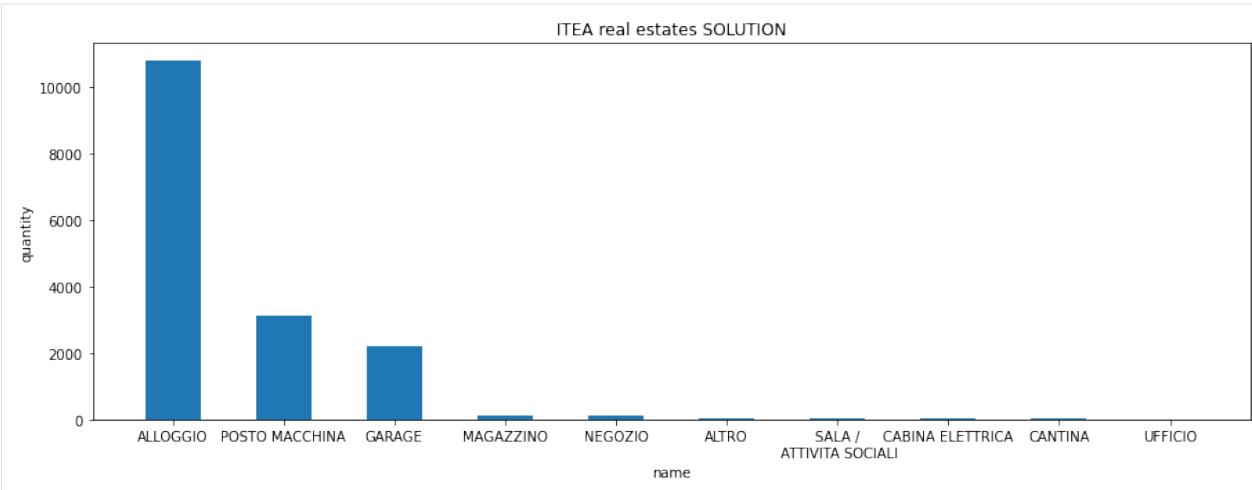
fig = plt.figure(figsize=(15,5))

plt.bar(xs, ys, 0.5, align='center')

plt.title("ITEA real estates SOLUTION")
plt.xticks(xs, xs_labels)

plt.xlabel('name')
plt.ylabel('quantity')

plt.show()
```



&lt;/div&gt;

[5] :

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

write here
```

## 10.2.10 Price catalog

### Download worked project

Browse files online<sup>474</sup>

Public administrations always need to know the up-to-date prices of all the various items they need, so to place appropriate orders to contractors. We'll analyze the dataset [EPPAT-2018-new-compact.csv](#), which is the price list for all products and services the Autonomous Province of Trento (Italy) may require.

Data source: [dati.trentino.it](#)<sup>475</sup>, released under Creative Commons Attribution 4.0<sup>476</sup> licence.

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
price-catalog-prj
 price-catalog.ipynb
 price-catalog-sol.ipynb
 EPPAT-2018-new-compact.csv
 jupman.py
```

<sup>474</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/price-catalog>

<sup>475</sup> <https://dati.trentino.it/dataset/prezzario-dei-lavori-pubblici-della-provincia-autonoma-di-trento>

<sup>476</sup> <http://creativecommons.org/licenses/by/4.0/deed.it>

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `price-catalog.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### The dataset

Let's have a quick look at the file `EPPAT-2018-new-compact.csv`. We will show examples with pandas, but it is not required to solve the exercises.

**DO NOT WASTE TIME LOOKING AT THE WHOLE DATASET!**

The dataset is quite complex, please focus on the few examples we provide

```
[1]: import pandas as pd
import numpy as np

pd.set_option('display.max_colwidth', None)
df = pd.read_csv('EPPAT-2018-new-compact.csv', encoding='latin-1')
```

The dataset contains several columns, but we will consider the following ones:

```
[2]: df=df[['Codice Prodotto','Descrizione Breve Prodotto','Categoria','Prezzo']]
df[:22]
```

	Codice Prodotto	Descrizione Breve Prodotto \
0	A.02.35.0050	ATTREZZATURA PER INFISSIONE PALI PILOTI
1	A.02.35.0050.010	Attrezzatura per infissione pali piloti.
2	A.02.40	ATTREZZATURE SPECIALI
3	A.02.40.0010	POMPA COMPLETA DI MOTORE
4	A.02.40.0010.010	fino a mm 50.
5	A.02.40.0010.020	oltre mm 50 fino a mm 100.
6	A.02.40.0010.030	oltre mm 100 fino a mm 150.
7	A.02.40.0010.040	oltre mm 150 fino a mm 200.
8	A.02.40.0010.050	oltre mm 200.
9	A.02.40.0020	GRUPPO ELETTRICO
10	A.02.40.0020.010	fino a 10 KW
11	A.02.40.0020.020	oltre 10 fino a 13 KW
12	A.02.40.0020.030	oltre 13 fino a 20 KW
13	A.02.40.0020.040	oltre 20 fino a 28 KW
14	A.02.40.0020.050	oltre 28 fino a 36 KW
15	A.02.40.0020.060	oltre 36 fino a 56 KW
16	A.02.40.0020.070	oltre 56 fino a 80 KW
17	A.02.40.0020.080	oltre 80 fino a 100 KW

(continues on next page)

(continued from previous page)

18	A.02.40.0020.090		oltre 100 fino a 120 KW
19	A.02.40.0020.100		oltre 120 fino a 156 KW
20	A.02.40.0020.110		oltre 156 fino a 184 KW
21	A.02.40.0030	NASTRO TRASPORTATORE CON MOTORE AD ARIA COMPRESSA	
		Categoria	Prezzo
0		NaN	NaN
1	Noli e trasporti	109.09	
2		NaN	NaN
3		NaN	NaN
4	Noli e trasporti	2.21	
5	Noli e trasporti	3.36	
6	Noli e trasporti	4.42	
7	Noli e trasporti	5.63	
8	Noli e trasporti	6.84	
9		NaN	NaN
10	Noli e trasporti	8.77	
11	Noli e trasporti	9.94	
12	Noli e trasporti	14.66	
13	Noli e trasporti	15.62	
14	Noli e trasporti	16.40	
15	Noli e trasporti	28.53	
16	Noli e trasporti	44.06	
17	Noli e trasporti	50.86	
18	Noli e trasporti	55.88	
19	Noli e trasporti	80.47	
20	Noli e trasporti	94.00	
21		NaN	NaN

### Pompa completa a motore Example

If we look at the dataset, in some cases we can spot a pattern (rows 3 to 8 included):

[3]: df[3:12]

	Codice Prodotto	Descrizione Breve Prodotto	Categoria	Prezzo
3	A.02.40.0010	POMPA COMPLETA DI MOTORE	NaN	NaN
4	A.02.40.0010.010	fino a mm 50.	Noli e trasporti	2.21
5	A.02.40.0010.020	oltre mm 50 fino a mm 100.	Noli e trasporti	3.36
6	A.02.40.0010.030	oltre mm 100 fino a mm 150.	Noli e trasporti	4.42
7	A.02.40.0010.040	oltre mm 150 fino a mm 200.	Noli e trasporti	5.63
8	A.02.40.0010.050	oltre mm 200.	Noli e trasporti	6.84
9	A.02.40.0020	GRUPPO ELETTROGENO	NaN	NaN
10	A.02.40.0020.010	fino a 10 KW	Noli e trasporti	8.77
11	A.02.40.0020.020	oltre 10 fino a 13 KW	Noli e trasporti	9.94

We see the first column holds product codes. If two rows share a code prefix, they belong to the same product type. As an example, we can take product A.02.40.0010, which has 'POMPA COMPLETA A MOTORE' as description ('Descrizione Breve Prodotto' column). The first row is basically telling us the product type, while the following rows are specifying several products of the same type (notice they all share the A.02.40.0010 prefix code until 'GRUPPO ELETTROGENO' excluded). Each description specifies a range of values for that product: *fino a* means *until to*, and *oltre* means *beyond*.

Notice that:

- first row has only one number
- intermediate rows have two numbers

- last row of the product series (row 8) has only one number and contains the word *oltre* (*beyond*) (in some other cases, last row of product series may have two numbers)

### A1 extract\_bounds

Write a function that given a Descrizione Breve Prodotto **as a single string** extracts the range contained within as a tuple.

If the string contains only one number n:

- if it contains UNTIL ('fino') it is considered a first row with bounds (0, n)
- if it contains BEYOND ('oltre') it is considered a last row with bounds (n, math.inf)

**DO NOT** use constants like measure units 'mm', 'KW', etc in the code

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[4]: import math

#use this list to remove unneeded stuff
PUNCTUATION=[',', '-', '.', '%']
UNTIL = 'fino'
BEYOND = 'oltre'

def extract_bounds(text):

 fixed_text = text
 for pun in PUNCTUATION:
 fixed_text = fixed_text.replace(pun, ' ')
 words = fixed_text.split()
 i = 0
 left = None
 right = None

 while i < len(words) and (not left or not right):

 if words[i].isdigit():
 if not left:
 left = int(words[i])
 elif not right:
 right = int(words[i])
 i += 1

 if not right:
 if BEYOND in text:
 right = math.inf
 else:
 right = left
 left = 0

 return (left,right)

assert extract_bounds('fino a mm 50.') == (0,50)
assert extract_bounds('oltre mm 50 fino a mm 100.') == (50,100)
```

(continues on next page)

(continued from previous page)

```

assert extract_bounds('oltre mm 200.') == (200, math.inf)
assert extract_bounds('da diametro 63 mm a diametro 127 mm') == (63, 127)
assert extract_bounds('fino a 10 KW') == (0,10)
assert extract_bounds('oltre 156 fino a 184 KW') == (156,184)
assert extract_bounds('fino a 170 A, avviamento elettrico') == (0,170)
assert extract_bounds('oltre 170 A fino a 250 A, avviamento elettrico') == (170, 250)
assert extract_bounds('oltre 300 A, avviamento elettrico') == (300, math.inf)
assert extract_bounds('tetti piani o con bassa pendenza - fino al 10%') == (0,10)
assert extract_bounds('tetti a media pendenza - oltre al 10% e fino al 45%') == (10,
 ↪45)
assert extract_bounds('tetti ad alta pendenza - oltre al 45%') == (45, math.inf)

```

&lt;/div&gt;

[4]: import math

```

#use this list to remove unneeded stuff
PUNCTUATION=[' ',',','-','.','%']
UNTIL = 'fino'
BEYOND = 'oltre'

def extract_bounds(text):
 raise Exception('TODO IMPLEMENT ME !')

assert extract_bounds('fino a mm 50.') == (0,50)
assert extract_bounds('oltre mm 50 fino a mm 100.') == (50,100)
assert extract_bounds('oltre mm 200.') == (200, math.inf)
assert extract_bounds('da diametro 63 mm a diametro 127 mm') == (63, 127)
assert extract_bounds('fino a 10 KW') == (0,10)
assert extract_bounds('oltre 156 fino a 184 KW') == (156,184)
assert extract_bounds('fino a 170 A, avviamento elettrico') == (0,170)
assert extract_bounds('oltre 170 A fino a 250 A, avviamento elettrico') == (170, 250)
assert extract_bounds('oltre 300 A, avviamento elettrico') == (300, math.inf)
assert extract_bounds('tetti piani o con bassa pendenza - fino al 10%') == (0,10)
assert extract_bounds('tetti a media pendenza - oltre al 10% e fino al 45%') == (10,
 ↪45)
assert extract_bounds('tetti ad alta pendenza - oltre al 45%') == (45, math.inf)

```

## A2 extract\_product

Write a function that given a filename, a code and a unit, parses the csv until it finds the corresponding code and RETURNS **one** dictionary with relevant information for that product

- Prezzo ( price ) **MUST be converted to float**
- USE a csv.DictReader for parsing, see example<sup>477</sup>
- USE latin-1 as encoding

[5]: # Suppose we want to get all info about A.02.40.0010 prefix:  
df[3:12]

	Codice Prodotto	Descrizione Breve Prodotto	Categoria	Prezzo
3	A.02.40.0010	POMPA COMPLETA DI MOTORE	NaN	NaN
4	A.02.40.0010.010	fino a mm 50. Noli e trasporti	2.21	

(continues on next page)

<sup>477</sup> [https://en.softpython.org/format2-csv-sol.html#Reading-as-dictionaries](https://en.softpython.org/formats/format2-csv-sol.html#Reading-as-dictionaries)

(continued from previous page)

5	A.02.40.0010.020	oltre mm 50 fino a mm 100.	Noli e trasporti	3.36
6	A.02.40.0010.030	oltre mm 100 fino a mm 150.	Noli e trasporti	4.42
7	A.02.40.0010.040	oltre mm 150 fino a mm 200.	Noli e trasporti	5.63
8	A.02.40.0010.050	oltre mm 200.	Noli e trasporti	6.84
9	A.02.40.0020	GRUPPO ELETTROGENO	NaN	NaN
10	A.02.40.0020.010	fino a 10 KW	Noli e trasporti	8.77
11	A.02.40.0020.020	oltre 10 fino a 13 KW	Noli e trasporti	9.94

A call to

```
pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm'))
```

Must produce:

```
{'category': 'Noli e trasporti',
'code': 'A.02.40.0010',
'description': 'POMPA COMPLETA DI MOTORE',
'measure_unit': 'mm',
'models': [{`bounds`}: (0, 50), `price`: 2.21, `subcode`: '010'},
{`bounds`}: (50, 100), `price`: 3.36, `subcode`: '020'},
{`bounds`}: (100, 150), `price`: 4.42, `subcode`: '030'},
{`bounds`}: (150, 200), `price`: 5.63, `subcode`: '040'},
{`bounds`}: (200, math.inf), `price`: 6.84, `subcode`: '050'}]}
```

Notice that if we append subcode to code (with a dot) we obtain the full product code.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[6]:

```
import csv
from pprint import pprint

def extract_product(filename, code, measure_unit):

 c = 0
 with open(filename, encoding='latin-1', newline='') as f:
 my_reader = csv.DictReader(f, delimiter=',') # Notice we now used DictReader
 for d in my_reader:

 if d['Codice Prodotto'] == code:
 ret = {}
 ret['description'] = d['Descrizione Breve Prodotto']
 ret['code'] = code
 ret['measure_unit'] = measure_unit
 ret['models'] = []

 if d['Codice Prodotto'].startswith(code + '.'):
 ret['category'] = d['Categoria']
 subdiz = {}
 subdiz['price'] = float(d['Prezzo'])
 subdiz['subcode'] = d['Codice Prodotto'][len(code)+1:]
 subdiz['bounds'] = extract_bounds(d['Descrizione Breve Prodotto'])
 ret['models'].append(subdiz)

 return ret
```

(continues on next page)

(continued from previous page)

```

pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm'))
assert extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm') == \
 {'category': 'Noli e trasporti',
 'code': 'A.02.40.0010',
 'description': 'POMPA COMPLETA DI MOTORE',
 'measure_unit': 'mm',
 'models': [{ 'bounds': (0, 50), 'price': 2.21, 'subcode': '010' },
 { 'bounds': (50, 100), 'price': 3.36, 'subcode': '020' },
 { 'bounds': (100, 150), 'price': 4.42, 'subcode': '030' },
 { 'bounds': (150, 200), 'price': 5.63, 'subcode': '040' },
 { 'bounds': (200, math.inf), 'price': 6.84, 'subcode': '050' }] }

#pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0020', 'KW'))
#pprint(extract_product('EPPAT-2018-new-compact.csv', 'B.02.10.0042', 'mm'))
#pprint(extract_product('EPPAT-2018-new-compact.csv', 'B.30.10.0010', '%'))

{'category': 'Noli e trasporti',
 'code': 'A.02.40.0010',
 'description': 'POMPA COMPLETA DI MOTORE',
 'measure_unit': 'mm',
 'models': [{ 'bounds': (0, 50), 'price': 2.21, 'subcode': '010' },
 { 'bounds': (50, 100), 'price': 3.36, 'subcode': '020' },
 { 'bounds': (100, 150), 'price': 4.42, 'subcode': '030' },
 { 'bounds': (150, 200), 'price': 5.63, 'subcode': '040' },
 { 'bounds': (200, inf), 'price': 6.84, 'subcode': '050' }] }

```

&lt;/div&gt;

[6]:

```

import csv
from pprint import pprint

def extract_product(filename, code, measure_unit):
 raise Exception('TODO IMPLEMENT ME !')

pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm'))
assert extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm') == \
 {'category': 'Noli e trasporti',
 'code': 'A.02.40.0010',
 'description': 'POMPA COMPLETA DI MOTORE',
 'measure_unit': 'mm',
 'models': [{ 'bounds': (0, 50), 'price': 2.21, 'subcode': '010' },
 { 'bounds': (50, 100), 'price': 3.36, 'subcode': '020' },
 { 'bounds': (100, 150), 'price': 4.42, 'subcode': '030' },
 { 'bounds': (150, 200), 'price': 5.63, 'subcode': '040' },
 { 'bounds': (200, math.inf), 'price': 6.84, 'subcode': '050' }] }

#pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0020', 'KW'))
#pprint(extract_product('EPPAT-2018-new-compact.csv', 'B.02.10.0042', 'mm'))
#pprint(extract_product('EPPAT-2018-new-compact.csv', 'B.30.10.0010', '%'))

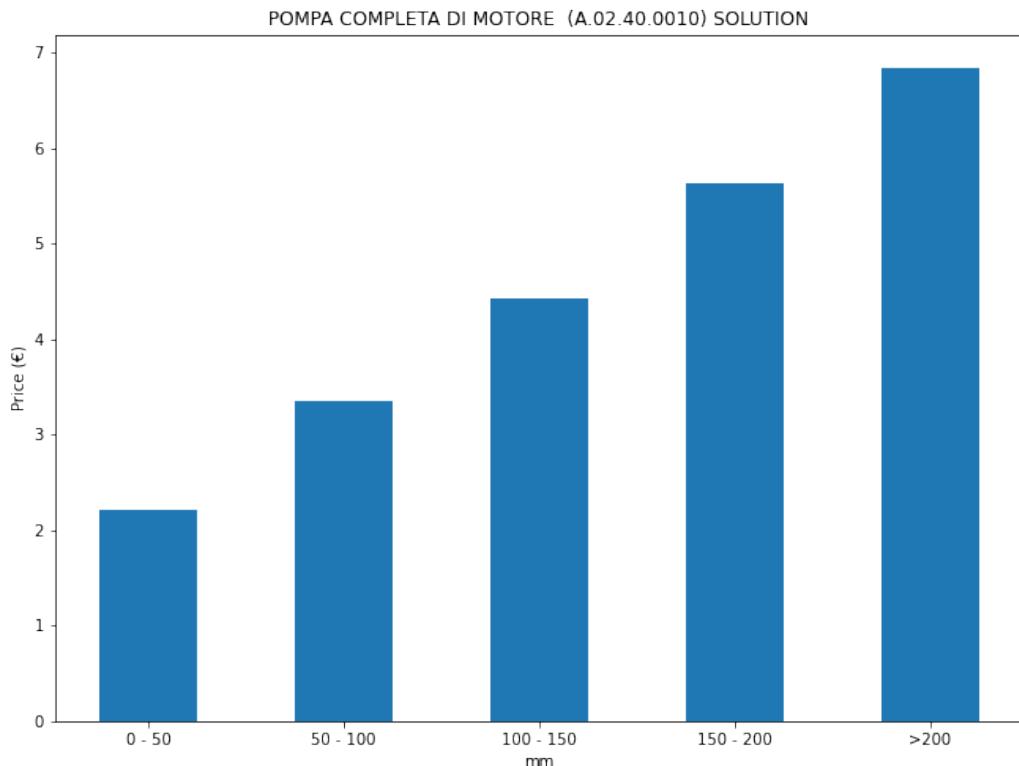
```

### A3 plot\_product

Implement following function that takes a dictionary as output by previous extract\_product and shows its price ranges.

- pay attention to display title and axis labels as shown, using input data and **not** constants.
- in case last range holds a `math.inf`, show a `>` sign
- **if you don't have a working extract\_product, just copy paste data from previous asserts.**

```
>>> extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm')
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[7]:

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def plot_product(product):

 models = product['models']
 xs = np.arange(len(models))
```

(continues on next page)

(continued from previous page)

```
ys = [model["price"] for model in models]

plt.bar(xs, ys, 0.5, align='center')

plt.title('%s (%s) SOLUTION' % (product['description'], product['code']))

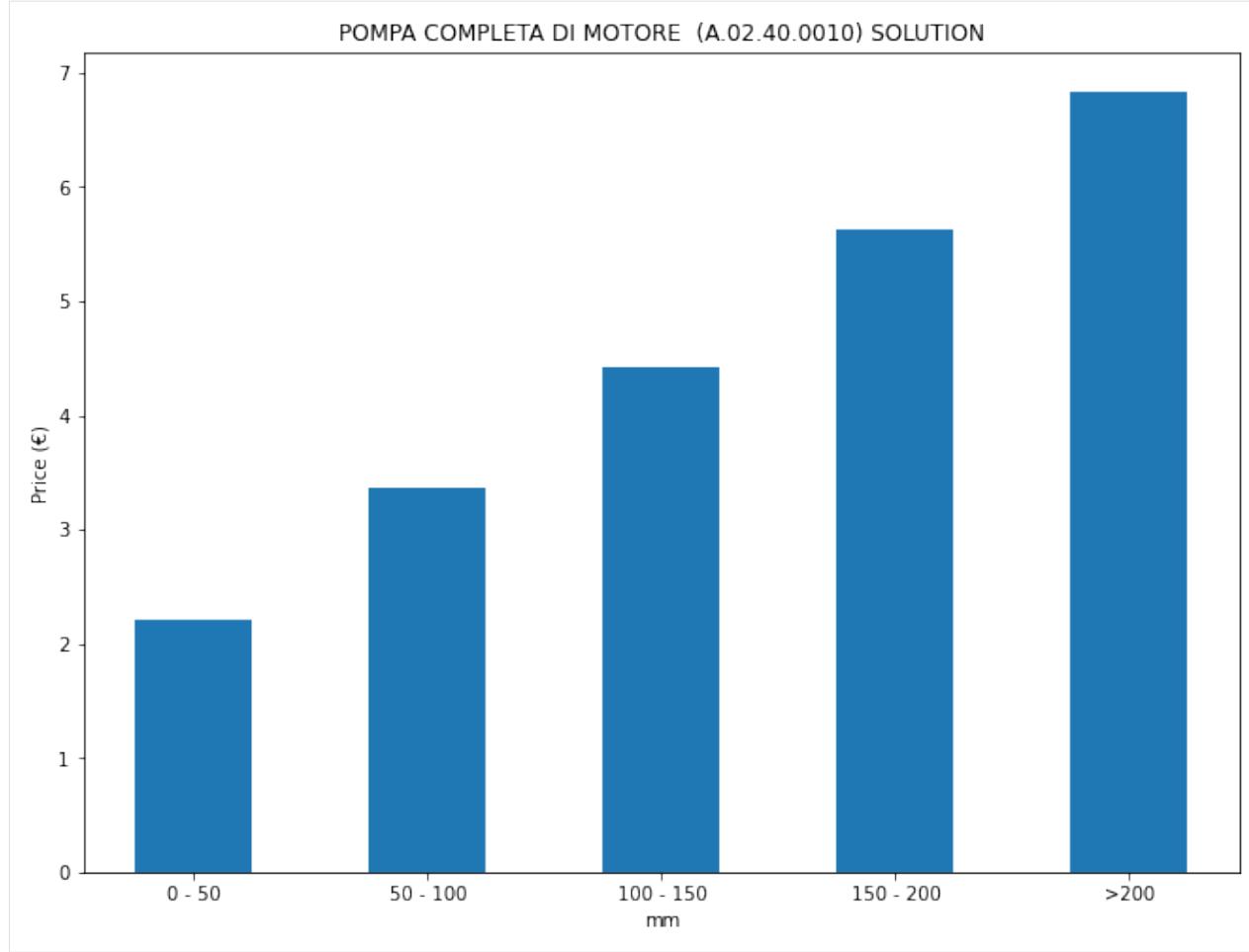
ticks = []
for model in models:
 bounds = model["bounds"]
 if bounds[1] == math.inf:
 ticks.append('>%s' % bounds[0])
 else:
 ticks.append('%s - %s' % (bounds[0], bounds[1]))

plt.xticks(xs, ticks)
plt.gcf().set_size_inches(11,8)
plt.xlabel(product['measure_unit'])
plt.ylabel('Price (€)')

plt.show()

product = extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm')
#product = extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0020', 'KW')
#product = extract_product('EPPAT-2018-new-compact.csv', 'B.02.10.0042', 'mm')
#product = extract_product('EPPAT-2018-new-compact.csv', 'B.30.10.0010', '%')

plot_product(product)
```



</div>

```
[7]:
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def plot_product(product):
 raise Exception('TODO IMPLEMENT ME !')

product = extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm')
#product = extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0020', 'KW')
#product = extract_product('EPPAT-2018-new-compact.csv', 'B.02.10.0042', 'mm')
#product = extract_product('EPPAT-2018-new-compact.csv', 'B.30.10.0010', '%')

plot_product(product)
```

## 10.2.11 EURES job offers

### Download worked project

After exiting your school prison, when looking for a job in Europe you will be shocked to discover a great variety of languages are spoken. Many job listings are provided by [Eures](#)<sup>478</sup> portal, which is easily searchable with many fields on which you can filter. For this exercise we will use a test dataset [offerte-lavoro.csv](#) which was generated just for a hackaton: it is a crude italian version of the job offers data, with many fields expressed in natural language. We will try to convert it to a dataset with more columns and translate some terms to English.

Data provider: Autonomous Province of Trento<sup>479</sup>

License: Creative Commons Zero 1.0<sup>480</sup>

#### Requirements:

We will manipulate the dataset with pandas which is a library for analytics: if you don't know it yet please read [the pandas tutorial](#)<sup>481</sup> first.

### What to do

1. If you haven't already, install Pandas:

Anaconda:

```
conda install pandas
```

Without Anaconda (--user installs in your home):

```
python3 -m pip install --user pandas
```

2. unzip exercises in a folder, you should get something like this:

```
pandas
eures.ipynb
eures-sol.ipynb
jupman.py
```

**WARNING 1:** to correctly visualize the notebook, it MUST be in an unzipped folder !

3. open Jupyter Notebook from that folder. Two things should open, first a console and then browser.
4. The browser should show a file list: navigate the list and open the notebook `eures.ipynb`

**WARNING 2:** DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

5. Go on reading that notebook, and follow instructions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter

<sup>478</sup> <https://ec.europa.eu/eures/public/homepage>

<sup>479</sup> <https://dati.trentino.it/dataset/offerte-di-lavoro-eures-test-odhb2019>

<sup>480</sup> <http://creativecommons.org/publicdomain/zero/1.0/deed.it>

<sup>481</sup> <https://en.softpython.org/pandas/pandas1-intro-sol.html>

- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### Introduction

#### WARNING: avoid constants in function bodies !!

In the exercises data you will find many names such as 'Austria', 'Giugno', etc. **DO NOT** put such constant names inside body of functions !! You have to write generic code which works with any input.

### offerte dataset

We will load the dataset offerte-lavoro.csv into Pandas:

```
[1]: import pandas as pd # we import pandas and for ease we rename it to 'pd'
import numpy as np # we import numpy and for ease we rename it to 'np'

remember the encoding !
offerte = pd.read_csv('offerte-lavoro.csv', encoding='UTF-8')
offerte.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53 entries, 0 to 52
Data columns (total 8 columns):
 # Column Non-Null Count Dtype
--- --
 0 RIFER. 53 non-null object
 1 SEDE LAVORO 53 non-null object
 2 POSTI 53 non-null int64
 3 IMPIEGO RICHIESTO 53 non-null object
 4 TIPO CONTRATTO 53 non-null object
 5 LINGUA RICHIESTA 51 non-null object
 6 RET. LORDA 53 non-null object
 7 DESCRIZIONE OFFERTA 53 non-null object
dtypes: int64(1), object(7)
memory usage: 3.4+ KB
```

It contains Italian column names, and many string fields:

```
[2]: offerte.head()

[2]: RIFER. SEDE LAVORO POSTI \n0 1833190100024 Norvegia 6
1 083PZMM Francia 1
2 4954752 Danimarca 1
3 - Berlino\nTrento 1
4 10531631 Svezia 1

 IMPIEGO RICHIESTO \\
0 Restaurant staff
1 Assistant export trilingue italien et anglais ...
2 Italian Sales Representative
3 Apprendista perito elettronico; Elettrotecnico
```

(continues on next page)

(continued from previous page)

```

4 Italian speaking purchase

 TIPO CONTRATTO \
0 Tempo determinato da maggio ad agosto
1 Non specificato
2 Non specificato
3 Inizialmente contratto di apprendistato con po...
4 Non specificato

 LINGUA RICHIESTA RET. LORDA \
0 Inglese fluente + Vedi testo Da 3500\nFr/\nmese
1 Inglese; italiano; francese fluente Da definire
2 Inglese; Italiano fluente Da definire
3 Inglese Buono (B1-B2); Tedesco base Min 1000\nMax\n1170\n€/mese
4 Inglese; italiano fluente Da definire

 DESCRIZIONE OFFERTA
0 We will be working together with sales, prepar...
1 Vos missions principales sont les suivantes : ...
2 Minimum 2 + years sales experience, preferably...
3 Ti stai diplomando e/o stai cercando un primo ...
4 This is a varied Purchasing role, where your m...

```

## rename columns

As first thing, we create a new dataframe `offers` with columns renamed into English:

```
[3]: replacements = ['Reference', 'Workplace', 'Positions', 'Qualification', 'Contract type',
 'Required languages', 'Gross retribution', 'Offer description']
diz = {}
i = 0
for col in offerte:
 diz[col] = replacements[i]
 i += 1
offers = offerte.rename(columns = diz)
```

```
[4]: offers
```

	Reference \
0	18331901000024
1	083PZMM
2	4954752
3	-
4	10531631
5	51485
6	4956299
7	-
8	2099681
9	12091902000474
10	10000-1169373760-S
11	10000-1168768920-S
12	082BMLG
13	23107550
14	11949-11273083-S
15	18331901000024

(continues on next page)

(continued from previous page)

```

16 ID-11252967
17 10000-1162270517-S
18 2100937
19 WBS697919
20 19361902000002
21 2095000
22 58699222
23 10000-1169431325-S
24 082QNLW
25 2101510
26 171767
27 14491903000005
28 10000-1167210671-S
29 507
30 846727
31 10531631
32 082ZFDB
33 1807568
34 2103264
35 ID-11146984
36 -
37 243096
38 9909319
39 WBS1253419
40 70cb25b1-5510-11e9-b89f-005056ac086d
41 10000-1170625924-S
42 2106868
43 23233743
44 ID-11478229
45 ID-11477956
46 6171903000036
47 9909319
48 ID-11239341
49 10000-1167068836-S
50 083PZMM
51 4956299
52 -

```

	Workplace	Positions	\
0	Norvegia	6	
1	Francia	1	
2	Danimarca	1	
3	Berlino\nTrento	1	
4	Svezia	1	
5	Islanda	1	
6	Danimarca	1	
7	Italia\nLazise	1	
8	Irlanda	11	
9	Norvegia	1	
10	Svizzera	1	
11	Germania	1	
12	Francia	1	
13	Svezia	1	
14	Austria	1	
15	Norvegia	6	
16	Austria	1	
17	Germania	1	

(continues on next page)

(continued from previous page)

18		Irlanda	1
19		Paesi Bassi	5
20		Norvegia	2
21		Spagna	15
22		Norvegia	1
23		Svizzera	1
24		Francia	1
25		Irlanda	1
26		Spagna	300
27	Norvegia\nMøre e Romsdal e Sogn og Fjordane.		6
28		Germania	1
29		Italia\ned\nestero	25
30		Belgio	1
31		Svezia\nLund	1
32		Francia	1
33		Regno Unito	1
34		Irlanda	1
35	Austria Klagenfurt		1
36		Berlino\nTrento	1
37		Spagna	1
38		Francia	1
39		Paesi\nBassi	1
40		Svizzera	1
41		Germania	1
42		Irlanda	1
43		Svezia	1
44	Italia\nAustria		1
45		Austria	1
46	Norvegia\nHesla Gaard		1
47		Finlandia	1
48	Cipro Grecia Spagna		5
49		Germania	2
50		Francia	1
51		Belgio	1
52	Austria\nPfenninger Alm		1

0		Qualification \
		Restaurant staff
1	Assistant export trilingue italien et anglais ...	
		Italian Sales Representative
3	Apprendista perito elettronico; Elettrotecnico	
4		Italian speaking purchase
5		Pizza chef
6	Regional Key account manager - Italy	
7		Receptionist
8	Customer Service Representative in Athens	
9		Dispatch personnel
10	Mitarbeiter (m/w/d) im Verkaufssinnendienst	
11		Vertriebs assistent
12		Second / Seconde de cuisine
13		Waiter/Waitress
14		Empfangskraft
15		Salesclerk
16	Verkaufssachbearbeiter für Italien (m/w)	
17		Koch/Köchin
18		Garden Centre Assistant
19	Strawberries and Rhubarb processors	

(continues on next page)

(continued from previous page)

20 Cleaners/renholdere Fishing Camp 2019 season  
 21 Customer service agent for solar energy  
 22 Receptionists tourist hotel  
 23 Reiseverkehrskaufmann/-frau - Touristik  
 24 Assistant administratif export avec Italie (H/F)  
 25 Receptionist  
 26 Seasonal worker in a strawberry farm  
 27 Guider  
 28 Sales Manager Südeuropa m/w  
 29 Animatori - coreografi - ballerini - istruttor...  
 30 Junior Buyer Italian /English (m/v)  
 31 Italian Speaking Sales Administration Officer  
 32 Assistant Administratif et Commercial Bilingue...  
 33 Account Manager - German, Italian, Spanish, Dutch  
 34 Receptionist - Summer  
 35 Nachwuchsführungs kraft im Agrarhandel / Trainee...  
 36 Apprendista perito elettronico; Elettrotecnico  
 37 Customer Service with French and Italian  
 38 Commercial Web Italie (H/F)  
 39 Customer service employee Dow  
 40 Hauswart/In  
 41 Monteur (m/w/d) Photovoltaik (Elektroanlagenmo...  
 42 Retail Store Assistant  
 43 E-commerce copywriter  
 44 Forstarbeiter/in  
 45 Koch/Köchin für italienische Küche in Teilzeit  
 46 Maid / Housekeeping assistant  
 47 Test Designer  
 48 Animateur 2019 (m/w)  
 49 Verkaufshilfe im Souvenirshop (m/w/d) 5 Tage-W...  
 50 Assistant export trilingue italien et anglais ...  
 51 ACCOUNT MANAGER EXPORT ITALIE - HAYS - StepSto...  
 52 Cameriere e Commis de rang

	Contract type \
0	Tempo determinato da maggio ad agosto
1	Non specificato
2	Non specificato
3	Inizialmente contratto di apprendistato con po...
4	Non specificato
5	Tempo determinato
6	Non specificato
7	Non specificato
8	Non specificato
9	Maggio - agosto 2019
10	Non specificato
11	Non specificato
12	Tempo determinato da aprile ad ottobre 2019
13	Non specificato
14	Non specificato
15	Da maggio ad ottobre
16	Non specificato
17	Non specificato
18	Non specificato
19	Da maggio a settembre
20	Tempo determinato da aprile ad ottobre 2019
21	Non specificato

(continues on next page)

(continued from previous page)

```

22 Da maggio a settembre o da giugno ad agosto
23 Non specificato
24 Non specificato
25 Non specificato
26 Da febbraio a giugno
27 Tempo determinato da maggio a settembre
28 Tempo indeterminato
29 Tempo determinato da aprile ad ottobre
30 Non specificato
31 Tempo indeterminato
32 Non specificato
33 Non specificato
34 Da maggio a settembre
35 Non specificato
36 Inizialmente contratto di apprendistato con po...
37 Non specificato
38 Non specificato
39 Tempo determinato
40 Non specificato
41 Non specificato
42 Non specificato
43 Non specificato
44 Aprile - maggio 2019
45 Non specificato
46 Tempo determinato da aprile a dicembre
47 Non specificato
48 Tempo determinato aprile-ottobre
49 Contratto stagionale fino a novembre 2019
50 Non specificato
51 Non specificato
52 Non specificato

```

```

Required languages \
0 Inglese fluente + Vedi testo
1 Inglese; italiano; francese fluente
2 Inglese; Italiano fluente
3 Inglese Buono (B1-B2); Tedesco base
4 Inglese; italiano fluente
5 Inglese Buono
6 Inglese; italiano fluente
7 Inglese; Tedesco fluente + Vedi testo
8 Italiano fluente; Inglese buono
9 Inglese fluente + Vedi testo
10 Tedesco fluente; francese e/o italiano buono
11 Tedesco ed inglese fluente + italiano e/o spag...
12 Francese discreto
13 Inglese ed Italiano buono
14 Tedesco ed Inglese Fluente + vedi testo
15 Inglese fluente + Vedi testo
16 Tedesco e italiano fluenti
17 Italiano e tedesco buono
18 Inglese fluente
19 NaN
20 Inglese fluente
21 Inglese e tedesco fluenti
22 Inglese Fluente; francese e/o spagnolo buoni
23 Tedesco Fluente + Vedi testo

```

(continues on next page)

(continued from previous page)

```
24 Francese ed italiano fluenti
25 Inglese fluente; Tedesco discreto
26 NaN
27 Tedesco e inglese fluente + Italiano buono
28 Inglese e tedesco fluente + Italiano e/o spagn...
29 Inglese Buono + Vedi testo
30 Inglese Ed italiano fluente
31 Inglese ed italiano fluente
32 Francese ed italiano fluente
33 Inglese Fluente + Vedi testo
34 Inglese fluente
35 Tedesco; Italiano buono
36 Inglese Buono (B1-B2); Tedesco base
37 Italiano; Francese fluente; Spagnolo buono
38 Italiano; Francese fluente
39 Inglese; italiano fluente + vedi testo
40 Tedesco buono
41 Tedesco e/o inglese buono
42 Inglese Fluente
43 Inglese Fluente + vedi testo
44 Tedesco italiano discreto
45 Tedesco buono
46 Inglese fluente
47 Inglese fluente
48 Tedesco; inglese buono
49 Tedesco buono; Inglese buono
50 Inglese francese; Italiano fluente
51 Inglese francese; Italiano fluente
52 Inglese buono; tedesco preferibile
```

```
 Gross retribution \
0 Da 3500\nFr/\nmese
1 Da definire
2 Da definire
3 Min 1000\nMax\n1170\n€/mese
4 Da definire
5 Da definire
6 Da definire
7 Min 1500€\nMax\n1800€\nnetto\nmese
8 Da definire
9 Da definire
10 Da definire
11 Da definire
12 Da definire
13 Da definire
14 Da definire
15 Da definire
16 2574,68 Euro/\nmese
17 Da definire
18 Da definire
19 Vedi testo
20 Da definire
21 €21,000 per annum + 3.500
22 Da definire
23 Da definire
24 Da definire
25 Da definire
```

(continues on next page)

(continued from previous page)

```

26 Da definire
27 20000 NOK /mese
28 Da definire
29 Vedi testo
30 Da definire
31 Da definire
32 Da definire
33 £25,000 per annum
34 Da definire
35 1.950\nEuro/ mese
36 Min 1000\nMax\n1170\n€/mese
37 Da definire
38 Da definire
39 Da definire
40 Da definire
41 Da definire
42 Da definire
43 Da definire
44 €9,50\n/ora
45 Da definire
46 20.000 NOK mese
47 Da definire
48 800\n€/mese
49 Da definire
50 Da definire
51 Da definire
52 1500–1600\n€/mese

```

## Offer description

```

0 We will be working together with sales, prepar...
1 Vos missions principales sont les suivantes : ...
2 Minimum 2 + years sales experience, preferably...
3 Ti stai diplomando e/o stai cercando un primo ...
4 This is a varied Purchasing role, where your m...
5 Job details/requirements: Experience in making...
6 Requirements: possess good business acumen; ar...
7 Camping Village Du Parc, Lazise, Italy is looki...
8 Responsibilities: Solving customers queries by...
9 The Dispatch Team works outside in all weather...
10 Was Sie erwartet: telefonische und persönliche...
11 Ihre Tätigkeit: enge Zusammenarbeit mit unsere...
12 Missions : Vous serez en charge de la mise en ...
13 Bar Robusta are looking for someone that speak...
14 Erfolgreich abgeschlossene Ausbildung in der H...
15 We will be working together with sales, prepar...
16 Unsere Anforderungen: Sie haben eine kaufmänni...
17 Kenntnisse und Fertigkeiten: Erfolgreich abges...
18 Applicants should have good plant knowledge an...
19 In this job you will be busy picking strawberr...
20 Torsvåg Havfiske, estbl. 2005, is a touristcom...
21 One of our biggest clients offer a wide range ...
22 The job also incl communication with the kitch...
23 Wir erwarten: Abgeschlossene Reisebüroausbildu...
24 Vous serez en charge des missions suivantes po...
25 Receptionist required for the 2019 Season. Kno...
26 Peon agricola (recolector fresa) / culegator d...
27 We require that you: are at least 20 years old...

```

(continues on next page)

(continued from previous page)

```
28 Ihr Profil :Idealerweise Erfahrung in der Text...
29 Padronanza di una o più lingue tra queste (ita...
30 You have a Bachelor degree. 2-3 years of profe...
31 You will focus on: Act as our main contact for...
32 Au sein de l'équipe administrative, vous trava...
33 Account Manager The Candidate You will be an e...
34 Assist with any ad-hoc project as required by ...
35 Ihre Qualifikationen: landwirtschaftliche Ausb...
36 Ti stai diplomando e/o stai cercando un primo ...
37 As an IT Helpdesk, you will be responsible for...
38 Profil : Première expérience réussie dans la v...
39 Requirements: You have a bachelor degree or hi...
40 Wir suchen in unserem Team einen Mitarbeiter m...
41 Anforderungen an die Bewerber/innen: abgeschlo...
42 Retail Store Assistant required for a SPAR sho...
43 We support 15 languages incl Chinese, Russian ...
44 ANFORDERUNGSPROFIL: Pflichtschulabschluss und ...
45 ANFORDERUNGSPROFIL:Erfahrung mit Pasta & Pizze...
46 Responsibility for cleaning off our apartments...
47 As Test Designer in R&D Devices team you will:...
48 Deine Fähigkeiten: Im Vordergrund steht Deine ...
49 Wir bieten: Einen zukunftssicheren, saisonalen...
50 Description : Au sein d'une équipe de 10 perso...
51 Votre profil : Pour ce poste, nous recherchons...
52 Lavoro estivo nella periferia di Salisburgo. E...
```

## 1. Rename countries

We would like to create a new column holding a list of countries where the job is to be done. You will also have to translate countries to their English name.

To allow for text processing, you are provided with some data as python data structures (you do not need to further edit it):

```
[5]:
connectives = ['e', 'ed']
punctuation = ['.', ';', ',']

countries = {
 'Austria':'Austria',
 'Belgio': 'Belgium',
 'Cipro':'Cyprus',
 'Danimarca': 'Denmark',
 'Irlanda':'Ireland',
 'Italia':'Italy',
 'Grecia':'Greece',
 'Finlandia' : 'Finland',
 'Francia' : 'France',
 'Norvegia': 'Norway',
 'Paesi Bassi':'Netherlands',
 'Regno Unito': 'United Kingdom',
 'Spagna': 'Spain',
 'Svezia':'Sweden',
 'Islanda':'Iceland',
 'Svizzera':'Switzerland',
```

(continues on next page)

(continued from previous page)

```

 'estero': 'abroad' # special case
}

cities = {
 'Pfenninger Alm': 'Pfenninger Alm',
 'Berlino': 'Berlin',
 'Trento': 'Trento',
 'Klagenfurt': 'Klagenfurt',
 'Lazise': 'Lazise',
 'Lund': 'Lund',
 'Møre e Romsdal': 'Møre og Romsdal',
 'Pfenninger Alm': 'Pfenninger Alm',
 'Sogn og Fjordane': 'Sogn og Fjordane',
 'Hesla Gaard': 'Hesla Gaard'
}

```

## 1.1 countries\_to\_list

⊕⊕ Implement function `countries_to_list` which given a string from Workplace column, RETURN a list holding country names in English **in the exact order they appear in the string**. The function will have to remove city names as well as punctuation, connectives and newlines using data define in the previous cell. There are various ways to solve the exercise: if you try the most straightforward one, most probably you will get countries which are not in the same order as in the string.

**NOTE:** this function only takes a single string as input!

Example:

```
>>> countries_to_list("Regno Unito, Italia ed estero")
['United Kingdom', 'Italy', 'abroad']
```

For other examples, see asserts.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```

def countries_to_list(s):

 ret = []
 i = 0
 ns = s.replace('\n', ' ')
 for connective in connectives:
 ns = ns.replace(' ' + connective + ' ', ' ')
 for p in punctuation:
 ns = ns.replace(p, '')

 while i < len(ns):
 for country in countries:
 if ns[i:].startswith(country):
 ret.append(countries[country])
 i += len(country)
 i += 1 # crude but works for this dataset ;-
 return ret

```

(continues on next page)

(continued from previous page)

```
single country
assert countries_to_list("Francia") == ['France']
country with a city
assert countries_to_list("Austria Klagenfurt") == ['Austria']
country with a space
assert countries_to_list("Paesi Bassi") == ['Netherlands']
one country, newline, one city
assert countries_to_list("Italia\nLazise") == ['Italy']
newline, multiple cities
assert countries_to_list("Norvegia\nMøre e Romsdal e Sogn og Fjordane.") == ['Norway']
multiple countries - order *must* be preserved !
assert countries_to_list('Cipro Grecia Spagna') == ['Cyprus', 'Greece', 'Spain']
punctuation and connectives, multiple countries - order *must* be preserved !
assert countries_to_list('Regno Unito, Italia ed estero') == ['United Kingdom', 'Italy
˓→', 'abroad']
```

&lt;/div&gt;

[6]:

```
def countries_to_list(s):
 raise Exception('TODO IMPLEMENT ME !')

single country
assert countries_to_list("Francia") == ['France']
country with a city
assert countries_to_list("Austria Klagenfurt") == ['Austria']
country with a space
assert countries_to_list("Paesi Bassi") == ['Netherlands']
one country, newline, one city
assert countries_to_list("Italia\nLazise") == ['Italy']
newline, multiple cities
assert countries_to_list("Norvegia\nMøre e Romsdal e Sogn og Fjordane.") == ['Norway']
multiple countries - order *must* be preserved !
assert countries_to_list('Cipro Grecia Spagna') == ['Cyprus', 'Greece', 'Spain']
punctuation and connectives, multiple countries - order *must* be preserved !
assert countries_to_list('Regno Unito, Italia ed estero') == ['United Kingdom', 'Italy
˓→', 'abroad']
```

## 1.2 Filling column Workplace Country

⊕ Now create a new column `Workplace Country` with data calculated using the function you just defined.

To do it, check method `transform` in Pandas worksheet<sup>482</sup>

[7]: # write here

```
<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"
 data-jupman-show="Show solution"
 data-jupman-hide="Hide">Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

<sup>482</sup> <https://en.softpython.org/pandas/pandas1-sol.html#6.3-Transforming-columns>

```
[8]: # SOLUTION

offers['Workplace Country'] = offerte['SEDE LAVORO']
offers['Workplace Country'] = offers['Workplace Country'].transform(countries_to_list)

</div>

[8]:
```

```
[9]: print()
print("***** SOLUTION OUTPUT *****")
offers

***** SOLUTION OUTPUT *****
```

	Reference \
0	18331901000024
1	083PZMM
2	4954752
3	-
4	10531631
5	51485
6	4956299
7	-
8	2099681
9	12091902000474
10	10000-1169373760-S
11	10000-1168768920-S
12	082BMLG
13	23107550
14	11949-11273083-S
15	18331901000024
16	ID-11252967
17	10000-1162270517-S
18	2100937
19	WBS697919
20	19361902000002
21	2095000
22	58699222
23	10000-1169431325-S
24	082QNLW
25	2101510
26	171767
27	14491903000005
28	10000-1167210671-S
29	507
30	846727
31	10531631
32	082ZFDB
33	1807568
34	2103264
35	ID-11146984
36	-
37	243096
38	9909319
39	WBS1253419
40	70cb25b1-5510-11e9-b89f-005056ac086d

(continues on next page)

(continued from previous page)

41		10000-1170625924-S	
42		2106868	
43		23233743	
44		ID-11478229	
45		ID-11477956	
46		6171903000036	
47		9909319	
48		ID-11239341	
49		10000-1167068836-S	
50		083PZMM	
51		4956299	
52		-	
			\
0	Workplace	Positions	
1	Norvegia	6	
2	Francia	1	
3	Danimarca	1	
4	Berlino\nTrento	1	
5	Svezia	1	
6	Islanda	1	
7	Danimarca	1	
8	Italia\nLazise	1	
9	Irlanda	11	
10	Norvegia	1	
11	Svizzera	1	
12	Germania	1	
13	Francia	1	
14	Svezia	1	
15	Austria	1	
16	Norvegia	6	
17	Austria	1	
18	Germania	1	
19	Irlanda	1	
20	Paesi Bassi	5	
21	Norvegia	2	
22	Spagna	15	
23	Norvegia	1	
24	Svizzera	1	
25	Francia	1	
26	Irlanda	1	
27	Spagna	300	
28	Norvegia\nMøre e Romsdal e Sogn og Fjordane.	6	
29	Germania	1	
30	Italia\ned\nestero	25	
31	Belgio	1	
32	Svezia\nLund	1	
33	Francia	1	
34	Regno Unito	1	
35	Irlanda	1	
36	Austria Klagenfurt	1	
37	Berlino\nTrento	1	
38	Spagna	1	
39	Francia	1	
40	Paesi\nBassi	1	
41	Svizzera	1	
42	Germania	1	
	Irlanda	1	

(continues on next page)

(continued from previous page)

43	Svezia	1
44	Italia\nAustria	1
45	Austria	1
46	Norvegia\nHesla Gaard	1
47	Finlandia	1
48	Cipro Grecia Spagna	5
49	Germania	2
50	Francia	1
51	Belgio	1
52	Austria\nPfenninger Alm	1
0	Qualification \	
1	Assistant export trilingue italien et anglais ...	
2	Italian Sales Representative	
3	Apprendista perito elettronico; Elettrotecnico	
4	Italian speaking purchase	
5	Pizza chef	
6	Regional Key account manager - Italy	
7	Receptionist	
8	Customer Service Representative in Athens	
9	Dispatch personnel	
10	Mitarbeiter (m/w/d) im Verkaufssinnendienst	
11	Vertriebs assistent	
12	Second / Seconde de cuisine	
13	Waiter/Waitress	
14	Empfangskraft	
15	Salesclerk	
16	Verkaufssachbearbeiter für Italien (m/w)	
17	Koch/Köchin	
18	Garden Centre Assistant	
19	Strawberries and Rhubarb processors	
20	Cleaners/renholdere Fishing Camp 2019 season	
21	Customer service agent for solar energy	
22	Receptionists tourist hotel	
23	Reiseverkehrskaufmann/-frau - Touristik	
24	Assistant administratif export avec Italie (H/F)	
25	Receptionist	
26	Seasonal worker in a strawberry farm	
27	Guider	
28	Sales Manager Südeuropa m/w	
29	Animatori - coreografi - ballerini - istruttor...	
30	Junior Buyer Italian /English (m/v)	
31	Italian Speaking Sales Administration Officer	
32	Assistant Administratif et Commercial Bilingue...	
33	Account Manager - German, Italian, Spanish, Dutch	
34	Receptionist - Summer	
35	Nachwuchsführungskraft im Agrarhandel / Trainee...	
36	Apprendista perito elettronico; Elettrotecnico	
37	Customer Service with French and Italian	
38	Commercial Web Italie (H/F)	
39	Customer service employee Dow	
40	Hauswart/In	
41	Monteur (m/w/d) Photovoltaik (Elektroanlagenmo...	
42	Retail Store Assistant	
43	E-commerce copywriter	
44	Forstarbeiter/in	

(continues on next page)

(continued from previous page)

```
45 Koch/Köchin für italienische Küche in Teilzeit
46 Maid / Housekeeping assistant
47 Test Designer
48 Animateur 2019 (m/w)
49 Verkaufshilfe im Souvenirshop (m/w/d) 5 Tage-W...
50 Assistant export trilingue italien et anglais ...
51 ACCOUNT MANAGER EXPORT ITALIE - HAYS - StepSto...
52 Cameriere e Commis de rang

 Contract type \
0 Tempo determinato da maggio ad agosto
1 Non specificato
2 Non specificato
3 Inizialmente contratto di apprendistato con po...
4 Non specificato
5 Tempo determinato
6 Non specificato
7 Non specificato
8 Non specificato
9 Maggio - agosto 2019
10 Non specificato
11 Non specificato
12 Tempo determinato da aprile ad ottobre 2019
13 Non specificato
14 Non specificato
15 Da maggio ad ottobre
16 Non specificato
17 Non specificato
18 Non specificato
19 Da maggio a settembre
20 Tempo determinato da aprile ad ottobre 2019
21 Non specificato
22 Da maggio a settembre o da giugno ad agosto
23 Non specificato
24 Non specificato
25 Non specificato
26 Da febbraio a giugno
27 Tempo determinato da maggio a settembre
28 Tempo indeterminato
29 Tempo determinato da aprile ad ottobre
30 Non specificato
31 Tempo indeterminato
32 Non specificato
33 Non specificato
34 Da maggio a settembre
35 Non specificato
36 Inizialmente contratto di apprendistato con po...
37 Non specificato
38 Non specificato
39 Tempo determinato
40 Non specificato
41 Non specificato
42 Non specificato
43 Non specificato
44 Aprile - maggio 2019
45 Non specificato
46 Tempo determinato da aprile a dicembre
```

(continues on next page)

(continued from previous page)

```

47 Non specificato
48 Tempo determinato aprile-ottobre
49 Contratto stagionale fino a novembre 2019
50 Non specificato
51 Non specificato
52 Non specificato

 Required languages \
0 Inglese fluente + Vedi testo
1 Inglese; italiano; francese fluente
2 Inglese; Italiano fluente
3 Inglese Buono (B1-B2); Tedesco base
4 Inglese; italiano fluente
5 Inglese Buono
6 Inglese; italiano fluente
7 Inglese; Tedesco fluente + Vedi testo
8 Italiano fluente; Inglese buono
9 Inglese fluente + Vedi testo
10 Tedesco fluente; francese e/o italiano buono
11 Tedesco ed inglese fluente + italiano e/o spag...
12 Francese discreto
13 Inglese ed Italiano buono
14 Tedesco ed Inglese Fluente + vedi testo
15 Inglese fluente + Vedi testo
16 Tedesco e italiano fluenti
17 Italiano e tedesco buono
18 Inglese fluente
19 NaN
20 Inglese fluente
21 Inglese e tedesco fluenti
22 Inglese Fluente; francese e/o spagnolo buoni
23 Tedesco Fluente + Vedi testo
24 Francese ed italiano fluenti
25 Inglese fluente; Tedesco discreto
26 NaN
27 Tedesco e inglese fluente + Italiano buono
28 Inglese e tedesco fluente + Italiano e/o spagn...
29 Inglese Buono + Vedi testo
30 Inglese Ed italiano fluente
31 Inglese ed italiano fluente
32 Francese ed italiano fluente
33 Inglese Fluente + Vedi testo
34 Inglese fluente
35 Tedesco; Italiano buono
36 Inglese Buono (B1-B2); Tedesco base
37 Italiano; Francese fluente; Spagnolo buono
38 Italiano; Francese fluente
39 Inglese; italiano fluente + vedi testo
40 Tedesco buono
41 Tedesco e/o inglese buono
42 Inglese Fluente
43 Inglese Fluente + vedi testo
44 Tedesco italiano discreto
45 Tedesco buono
46 Inglese fluente
47 Inglese fluente
48 Tedesco; inglese buono

```

(continues on next page)

(continued from previous page)

```
49 Tedesco buono; Inglese buono
50 Inglese francese; Italiano fluente
51 Inglese francese; Italiano fluente
52 Inglese buono; tedesco preferibile

 Gross retribution \
0 Da 3500\nFr/\nmese
1 Da definire
2 Da definire
3 Min 1000\nMax\n1170\n€/mese
4 Da definire
5 Da definire
6 Da definire
7 Min 1500€\nMax\n1800€\nnetto\nmese
8 Da definire
9 Da definire
10 Da definire
11 Da definire
12 Da definire
13 Da definire
14 Da definire
15 Da definire
16 2574,68 Euro/\nmese
17 Da definire
18 Da definire
19 Vedi testo
20 Da definire
21 €21,000 per annum + 3.500
22 Da definire
23 Da definire
24 Da definire
25 Da definire
26 Da definire
27 20000 NOK /mese
28 Da definire
29 Vedi testo
30 Da definire
31 Da definire
32 Da definire
33 £25,000 per annum
34 Da definire
35 1.950\nEuro/ mese
36 Min 1000\nMax\n1170\n€/mese
37 Da definire
38 Da definire
39 Da definire
40 Da definire
41 Da definire
42 Da definire
43 Da definire
44 €9,50\n/ora
45 Da definire
46 20.000 NOK mese
47 Da definire
48 800\n€/mese
49 Da definire
50 Da definire
```

(continues on next page)

(continued from previous page)

51	Da definire	
52	1500-1600\n€/mese	
	Offer description	Workplace Country
0	We will be working together with sales, prepar...	[Norway]
1	Vos missions principales sont les suivantes : ...	[France]
2	Minimum 2 + years sales experience, preferably...	[Denmark]
3	Ti stai diplomando e/o stai cercando un primo ...	[]
4	This is a varied Purchasing role, where your m...	[Sweden]
5	Job details/requirements: Experience in making...	[Iceland]
6	Requirements: possess good business acumen; ar...	[Denmark]
7	Camping Village Du Parc, Lazise, Italy is looki...	[Italy]
8	Responsibilities: Solving customers queries by...	[Ireland]
9	The Dispatch Team works outside in all weather...	[Norway]
10	Was Sie erwartet: telefonische und persönliche...	[Switzerland]
11	Ihre Tätigkeit: enge Zusammenarbeit mit unsere...	[]
12	Missions : Vous serez en charge de la mise en ...	[France]
13	Bar Robusta are looking for someone that speak...	[Sweden]
14	Erfolgreich abgeschlossene Ausbildung in der H...	[Austria]
15	We will be working together with sales, prepar...	[Norway]
16	Unsere Anforderungen: Sie haben eine kaufmänni...	[Austria]
17	Kenntnisse und Fertigkeiten: Erfolgreich abges...	[]
18	Applicants should have good plant knowledge an...	[Ireland]
19	In this job you will be busy picking strawberr...	[Netherlands]
20	Torsvåg Havfiske, estbl. 2005, is a touristcom...	[Norway]
21	One of our biggest clients offer a wide range ...	[Spain]
22	The job also incl communication with the kitch...	[Norway]
23	Wir erwarten: Abgeschlossene Reisebüroausbildung...	[Switzerland]
24	Vous serez en charge des missions suivantes po...	[France]
25	Receptionist required for the 2019 Season. Kno...	[Ireland]
26	Peon agricola (recolector fresa) / culegator d...	[Spain]
27	We require that you: are at least 20 years old...	[Norway]
28	Ihr Profil : Idealerweise Erfahrung in der Text...	[]
29	Padronanza di una o più lingue tra queste (ita...	[Italy, abroad]
30	You have a Bachelor degree. 2-3 years of profe...	[Belgium]
31	You will focus on: Act as our main contact for...	[Sweden]
32	Au sein de l'équipe administrative, vous trava...	[France]
33	Account Manager The Candidate You will be an e...	[United Kingdom]
34	Assist with any ad-hoc project as required by ...	[Ireland]
35	Ihre Qualifikationen: landwirtschaftliche Ausb...	[Austria]
36	Ti stai diplomando e/o stai cercando un primo ...	[]
37	As an IT Helpdesk, you will be responsible for...	[Spain]
38	Profil : Première expérience réussie dans la v...	[France]
39	Requirements: You have a bachelor degree or hi...	[Netherlands]
40	Wir suchen in unserem Team einen Mitarbeiter m...	[Switzerland]
41	Anforderungen an die Bewerber/innen: abgeschlo...	[]
42	Retail Store Assistant required for a SPAR sho...	[Ireland]
43	We support 15 languages incl Chinese, Russian ...	[Sweden]
44	ANFORDERUNGSPROFIL: Pflichtschulabschluss und ...	[Italy, Austria]
45	ANFORDERUNGSPROFIL:Erfahrung mit Pasta & Pizze...	[Austria]
46	Responsibility for cleaning off our apartments...	[Norway]
47	As Test Designer in R&D Devices team you will:...	[Finland]
48	Deine Fähigkeiten: Im Vordergrund steht Deine ...	[Cyprus, Greece, Spain]
49	Wir bieten: Einen zukunftssicheren, saisonalen...	[]
50	Description : Au sein d'une équipe de 10 perso...	[France]
51	Votre profil : Pour ce poste, nous recherchons...	[Belgium]
52	Lavoro estivo nella periferia di Salisburgo. E...	[Austria]

## 2. Work dates

You will add columns holding the dates of when a job start and when a job ends.

### 2.1 from\_to function

⊕⊕ First define `from_to` function, which takes some text from column "Contract type" and RETURNS a tuple holding the extracted month numbers (starting from ONE, not zero!)

Example:

In this case result is (5, 8) because May is the fifth month and August is the eighth:

```
>>> from_to("Tempo determinato da maggio ad agosto")
(5, 8)
```

If it is not possible to extract the text, the function should return a tuple holding NaNs:

```
>>> from_to('Non specificato')
(np.nan, np.nan)
```

Beware NaNs can lead to puzzling results, make sure you have read NaN and Infinities section in [Numpy Matrices notebook](#)<sup>483</sup>

For other patterns to check, see asserts.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[10]: months = ['gennaio', 'febbraio', 'marzo', 'aprile', 'maggio', 'giugno',
 'luglio', 'agosto', 'settembre', 'ottobre', 'novembre', 'dicembre']

def from_to(text):

 ntext = text.lower().replace('ad ', 'a ')
 found = False

 if 'da ' in ntext:
 from_pos = ntext.find('da ') + 3
 from_month = text[from_pos:].split(' ')[0]
 if ' a ' in ntext:
 to_pos = ntext.find(' a ') + 3
 to_month = ntext[to_pos:].split(' ')[0]
 found = True
 if '-' in ntext:
 from_month = ntext.split(' - ')[0]
 to_month = ntext.split(' - ')[0].split(' ')[0]
 found = True

 if found:
 from_number = months.index(from_month) + 1
 to_number = months.index(to_month) + 1
 return (from_number,to_number)
 else:
```

(continues on next page)

<sup>483</sup> <https://en.softpython.org/matrices-numpy/matrices-numpy-sol.html#NaNs-and-infinities>

(continued from previous page)

```

 return (np.nan, np.nan)

assert from_to('Da maggio a settembre') == (5, 9)
assert from_to('Da maggio ad ottobre') == (5, 10)
assert from_to('Tempo determinato da maggio ad agosto') == (5, 8)
Unspecified
assert from_to('Non specificato') == (np.nan, np.nan)
WARNING: BE SUPERCAREFUL ABOUT THIS ONE: SYMBOL - IS *NOT* A MINUS !!
COPY AND PASTE IT EXACTLY AS YOU FIND IT HERE
(BUT OF COURSE *DO NOT COPY* THE MONTH NAMES !)
assert from_to('Maggio - agosto 2019') == (5, 5)
special case 'or', we just consider first interval and ignore the following one.
assert from_to('Da maggio a settembre o da giugno ad agosto') == (5, 9)
special case only right side, we ignore all of it
assert from_to('Contratto stagionale fino a novembre 2019') == (np.nan, np.nan)

```

&lt;/div&gt;

```
[10]: months = ['gennaio', 'febbraio', 'marzo' , 'aprile' , 'maggio' , 'giugno',
 'luglio' , 'agosto' , 'settembre', 'ottobre', 'novembre', 'dicembre']

def from_to(text):
 raise Exception('TODO IMPLEMENT ME !!')

assert from_to('Da maggio a settembre') == (5, 9)
assert from_to('Da maggio ad ottobre') == (5, 10)
assert from_to('Tempo determinato da maggio ad agosto') == (5, 8)
Unspecified
assert from_to('Non specificato') == (np.nan, np.nan)
WARNING: BE SUPERCAREFUL ABOUT THIS ONE: SYMBOL - IS *NOT* A MINUS !!
COPY AND PASTE IT EXACTLY AS YOU FIND IT HERE
(BUT OF COURSE *DO NOT COPY* THE MONTH NAMES !)
assert from_to('Maggio - agosto 2019') == (5, 5)
special case 'or', we just consider first interval and ignore the following one.
assert from_to('Da maggio a settembre o da giugno ad agosto') == (5, 9)
special case only right side, we ignore all of it
assert from_to('Contratto stagionale fino a novembre 2019') == (np.nan, np.nan)
```

## 2.2. From To columns

⊕ MODIFY offers dataframe by adding From and To columns.

- **HINT 1:** You can call transform, see Transforming section in Pandas worksheet<sup>484</sup>
- **HINT 2 :** to extract the element you want from the tuple, you can pass to the transform a function on the fly with lambda. See lambdas section in Functions worksheet<sup>485</sup>

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

<sup>484</sup> <https://en.softpython.org/pandas/pandas1-sol.html#6.3-Transforming-columns>

<sup>485</sup> <https://en.softpython.org/functions/fun1-intro-sol.html#Lambda-functions>

```
[11]: # write here

offers['From'] = offers['Contract type'].transform(lambda t: from_to(t)[0])
offers['To'] = offers['Contract type'].transform(lambda t: from_to(t)[1])

</div>
```

```
[11]: # write here
```

```
[12]: print()
print(" ***** SOLUTION OUTPUT *****")
offers
```

```
***** SOLUTION OUTPUT *****
[12]: Reference \
0 18331901000024
1 083PZMM
2 4954752
3 -
4 10531631
5 51485
6 4956299
7 -
8 2099681
9 12091902000474
10 10000-1169373760-S
11 10000-1168768920-S
12 082BMLG
13 23107550
14 11949-11273083-S
15 18331901000024
16 ID-11252967
17 10000-1162270517-S
18 2100937
19 WBS697919
20 19361902000002
21 2095000
22 58699222
23 10000-1169431325-S
24 082QNLW
25 2101510
26 171767
27 14491903000005
28 10000-1167210671-S
29 507
30 846727
31 10531631
32 082ZFDB
33 1807568
34 2103264
35 ID-11146984
36 -
37 243096
38 9909319
```

(continues on next page)

(continued from previous page)

39		WBS1253419	
40	70cb25b1-5510-11e9-b89f-005056ac086d	10000-1170625924-S	
41		2106868	
42		23233743	
43		ID-11478229	
44		ID-11477956	
45		6171903000036	
46		9909319	
47		ID-11239341	
48		10000-1167068836-S	
49		083PZMM	
50		4956299	
51		-	
52			
		Workplace	Positions \
0		Norvegia	6
1		Francia	1
2		Danimarca	1
3		Berlino\nTrento	1
4		Svezia	1
5		Islanda	1
6		Danimarca	1
7		Italia\nLazise	1
8		Irlanda	11
9		Norvegia	1
10		Svizzera	1
11		Germania	1
12		Francia	1
13		Svezia	1
14		Austria	1
15		Norvegia	6
16		Austria	1
17		Germania	1
18		Irlanda	1
19		Paesi Bassi	5
20		Norvegia	2
21		Spagna	15
22		Norvegia	1
23		Svizzera	1
24		Francia	1
25		Irlanda	1
26		Spagna	300
27	Norvegia\nMøre e Romsdal e Sogn og Fjordane.		6
28		Germania	1
29		Italia\nned\nestero	25
30		Belgio	1
31		Svezia\nLund	1
32		Francia	1
33		Regno Unito	1
34		Irlanda	1
35	Austria Klagenfurt		1
36		Berlino\nTrento	1
37		Spagna	1
38		Francia	1
39		Paesi\nBassi	1
40		Svizzera	1

(continues on next page)

(continued from previous page)

41		Germania	1
42		Irlanda	1
43		Svezia	1
44		Italia\nAustria	1
45		Austria	1
46		Norvegia\nHesla Gaard	1
47		Finlandia	1
48		Cipro Grecia Spagna	5
49		Germania	2
50		Francia	1
51		Belgio	1
52		Austria\nPfenninger Alm	1
		Qualification \	
0		Restaurant staff	
1	Assistant export trilingue italien et anglais ...		
2		Italian Sales Representative	
3	Apprendista perito elettronico; Elettrotecnico		
4		Italian speaking purchase	
5		Pizza chef	
6		Regional Key account manager - Italy	
7		Receptionist	
8	Customer Service Representative in Athens		
9		Dispatch personnel	
10	Mitarbeiter (m/w/d) im Verkaufssinnendienst		
11		Vertriebs assistent	
12		Second / Seconde de cuisine	
13		Waiter/Waitress	
14		Empfangskraft	
15		Salesclerk	
16	Verkaufssachbearbeiter für Italien (m/w)		
17		Koch/Köchin	
18		Garden Centre Assistant	
19		Strawberries and Rhubarb processors	
20	Cleaners/renholdere Fishing Camp 2019 season		
21		Customer service agent for solar energy	
22		Receptionists tourist hotel	
23		Reiseverkehrskaufmann/-frau - Touristik	
24	Assistant administratif export avec Italie (H/F)		
25		Receptionist	
26		Seasonal worker in a strawberry farm	
27		Guider	
28		Sales Manager Südeuropa m/w	
29	Animatori - coreografi - ballerini - istruttori...		
30		Junior Buyer Italian /English (m/v)	
31	Italian Speaking Sales Administration Officer		
32	Assistant Administratif et Commercial Bilingue...		
33	Account Manager - German, Italian, Spanish, Dutch		
34		Receptionist - Summer	
35	Nachwuchsführkraft im Agrarhandel / Trainee...		
36	Apprendista perito elettronico; Elettrotecnico		
37		Customer Service with French and Italian	
38		Commercial Web Italie (H/F)	
39		Customer service employee Dow	
40		Hauswart/In	
41	Monteur (m/w/d) Photovoltaik (Elektroanlagenmo...		
42		Retail Store Assistant	

(continues on next page)

(continued from previous page)

```

43 E-commerce copywriter
44 Forstarbeiter/in
45 Koch/Köchin für italienische Küche in Teilzeit
46 Maid / Housekeeping assistant
47 Test Designer
48 Animateur 2019 (m/w)
49 Verkaufshilfe im Souvenirshop (m/w/d) 5 Tage-W...
50 Assistant export trilingue italien et anglais ...
51 ACCOUNT MANAGER EXPORT ITALIE - HAYS - StepSto...
52 Cameriere e Commis de rang

```

	Contract type \
0	Tempo determinato da maggio ad agosto
1	Non specificato
2	Non specificato
3	Inizialmente contratto di apprendistato con po...
4	Non specificato
5	Tempo determinato
6	Non specificato
7	Non specificato
8	Non specificato
9	Maggio - agosto 2019
10	Non specificato
11	Non specificato
12	Tempo determinato da aprile ad ottobre 2019
13	Non specificato
14	Non specificato
15	Da maggio ad ottobre
16	Non specificato
17	Non specificato
18	Non specificato
19	Da maggio a settembre
20	Tempo determinato da aprile ad ottobre 2019
21	Non specificato
22	Da maggio a settembre o da giugno ad agosto
23	Non specificato
24	Non specificato
25	Non specificato
26	Da febbraio a giugno
27	Tempo determinato da maggio a settembre
28	Tempo indeterminato
29	Tempo determinato da aprile ad ottobre
30	Non specificato
31	Tempo indeterminato
32	Non specificato
33	Non specificato
34	Da maggio a settembre
35	Non specificato
36	Inizialmente contratto di apprendistato con po...
37	Non specificato
38	Non specificato
39	Tempo determinato
40	Non specificato
41	Non specificato
42	Non specificato
43	Non specificato
44	Aprile - maggio 2019

(continues on next page)

(continued from previous page)

```

45 Non specificato
46 Tempo determinato da aprile a dicembre
47 Non specificato
48 Tempo determinato aprile-ottobre
49 Contratto stagionale fino a novembre 2019
50 Non specificato
51 Non specificato
52 Non specificato

 Required languages \
0 Inglese fluente + Vedi testo
1 Inglese; italiano; francese fluente
2 Inglese; Italiano fluente
3 Inglese Buono (B1-B2); Tedesco base
4 Inglese; italiano fluente
5 Inglese Buono
6 Inglese; italiano fluente
7 Inglese; Tedesco fluente + Vedi testo
8 Italiano fluente; Inglese buono
9 Inglese fluente + Vedi testo
10 Tedesco fluente; francese e/o italiano buono
11 Tedesco ed inglese fluente + italiano e/o spag...
12 Francese discreto
13 Inglese ed Italiano buono
14 Tedesco ed Inglese Fluente + vedi testo
15 Inglese fluente + Vedi testo
16 Tedesco e italiano fluenti
17 Italiano e tedesco buono
18 Inglese fluente
19 NaN
20 Inglese fluente
21 Inglese e tedesco fluenti
22 Inglese Fluente; francese e/o spagnolo buoni
23 Tedesco Fluente + Vedi testo
24 Francese ed italiano fluenti
25 Inglese fluente; Tedesco discreto
26 NaN
27 Tedesco e inglese fluente + Italiano buono
28 Inglese e tedesco fluente + Italiano e/o spagn...
29 Inglese Buono + Vedi testo
30 Inglese Ed italiano fluente
31 Inglese ed italiano fluente
32 Francese ed italiano fluente
33 Inglese Fluente + Vedi testo
34 Inglese fluente
35 Tedesco; Italiano buono
36 Inglese Buono (B1-B2); Tedesco base
37 Italiano; Francese fluente; Spagnolo buono
38 Italiano; Francese fluente
39 Inglese; italiano fluente + vedi testo
40 Tedesco buono
41 Tedesco e/o inglese buono
42 Inglese Fluente
43 Inglese Fluente + vedi testo
44 Tedesco italiano discreto
45 Tedesco buono
46 Inglese fluente

```

(continues on next page)

(continued from previous page)

```

47 Inglese fluente
48 Tedesco; inglese buono
49 Tedesco buono; Inglese buono
50 Inglese francese; Italiano fluente
51 Inglese francese; Italiano fluente
52 Inglese buono; tedesco preferibile

 Gross retribution \
0 Da 3500\nFr/\nmese
1 Da definire
2 Da definire
3 Min 1000\nMax\n1170\n€/mese
4 Da definire
5 Da definire
6 Da definire
7 Min 1500€\nMax\n1800€\nnetto\nmese
8 Da definire
9 Da definire
10 Da definire
11 Da definire
12 Da definire
13 Da definire
14 Da definire
15 Da definire
16 2574,68 Euro/\nmese
17 Da definire
18 Da definire
19 Vedi testo
20 Da definire
21 €21,000 per annum + 3.500
22 Da definire
23 Da definire
24 Da definire
25 Da definire
26 Da definire
27 20000 NOK /mese
28 Da definire
29 Vedi testo
30 Da definire
31 Da definire
32 Da definire
33 £25,000 per annum
34 Da definire
35 1.950\nEuro/\n mese
36 Min 1000\nMax\n1170\n€/mese
37 Da definire
38 Da definire
39 Da definire
40 Da definire
41 Da definire
42 Da definire
43 Da definire
44 €9,50\n/ora
45 Da definire
46 20.000 NOK mese
47 Da definire
48 800\n€/mese

```

(continues on next page)

(continued from previous page)

49	Da definire
50	Da definire
51	Da definire
52	1500-1600\n€/mese
	Offer description \
0	We will be working together with sales, prepar...
1	Vos missions principales sont les suivantes : ...
2	Minimum 2 + years sales experience, preferably...
3	Ti stai diplomando e/o stai cercando un primo ...
4	This is a varied Purchasing role, where your m...
5	Job details/requirements: Experience in making...
6	Requirements: possess good business acumen; ar...
7	Camping Village Du Parc, Lazise, Italy is looki...
8	Responsibilities: Solving customers queries by...
9	The Dispatch Team works outside in all weather...
10	Was Sie erwartet: telefonische und persönliche...
11	Ihre Tätigkeit: enge Zusammenarbeit mit unsere...
12	Missions : Vous serez en charge de la mise en ...
13	Bar Robusta are looking for someone that speak...
14	Erfolgreich abgeschlossene Ausbildung in der H...
15	We will be working together with sales, prepar...
16	Unsere Anforderungen: Sie haben eine kaufmänni...
17	Kenntnisse und Fertigkeiten: Erfolgreich abges...
18	Applicants should have good plant knowledge an...
19	In this job you will be busy picking strawberri...
20	Torsvåg Havfiske, estbl. 2005, is a touristcom...
21	One of our biggest clients offer a wide range ...
22	The job also incl communication with the kitch...
23	Wir erwarten: Abgeschlossene Reisebüroausbildu...
24	Vous serez en charge des missions suivantes po...
25	Receptionist required for the 2019 Season. Kno...
26	Peon agricola (recolector fresa) / culegator d...
27	We require that you: are at least 20 years old...
28	Ihr Profil : Idealerweise Erfahrung in der Text...
29	Padronanza di una o più lingue tra queste (ita...
30	You have a Bachelor degree. 2-3 years of profe...
31	You will focus on: Act as our main contact for...
32	Au sein de l'équipe administrative, vous trava...
33	Account Manager The Candidate You will be an e...
34	Assist with any ad-hoc project as required by ...
35	Ihre Qualifikationen: landwirtschaftliche Ausb...
36	Ti stai diplomando e/o stai cercando un primo ...
37	As an IT Helpdesk, you will be responsible for...
38	Profil : Première expérience réussie dans la v...
39	Requirements: You have a bachelor degree or hi...
40	Wir suchen in unserem Team einen Mitarbeiter m...
41	Anforderungen an die Bewerber/innen: abgeschlo...
42	Retail Store Assistant required for a SPAR sho...
43	We support 15 languages incl Chinese, Russian ...
44	ANFORDERUNGSPROFIL: Pflichtschulabschluss und ...
45	ANFORDERUNGSPROFIL:Erfahrung mit Pasta & Pizze...
46	Responsibility for cleaning off our apartments...
47	As Test Designer in R&D Devices team you will:...
48	Deine Fähigkeiten: Im Vordergrund steht Deine ...
49	Wir bieten: Einen zukunftssicheren, saisonalen...
50	Description : Au sein d'une équipe de 10 perso...

(continues on next page)

(continued from previous page)

51 Votre profil : Pour ce poste, nous recherchons...  
 52 Lavoro estivo nella periferia di Salisburgo. E...

	Workplace	Country	From	To
0	[Norway]	5.0	8.0	
1	[France]	NaN	NaN	
2	[Denmark]	NaN	NaN	
3	[]	NaN	NaN	
4	[Sweden]	NaN	NaN	
5	[Iceland]	NaN	NaN	
6	[Denmark]	NaN	NaN	
7	[Italy]	NaN	NaN	
8	[Ireland]	NaN	NaN	
9	[Norway]	5.0	5.0	
10	[Switzerland]	NaN	NaN	
11	[]	NaN	NaN	
12	[France]	4.0	10.0	
13	[Sweden]	NaN	NaN	
14	[Austria]	NaN	NaN	
15	[Norway]	5.0	10.0	
16	[Austria]	NaN	NaN	
17	[]	NaN	NaN	
18	[Ireland]	NaN	NaN	
19	[Netherlands]	5.0	9.0	
20	[Norway]	4.0	10.0	
21	[Spain]	NaN	NaN	
22	[Norway]	5.0	9.0	
23	[Switzerland]	NaN	NaN	
24	[France]	NaN	NaN	
25	[Ireland]	NaN	NaN	
26	[Spain]	2.0	6.0	
27	[Norway]	5.0	9.0	
28	[]	NaN	NaN	
29	[Italy, abroad]	4.0	10.0	
30	[Belgium]	NaN	NaN	
31	[Sweden]	NaN	NaN	
32	[France]	NaN	NaN	
33	[United Kingdom]	NaN	NaN	
34	[Ireland]	5.0	9.0	
35	[Austria]	NaN	NaN	
36	[]	NaN	NaN	
37	[Spain]	NaN	NaN	
38	[France]	NaN	NaN	
39	[Netherlands]	NaN	NaN	
40	[Switzerland]	NaN	NaN	
41	[]	NaN	NaN	
42	[Ireland]	NaN	NaN	
43	[Sweden]	NaN	NaN	
44	[Italy, Austria]	4.0	4.0	
45	[Austria]	NaN	NaN	
46	[Norway]	4.0	12.0	
47	[Finland]	NaN	NaN	
48	[Cyprus, Greece, Spain]	NaN	NaN	
49	[]	NaN	NaN	
50	[France]	NaN	NaN	
51	[Belgium]	NaN	NaN	
52	[Austria]	NaN	NaN	

### 3. Required languages

Now we will try to extract required languages.

#### 3.1 function reqlan

⊕⊕⊕ First implement function `reqlan` that given a string from column 'Required language' produces a dictionary with extracted languages and associated level code in CEFR standard (Common European Framework of Reference for Languages).

Example:

```
>>> reqlan("Italiano; Francese fluente; Spagnolo buono")
{'italian': 'C1', 'french': 'C1', 'spanish': 'B2'}
```

To know what italian words are to be translated to, use dictionaries provided in the following cell.

See tests for more cases to handle.

**WARNING 1:** function takes a **single** string !!

**WARNING 2: BE VERY CAREFUL WITH NaN input !**

Function might also take a `NaN` value (`math.nan` or `np.nan` they are the same), in which case it should RETURN an empty dictionary:

```
>>> reqlan(np.nan)
{}
```

If you are checking for a `NaN`, **DO NOT** write

```
if text == np.nan: # WRONG !
```

To see why, do read **NaN**s and Infinities section in Numpy Matrices worksheet<sup>486</sup> !

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[13]:

```
languages = {
 'italiano':'italian',
 'tedesco':'german',
 'francese':'french',
 'inglese':'english',
 'spagnolo':'spanish',
}

lang_levels = {
 'discreto':'B1',
 'buono':'B2',
 'fluente':'C1',
```

(continues on next page)

<sup>486</sup> <https://en.softpython.org/matrices-numumpy/matrices-numumpy-sol.html#NaNs-and-infinities>

(continued from previous page)

```

}

def reqlan(text):

 import math
 if type(text) != str and math.isnan(text):
 return {}

 ret = {}
 ntext = text.lower().replace('+ vedi testo', '')
 ntext = ntext.replace('e/o','; ')
 ntext = ntext.replace(' e ','; ')
 words = ntext.replace(';', '').split(' ')

 found_langs = []
 for w in words:
 if w in languages:
 found_langs.append(w)
 if w in lang_levels or (w[:-1] +'e' in lang_levels):
 if w in lang_levels:
 label = lang_levels[w]
 else:
 label = lang_levels[w[:-1] + 'e']
 for lang in found_langs:
 ret[languages[lang]] = label
 found_langs = [] # reset

 return ret

different languages may have different skills
assert reqlan("Italiano fluente; Inglese buono") == {'italian': 'C1',
 'english': 'B2'}

a sequence of languages terminating with a level is assumed to have that same level
assert reqlan("Inglese; italiano; francese fluente") == {'english': 'C1',
 'italian': 'C1',
 'french' : 'C1'}

semicolon absence shouldn't be a problem
assert reqlan("Tedesco italiano discreto") == {
 'german':'B1',
 'italian': 'B1'
 }

we can have multiple sequences
assert reqlan("Italiano; Francese fluente; Spagnolo buono") == {'italian': 'C1',
 'french': 'C1',
 'spanish': 'B2'}

text after plus needs to be removed
assert reqlan("Inglese fluente + Vedi testo") == {'english': 'C1'}

plural.
NOTE: to do this, assume all plurals in the world

```

(continues on next page)

(continued from previous page)

```
are constructed by substituting 'i' to last character of singular words
assert reqlan("Tedesco e italiano fluenti") == {'german':'C1',
 'italian':'C1'}
```

```
special case: we ignore codes in parentheses and just put B2
assert reqlan("Inglese Buono (B1-B2); Tedesco base") == {'english': 'B2'}
```

```
e/o: and / or case. We simplify and just list them as others
```

```
assert reqlan("Tedesco fluente; francese e/o italiano buono") == { 'german':'C1',
 'french':'B2',
 'italian':'B2'
 }
```

```
of course there is a cell which is NaN :P
assert reqlan(np.nan) == {}
```

&lt;/div&gt;

[13]:

```
languages = {
 'italiano':'italian',
 'tedesco':'german',
 'francese':'french',
 'inglese':'english',
 'spagnolo':'spanish',
}
```

```
lang_levels = {
 'discreto':'B1',
 'buono':'B2',
 'fluente':'C1',
}
```

```
def reqlan(text):
 raise Exception('TODO IMPLEMENT ME !')
```

```
different languages may have different skills
assert reqlan("Italiano fluente; Inglese buono") == {'italian': 'C1',
 'english': 'B2'}
```

```
a sequence of languages terminating with a level is assumed to have that same level
assert reqlan("Inglese; italiano; francese fluente") == {'english': 'C1',
 'italian': 'C1',
 'french' : 'C1'}
```

```
semicolon absence shouldn't be a problem
assert reqlan("Tedesco italiano discreto") == {
 'german':'B1',
 'italian': 'B1'
}
```

```
we can have multiple sequences
assert reqlan("Italiano; Francese fluente; Spagnolo buono") == {'italian': 'C1',
 'french': 'C1',
 'spanish': 'B2'}
```

(continues on next page)

(continued from previous page)

```
text after plus needs to be removed
assert reqlan("Inglese fluente + Vedi testo") == {'english': 'C1'}
```

```
plural.
NOTE: to do this, assume all plurals in the world
are constructed by substituting 'i' to last character of singular words
assert reqlan("Tedesco e italiano fluenti") == {'german':'C1',
 'italian':'C1'}
```

```
special case: we ignore codes in parentheses and just put B2
assert reqlan("Inglese Buono (B1-B2); Tedesco base") == {'english': 'B2'}
```

```
e/o: and / or case. We simplify and just list them as others
```

```
assert reqlan("Tedesco fluente; francese e/o italiano buono") == { 'german':'C1',
 'french':'B2',
 'italian':'B2'
 }
```

```
of course there is a cell which is NaN :P
assert reqlan(np.nan) == {}
```

### 3.2 Languages column

⊕ Now add the languages column using the previously defined reqlan function:

Show solution</div><div class="jupman-sol-jupman-sol-code" style="display:none">

```
[14]: # write here
offers['Languages'] = offers['Required languages'].transform(reqlan)
</div>
```

```
[14]: # write here
```

```
[15]: print()
print("***** SOLUTION OUTPUT *****")
offers
```

```
***** SOLUTION OUTPUT *****
Reference \
0 18331901000024
1 083PZMM
2 4954752
3 -
4 10531631
5 51485
6 4956299
7 -
8 2099681
```

(continues on next page)

(continued from previous page)

9	12091902000474
10	10000-1169373760-S
11	10000-1168768920-S
12	082BMLG
13	23107550
14	11949-11273083-S
15	18331901000024
16	ID-11252967
17	10000-1162270517-S
18	2100937
19	WBS697919
20	19361902000002
21	2095000
22	58699222
23	10000-1169431325-S
24	082QNLW
25	2101510
26	171767
27	14491903000005
28	10000-1167210671-S
29	507
30	846727
31	10531631
32	082ZFDB
33	1807568
34	2103264
35	ID-11146984
36	-
37	243096
38	9909319
39	WBS1253419
40	70cb25b1-5510-11e9-b89f-005056ac086d
41	10000-1170625924-S
42	2106868
43	23233743
44	ID-11478229
45	ID-11477956
46	6171903000036
47	9909319
48	ID-11239341
49	10000-1167068836-S
50	083PZMM
51	4956299
52	-
	Workplace    Positions \
0	Norvegia                6
1	Francia                1
2	Danimarca                1
3	Berlino\nTrento                1
4	Svezia                1
5	Islanda                1
6	Danimarca                1
7	Italia\nLazise                1
8	Irlanda                11
9	Norvegia                1
10	Svizzera                1

(continues on next page)

(continued from previous page)

11		Germania	1
12		Francia	1
13		Svezia	1
14		Austria	1
15		Norvegia	6
16		Austria	1
17		Germania	1
18		Irlanda	1
19		Paesi Bassi	5
20		Norvegia	2
21		Spagna	15
22		Norvegia	1
23		Svizzera	1
24		Francia	1
25		Irlanda	1
26		Spagna	300
27	Norvegia\nMøre e Romsdal e Sogn og Fjordane.		6
28		Germania	1
29		Italia\ned\nestero	25
30		Belgio	1
31		Svezia\nLund	1
32		Francia	1
33		Regno Unito	1
34		Irlanda	1
35	Austria Klagenfurt		1
36		Berlino\nTrento	1
37		Spagna	1
38		Francia	1
39		Paesi\nBassi	1
40		Svizzera	1
41		Germania	1
42		Irlanda	1
43		Svezia	1
44		Italia\nAustria	1
45		Austria	1
46	Norvegia\nHesla Gaard		1
47		Finlandia	1
48	Cipro Grecia Spagna		5
49		Germania	2
50		Francia	1
51		Belgio	1
52	Austria\nPfenninger Alm		1
0		Qualification \ Restaurant staff	
1	Assistant export trilingue italien et anglais ...		
2		Italian Sales Representative	
3	Apprendista perito elettronico; Elettrotecnico		
4		Italian speaking purchase	
5		Pizza chef	
6		Regional Key account manager - Italy	
7		Receptionist	
8	Customer Service Representative in Athens		
9		Dispatch personnel	
10	Mitarbeiter (m/w/d) im Verkaufssinnendienst		
11		Vertriebs assistent	
12		Second / Seconde de cuisine	

(continues on next page)

(continued from previous page)

13	Waiter/Waitress
14	Empfangskraft
15	Salesclerk
16	Verkaufssachbearbeiter für Italien (m/w)
17	Koch/Köchin
18	Garden Centre Assistant
19	Strawberries and Rhubarb processors
20	Cleaners/renholdere Fishing Camp 2019 season
21	Customer service agent for solar energy
22	Receptionists tourist hotel
23	Reiseverkehrskaufmann/-frau - Touristik
24	Assistant administratif export avec Italie (H/F)
25	Receptionist
26	Seasonal worker in a strawberry farm
27	Guider
28	Sales Manager Südeuropa m/w
29	Animatori - coreografi - ballerini - istruttori...
30	Junior Buyer Italian /English (m/v)
31	Italian Speaking Sales Administration Officer
32	Assistant Administratif et Commercial Bilingue...
33	Account Manager - German, Italian, Spanish, Dutch
34	Receptionist - Summer
35	Nachwuchsführungskraft im Agrarhandel / Trainee...
36	Apprendista perito elettronico; Elettrotecnico
37	Customer Service with French and Italian
38	Commercial Web Italie (H/F)
39	Customer service employee Dow
40	Hauswart/In
41	Monteur (m/w/d) Photovoltaik (Elektroanlagenmo...
42	Retail Store Assistant
43	E-commerce copywriter
44	Forstarbeiter/in
45	Koch/Köchin für italienische Küche in Teilzeit
46	Maid / Housekeeping assistant
47	Test Designer
48	Animateur 2019 (m/w)
49	Verkaufshilfe im Souvenirshop (m/w/d) 5 Tage-W...
50	Assistant export trilingue italien et anglais ...
51	ACCOUNT MANAGER EXPORT ITALIE - HAYS - StepSto...
52	Cameriere e Commis de rang

0	Contract type \
1	Tempo determinato da maggio ad agosto
2	Non specificato
3	Inizialmente contratto di apprendistato con po...
4	Non specificato
5	Tempo determinato
6	Non specificato
7	Non specificato
8	Non specificato
9	Maggio - agosto 2019
10	Non specificato
11	Non specificato
12	Tempo determinato da aprile ad ottobre 2019
13	Non specificato
14	Non specificato

(continues on next page)

(continued from previous page)

```

15 Da maggio ad ottobre
16 Non specificato
17 Non specificato
18 Non specificato
19 Da maggio a settembre
20 Tempo determinato da aprile ad ottobre 2019
21 Non specificato
22 Da maggio a settembre o da giugno ad agosto
23 Non specificato
24 Non specificato
25 Non specificato
26 Da febbraio a giugno
27 Tempo determinato da maggio a settembre
28 Tempo indeterminato
29 Tempo determinato da aprile ad ottobre
30 Non specificato
31 Tempo indeterminato
32 Non specificato
33 Non specificato
34 Da maggio a settembre
35 Non specificato
36 Inizialmente contratto di apprendistato con po...
37 Non specificato
38 Non specificato
39 Tempo determinato
40 Non specificato
41 Non specificato
42 Non specificato
43 Non specificato
44 Aprile - maggio 2019
45 Non specificato
46 Tempo determinato da aprile a dicembre
47 Non specificato
48 Tempo determinato aprile-ottobre
49 Contratto stagionale fino a novembre 2019
50 Non specificato
51 Non specificato
52 Non specificato

```

```

 Required languages \
0 Inglese fluente + Vedi testo
1 Inglese; italiano; francese fluente
2 Inglese; Italiano fluente
3 Inglese Buono (B1-B2); Tedesco base
4 Inglese; italiano fluente
5 Inglese Buono
6 Inglese; italiano fluente
7 Inglese; Tedesco fluente + Vedi testo
8 Italiano fluente; Inglese buono
9 Inglese fluente + Vedi testo
10 Tedesco fluente; francese e/o italiano buono
11 Tedesco ed inglese fluente + italiano e/o spag...
12 Francese discreto
13 Inglese ed Italiano buono
14 Tedesco ed Inglese Fluente + vedi testo
15 Inglese fluente + Vedi testo
16 Tedesco e italiano fluenti

```

(continues on next page)

(continued from previous page)

```

17 Italiano e tedesco buono
18 Inglese fluente
19 NaN
20 Inglese fluente
21 Inglese e tedesco fluenti
22 Inglese Fluente; francese e/o spagnolo buoni
23 Tedesco Fluente + Vedi testo
24 Francese ed italiano fluenti
25 Inglese fluente; Tedesco discreto
26 NaN
27 Tedesco e inglese fluente + Italiano buono
28 Inglese e tedesco fluente + Italiano e/o spagn...
29 Inglese Buono + Vedi testo
30 Inglese Ed italiano fluente
31 Inglese ed italiano fluente
32 Francese ed italiano fluente
33 Inglese Fluente + Vedi testo
34 Inglese fluente
35 Tedesco; Italiano buono
36 Inglese Buono (B1-B2); Tedesco base
37 Italiano; Francese fluente; Spagnolo buono
38 Italiano; Francese fluente
39 Inglese; italiano fluente + vedi testo
40 Tedesco buono
41 Tedesco e/o inglese buono
42 Inglese Fluente
43 Inglese Fluente + vedi testo
44 Tedesco italiano discreto
45 Tedesco buono
46 Inglese fluente
47 Inglese fluente
48 Tedesco; inglese buono
49 Tedesco buono; Inglese buono
50 Inglese francese; Italiano fluente
51 Inglese francese; Italiano fluente
52 Inglese buono; tedesco preferibile

 Gross retribution \
0 Da 3500\nFr/\nmese
1 Da definire
2 Da definire
3 Min 1000\nMax\n1170\n€/mese
4 Da definire
5 Da definire
6 Da definire
7 Min 1500€\nMax\n1800€\nnetto\nmese
8 Da definire
9 Da definire
10 Da definire
11 Da definire
12 Da definire
13 Da definire
14 Da definire
15 Da definire
16 2574,68 Euro/\nmese
17 Da definire
18 Da definire

```

(continues on next page)

(continued from previous page)

```

19 Vedi testo
20 Da definire
21 €21,000 per annum + 3.500
22 Da definire
23 Da definire
24 Da definire
25 Da definire
26 Da definire
27 20000 NOK /mese
28 Da definire
29 Vedi testo
30 Da definire
31 Da definire
32 Da definire
33 £25,000 per annum
34 Da definire
35 1.950\nEuro/ mese
36 Min 1000\nMax\n1170\n€/mese
37 Da definire
38 Da definire
39 Da definire
40 Da definire
41 Da definire
42 Da definire
43 Da definire
44 €9,50\n/ora
45 Da definire
46 20.000 NOK mese
47 Da definire
48 800\n€/mese
49 Da definire
50 Da definire
51 Da definire
52 1500-1600\n€/mese

```

## Offer description \

```

0 We will be working together with sales, prepar...
1 Vos missions principales sont les suivantes : ...
2 Minimum 2 + years sales experience, preferably...
3 Ti stai diplomando e/o stai cercando un primo ...
4 This is a varied Purchasing role, where your m...
5 Job details/requirements: Experience in making...
6 Requirements: possess good business acumen; ar...
7 Camping Village Du Parc, Lazise, Italy is looki...
8 Responsibilities: Solving customers queries by...
9 The Dispatch Team works outside in all weather...
10 Was Sie erwartet: telefonische und persönliche...
11 Ihre Tätigkeit: enge Zusammenarbeit mit unsere...
12 Missions : Vous serez en charge de la mise en ...
13 Bar Robusta are looking for someone that speak...
14 Erfolgreich abgeschlossene Ausbildung in der H...
15 We will be working together with sales, prepar...
16 Unsere Anforderungen: Sie haben eine kaufmänn...
17 Kenntnisse und Fertigkeiten: Erfolgreich abges...
18 Applicants should have good plant knowledge an...
19 In this job you will be busy picking strawberr...
20 Torsvåg Havfiske, estbl. 2005, is a touristcom...

```

(continues on next page)

(continued from previous page)

21 One of our biggest clients offer a wide range ...
 22 The job also incl communication with the kitch...
 23 Wir erwarten: Abgeschlossene Reisebüroausbildu...
 24 Vous serez en charge des missions suivantes po...
 25 Receptionist required for the 2019 Season. Kno...
 26 Peon agricola (recolector fresa) / culegator d...
 27 We require that you: are at least 20 years old...
 28 Ihr Profil : Idealerweise Erfahrung in der Text...
 29 Padronanza di una o più lingue tra queste (ita...
 30 You have a Bachelor degree. 2-3 years of profe...
 31 You will focus on: Act as our main contact for...
 32 Au sein de l'équipe administrative, vous trava...
 33 Account Manager The Candidate You will be an e...
 34 Assist with any ad-hoc project as required by ...
 35 Ihre Qualifikationen: landwirtschaftliche Ausb...
 36 Ti stai diplomando e/o stai cercando un primo ...
 37 As an IT Helpdesk, you will be responsible for...
 38 Profil : Première expérience réussie dans la v...
 39 Requirements: You have a bachelor degree or hi...
 40 Wir suchen in unserem Team einen Mitarbeiter m...
 41 Anforderungen an die Bewerber/innen: abgeschlo...
 42 Retail Store Assistant required for a SPAR sho...
 43 We support 15 languages incl Chinese, Russian ...
 44 ANFORDERUNGSPROFIL: Pflichtschulabschluss und ...
 45 ANFORDERUNGSPROFIL:Erfahrung mit Pasta & Pizze...
 46 Responsibility for cleaning off our apartments...
 47 As Test Designer in R&D Devices team you will:...
 48 Deine Fähigkeiten: Im Vordergrund steht Deine ...
 49 Wir bieten: Einen zukunftssicheren, saisonalen...
 50 Description : Au sein d'une équipe de 10 perso...
 51 Votre profil : Pour ce poste, nous recherchons...
 52 Lavoro estivo nella periferia di Salisburgo. E...

	Workplace	Country	From	To	\
0		[Norway]	5.0	8.0	
1		[France]	NaN	NaN	
2		[Denmark]	NaN	NaN	
3		[]	NaN	NaN	
4		[Sweden]	NaN	NaN	
5		[Iceland]	NaN	NaN	
6		[Denmark]	NaN	NaN	
7		[Italy]	NaN	NaN	
8		[Ireland]	NaN	NaN	
9		[Norway]	5.0	5.0	
10		[Switzerland]	NaN	NaN	
11		[]	NaN	NaN	
12		[France]	4.0	10.0	
13		[Sweden]	NaN	NaN	
14		[Austria]	NaN	NaN	
15		[Norway]	5.0	10.0	
16		[Austria]	NaN	NaN	
17		[]	NaN	NaN	
18		[Ireland]	NaN	NaN	
19		[Netherlands]	5.0	9.0	
20		[Norway]	4.0	10.0	
21		[Spain]	NaN	NaN	
22		[Norway]	5.0	9.0	

(continues on next page)

(continued from previous page)

```

23 [Switzerland] NaN NaN
24 [France] NaN NaN
25 [Ireland] NaN NaN
26 [Spain] 2.0 6.0
27 [Norway] 5.0 9.0
28 [] NaN NaN
29 [Italy, abroad] 4.0 10.0
30 [Belgium] NaN NaN
31 [Sweden] NaN NaN
32 [France] NaN NaN
33 [United Kingdom] NaN NaN
34 [Ireland] 5.0 9.0
35 [Austria] NaN NaN
36 [] NaN NaN
37 [Spain] NaN NaN
38 [France] NaN NaN
39 [Netherlands] NaN NaN
40 [Switzerland] NaN NaN
41 [] NaN NaN
42 [Ireland] NaN NaN
43 [Sweden] NaN NaN
44 [Italy, Austria] 4.0 4.0
45 [Austria] NaN NaN
46 [Norway] 4.0 12.0
47 [Finland] NaN NaN
48 [Cyprus, Greece, Spain] NaN NaN
49 [] NaN NaN
50 [France] NaN NaN
51 [Belgium] NaN NaN
52 [Austria] NaN NaN

```

```

 Languages
0 {'english': 'C1'}
1 {'english': 'C1', 'italian': 'C1', 'french': '...'
2 {'english': 'C1', 'italian': 'C1'}
3 {'english': 'B2'}
4 {'english': 'C1', 'italian': 'C1'}
5 {'english': 'B2'}
6 {'english': 'C1', 'italian': 'C1'}
7 {'english': 'C1', 'german': 'C1'}
8 {'italian': 'C1', 'english': 'B2'}
9 {'english': 'C1'}
10 {'german': 'C1', 'french': 'B2', 'italian': 'B2'}
11 {'german': 'C1', 'english': 'C1', 'italian': '...'
12 {'french': 'B1'}
13 {'english': 'B2', 'italian': 'B2'}
14 {'german': 'C1', 'english': 'C1'}
15 {'english': 'C1'}
16 {'german': 'C1', 'italian': 'C1'}
17 {'italian': 'B2', 'german': 'B2'}
18 {'english': 'C1'}
19 {}
20 {'english': 'C1'}
21 {'english': 'C1', 'german': 'C1'}
22 {'english': 'C1'}
23 {'german': 'C1'}
24 {'french': 'C1', 'italian': 'C1'}

```

(continues on next page)

(continued from previous page)

```
25 {'english': 'C1', 'german': 'B1'}
26 {}
27 {'german': 'C1', 'english': 'C1', 'italian': '...'
28 {'english': 'C1', 'german': 'C1', 'italian': '...'
29 {'english': 'B2'}
30 {'english': 'C1', 'italian': 'C1'}
31 {'english': 'C1', 'italian': 'C1'}
32 {'french': 'C1', 'italian': 'C1'}
33 {'english': 'C1'}
34 {'english': 'C1'}
35 {'german': 'B2', 'italian': 'B2'}
36 {'english': 'B2'}
37 {'italian': 'C1', 'french': 'C1', 'spanish': '...'
38 {'italian': 'C1', 'french': 'C1'}
39 {'english': 'C1', 'italian': 'C1'}
40 {'german': 'B2'}
41 {'german': 'B2', 'english': 'B2'}
42 {'english': 'C1'}
43 {'english': 'C1'}
44 {'german': 'B1', 'italian': 'B1'}
45 {'german': 'B2'}
46 {'english': 'C1'}
47 {'english': 'C1'}
48 {'german': 'B2', 'english': 'B2'}
49 {'german': 'B2', 'english': 'B2'}
50 {'english': 'C1', 'french': 'C1', 'italian': '...'
51 {'english': 'C1', 'french': 'C1', 'italian': '...'
52 {'english': 'B2'}
```

## Continue

Go on with the challenges<sup>487</sup>

## 10.3 Relational data

### 10.3.1 Trans-Atlantic Slave Trade

#### Download worked project

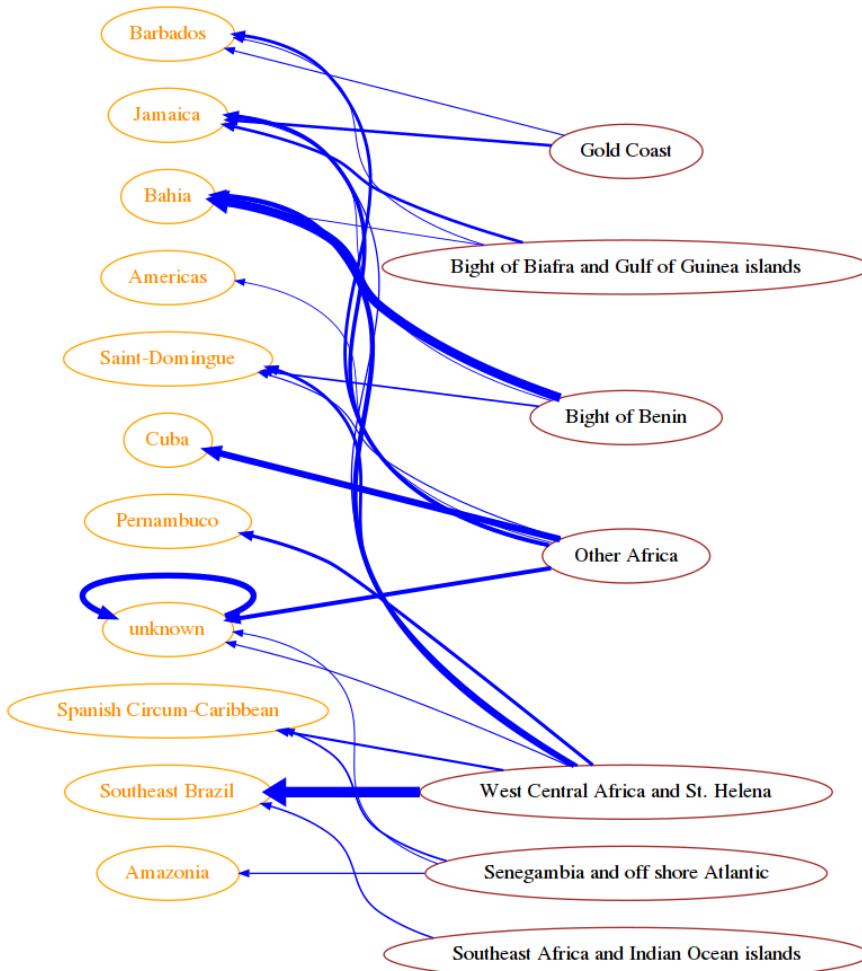
Browse files online<sup>488</sup>

Two centuries ago the shipping of enslaved Africans across the Atlantic was morally indistinguishable from shipping sugar or textiles. This migration experience covers an era of very dramatic shifts in perceptions of good and evil, which provided the Americas with a crucial labor force for their own economic development.

---

<sup>487</sup> <https://en.softpython.org/pandas/pandas3-chal.html>

<sup>488</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/slave-trade>



**Data provider:** The Trans-Atlantic Slave Trade Database. 2020. SlaveVoyages. <https://www.slavevoyages.org>. You are encouraged to explore the dataset with the [very interesting online tool<sup>489</sup>](#) they built, in particular check out the [Maps<sup>490</sup>](#) and [Timelapse<sup>491</sup>](#) tabs.

#### Data license:

- Historical data: The Trans-Atlantic Slave Trade Database. 2020. SlaveVoyages. <https://www.slavevoyages.org> (accessed June 9, 2020). License: Public domain (use restrictions do not apply).
- Imputed data: Estimates. 2020. SlaveVoyages. <https://slavevoyages.org/assessment/estimates> (accessed June 9, 2020). License: Creative Commons Attribution-Noncommercial 3.0 United States License.

<sup>489</sup> <https://www.slavevoyages.org/voyage/database>

<sup>490</sup> <https://www.slavevoyages.org/voyage/database#maps>

<sup>491</sup> <https://www.slavevoyages.org/voyage/database#timelapse>

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
slave-trade-prj
 slave-trade.ipynb
 slave-trade-sol.ipynb
 slave-trade.csv
 region-codes.csv
 soft.py
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `slave-trade.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press **Control + Enter**
- to execute Python code inside a Jupyter cell AND select next cell, press **Shift + Enter**
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press **Alt + Enter**
- If the notebooks look stuck, try to select **Kernel -> Restart**

#### 1. `read_trade`

Each line in `slave-trade.csv` represents a ship voyage from a purchase place to a landing place. Parse it with a csv reader and output a list of dictionaries, one per voyage according to the output excerpt.

- Each ship has a nation flag NATINIMP
- Each voyage has purchase place code MJBYPTIMP and a landing place code MJSPLPTIMP with five digits format xyzvt that indicate a specific town: you **MUST** save more generic codes of the form xyz00 which indicate broader regions.
- **WARNING 1:** convert to int **only** VOYAGEID and YEARAM, leave MJBYPTIMP and MJSPLPTIMP as strings
- **WARNING 2:** some codes in `slave-trade.csv` have a space instead of a number, in those cases save code 00000

```
[1]: import pandas as pd
import numpy as np
df = pd.read_csv('slave-trade.csv', encoding='UTF-8')
df[df.VOYAGEID.isin([1, 2024, 2393, 4190])]
```

```
[1]: VOYAGEID YEARAM NATINIMP MJBYPTIMP MJSPLPTIMP
0 1 1817 Portugal/Brazil 60820 50299
2000 2024 1840 U.S.A. 60615 31399
2361 2393 1829 Spain/Uruguay 60212
4000 4190 1854 U.S.A. 60515 31301
```

**Region labels:** For each location you need to also save its label, which you can find in separate file `region-codes.csv` (load the file with a csv reader)

- **WARNING 1:** in `region-codes.csv` there are **only** codes in format xyz00
- **WARNING 2:** some region codes are missing, in those cases place label 'unknown'

```
[2]: import pandas as pd
dfr = pd.read_csv('region-codes.csv', encoding='UTF-8', dtype=str)
dfr[dfr.Value.isin(['60800', '60600', '31300', '50200', '60500'])]
```

	Value	Region
47	31300	Cuba
84	50200	Bahia
92	60500	Bight of Benin
93	60600	Bight of Biafra and Gulf of Guinea islands
95	60800	Southeast Africa and Indian Ocean islands

**Output excerpt:** (for full output see `expected_db.py`)

```
>>> read_voyages('slave-trade.csv', 'region-codes.csv')
[
{'flag': 'Portugal/Brazil',
 'id': 1,
 'landing_id': '50200',
 'landing_label': 'Bahia',
 'purchase_id': '60800',
 'purchase_label': 'Southeast Africa and Indian Ocean islands',
 'year': 1817},
 {'flag': 'U.S.A.',
 'id': 2024,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60600',
 'purchase_label': 'Bight of Biafra and Gulf of Guinea islands',
 'year': 1840},
 {'flag': 'Spain/Uruguay',
 'id': 2393,
 'landing_id': '00000',
 'landing_label': 'unknown',
 'purchase_id': '60200',
 'purchase_label': 'Sierra Leone',
 'year': 1829},
 {'flag': 'U.S.A.',
 'id': 4190,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60500',
 'purchase_label': 'Bight of Benin',
 'year': 1854},
 .
 .
]
```

Show solutionHide

```
[3]: import csv

def read_voyages(slave_trade_csv, region_codes_csv):
```

(continues on next page)

(continued from previous page)

```
with open(region_codes_csv, encoding='utf-8', newline='') as fregions:
 my_reader = csv.DictReader(fregions, delimiter=',')
 regions_db = {}
 for d in my_reader:
 regions_db[d['Value']] = d['Region']

with open(slave_trade_csv, encoding='utf-8', newline='') as f:
 my_reader = csv.DictReader(f, delimiter=',')
 ret = []
 for d in my_reader:
 voyage = {}
 voyage['id'] = int(d['VOYAGEID'])
 voyage['year'] = int(d['YEARAM'])
 voyage['flag'] = d['NATINIMP']

 if d['MJBYPTIMP'].strip():
 pur_reg = d['MJBYPTIMP'][:3] + '00'
 voyage['purchase_id'] = pur_reg
 if pur_reg in regions_db:
 voyage['purchase_label'] = regions_db[pur_reg]
 else:
 voyage['purchase_label'] = 'unknown'
 else:
 voyage['purchase_id'] = '00000'
 voyage['purchase_label'] = 'unknown'

 if d['MJSLPTIMP'].strip():
 lan_reg = d['MJSLPTIMP'][:3] + '00'
 voyage['landing_id'] = lan_reg
 if lan_reg in regions_db:
 voyage['landing_label'] = regions_db[lan_reg]
 else:
 voyage['landing_label'] = 'unknown'
 else:
 voyage['landing_id'] = '00000'
 voyage['landing_label'] = 'unknown'

 ret.append(voyage)

 return ret

voyages_db = read_voyages('slave-trade.csv', 'region-codes.csv')

print('OUTPUT EXCERPT:')
from pprint import pformat
print('[' + '\n' + ',\n'.join([pformat(voyages_db[vid]) for vid in [0,2000,2361, 4000]]) + '\n' + ',\n' + '.\n' + '.\n'] + ')')

OUTPUT EXCERPT:
[
{'flag': 'Portugal/Brazil',
```

(continues on next page)

(continued from previous page)

```

'id': 1,
'landing_id': '50200',
'landing_label': 'Bahia',
'purchase_id': '60800',
'purchase_label': 'Southeast Africa and Indian Ocean islands',
'year': 1817},
{'flag': 'U.S.A.',
'id': 2024,
'landing_id': '31300',
'landing_label': 'Cuba',
'purchase_id': '60600',
'purchase_label': 'Bight of Biafra and Gulf of Guinea islands',
'year': 1840},
{'flag': 'Spain/Uruguay',
'id': 2393,
'landing_id': '00000',
'landing_label': 'unknown',
'purchase_id': '60200',
'purchase_label': 'Sierra Leone',
'year': 1829},
{'flag': 'U.S.A.',
'id': 4190,
'landing_id': '31300',
'landing_label': 'Cuba',
'purchase_id': '60500',
'purchase_label': 'Bight of Benin',
'year': 1854},
.
.
]

```

&lt;/div&gt;

```
[3]:
import csv

def read_voyages(slave_trade_csv, region_codes_csv):
 raise Exception('TODO IMPLEMENT ME !')

voyages_db = read_voyages('slave-trade.csv', 'region-codes.csv')

print('OUTPUT EXCERPT:')
from pprint import pformat
print('[\n' +''.join([pformat(voyages_db[vid]) for vid in [0,2000,2361, 4000]]) +
 '\n', '\n' .\n .\n]')
```

```
[4]: # TESTING
from pprint import pformat; from expected_db import expected_db
for i in range(0, len(expected_db)):
 if expected_db[i] != voyages_db[i]:
 print('\nERROR at index', i, ':')
 print(' ACTUAL:\n', pformat(voyages_db[i]))
 print(' EXPECTED:\n', pformat(expected_db[i]))
 break
if len(voyages_db) != len(expected_db):
 print("ERROR: Different lengths! voyages_db:", len(voyages_db), " expected_db:",
 len(expected_db))
```

## 2. Deportation

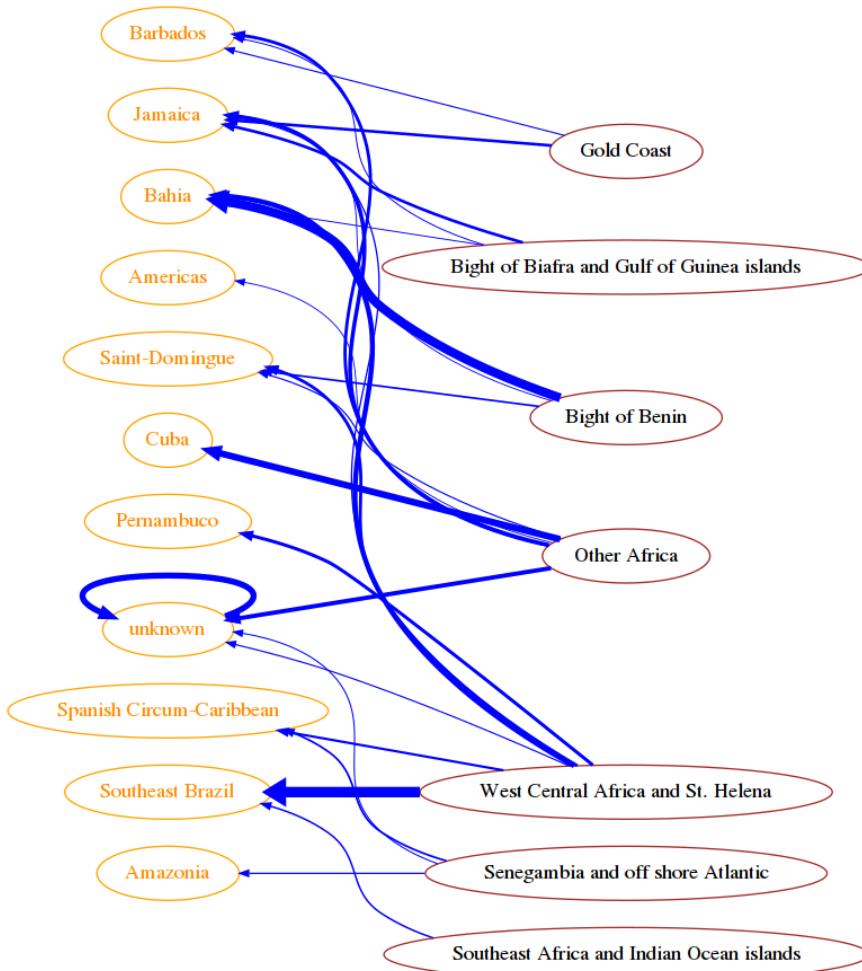
For each link purchase -> landing place, count in how many voyages it was present, then draw result in networkx.

- as edge weight use a normalized value from 0.0 to 1.0 (maximal count found in the graph)
- show only edges with weight greater or equal to `min_weight`
- to display the graph from right to left, set `G.graph['graph'] = {'rankdir': 'RL'}`
- for networkx attributes see this example<sup>492</sup>, make sure to display edges proportionally to the weight

**Example:**

```
>>> show_deportation(voyages_db, 0.09)
COUNTS EXCERPT SOLUTION:
{
 ('60800', '50200') : 48,
 ('60700', '50200') : 1301,
 ('60700', '50400') : 2770,
 ('60800', '50400') : 443,
 ('60900', '50400') : 196,
 .
 .
}
```

<sup>492</sup> <https://en.softpython.org/relational/relational1-intro-sol.html#Fancy-networkx-graphs>



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5] :

```

import networkx as nx
from soft import draw_nx

def show_deportation(voyages, min_weight):

 edges = {}

 for v in voyages_db:
 t = (v['purchase_id'], v['landing_id'])
 if t in edges:
 edges[t] += 1
 else:
 edges[t] = 1

 G = nx.DiGraph()
 G.graph['graph'] = {'rankdir': 'RL'} # force horiz right to left layout

```

(continues on next page)

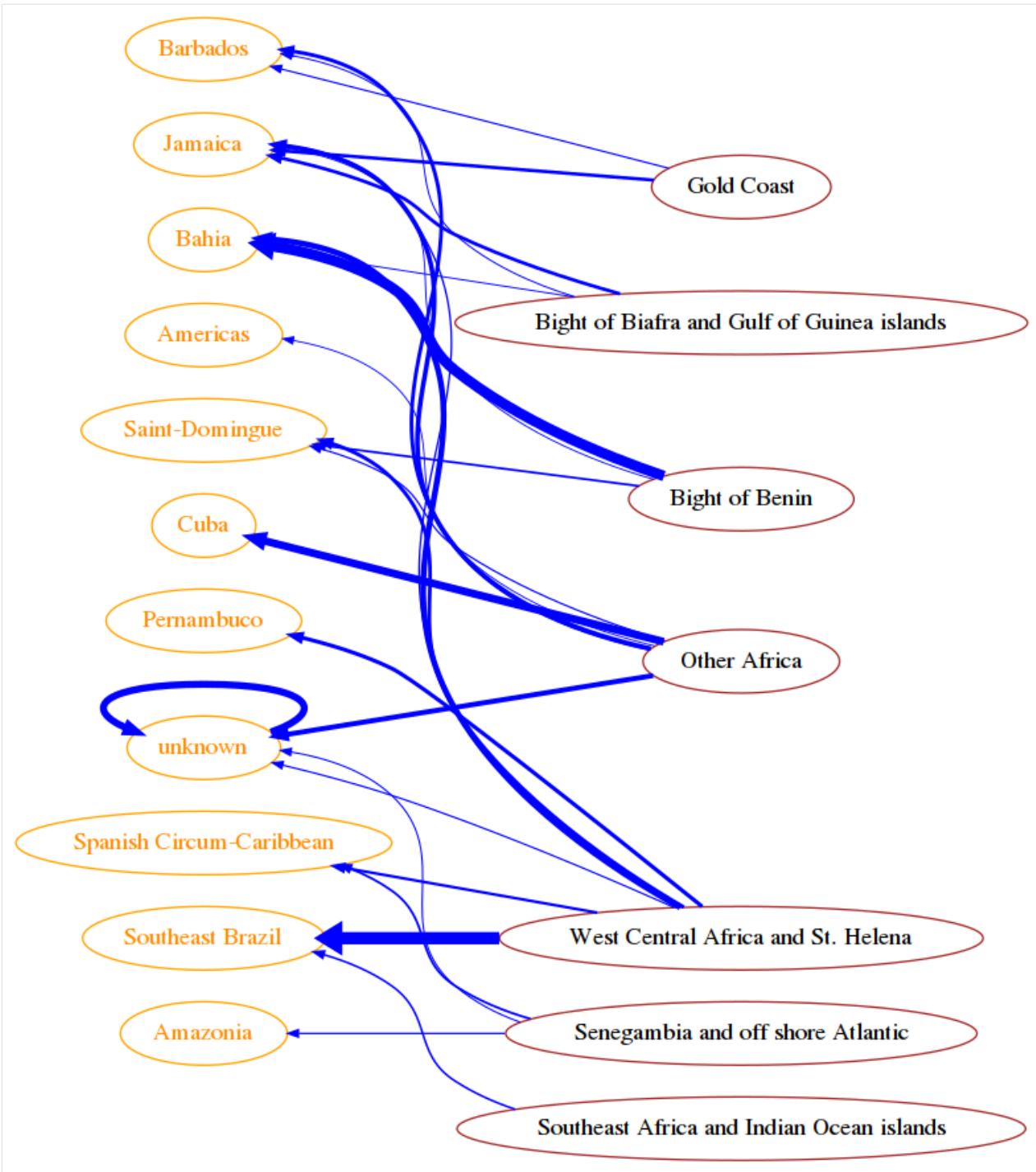
(continued from previous page)

```
mx = max(edges.values())
for v in voyages_db:
 s = edges[(v['purchase_id'], v['landing_id'])]
 r = s / mx
 if r >= min_weight:
 G.add_node(v['purchase_id'], fontcolor='black', color='brown', label=v['purchase_label'])
 G.add_node(v['landing_id'], fontcolor='darkorange', color='orange', label=v['landing_label'])
 G.add_edge(v['purchase_id'], v['landing_id'],
 color="blue",
 penwidth= 7 * r,
 weight=r)

draw_nx(G,
)

show_deportation(voyages_db, 0.09)
#show_deportation(voyages_db, 0.06)

COUNTS EXCERPT SOLUTION:
{
 ('60800', '50200') : 48,
 ('60700', '50200') : 1301,
 ('60700', '50400') : 2770,
 ('60800', '50400') : 443,
 ('60900', '50400') : 196,
 .
 .
}
Image saved to file: expected-deportation-plot.png
```



&lt;/div&gt;

```
[5]:
import networkx as nx
from soft import draw_nx

def show_deportation(voyages, min_weight):
 raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
show_deportation(voyages_db, 0.09)
#show_deportation(voyages_db, 0.06)
```

### 3. The time to stop

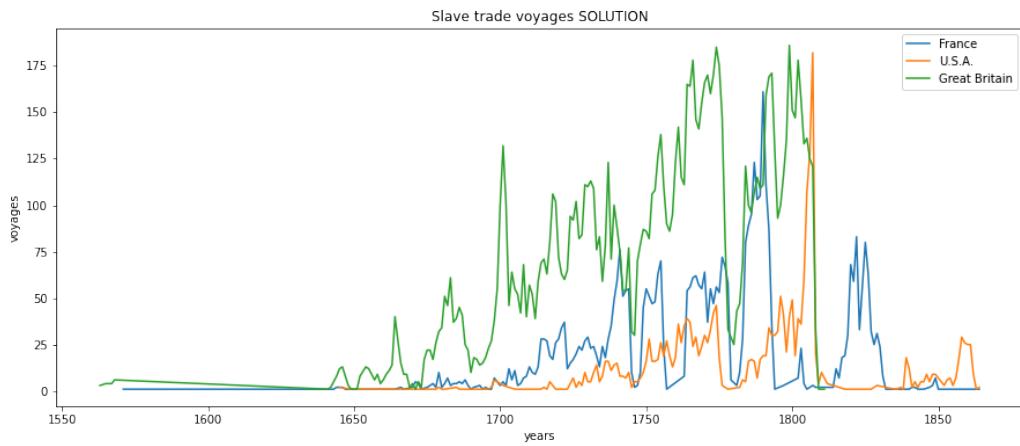
Given a nation flag, plot inside draw\_time the number of voyages per year done by ships belonging to that flag.

**DO NOT** call plt.show nor plt.figure

- we show some counts example but to calculate the data feel free to use any method you want
- **to associate a plot to a label, use i.e.** plt.plot(xs, ys, label='France')

**Example:**

```
>>> fig = plt.figure(figsize=(15, 6))
>>> draw_time(voyages_db, 'France')
>>> draw_time(voyages_db, 'U.S.A.')
>>> draw_time(voyages_db, 'Great Britain')
>>> plt.legend()
>>> plt.show()
France COUNTS EXCERPT SOLUTION:
{
 1816:7,
 1819:30,
 1821:59,
 .
 .
}
U.S.A. COUNTS EXCERPT SOLUTION:
{
 1821:1,
 1827:1,
 1837:2,
 .
 .
}
Great Britain COUNTS EXCERPT SOLUTION:
{
 1810:1,
 1809:1,
 1811:1,
 .
 .
}
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[6]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def draw_time(voyages, flag):

 plt.title("Slave trade voyages SOLUTION")
 plt.xlabel('years')
 plt.ylabel('voyages')

 counts = {}
 for v in voyages:
 if v['flag'] == flag:
 y = v['year']
 if y in counts:
 counts[y] += 1
 else:
 counts[y] = 1

 xs = sorted(counts.keys())
 ys = [counts[x] for x in xs]
 plt.plot(xs, ys, label=flag)

fig = plt.figure(figsize=(15, 6))
draw_time(voyages_db, 'France')
draw_time(voyages_db, 'U.S.A.')
draw_time(voyages_db, 'Great Britain')
plt.legend()

plt.show()
France COUNTS EXCERPT SOLUTION:
{
```

(continues on next page)

(continued from previous page)

```

1816:7,
1819:30,
1821:59,
.

.

}

U.S.A. COUNTS EXCERPT SOLUTION:
{
 1821:1,
 1827:1,
 1837:2,
.

.

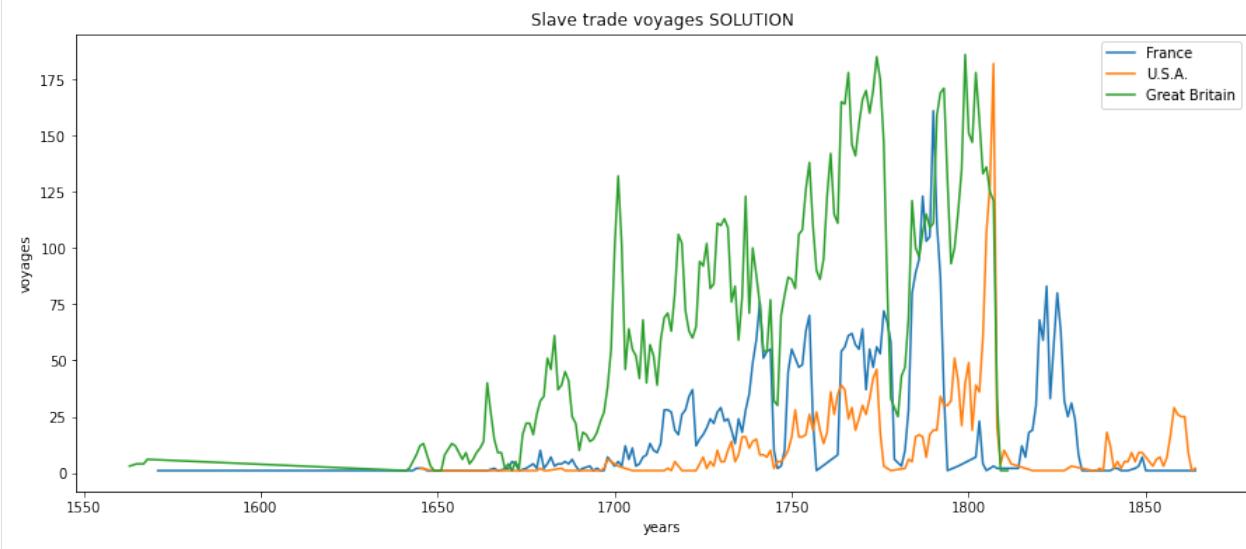
}

Great Britain COUNTS EXCERPT SOLUTION:
{
 1810:1,
 1809:1,
 1811:1,
.

.

}

```



&lt;/div&gt;

```
[6]:
%matplotlib inline
import matplotlib.pyplot as plt

def draw_time(voyages, flag):
 raise Exception('TODO IMPLEMENT ME !')

fig = plt.figure(figsize=(15,6))
draw_time(voyages_db, 'France')
draw_time(voyages_db, 'U.S.A.')
draw_time(voyages_db, 'Great Britain')
plt.legend()
```

(continues on next page)

(continued from previous page)

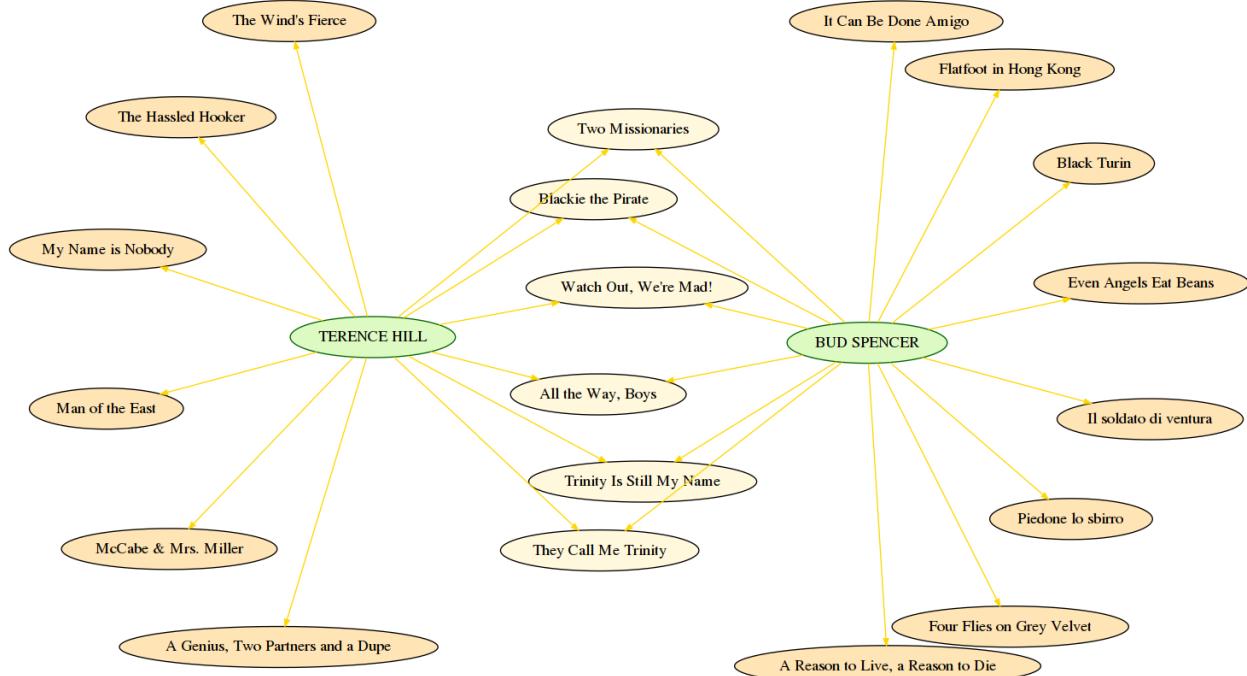
```
plt.show()
```

[ ] :

### 10.3.2 Bud Spencer and Terence Hill movies

## Download worked project

[Browse files online](#)<sup>493</sup>



Among the greatest gifts of Italy to the world we can certainly count Terence Hill and Bud Spencer movies.

Their film career can be found in Wikidata<sup>494</sup>, a project by the Wikimedia foundation which aims to store only machine-readable data, like numbers, strings, and so on interlinked with many references. Each entity in Wikidata has an identifier, for example Terence Hill is the entity Q243430<sup>495</sup> and Bud Spencer is Q221074<sup>496</sup>.

Wikidata can be queried using the SPARQL language: we performed [this query](#)<sup>497</sup> repeated for several languages, and downloaded CSV files (among the many formats which can be chosen). Even if not necessary for the purposes of the

<sup>493</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/bud-spencer-terence-hill-movies>

<sup>494</sup> <https://wikidata.org/>

<sup>495</sup> <http://www.wikidata.org/entity/Q243430>

<sup>496</sup> <http://www.wikidata.org/entity/Q221074>

<sup>497</sup> <https://query.wikidata.org/sparql?format=json> https://query.wikidata.org/sparql?format=json&query=SELECT%20%3Fstar%20%3FstarLabel%20%3Fitem%20%3FitemLabel%20%28MIN%28%3Fdate%29%20AS%20%3FfirstReleased%29%0AWHERE%20%7B%0A%20%20%3Fitem%20wdt%3AP161%20%3Fstar%3B%0A%20%20%20%20%20%20%20wdt%3AP577%20%3Fdate.%0A%20%20%20%20%0A%20%20FILTER%20%28%3Fstar%20%3D%20wdt%3AQ221074%20%7C%7C%20%3Fstar%20%3D%20wdt%3AQ243430%29%20%20%0A%20%20%20%20%20%20%20%20%20%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22en%22.%20%7D%0A%20%20OPTIONAL%20%7B%20%3Fitem%20wdt%3AP18%20%3F\_image.%20%7D%0A%7D%20GROUP%20BY%20%3Fstar%20%3FstarLabel%20%3Fitem%20%3FitemLabel%20%3F\_image%0AORDER%20BY%20%28%3Fdate%29

exercise, you are invited to play a bit with the interface, like trying different visualizations (i.e. try clicking the eye in the middle-left corner and then select Graph) - or see other examples<sup>498</sup>.

**REQUIREMENTS:** Having read Relational data tutorial<sup>499</sup>, which contains also instructions for installing required libraries.

### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
bud-spencer-terence-hill-movies-prj
 bud-spencer-terence-hill-movies.ipynb
 bud-spencer-terence-hill-movies-sol.ipynb
 bud-spencer-terence-hill-movies-de.csv
 bud-spencer-terence-hill-movies-en.csv
 bud-spencer-terence-hill-movies-es.csv
 bud-spencer-terence-hill-movies-it.csv
 soft.py
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook bud-spencer-terence-hill-movies.ipynb
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### The datasets

You are given some CSVs of movies, all having names ending in `-xy.csv`, where `xy` can be a language tag like `it`, `en`, `de`, `es`... They mostly contain the same data except for the movie labels which are in the corresponding language. The final goal will be displaying the network of movies and put in evidence the ones co-starring the famous duo.

Each file row contains info about a single actor starring in a movie. Multiple lines with same movie id will mean multiple actors are co-starring. We can see an excerpt of **first four** lines of english version: notice second movie has id [Q180638](#)<sup>500</sup> and is co-starred by both Bud Spencer and Terence Hill

`star,starLabel,movie,movieLabel,firstReleased`

<http://www.wikidata.org/entity/Q221074>, Bud Spencer, <http://www.wikidata.org/entity/Q116187>, Thieves and Robbers, 1983-02-11T00:00:00Z

(continues on next page)

<sup>498</sup> [https://www.wikidata.org/wiki/Wikidata:SPARQL\\_query\\_service/queries/examples/human](https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples/human)

<sup>499</sup> <https://en.softpython.org/relational/relational1-intro-sol.html#>

<sup>500</sup> <http://www.wikidata.org/entity/Q180638>

(continued from previous page)

```
http://www.wikidata.org/entity/Q221074, Bud Spencer, http://www.wikidata.org/entity/
→Q180638, Odds and Evens, 1978-10-28T00:00:00Z
```

```
http://www.wikidata.org/entity/Q243430, Terence Hill, http://www.wikidata.org/entity/
→Q180638, Odds and Evens, 1978-10-28T00:00:00Z
```

## 1. load

Write a function that given a `filename_prefix` and list of `languages`, parses the corresponding files and RETURNS a **dictionary of dictionaries**, which maps movies id to movies data, in the format as in the excerpt.

- When a label is missing, you will find instead an id like Q3778078: substitute it with empty string (HINT: to recognize ids you might use `is_digit()` method)
- convert date numbers to proper integers
- **DO NOT** put constant ids nor language tags in the code (so no 'Q221074' nor 'it' ...)

**Example** (complete output can be found in `expected_db.py`):

```
>>> load('bud-spencer-terence-hill-movies', ['en', 'it', 'de'])
{
 'Q116187': {
 'actors': [('Q221074', 'Bud Spencer')],
 'first_release': (1983, 2, 11),
 'names': {'de': 'Bud, der Ganovenschreck',
 'en': 'Thieves and Robbers',
 'it': 'Cane e gatto'}
 }
 'Q180638': {
 'actors': [('Q221074', 'Bud Spencer'), ('Q243430', 'Terence Hill')],
 'first_release': (1978, 10, 28),
 'names': {'de': 'Zwei sind nicht zu bremsen',
 'en': 'Odds and Evens',
 'it': 'Pari e dispari'}
 }
 'Q231967': {
 'actors': [('Q221074', 'Bud Spencer'), ('Q243430', 'Terence Hill')],
 'first_release': (1981, 1, 1),
 'names': {'de': 'Zwei Asse trümpfen auf',
 'en': 'A Friend Is a Treasure',
 'it': 'Chi trova un amico, trova un tesoro'}
 }
 .
 .
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[2]:

```
import csv
```

(continues on next page)

(continued from previous page)

```

def load(filename_prefix, languages):

 first_lang = True
 ret = {}
 for lang in languages:
 fn = '%s-%s.csv' % (filename_prefix, lang)
 #print("Reading", fn)
 with open(fn, encoding='utf-8', newline='') as f:

 my_reader = csv.DictReader(f, delimiter=',')
 for d in my_reader:
 movie_id = d['movie'][len('http://www.wikidata.org/entity/'):]
 actor = (d['star'][len('http://www.wikidata.org/entity/'):], d[
 'starLabel'])

 if d['movieLabel'][0] == 'Q' and d['movieLabel'][1].isdigit():
 #print('FOUND MISSING LABEL', d['movieLabel'], 'FOR', lang)
 movie_label_fixed = ''
 else:
 movie_label_fixed = d['movieLabel']

 if first_lang:
 if movie_id in ret:
 ret[movie_id]['actors'].append(actor)
 else:
 ret[movie_id] = {'actors': [actor],
 'names' : {lang: movie_label_fixed},
 'first_release' : tuple([int(s) for s in d[
 'firstReleased'][:10].split('-')])
 }
 else:
 ret[movie_id]['names'][lang] = movie_label_fixed

 #print("Found", len(ret), "movies")
 first_lang = False

 return ret

movies_db = load('bud-spencer-terence-hill-movies', ['en', 'it', 'de'])

#movies_db = load('bud-spencer-terence-hill-movies', ['es', 'en', 'de', 'it'])
movies_db

```

EXERPT:

```
{
 'Q116187': {
 'actors': [('Q221074', 'Bud Spencer')],
 'first_release': (1983, 2, 11),
 'names': {'de': 'Bud, der Ganovenschreck',
 'en': 'Thieves and Robbers',
 'it': 'Cane e gatto'}
 }
}
```

(continues on next page)

(continued from previous page)

```

'Q180638': {
 'actors': [('Q221074', 'Bud Spencer'), ('Q243430', 'Terence Hill')],
 'first_release': (1978, 10, 28),
 'names': {'de': 'Zwei sind nicht zu bremsen',
 'en': 'Odds and Evens',
 'it': 'Pari e dispari'}
}
'Q231967': {
 'actors': [('Q221074', 'Bud Spencer'), ('Q243430', 'Terence Hill')],
 'first_release': (1981, 1, 1),
 'names': {'de': 'Zwei Asse trumpfen auf',
 'en': 'A Friend Is a Treasure',
 'it': 'Chi trova un amico, trova un tesoro'}
}
.
.
.
}
```

&lt;/div&gt;

```
[2]: import csv

def load(filename_prefix, languages):
 raise Exception('TODO IMPLEMENT ME !')

movies_db = load('bud-spencer-terence-hill-movies', ['en', 'it', 'de'])

#movies_db = load('bud-spencer-terence-hill-movies', ['es', 'en', 'de', 'it'])
movies_db
```

```
[3]: # TESTING
from pprint import pformat; from expected_movies_db import expected_movies_db
for sid in expected_movies_db.keys():
 if sid not in movies_db: print('\nERROR: MISSING movie', sid); break
 for k in expected_movies_db[sid]:
 if k not in movies_db[sid]:
 print('\nERROR at movie', sid, '\n\n MISSING key:', k); break
 if expected_movies_db[sid][k] != movies_db[sid][k]:
 print('\nERROR at movie', sid, 'key:', k)
 print(' ACTUAL:\n', pformat(movies_db[sid][k]))
 print(' EXPECTED:\n', pformat(expected_movies_db[sid][k]))
 break
if len(movies_db) > len(expected_movies_db):
 print('ERROR! There are more movies than expected!')
 print(' ACTUAL:\n', len(movies_db))
 print(' EXPECTED:\n', len(expected_movies_db))
```

### 2. save\_table

Write a function that given a movies db and a list of languages, writes a new file merged.csv

- separate actor names with and
- use only the year as date
- file must be formatted like this

```
movie_id,name en,name it,first_release,actors
Q116187,Thieves and Robbers,Cane e gatto,1983,Bud Spencer
Q180638,Odds and Evens,Pari e dispari,1978,Bud Spencer and Terence Hill
```

Complete expected file is in `expected-merged.csv`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[5]: import csv

def save_table(movies, languages):

 with open('merged.csv', 'w', encoding='utf-8', newline='') as csv_out:
 my_writer = csv.writer(csv_out, delimiter=',')

 header = ['movie_id'] + ['name ' + lan for lan in languages] + ['first_release']
 my_writer.writerow(header)

 for movie_id in movies:
 movie = movies[movie_id]
 actors_names = ' and '.join([actor[1] for actor in movie['actors']])
 row = [movie_id]
 row.extend([movie['names'][language] for language in languages])
 row.extend([movie['first_release'][0], actors_names])

 my_writer.writerow(row)
 print('saved file to merged.csv')

 save_table(movies_db, ['en', 'it'])
 #save_table(movies_db, ['de'])

 saved file to merged.csv
```

</div>

```
[5]: import csv

def save_table(movies, languages):
 raise Exception('TODO IMPLEMENT ME !')

save_table(movies_db, ['en', 'it'])
#save_table(movies_db, ['de'])
```

```
saved file to merged.csv
```

```
[6]: # TESTING
with open('expected-merged.csv',encoding='utf-8', newline='') as expected_f:
 with open('merged.csv',encoding='utf-8', newline='') as f:
 expected_reader = csv.reader(expected_f, delimiter=',')
 reader = csv.reader(f, delimiter=',')
 i = 0
 for expected_row in expected_reader:
 try:
 row = next(reader)
 except:
 print('ERROR at row', i, ': ACTUAL rows are less than EXPECTED!')
 break
 for j in range(len(expected_row)):
 if expected_row[j] != row[j]:
 print('ERROR at row', i, ' cell index', j)
 print(row)
 print('\nACTUAL :', row[j])
 print('\nEXPECTED:', expected_row[j])
 break
 i += 1
```

### 3. show\_graph

Display a NetworkX graph of movies (see examples<sup>501</sup>) from since\_year (included) to until\_year (included), in the given language

- display actor names as capitalized
- display co-starred movies, non co-starred movies and actors with different colors by setting node attributes style='filled' and i.e. fillcolor='green' (see some color names<sup>502</sup>)

**DO NOT** use labels as node ids

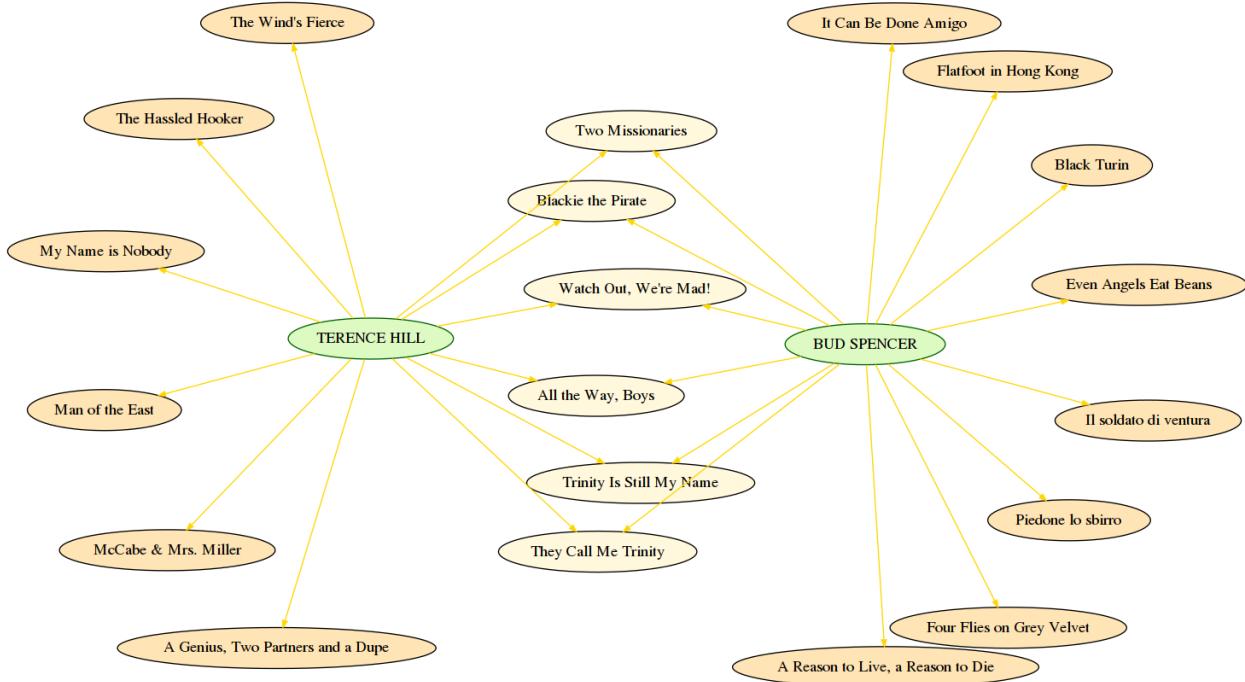
**DO NOT** write constants in your code, so no 'Terence' nor 'TERENCE'...

#### Example 1

```
>>> show_graph(movies_db, 1970, 1975, 'en')
```

<sup>501</sup> <https://en.softpython.org/relational/relational1-intro-sol.html#Fancy-networkx-graphs>

<sup>502</sup> <https://www.w3.org/wiki/CSS/Properties/color/keywords>



## Example 2

```
>>> show_graph(movies_db, 1970, 1974, 'it')
```



```
Show solution<div class="jupman-sol jupman-sol-code" style="display:none">
```

[7]:

```

import networkx as nx
from soft import draw_nx

def show_graph(movies, since_year, until_year, language):

 G = nx.DiGraph()
 G.graph['graph'] = { 'layout':'neato'} # don't delete these!

 for movie_id in movies:
 movie = movies[movie_id]
 if movie['first_release'][0] >= since_year and movie['first_release'][0] <=
until_year:
 if len(movie['actors']) > 1:
 fillcolor = 'cornsilk'
 else:
 fillcolor = 'moccasin'
 G.add_node(movie_id, label=movie['names'][language], fillcolor=fillcolor,
color='black', style='filled', fontcolor='black')
 for actor in movie['actors']:
 G.add_node(actor[0], fillcolor='#defac3', fontcolor='black', color=
'darkgreen', style='filled', label=actor[1].upper())
 # IMPORTANT: edges connect movie_ids , NOT labels !!!!
 G.add_edge(actor[0], movie_id, color='gold')

 draw_nx(G, save_to='expected-%s-%s-%s.png' % (since_year,until_year,language))

show_graph(movies_db, 1970, 1975, 'en')

```

Image saved to file: expected-1970-1975-en.png



&lt;/div&gt;

[7]:

```
import networkx as nx
from soft import draw_nx

def show_graph(movies, since_year, until_year, language):

 G = nx.DiGraph()
 G.graph['graph']= { 'layout':'neato'} # don't delete these!

 raise Exception('TODO IMPLEMENT ME !')

show_graph(movies_db, 1970, 1975, 'en')
```

[8]:

```
show_graph(movies_db, 1970, 1974, 'it')
```

[ ]:

### 10.3.3 Bus network

#### Download worked project

Browse files online<sup>503</sup>

In this worked project we will visualize intercity bus network in GTFS format. Original data was split in several files which we merged into dataset [network-short.csv](#).

Data source: [dati.trentino.it](#)<sup>504</sup>, MITT service, released under Creative Commons Attribution 4.0<sup>505</sup> licence.

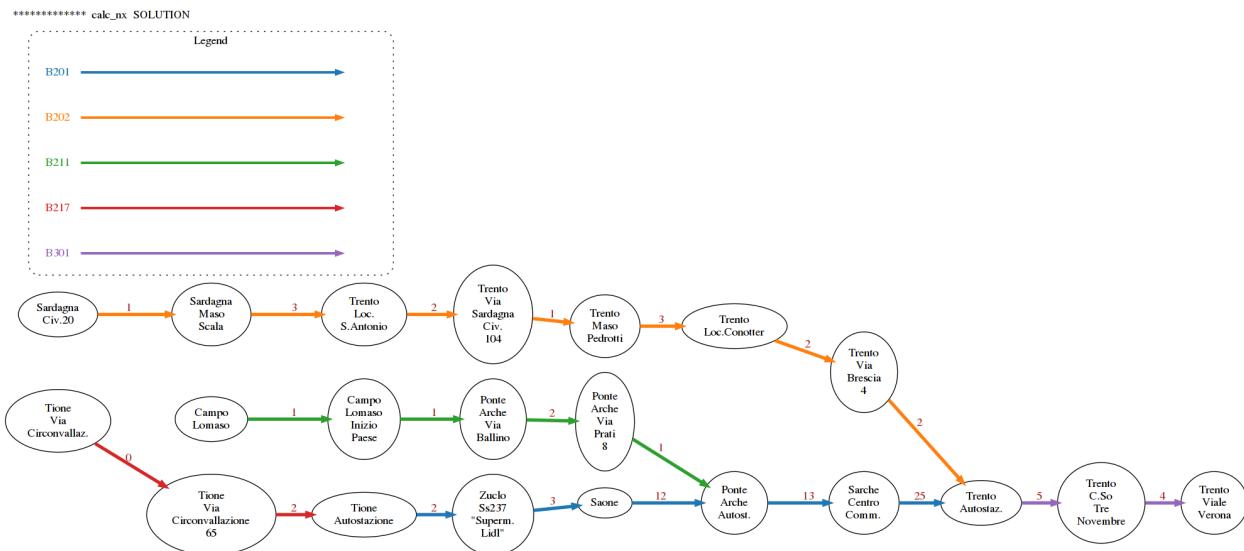
**REQUIREMENTS: Having read Relational data tutorial<sup>506</sup>, which contains also instructions for installing required libraries.**

<sup>503</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/bus-network>

<sup>504</sup> <https://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs>

<sup>505</sup> <http://creativecommons.org/licenses/by/4.0/deed.it>

<sup>506</sup> <https://en.softpython.org/relational/relational1-intro-sol.html#>



## What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
bus-network-prj
 bus-network.ipynb
 bus-network-sol.ipynb
 soft.py
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `bus-network.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

### Introduction

To visualize data, we will use `networkx`<sup>507</sup> library. Let's first see an example on how to do it:

```
[2]: import networkx as nx
from soft import draw_nx

Gex = nx.DiGraph()

we can force horizontal layout like this:

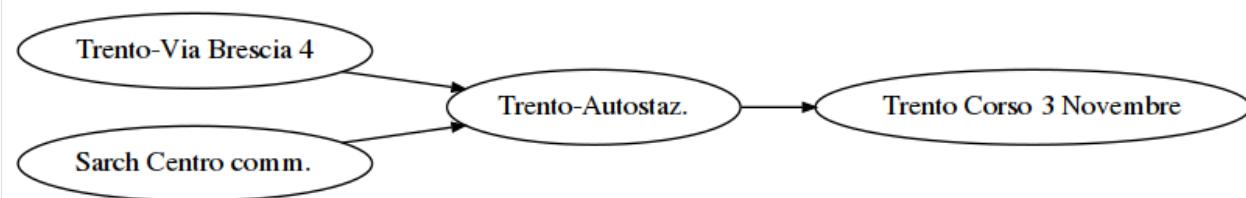
Gex.graph['graph'] = {
 'rankdir': 'LR',
}

When we add nodes, we can identify them with an identifier like the
stop_id which is separate from the label, because in some unfortunate
case two different stops can share the same label.

Gex.add_node('1', label='Trento-Autostaz.',
 color='black', fontcolor='black')
Gex.add_node('723', label='Trento-Via Brescia 4',
 color='black', fontcolor='black')
Gex.add_node('870', label='Sarch Centro comm.',
 color='black', fontcolor='black')
Gex.add_node('1180', label='Trento Corso 3 Novembre',
 color='black', fontcolor='black')

IMPORTANT: edges connect stop_ids , NOT labels !!!!
Gex.add_edge('870', '1')
Gex.add_edge('723', '1')
Gex.add_edge('1', '1180')

function defined in sciprog.py :
draw_nx(Gex)
```



### Colors and additional attributes

Since we have a bus stop netowrk, we might want to draw edges according to the route they represent. Here we show how to do it only with the edge from *Trento-Autostaz* to *Trento Corso 3 Novembre*:

```
[3]: # we can retrieve an edge like this:

edge = Gex['1']['1180']
```

(continues on next page)

<sup>507</sup> <https://networkx.github.io/>

(continued from previous page)

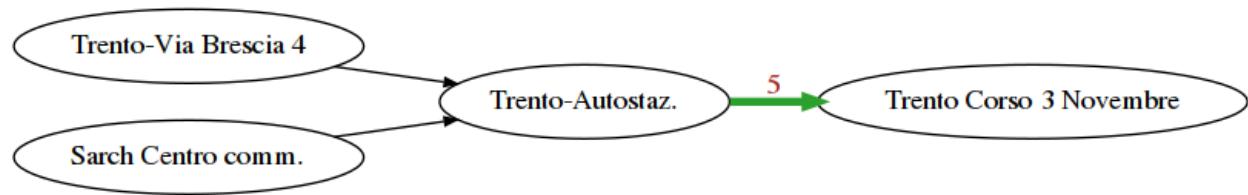
```
and set attributes, like these:

edge['weight'] = 5 # it takes 5 minutes to go from Trento-Autostaz
 # to Trento Corso 3 Novembre
edge['label'] = str(5) # the label is a string

edge['color'] = '#2ca02c' # we can set some style for the edge, such as color
edge['penwidth']= 4 # and thickness

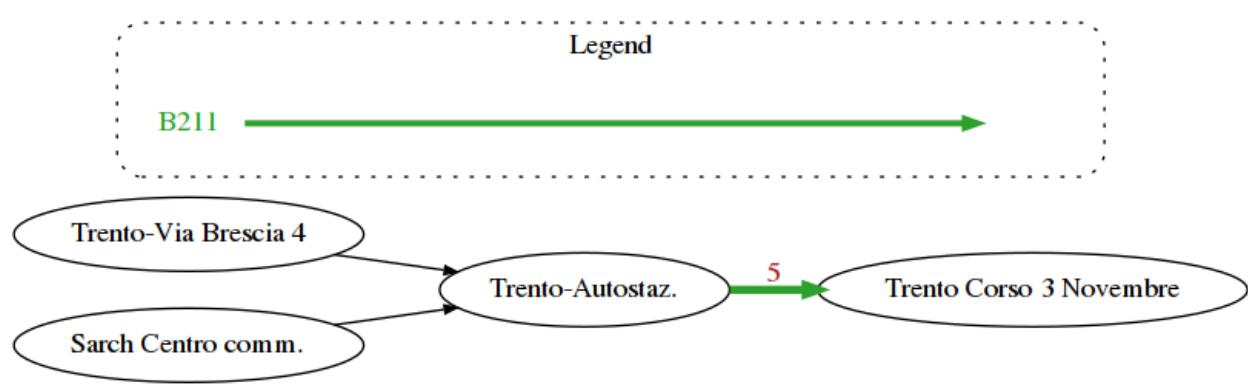
edge['route_short_name'] = 'B301' # we can add any attribute we want,
 # Note these custom ones won't show in the graph
```

```
draw_nx(Gex)
```



To be more explicit, we can also add a legend this way:

```
[4]: draw_nx(Gex, [{"color': '#2ca02c', 'label': 'B211'}])
```



```
[5]: # Note an edge is a simple dictionary:
```

```
print(edge)

{'weight': 5, 'label': '5', 'color': '#2ca02c', 'penwidth': 4, 'route_short_name':
→'B301'}
```

**load\_stops**

To load `network-short.csv`, we provide this function:

```
[6]: def load_stops():
 """Loads file data and RETURN a list of dictionaries with the stop times
 """

 import csv
 with open('network-short.csv', newline='', encoding='UTF-8') as csvfile:
 reader = csv.DictReader(csvfile)
 lst = []
 for d in reader:
 lst.append(d)
 return lst
```

```
[7]: stops = load_stops()

#IMPORTANT: NOTICE *ALL* VALUES ARE *STRINGS* !!!!!!!!!

stops[0:2]

[7]: [OrderedDict([('3',
 ('route_id', '76'),
 ('agency_id', '12'),
 ('route_short_name', 'B202'),
 ('route_long_name',
 'Trento-Sardagna-Candriai-Vaneze-Vason-Viote'),
 ('route_type', '3'),
 ('service_id', '22018091220190621'),
 ('trip_id', '0002402742018091220190621'),
 ('trip_headsign', 'Trento-Autostaz.'),
 ('direction_id', '0'),
 ('arrival_time', '06:27:00'),
 ('departure_time', '06:27:00'),
 ('stop_id', '5025'),
 ('stop_sequence', '4'),
 ('stop_code', '2620VE'),
 ('stop_name', 'Sardagna Civ.20'),
 ('stop_desc', ''),
 ('stop_lat', '46.073125'),
 ('stop_lon', '11.093579'),
 ('zone_id', '2620.0'))),
 OrderedDict([('4',
 ('route_id', '76'),
 ('agency_id', '12'),
 ('route_short_name', 'B202'),
 ('route_long_name',
 'Trento-Sardagna-Candriai-Vaneze-Vason-Viote'),
 ('route_type', '3'),
 ('service_id', '22018091220190621'),
 ('trip_id', '0002402742018091220190621'),
 ('trip_headsign', 'Trento-Autostaz.'),
 ('direction_id', '0'),
 ('arrival_time', '06:28:00'),
 ('departure_time', '06:28:00'),
 ('stop_id', '843'))]]
```

(continues on next page)

(continued from previous page)

```
('stop_sequence', '5'),
('stop_code', '2620MS'),
('stop_name', 'Sardagna-Maso Scala'),
('stop_desc', ''),
('stop_lat', '46.069871'),
('stop_lon', '11.097749'),
('zone_id', '2620.0'))]
```

## 1. extract\_routes

Implement a function that extracts all route\_short\_name from the stops list and RETURNS an alphabetically sorted list of them, without duplicates (see example)

Example:

```
>>> stops = load_stops()
>>> extract_routes(stops)
['B201', 'B202', 'B211', 'B217', 'B301']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8]:

```
import networkx as nx
from soft import draw_nx

def extract_routes(stps):

 s = set()
 for diz in stps:
 s.add(diz['route_short_name'])
 ret = list(s)
 ret.sort()
 return ret

extract_routes(stops)
```

[8]:

```
['B201', 'B202', 'B211', 'B217', 'B301']
```

</div>

[8]:

```
import networkx as nx
from soft import draw_nx

def extract_routes(stps):
 raise Exception('TODO IMPLEMENT ME !')

extract_routes(stops)
```

### 2. to\_int\_min

Implement a function that takes a time string in the format like 08:27:42 and RETURN the time since midnight in minutes, ignoring the seconds (es 507)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[9]:

```
def to_int_min(time_string):

 hours = int(time_string[0:2])
 mins = int(time_string[3:5])
 return (hours * 60 + mins)

to_int_min('08:27:42')
```

[9]:

```
507
```

</div>

[9]:

```
def to_int_min(time_string):
 raise Exception('TODO IMPLEMENT ME !')
to_int_min('08:27:42')
```

### 3. get\_legend\_edges

If you have n routes numbered from 0 to n-1, and you want to assign to each of them a different color, we provide this function:

```
[10]: def get_color(i, n):
 """ RETURN the i-th color chosen from n possible colors, in
 hex format (i.e. #ff0018).

 - if i < 0 or i >= n, raise ValueError
 """
 if n < 1:
 raise ValueError("Invalid n: %s" % n)
 if i < 0 or i >= n:
 raise ValueError("Invalid i: %s" % i)

 #HACKY, just for matplotlib < 3
 lst = ['#1f77b4',
 '#ff7f0e',
 '#2ca02c',
 '#d62728',
 '#9467bd',
 '#8c564b',
 '#e377c2',
 '#7f7f7f',
 '#bcbd22',
 '#17becf']

 return lst[i % 10]
```

```
[11]: get_color(4,5)
[11]: '#9467bd'
```

Now implement a function that RETURNS a list of dictionaries, where each dictionary represent a route with label and associated color. Dictionaries are in the order returned by extract\_routes() function.

**Example:**

```
>>> get_legend_edges()
[{'label': 'B201', 'color': '#1f77b4'},
 {'label': 'B202', 'color': '#ff7f0e'},
 {'label': 'B211', 'color': '#2ca02c'},
 {'label': 'B217', 'color': '#d62728'},
 {'label': 'B301', 'color': '#9467bd'}]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[12]: def get_legend_edges():

 legend_edges = []
 i = 0
 routes = extract_routes(stops)

 for route_short_name in routes:
 legend_edges.append({
 'label': route_short_name,
 'color': get_color(i, len(routes))
 })
 i += 1
 return legend_edges
```

```
get_legend_edges()
```

```
[12]: [{}'label': 'B201', 'color': '#1f77b4'},
 {'label': 'B202', 'color': '#ff7f0e'},
 {'label': 'B211', 'color': '#2ca02c'},
 {'label': 'B217', 'color': '#d62728'},
 {'label': 'B301', 'color': '#9467bd'}]
```

```
</div>
```

```
[12]: def get_legend_edges():
 raise Exception('TODO IMPLEMENT ME !')

get_legend_edges()
```

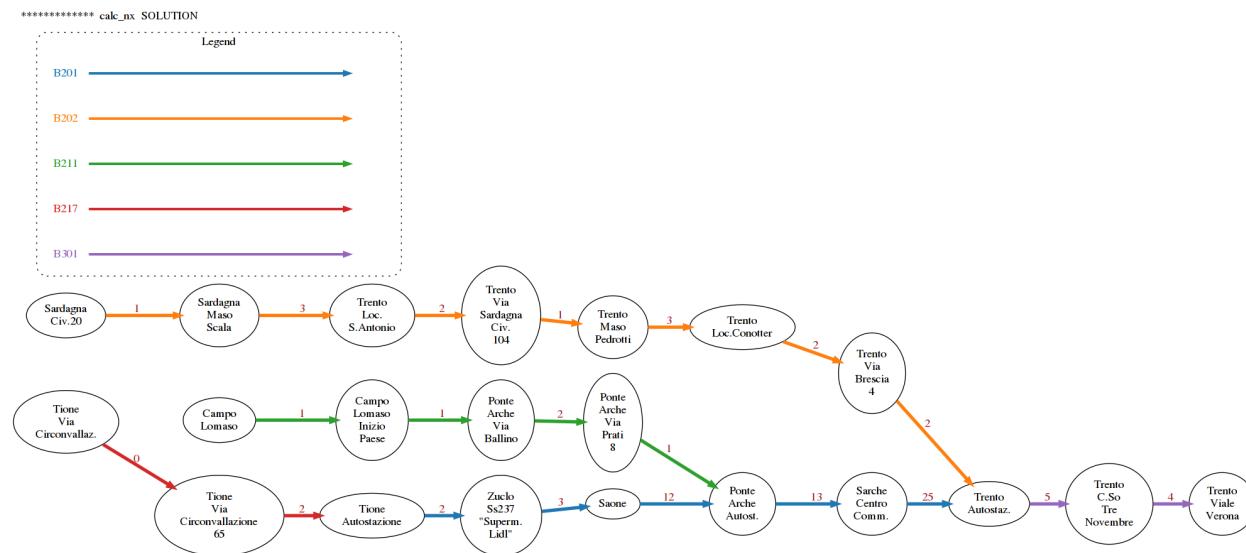
#### 4. calc\_nx

Implement function `calc_nx` which RETURN a NetworkX DiGraph representing the bus stop network

- To keep things simple, we suppose routes NEVER overlap (no edge is ever shared by two routes), so we need only a DiGraph and not a MultiGraph
- as label for nodes, use the stop\_name, and try to format it nicely.
- as 'weight' for the edges, use the time in minutes between one stop and the next one
- as custom property, add `route_short_name`
- as 'color' for the edges, use the color given by provided `get_color(i, n)` function
- as 'penwidth' for edges, set 4

**IMPORTANT:** notice stops are already ordered by arrival\_time, this makes it easy to find edges !

**HINT:** to make sure you're on the right track, try first to represent one single route, like B202



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);>Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[13] :

```
def calc_nx(stops):

 G = nx.DiGraph()

 G.graph['graph'] = {
 'rankdir': 'LR', # horizontal layout ,
 }

 G.name = '***** calc_nx SOLUTION '

 routes = extract_routes(stops)
```

(continues on next page)

(continued from previous page)

```

i = 0

for route_short_name in routes:

 prev_diz = None

 for diz in stops:

 if diz['route_short_name'] == route_short_name:

 G.add_node(diz['stop_id'],
 label=diz['stop_name'].replace(' ', '\n').replace('-', '\n'
→') ,
 color='black',
 fontcolor='black')

 if prev_diz:

 G.add_edge(prev_diz['stop_id'], diz['stop_id'])
 delta_time = to_int_min(diz['arrival_time']) - to_int_min(prev_
→diz['arrival_time']))

 edge = G[prev_diz['stop_id']][diz['stop_id']]
 edge['weight'] = delta_time
 edge['label'] = str(delta_time)

 edge['route_short_name'] = route_short_name

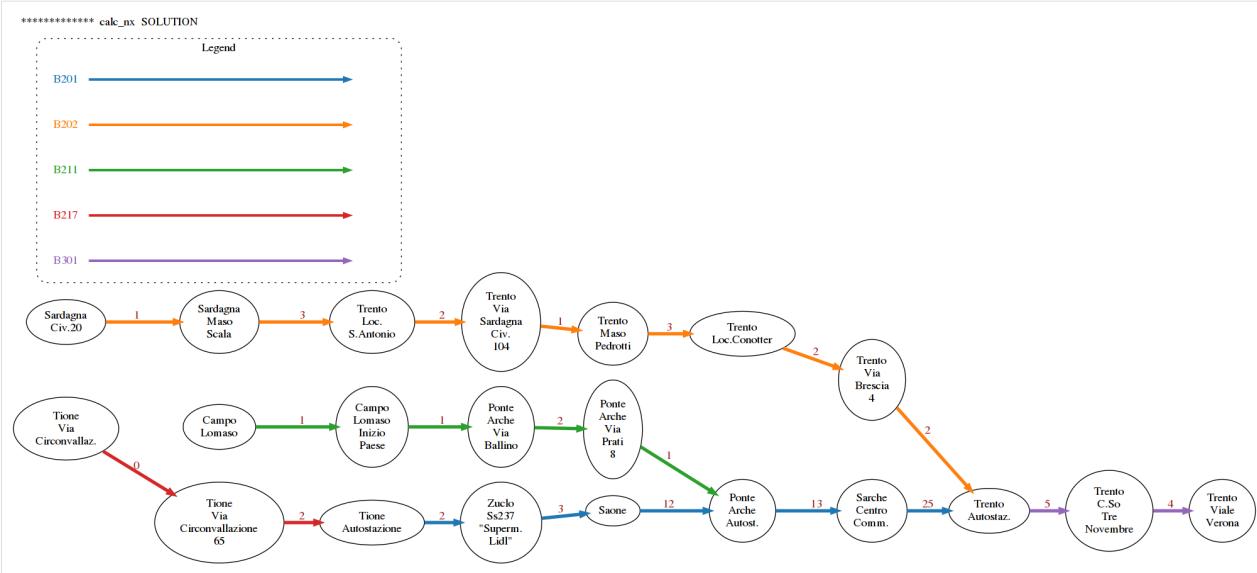
 edge['color'] = get_color(i, len(routes))
 edge['penwidth']= 4

 prev_diz = diz
 i += 1
 return G

G = calc_nx(stops)

draw_nx(G, get_legend_edges(),
)
Image saved to file: expected-network.png

```



</div>

[13]:

```
def calc_nx(stops):
 raise Exception('TODO IMPLEMENT ME !')

G = calc_nx(stops)

draw_nx(G, get_legend_edges(),

)

```

## 5. Hubs

A *hub* is a node that allows to switch route, that is, it is touched by *at least* two different routes.

For example, *Trento-Autostaz* is touched by three routes, which is more than one, so it is a hub. Let's examine the node - we know it has `stop_id='1'`:

```
[14]: G.node['1']
```

```
[14]: {'label': 'Trento\nAutostaz.', 'color': 'black', 'fontcolor': 'black'}
```

If we examine its `in_edges`, we find it has incoming edges from `stop_id` '723' and '870', which represent respectively *Trento Via Brescia* and *Sarche Centro Commerciale*:

```
[15]: G.in_edges('1')
```

```
[15]: InEdgeDataView([('870', '1'), ('723', '1')])
```

If you get a View object, if needed you can easily transform to a list:

```
[16]: list(G.in_edges('1'))
[16]: [('870', '1'), ('723', '1')]
```

```
[17]: G.node['723']
[17]: {'label': 'Trento\nVia\nBrescia\n4', 'color': 'black', 'fontcolor': 'black'}
```

```
[18]: G.node['870']
[18]: {'label': 'Sarche\nCentro\nComm.', 'color': 'black', 'fontcolor': 'black'}
```

There is only an outgoing edge toward *Trento Corso 3 Novembre*:

```
[19]: G.out_edges('1')
[19]: OutEdgeDataView([('1', '1108')])
```

```
[20]: G.node['1108']
[20]: {'label': 'Trento\nC.So\nTre\nNovembre',
 'color': 'black',
 'fontcolor': 'black'}
```

If, for example, we want to know the `route_id` of this outgoing edge, we can access it this way:

```
[21]: G['1']['1108']
[21]: {'weight': 5,
 'label': '5',
 'route_short_name': 'B301',
 'color': '#9467bd',
 'penwidth': 4}
```

If you want to change the color attribute of the node '1', you can write like this:

```
[22]: G.node['1']['color'] = 'red'
G.node['1']['fontcolor'] = 'red'
```

## Implement color\_hubs

Implement a function which prints the hubs in the graph G as text, and then draws the graph with the hubs colored in red.

**NOTE:** you don't need to recalculate the graph, just set the relevant nodes color to red

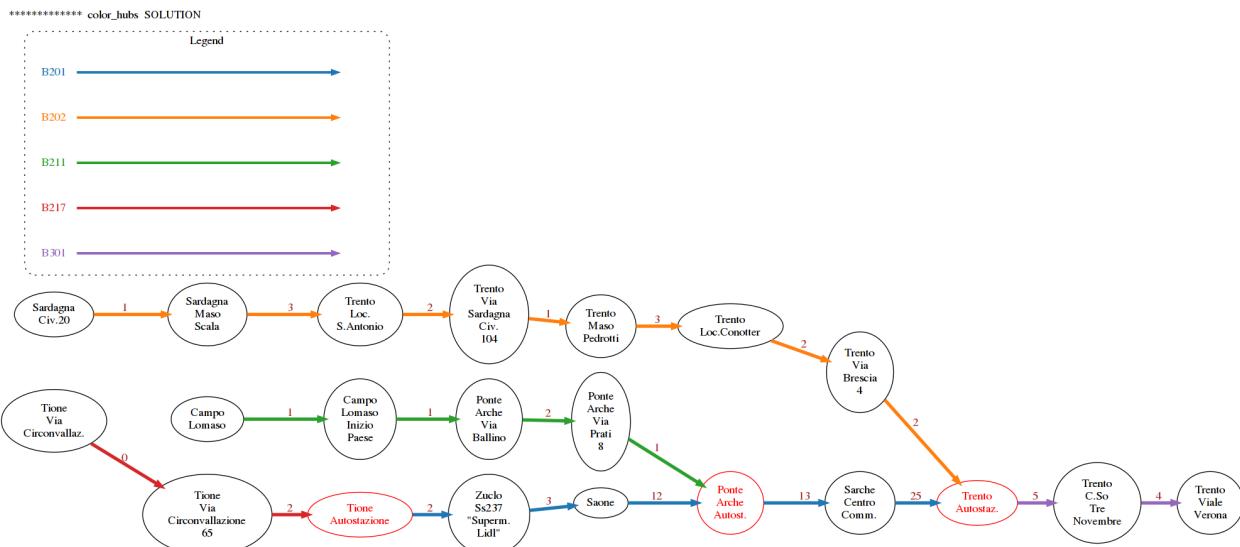
**Example:**

```
>>> color_hubs(G)
SOLUTION: The hubs are:

stop_id:757
Tione
Autostazione

stop_id:742
Ponte
Arche
Autost.

stop_id:1
Trento
Autostaz.
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[23] :

```

def color_hubs(G):

 G.name = '***** color_hubs SOLUTION'

 hubs = []
 for node in G.nodes():
 edges = list(G.in_edges(node)) + list(G.out_edges(node))
 route_short_names = set()
 for edge in edges:
 route_short_names.add(G[edge[0]][edge[1]]['route_short_name'])
 if len(route_short_names) > 1:
 hubs.append(node)

 print("SOLUTION: The hubs are:")
 print()

 for hub in hubs:
 print("stop_id:%s\n%s\n" % (hub, G.node[hub]['label']))
 G.node[hub]['color'] = 'red'
 G.node[hub]['fontcolor'] = 'red'

 draw_nx(G, legend_edges=get_legend_edges())

color_hubs(G)

```

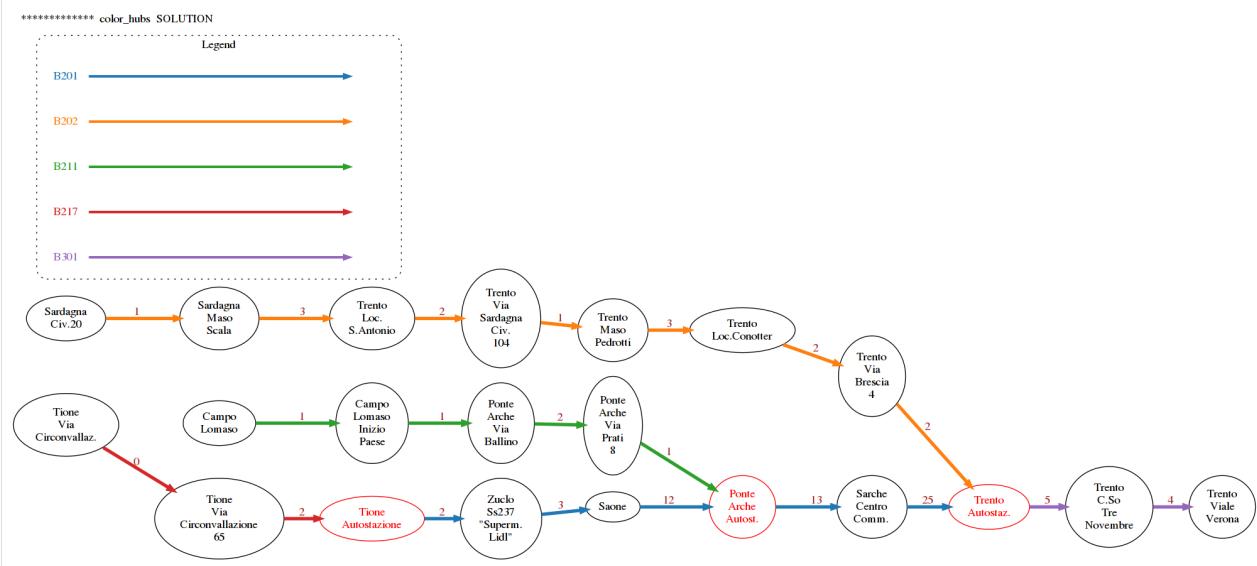
SOLUTION: The hubs are:

```
stop_id:757
Tione
Autostazione
```

```
stop_id:742
Ponte
Arche
Autost.
```

```
stop_id:1
Trento
Autostaz.
```

Image saved to file: expected-hubs.png



</div>

[23] :

```
def color_hubs (G) :
 raise Exception ('TODO IMPLEMENT ME !')

color_hubs (G)
```

## 6. plot\_timings

To extract bus times from G, use this:

```
[24]: G.edges()
[24]: OutEdgeView([('757', '746'), ('746', '857'), ('857', '742'), ('742', '870'), ('870',
 ↪ '1'), ('1', '1108'), ('5025', '843'), ('843', '842'), ('842', '3974'), ('3974', '841'
 ↪), ('841', '881'), ('881', '723'), ('723', '1'), ('1556', '4392'), ('4392', '4391'
 ↪), ('4391', '4390'), ('4390', '742'), ('829', '3213'), ('3213', '757'), ('1108',
 ↪ '1109')])
```

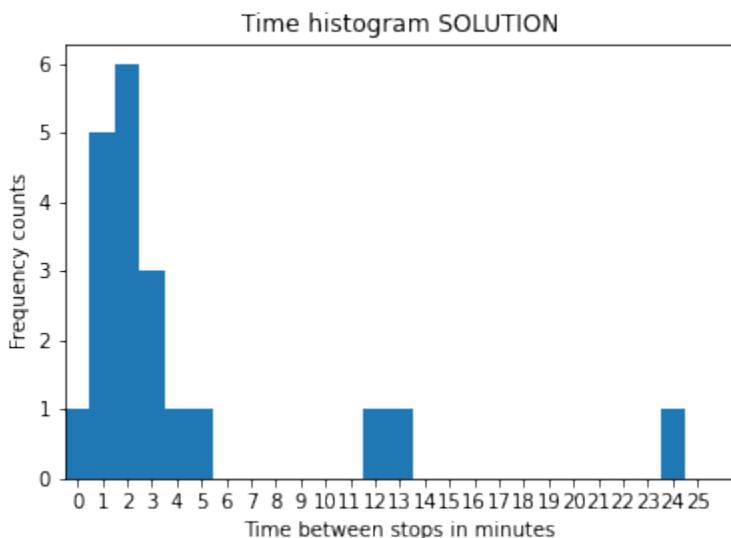
If you get a View, you can iterate through the sequence like it were a list

To get the data from an edge, you can use this:

```
[25]: G.get_edge_data('1', '1108')
[25]: {'weight': 5,
 'label': '5',
 'route_short_name': 'B301',
 'color': '#9467bd',
 'penwidth': 4}
```

Now implement the function `plot_timings`, which given a networkx DiGraph G plots a frequency histogram of the time between bus stops.

**Expected output:**



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"  
data-jupman-show="Show solution"  
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[26]:
def plot_timings(G):

 import numpy as np
 import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```

timings = [G.get_edge_data(edge[0], edge[1])['weight'] for edge in G.edges()]

import matplotlib.pyplot as plt
import numpy as np

add histogram

min_x = min(timings)
max_x = max(timings)
bar_width = 1.0

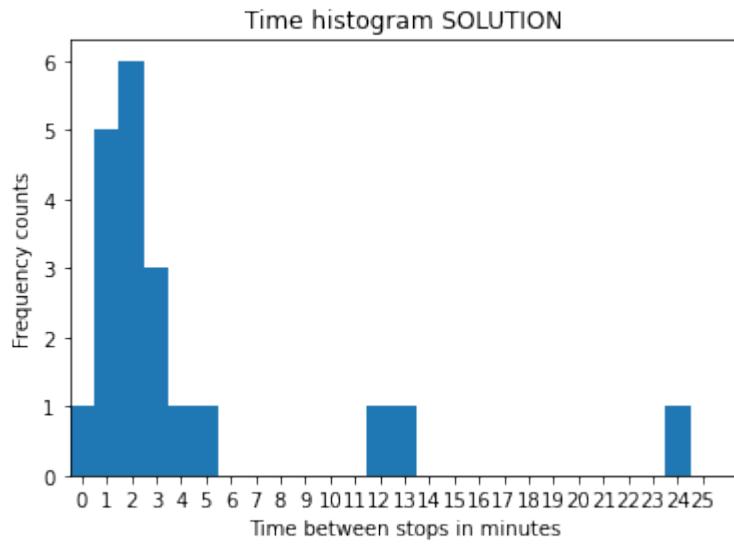
in this case hist returns a tuple of three values
we put in three variables
n, bins, columns = plt.hist(timings,
 bins=range(min_x,max_x + 1),
 width=1.0) # graphical width of the bars

xs = np.arange(min_x,max_x + 1)
plt.xlabel('Time between stops in minutes')
plt.ylabel('Frequency counts')
plt.title('Time histogram SOLUTION')
plt.xlim(0, max(timings) + 2)
plt.xticks(xs + bar_width / 2, # position of ticks
 xs)

plt.show()

```

plot\_timings(G)



&lt;/div&gt;

[26]:

```

def plot_timings(G):
 raise Exception('TODO IMPLEMENT ME !')

```

(continues on next page)

(continued from previous page)

plot\_timings(G)

### 10.3.4 Wikispeedia

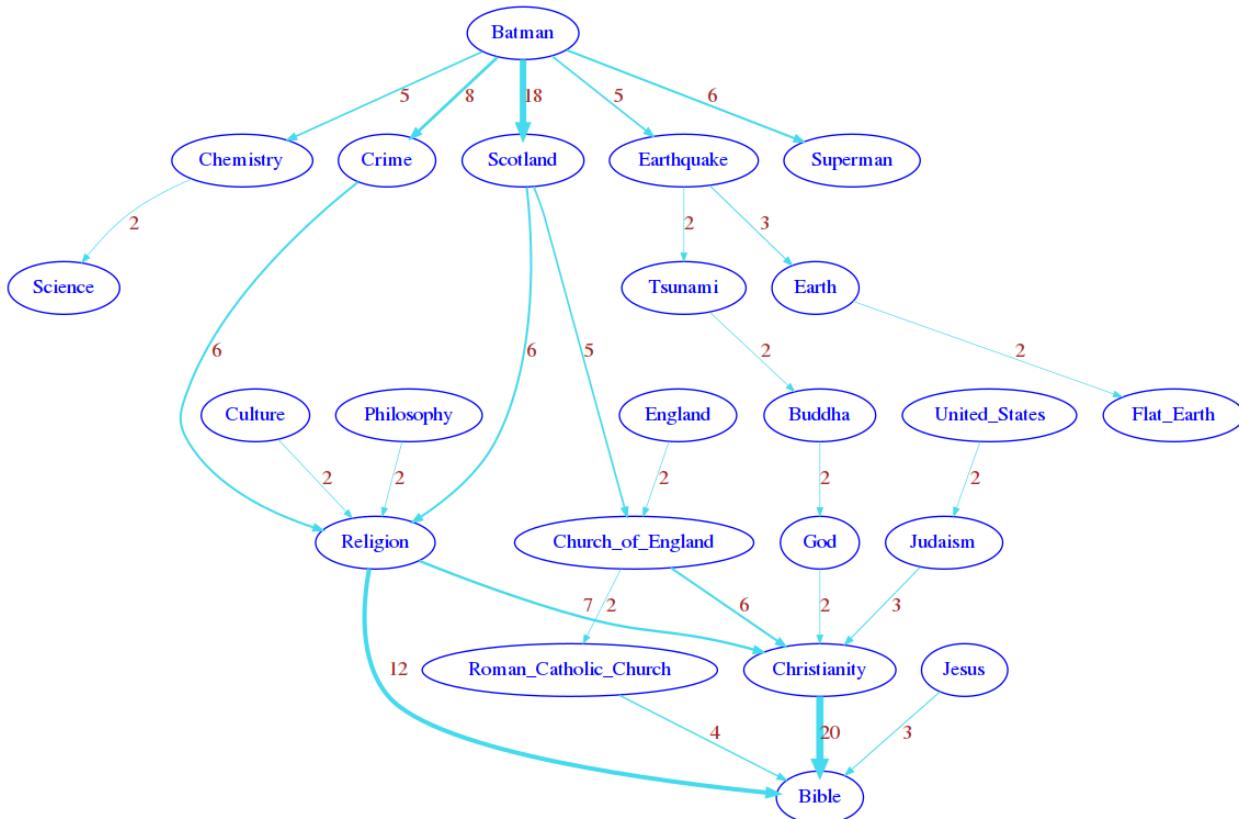
#### Download worked project

Browse files online<sup>508</sup>

What is the semantic distance between Batman and the Bible? Wikispeedia<sup>509</sup> is a fun game where you are given (or can choose) apparently unrelated source and a target Wikipedia pages, and you are asked to reach target page by only clicking links you find along the pages you visit. These click paths provide valuable information regarding human behaviour and the semantic connection between different topics, for example search engines might use such information to better understand user queries. You will analyze a dataset of such paths.

Data source: <https://snap.stanford.edu/data/wikispeedia.html>

- Robert West and Jure Leskovec: Human Wayfinding in Information Networks. 21st International World Wide Web Conference (WWW), 2012.
- Robert West, Joelle Pineau, and Doina Precup: Wikispeedia: An Online Game for Inferring Semantic Distances between Concepts. 21st International Joint Conference on Artificial Intelligence (IJCAI), 2009.



<sup>508</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/wikispeedia>

<sup>509</sup> <https://dlab.epfl.ch/wikispeedia/play/>

**REQUIREMENTS:** Having read Relational data tutorial<sup>510</sup>, which contains also instructions for installing required libraries.

## What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
wikispeedia-prj
wikispeedia.ipynb
wikispeedia-sol.ipynb
paths_finished.tsv
soft.py
jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `wikispeedia.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## The dataset

Each row of the dataset `paths_finished.tsv` is a user *session*, where user navigates from start page to end page. Columns are `hashedIpAddress`, `timestamp`, `durationInSec`, `path` and `rating`.

We define a **session group** as all sessions which have same start page and same end page, for example all these paths start with Linguistics and end in Rome:

```
[1]: import pandas as pd
import numpy as np
pd.options.display.max_colwidth = None
df = pd.read_csv('paths_finished.tsv', encoding='UTF-8', skiprows=16, header=None,
sep='\t')
df[5890:5902]
```

	0	1	2	\
5890	2f8c281d5e0b0e93	1248913182	106	
5891	389b67fa365b727a	1249604227	89	
5892	0299542414c3f20a	1257970576	94	
5893	2b6e83d366a7514d	1260188882	113	
5894	0d57c8c57d75e2f5	1282028286	153	
5895	0d57c8c57d75e2f5	1295244051	62	

(continues on next page)

<sup>510</sup> <https://en.softpython.org/relational/relational1-intro-sol.html#>

(continued from previous page)

5896	772843f73d9cf93d	1307880434	177	
5897	654430d34a08a0f5	1339755238	81	
5898	12470aee3d5ad152	1344167616	40	
5899	6365b049395c53ce	1345421532	67	
5900	05786cb24102850d	1347671980	65	
5901	369a3f7e77700217	1349302559	56	
				3 \
5890	Linguistics;Philosophy;Aristotle;Ancient_Greece;Italy;Rome			
5891	Linguistics;Language;English_language;Latin;Rome			
5892	Linguistics;Philosophy;Plato;Latin;Rome			
5893	Linguistics;Philosophy;Thomas_Aquinas;Italy;Rome			
5894	Linguistics;Language;Spanish_language;Vulgar_Latin;Roman_Empire;Rome			
5895	Linguistics;Philosophy;Socrates;Ancient_Greece;Ancient_Rome;Rome			
5896	Linguistics;Language;German_language;Latin_alphabet;<;Italy;Rome			
5897	Linguistics;Philosophy;Augustine_of_Hippo;Italy;Rome			
5898	Linguistics;Philosophy;Plato;Italy;Rome			
5899	Linguistics;Culture;Ancient_Rome;Rome			
5900	Linguistics;Language;English_language;Latin;Rome			
5901	Linguistics;Language;English_language;Latin;Rome			
	4			
5890	3.0			
5891	2.0			
5892	NaN			
5893	2.0			
5894	NaN			
5895	1.0			
5896	NaN			
5897	2.0			
5898	NaN			
5899	1.0			
5900	2.0			
5901	NaN			

In this other session group, all sessions start with Pikachu and end with Sun:

[2]:	df[45121:45138]	0	1	2	\
	45121	0d57c8c57d75e2f5	1278403914	17	
	45122	0d57c8c57d75e2f5	1278403938	42	
	45123	0d57c8c57d75e2f5	1278403972	17	
	45124	0d57c8c57d75e2f5	1278403991	8	
	45125	0d57c8c57d75e2f5	1278404007	9	
	45126	0d57c8c57d75e2f5	1278404048	8	
	45127	0d57c8c57d75e2f5	1278404061	16	
	45128	0d57c8c57d75e2f5	1278404065	8	
	45129	0d57c8c57d75e2f5	1278404070	8	
	45130	0d57c8c57d75e2f5	1278404089	8	
	45131	0d57c8c57d75e2f5	1278404095	7	
	45132	0d57c8c57d75e2f5	1278404101	9	
	45133	0d57c8c57d75e2f5	1278404117	6	
	45134	0d57c8c57d75e2f5	1278404124	10	
	45135	0d57c8c57d75e2f5	1278404129	9	
	45136	0d57c8c57d75e2f5	1278404142	7	
	45137	0d57c8c57d75e2f5	1278404145	6	

(continues on next page)

(continued from previous page)

		3	4
45121	Pikachu;North_America;Earth;Planet;Sun	1.0	
45122	Pikachu;Tree;Sunlight;Sun	NaN	
45123	Pikachu;Tree;Plant;<;Sunlight;Sun	1.0	
45124	Pikachu;Tree;Sunlight;Sun	NaN	
45125	Pikachu;Tree;Sunlight;Sun	1.0	
45126	Pikachu;Tree;Sunlight;Sun	NaN	
45127	Pikachu;Tree;Sunlight;Sun	NaN	
45128	Pikachu;Tree;Sunlight;Sun	NaN	
45129	Pikachu;Tree;Sunlight;Sun	NaN	
45130	Pikachu;Tree;Sunlight;Sun	NaN	
45131	Pikachu;Tree;Sunlight;Sun	NaN	
45132	Pikachu;Tree;Sunlight;Sun	NaN	
45133	Pikachu;Tree;Sunlight;Sun	NaN	
45134	Pikachu;Tree;Sunlight;Sun	NaN	
45135	Pikachu;Tree;Sunlight;Sun	NaN	
45136	Pikachu;Tree;Sunlight;Sun	NaN	
45137	Pikachu;Tree;Sunlight;Sun	NaN	

## 1. filter\_back

Whenever a user clicks the *Back* button, she navigates back one page. This fact is tracked in the data by the presence of a '<' symbol. Write a function which RETURN a NEW path without pages which were navigated back.

**NOTE: you can have duplicates** even without presence of <, because a user might end up to a previous page just by following circular links.

**DO NOT** misuse search methods, I'm watching you }:-[

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[3]: def filter_back(path):
 ret = []
 for page in path:
 if page == '<':
 if len(ret) > 0:
 ret.pop()
 else:
 ret.append(page)
 return ret

assert filter_back([]) == []
assert filter_back(['alfa']) == ['alfa']
assert filter_back(['beta', 'alfa', 'charlie']) == ['beta', 'alfa', 'charlie']

assert filter_back(['charlie', 'tango', '<']) == ['charlie']
inp = ['charlie', 'tango', '<']
assert filter_back(inp) == ['charlie'] # new
assert inp == ['charlie', 'tango', '<']
assert filter_back(['alfa', 'beta', 'charlie', '<', '<', 'delta']) == ['alfa', 'delta']
assert filter_back(['alfa', 'beta', 'charlie', 'delta', 'eagle', '<', '<', 'golf', '<', '<', 'hotel']) \
```

(continues on next page)

(continued from previous page)

```
== ['alfa', 'beta', 'hotel']

circular paths
assert filter_back(['alfa', 'beta', 'alfa', 'alfa', 'beta']) == ['alfa', 'beta', 'alfa',
 ↪ 'alfa', 'beta']
assert filter_back(['alfa', 'beta', 'alfa', '<', 'charlie', 'charlie', 'delta', 'charlie', '<
 ↪ ', 'charlie', 'delta']) \
 == ['alfa', 'beta', 'charlie', 'charlie', 'delta', 'charlie', 'delta']
```

&lt;/div&gt;

```
[3]: def filter_back(path):
 raise Exception('TODO IMPLEMENT ME !')

assert filter_back([]) == []
assert filter_back(['alfa']) == ['alfa']
assert filter_back(['beta', 'alfa', 'charlie']) == ['beta', 'alfa', 'charlie']

assert filter_back(['charlie', 'tango', '<']) == ['charlie']
inp = ['charlie', 'tango', '<']
assert filter_back(inp) == ['charlie'] # new
assert inp == ['charlie', 'tango', '<']
assert filter_back(['alfa', 'beta', 'charlie', '<', '<', 'delta']) == ['alfa', 'delta']
assert filter_back(['alfa', 'beta', 'charlie', 'delta', 'eagle', '<', '<', 'golf', '<', '<',
 ↪ 'hotel']) \
 == ['alfa', 'beta', 'hotel']

circular paths
assert filter_back(['alfa', 'beta', 'alfa', 'alfa', 'beta']) == ['alfa', 'beta', 'alfa',
 ↪ 'alfa', 'beta']
assert filter_back(['alfa', 'beta', 'alfa', '<', 'charlie', 'charlie', 'delta', 'charlie', '<
 ↪ ', 'charlie', 'delta']) \
 == ['alfa', 'beta', 'charlie', 'charlie', 'delta', 'charlie', 'delta']
```

## 2. load\_db

Load the **tab**-separated file `paths_finished.tsv` with a CSV Reader. The file has some rows to skip and no column names: parse it and RETURN a list of dictionaries, with `hashedIpAddress`, `timestamp`, `durationInSec`, `path` and `rating` as fields:

- `path`: convert it with `filter_back` function
- `timestamp` and `durationInSec`: convert to integer
- `rating`: convert to integer, if `NULL`, set it to `None`

### Example:

```
>>> sessions_db = load_db('paths_finished.tsv')

Parsed 51318 sessions

>>> sessions_db[:2]
[{'hashedIpAddress': '6a3701d319fc3754',
 'timestamp': 1297740409,
```

(continues on next page)

(continued from previous page)

```
'durationInSec': 166,
'path': ['14th_century',
'15th_century',
'16th_century',
'Pacific_Ocean',
'Atlantic_Ocean',
'Accra',
'Africa',
'Atlantic_slave_trade',
'African_slave_trade'],
'rating': None},
{'hashedIpAddress': '3824310e536af032',
'timestamp': 1344753412,
'durationInSec': 88,
'path': ['14th_century',
'Europe',
'Africa',
'Atlantic_slave_trade',
'African_slave_trade'],
'rating': 3}]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[4]:

```
import csv

def load_db(filename):

 with open(filename, encoding='utf-8', newline='') as f:
 my_reader = csv.reader(f, delimiter='\t')

 ret = []

 for row in my_reader:
 if len(row) < 2:
 continue

 ses = { "hashedIpAddress": row[0],
 "timestamp" : int(row[1]),
 "durationInSec" : int(row[2]),
 "path" : filter_back(row[3].split(';')),
 "rating": int(row[4]) if row[4] != 'NULL' else None
 }

 ret.append(ses)

 print('Parsed', len(ret), 'sessions')

 return ret

sessions_db = load_db('paths_finished.tsv')
sessions_db[:2]
```

```
Parsed 51318 sessions
```

```
[4]: [{"hashedIpAddress": "6a3701d319fc3754",
 "timestamp": 1297740409,
 "durationInSec": 166,
 "path": ["14th_century",
 "15th_century",
 "16th_century",
 "Pacific_Ocean",
 "Atlantic_Ocean",
 "Accra",
 "Africa",
 "Atlantic_slave_trade",
 "African_slave_trade"],
 "rating": None},
 {"hashedIpAddress": "3824310e536af032",
 "timestamp": 1344753412,
 "durationInSec": 88,
 "path": ["14th_century",
 "Europe",
 "Africa",
 "Atlantic_slave_trade",
 "African_slave_trade"],
 "rating": 3}]]
```

```
</div>
```

```
[4]:
import csv

def load_db(filename):
 raise Exception('TODO IMPLEMENT ME !')

sessions_db = load_db('paths_finished.tsv')

sessions_db[:2]
```

```
[5]: # TESTING
from pprint import pprint
from expected_db import expected_db
for i in range(len(expected_db)):
 if expected_db[i] != sessions_db[i]:
 print('\nERROR at index', i, ':')
 print(' ACTUAL:')
 pprint(sessions_db[i])
 print(' EXPECTED:')
 pprint(expected_db[i])
 break
if len(sessions_db) != len(expected_db):
 print("ERROR: different lengths! sessions_db:", len(sessions_db), "expected_db",
 len(expected_db))
```

### 3. calc\_stats

Write a function which takes the sessions db and RETURN a NEW dictionary which maps sessions groups expressed as tuples (start, end) to a dictionary of statistics about them

- dictionary key: tuple with start,end page
- sessions: the number of sessions in that group
- avg\_len: the average length (as number of edges) of all paths in the group
- pages: the total number of **DISTINCT** pages found among all sessions in that group
- freqs: a dictionary which maps edges found in all sessions of that group to their count
- max\_freq: the highest count among all freqs

**Output example** (for complete output see [expected\\_stats\\_db.py](#)):

```
>>> calc_stats(sessions_db)
{
 ('Linguistics', 'Rome'): {'avg_len': 4.166666666666667,
 'freqs': {('Ancient_Greece', 'Ancient_Rome'): 1,
 ('Ancient_Greece', 'Italy'): 1,
 ('Ancient_Rome', 'Rome'): 2,
 ('Aristotle', 'Ancient_Greece'): 1,
 ('Augustine_of_Hippo', 'Italy'): 1,
 ('Culture', 'Ancient_Rome'): 1,
 ('English_language', 'Latin'): 3,
 ('German_language', 'Italy'): 1,
 ('Italy', 'Rome'): 5,
 ('Language', 'English_language'): 3,
 ('Language', 'German_language'): 1,
 ('Language', 'Spanish_language'): 1,
 ('Latin', 'Rome'): 4,
 ('Linguistics', 'Culture'): 1,
 ('Linguistics', 'Language'): 5,
 ('Linguistics', 'Philosophy'): 6,
 ('Philosophy', 'Aristotle'): 1,
 ('Philosophy', 'Augustine_of_Hippo'): 1,
 ('Philosophy', 'Plato'): 2,
 ('Philosophy', 'Socrates'): 1,
 ('Philosophy', 'Thomas_Aquinas'): 1,
 ('Plato', 'Italy'): 1,
 ('Plato', 'Latin'): 1,
 ('Roman_Empire', 'Rome'): 1,
 ('Socrates', 'Ancient_Greece'): 1,
 ('Spanish_language', 'Vulgar_Latin'): 1,
 ('Thomas_Aquinas', 'Italy'): 1,
 ('Vulgar_Latin', 'Roman_Empire'): 1},
 'max_freq': 6,
 'pages': 19,
 'sessions': 12},
 ('Pikachu', 'Sun'): {'avg_len': 3.0588235294117645,
 'freqs': {('Earth', 'Planet'): 1,
 ('North_America', 'Earth'): 1,
 ('Pikachu', 'North_America'): 1,
 ('Pikachu', 'Tree'): 16,
 ('Planet', 'Sun'): 1,
 ('Sunlight', 'Sun'): 16,
 ('Tree', 'Sun'): 16}
 }
}
```

(continues on next page)

(continued from previous page)

```
 ('Tree', 'Sunlight'): 16},
 'max_freq': 16,
 'pages': 7,
 'sessions': 17},
.
.
.
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution"

data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[6]: def calc_stats(sessions):

 ret = {}

 for session in sessions:
 path = session['path']

 t = (path[0], path[-1])
 if t in ret:
 ret[t]['avg_len'] += len(path) - 1
 ret[t]['sessions'] += 1

 ret[t]['pages_set'].update(path)

 else:
 ret[t] = { 'avg_len' : len(path) - 1,
 'sessions' : 1,
 'pages_set' : set(path),
 'freqs' : {}}

 freqs = ret[t]['freqs']

 for i in range(1, len(path)):
 seg = tuple([path[i-1], path[i]])
 if seg in freqs:
 freqs[seg] += 1
 else:
 freqs[seg] = 1

 for t in ret:
 ret[t]['avg_len'] = ret[t]['avg_len'] / ret[t]['sessions']
 ret[t]['pages'] = len(ret[t]['pages_set'])
 freq_vs = ret[t]['freqs'].values()
 ret[t]['max_freq'] = max(freq_vs) if len(freq_vs) > 0 else 0
 del ret[t]['pages_set']

 return ret
```

```
stats_db = calc_stats(sessions_db)
```

</div>

```
[6]: def calc_stats(sessions):
 raise Exception('TODO IMPLEMENT ME !')

stats_db = calc_stats(sessions_db)
```

```
[7]: # TESTING
from pprint import pprint
from expected_stats_db import expected_stats_db
for t in expected_stats_db:
 if not t in stats_db:
 print('\nERROR: missing key for session group', t)
 break
 elif expected_stats_db[t] != stats_db[t]:
 print('\nERROR at key for session group', t, ':')
 print(' ACTUAL:')
 pprint(stats_db[t])
 print(' EXPECTED:')
 pprint(expected_stats_db[t])
 break

diff = stats_db.keys() - expected_stats_db.keys()
if len(diff) > 0:
 print('ERROR! Found these extra keys in stats_db:', diff)
```

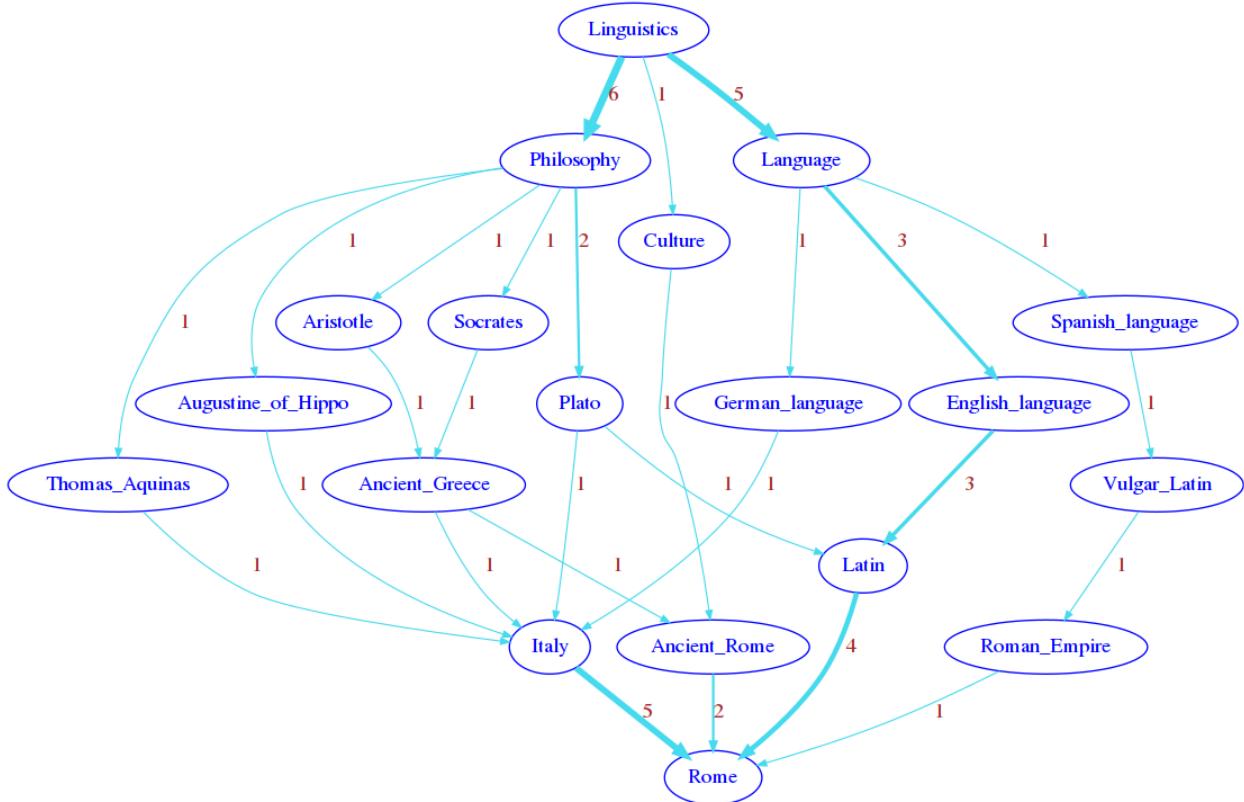
#### 4. plot\_network

Given a sessions group (start\_page, target\_page), we want to display the pages clicked by users for all its sessions. Since some sessions share edges, we will show their click frequency.

- Set edges attributes 'weight', 'label' as  $freq$ , 'penwidth' as  $5 \frac{freq}{max\_freq}$  and 'color' as '#45daed'
- Only display edges (and pages connected by such edges) for which the click count is *strictly greater* than the given threshold
- **NOTE:** when applying a threshold, it's fine if some nodes don't appear linked to either source or target

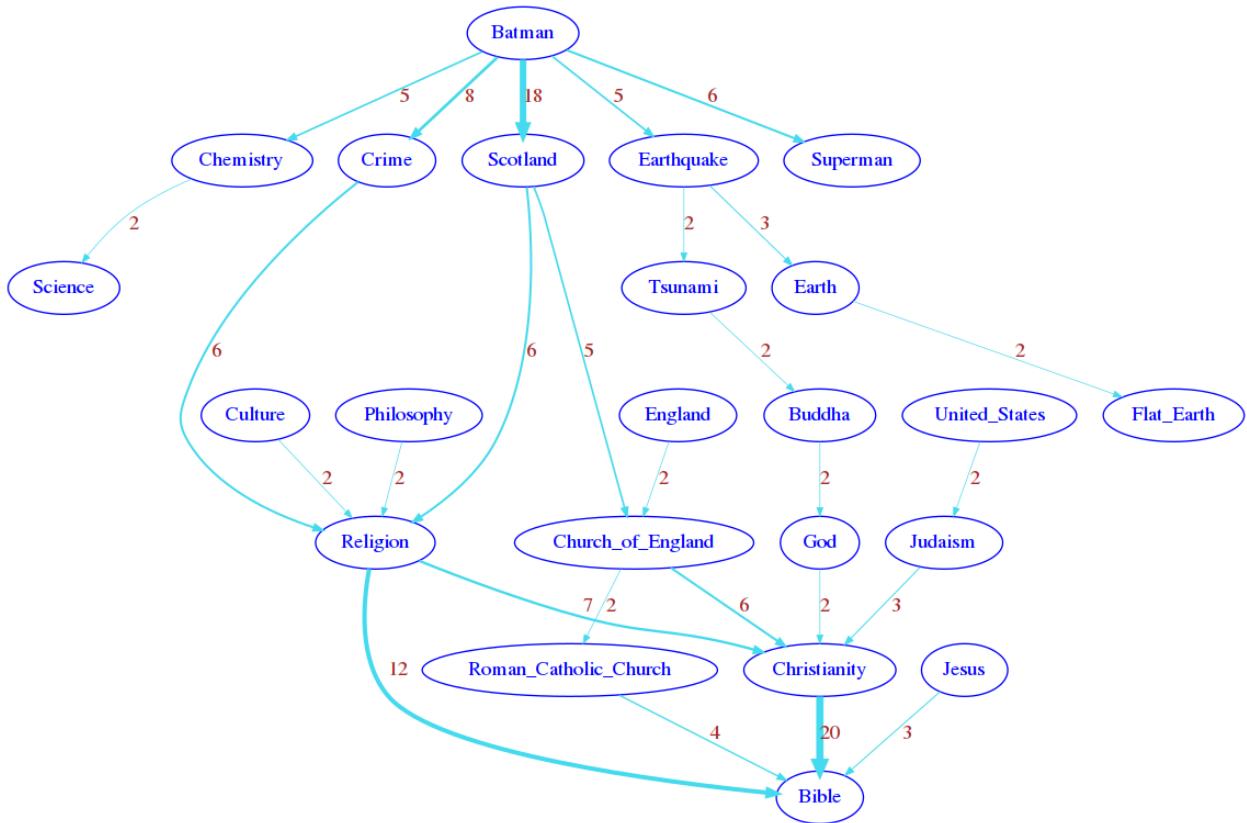
##### Example 1:

```
>>> plot_network(stats_db, 'Linguistics', 'Rome')
```



**Example 2:**

```
>>> plot_network(stats_db, 'Batman', 'Bible', 0) # default threshold zero, big graph
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);> Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[9] :

```
from soft import draw_nx
import networkx as nx

def plot_network(stats, source_page, target_page, threshold=0):

 G = nx.DiGraph()

 st = stats[(source_page, target_page)]

 for p1,p2 in st['freqs']:
 d = st['freqs'][(p1,p2)]
 if d > threshold:
 G.add_node(p2)
 G.add_edge(p1, p2)
 G[p1][p2]['weight'] = d
 G[p1][p2]['label'] = d
 G[p1][p2]['penwidth'] = 5*d/st['max_freq']
 G[p1][p2]['color'] = '#45daed'
```

(continues on next page)

(continued from previous page)

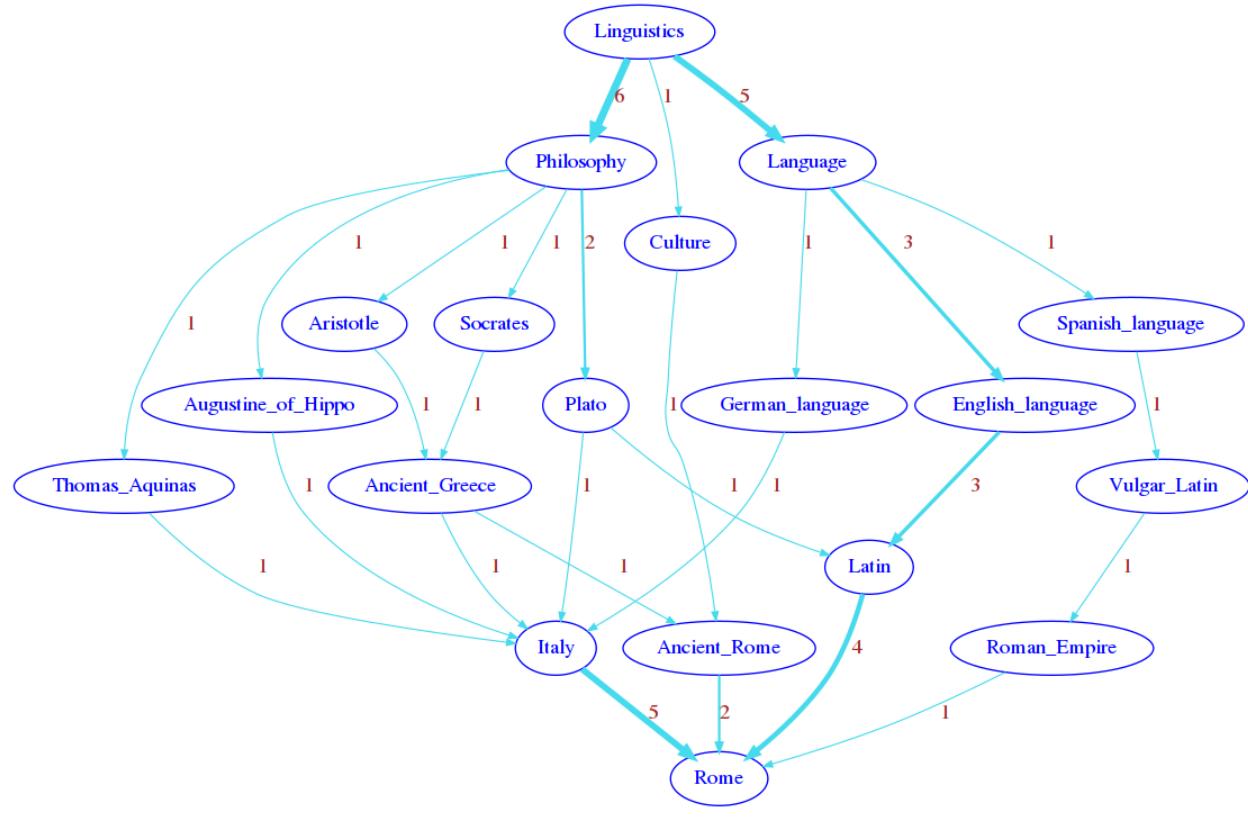
```

draw_nx(G
)

plot_network(stats_db, 'Linguistics', 'Rome')

```

Image saved to file: expected-Linguistics-Rome-0.png



&lt;/div&gt;

```
[9]:
from soft import draw_nx
import networkx as nx

def plot_network(stats, source_page, target_page, threshold=0):
 raise Exception('TODO IMPLEMENT ME !')

plot_network(stats_db, 'Linguistics', 'Rome')
```

```
[10]:
plot_network(stats_db, 'Batman', 'Bible', 0) # default threshold zero, big graph
```

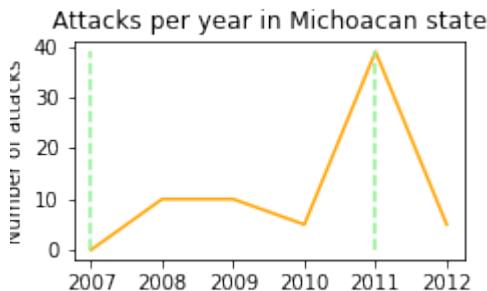
```
[11]:
plot_network(stats_db, 'Batman', 'Bible', 1) # we take only edges > 1
```

### 10.3.5 Mexican Drug Wars

#### Download worked project

Browse files online<sup>511</sup>

Drug cartels carried out a shocking wave of lethal attacks against hundreds of local elected officials and party candidates in Mexico during years 2007-2012, attempting to establish criminal governance regimes and conquer local governments, populations, and territories. This quickly forced Mexican authorities to deploy armored vehicles with heavy weapons to perform military operations within their own borders. You will analyze cartels attacks frequency and where they occurred.



Data sources:

- Trejo, Guillermo; Ley, Sandra, 2019, “Replication Data for: High-Profile Criminal Violence. Why Drug Cartels Murder Government Officials and Party Candidates in Mexico”, <https://doi.org/10.7910/DVN/VIXNNE>, Harvard Dataverse, V1, UNF:6:BcqInKD9NBX3NkI48CdqpQ== [fileUNF] License: CC0 - “Public Domain Dedication”
- Coscia, Michele and Viridiana Rios (2012). Knowing Where and How Criminal Organizations Operate Using Web Content. CIKM, 12 (October – November). [https://www.michelecoscia.com/?page\\_id=1032](https://www.michelecoscia.com/?page_id=1032)

#### What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
bus-speed-prj
mexican-drug-wars.ipynb
mexican-drug-wars-sol.ipynb

jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `mexican-drug-wars.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

<sup>511</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/mexican-drug-wars>

### Election attacks dataset

In the file `Dataset_HighProfileCriminalViolence.tab` are listed the number of attacks occurred to elected officials in Mexico from years 2007 to 2012. Focus only on columns `cve_inegi, state, year, aggr_sum, elect_local`:

	cve_inegi	state	year	aggr_sum	elect_local
2278	12031	Guerrero	2012	2	1.0
2279	12032	Guerrero	2007	0	0.0
2280	12032	Guerrero	2008	0	1.0
2281	12032	Guerrero	2009	1	0.0

- Municipalities where the attack occurred are identified by a 5 digits `cve_inegi` code: first two digits indicate the state, 3 last ones the town. **NOTE:** first file entries only have 4 digits as the leading zero is implied, take care of this case
- `aggr_sum`: number of attacks occurred in a particular municipality / year.
- `elect_local`: 1.0 if a **local** election occurred in the year of the attack (ignore other `elect_*`)

#### 1. load\_mexico

Extract Mexican state codes, names, the counts of attacks, and the years when **local** elections occurred, and RETURN a dictionary of dictionaries mapping **two digit** state codes **as strings** to the extracted info.

- use `csv.DictReader` with `delimiter='\\t'` and `utf8` encoding (municipalities will look weird but we don't use them)
- use exactly 6 cells for `attacks` lists: assume all were carried out between 2007 and 2012 included
- **DO NOT** assume the years in rows repeat with a pattern, for example municipality 21132 has two successive 2012 years!

**Example EXCEPT** (note keys order doesn't matter, complete expected output can be found in `expected_mexico_db.py`)

```
>>> load('Dataset_HighProfileCriminalViolence.tab')

{
 '08': {
 'attacks': [0, 5, 7, 12, 7, 2],
 'local_election_years': [2007, 2010],
 'state_code': '08',
 'state_name': 'Chihuahua'
 }
 '12': {
 'attacks': [4, 11, 11, 9, 3, 10],
 'local_election_years': [2008, 2011, 2012],
 'state_code': '12',
 'state_name': 'Guerrero'
 }
 .
 .
}
```

[Show solution](#)

[2]:

```
import csv
```

(continues on next page)

(continued from previous page)

```

def load(filename):

 with open(filename, encoding='utf8', newline='') as csvfile_in:
 my_reader = csv.DictReader(csvfile_in, delimiter='\t')

 ret = {}

 for d in my_reader:
 if len(d['cve_inegi']) == 4:
 inegi = '0' + d['cve_inegi']
 else:
 inegi = d['cve_inegi']

 state_code = inegi[:2]

 if not state_code in ret:
 ret[state_code] = {'attacks':[0]*6,
 'local_election_years' : [],
 'state_name': '',
 'state_code': state_code}

 rd = ret[state_code]
 year = int(d['year'])
 rd['state_code'] = state_code
 rd['state_name'] = d['state']
 rd['attacks'][year - 2007] += int(d['aggr_sum'])

 if d['elect_local'] == '1.0':
 if year not in rd['local_election_years']:
 rd['local_election_years'].append(year)

 for rd in ret:
 ret[rd]['local_election_years'].sort()

 return ret

```

```
mexico_db = load('Dataset_HighProfileCriminalViolence.tab')
mexico_db
```

&lt;/div&gt;

```
[2]:
import csv

def load(filename):
 raise Exception('TODO IMPLEMENT ME !')

mexico_db = load('Dataset_HighProfileCriminalViolence.tab')
mexico_db
```

```
[4]: # TESTING
from pprint import pformat; from expected_mexico_db import expected_mexico_db
(continues on next page)
```

(continued from previous page)

```

for sid in expected_mexico_db.keys():
 if sid not in mexico_db: print('\nERROR: MISSING state', sid); break
 for k in expected_mexico_db[sid]:
 if k not in mexico_db[sid]:
 print('\nERROR at state', sid, '\n\n MISSING key:', k); break
 if expected_mexico_db[sid][k] != mexico_db[sid][k]:
 print('\nERROR at state', sid, 'key:', k)
 print(' ACTUAL:\n', pformat(mexico_db[sid][k]))
 print(' EXPECTED:\n', pformat(expected_mexico_db[sid][k]))
 break
if len(mexico_db) > len(expected_mexico_db):
 print('ERROR! There are more states than expected!')
 print(' ACTUAL:\n', len(mexico_db))
 print(' EXPECTED:\n', len(expected_mexico_db))

```

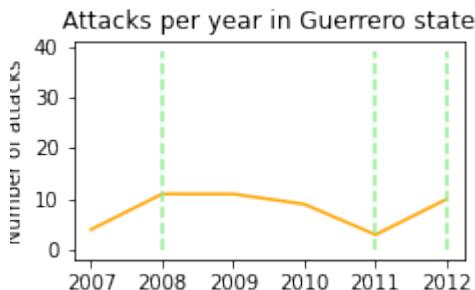
## 2. show\_attacks

Given a state\_code and , display a chart of the attack counts over the years.

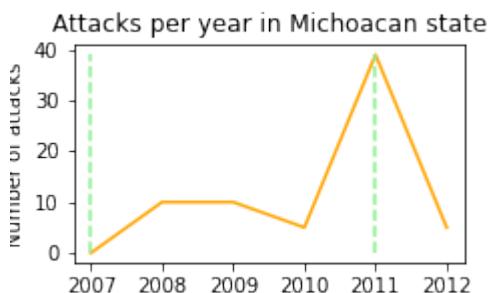
- normalize the height so to have all charts as high as the maximum possible attack count in the db
- show vertical dashed lines in proximity of election years (use linestyle='dashed'), using the same color
- you are allowed to use constants for years
- make sure vertical lines on borders are clearly visible** and separated from borders by setting proper limits
- remember** to also print the maximum possible attack count in the db

Examples:

```
>>> show_attacks('12', mexico_db)
max attacks happened in any state: 39
```



```
>>> show_attacks('16', mexico_db)
max attacks happened in any state: 39
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[5]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def show_attacks(state_code, mexdb):

 attacks = mexdb[state_code] ['attacks']

 plt.figure(figsize=(3.5,2))
 maxy = 0
 for s in mexdb:
 maxy = max(maxy, max(mexdb[s] ['attacks']))

 print("max attacks happened in any state:", maxy)

 xs = range(2007, 2013)
 ys = mexdb[state_code] ['attacks']

 plt.plot(xs, ys, color='orange')

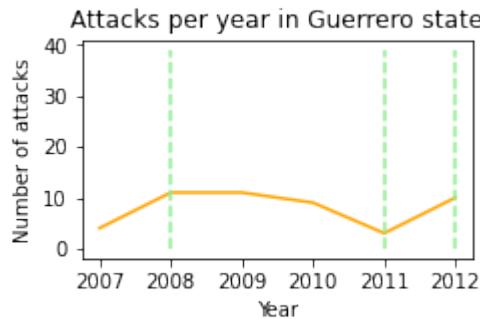
 plt.title("Attacks per year in %s state" % mexdb[state_code] ['state_name'])
 plt.xlabel('Year')
 plt.ylabel('Number of attacks')

 for year in mexdb[state_code] ['local_election_years']:
 plt.plot([year, year], [0, maxy], color='lightgreen', linestyle='dashed')

 plt.savefig('expected-plot-%s.png' % state_code)
 plt.show()
```

show\_attacks('12', mexico\_db) # Guerrero

max attacks happened in any state: 39



</div>

[5]:

```
%matplotlib inline
import matplotlib.pyplot as plt

def show_attacks(state_code, mexdb):
```

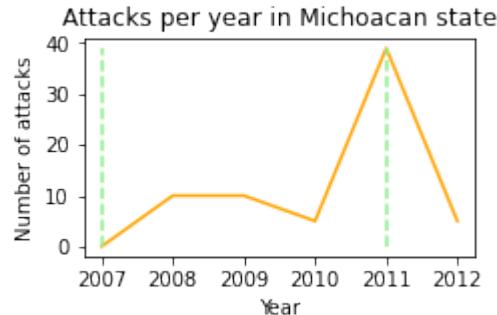
(continues on next page)

(continued from previous page)

```
raise Exception('TODO IMPLEMENT ME !')

show_attacks('12', mexico_db) # Guerrero
```

[6]: show\_attacks('16', mexico\_db) # Michoacan  
max attacks happened in any state: 39



### 3. Cartels

The file `CosciaRios2012_DataBase.csv` lists attacks performed by criminal organizations (cartels) in various years. For each row, the columns from 3-12 have a 1 if the corresponding cartel named in the header was involved in the attack, and 0 otherwise. Example:

[7]:

	Code	State	Year	Beltran_Leyva	Beltran_Leyva_Family	Familia	Golfo	\
17	1001	1	2007	0	0	1	0	
18	1001	1	2008	0	0	1	0	
19	1001	1	2009	0	0	1	1	

	Juarez	Sinaloa	Sinaloa_Family	Tijuana	Zetas	Otros
17	0	0	0	0	1	0
18	1	0	0	0	0	0
19	0	1	0	0	1	0

Write a function which given a `filename` and a `year`, processes the dataset and RETURN a dictionary mapping cartel names to a list of **sorted** states (no duplicates) where the cartel performed attacks in the given year.

- use a `csv.reader` with `utf8` encoding
- pick state code from `State` column and state names from previous `mexico_db` (you only need names) - if missing put state code (i.e. 09)
- **NOTE:** Sinaloa is a special case, since it is both a state and a cartel.

Example:

```
>>> cartels('CosciaRios2012_DataBase.csv', mexico_db, 2003)

{
 'Beltran_Leyva': ['Colima', 'Morelos', 'Sinaloa'],
 'Beltran_Leyva_Family': [],
 'Familia': [],
 'Golfo': ['Campeche', 'Chihuahua', 'Coahuila', 'Durango', 'Mexico',
 'Nuevo Leon', 'San Luis Potosi', 'Tamaulipas', 'Veracruz', 'Yucatan'],
 'Juarez': ['Baja California', 'Chihuahua', 'Coahuila', 'Colima',
```

(continues on next page)

(continued from previous page)

```

'Durango', 'Guerrero', 'Jalisco', 'Sinaloa', 'Tamaulipas'],
'Otros': ['Veracruz'],
'Sinaloa': ['Chiapas', 'Colima', 'Jalisco', 'Mexico',
 'Nayarit', 'Nuevo Leon', 'Sinaloa', 'Sonora', 'Tamaulipas'],
'Sinaloa_Family': ['Guerrero'],
'Tijuana': ['Aguascalientes', 'Baja California', 'Chiapas', 'Chihuahua', 'Coahuila',
 'Guerrero',
 'Jalisco', 'Mexico', 'Michoacan', 'Nuevo Leon', 'Puebla',
 'Sinaloa',
 'Sonora'],
'Zetas': ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico',
 'Nuevo Leon', 'Sinaloa', 'Tamaulipas', 'Veracruz', 'Yucatan']
}

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[8]:

```

import csv
def cartels(filename, mexdb, year):

 work = {}

 with open(filename, encoding='utf8', newline='') as csvfile_in:
 my_reader = csv.reader(csvfile_in, delimiter=',')

 header = next(my_reader)

 for cartel_name in header[3:]:
 work[cartel_name] = set()

 for row in my_reader:
 row_year = int(row[2])
 if row_year == year:
 inegi = row[0]
 if len(row[1]) == 1:
 state_code = '0' + row[1]
 else:
 state_code = row[1]
 if state_code in mexdb:
 state_name = mexdb[state_code]['state_name']
 else:
 state_name = state_code

 for j in range(3, len(row)):
 if row[j] == '1':
 org_name = header[j]
 work[org_name].add(state_name)

 ret = {}
 for k,v in work.items():
 ret[k] = sorted(v)
 return ret

```

(continues on next page)

(continued from previous page)

```

cartels2003 = cartels('CosciaRios2012_DataBase.csv', mexico_db, 2003)
from pprint import pprint
pprint(cartels2003, width=190)

assert cartels2003['Beltran_Leyva'] == ['Colima', 'Morelos', 'Sinaloa']
assert cartels2003['Otros'] == ['Veracruz']
assert cartels2003['Zetas'] == ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico',
 ↪'Nuevo Leon', 'Sinaloa', 'Tamaulipas', 'Veracruz', 'Yucatan']

{'Beltran_Leyva': ['Colima', 'Morelos', 'Sinaloa'],
 'Beltran_Leyva_Family': [],
 'Familia': [],
 'Golfo': ['Campeche', 'Chihuahua', 'Coahuila', 'Durango', 'Mexico', 'Nuevo Leon',
 ↪'San Luis Potosi', 'Tamaulipas', 'Veracruz', 'Yucatan'],
 'Juarez': ['Baja California', 'Chihuahua', 'Coahuila', 'Colima', 'Durango', 'Guerrero
 ↪', 'Jalisco', 'Sinaloa', 'Tamaulipas'],
 'Otros': ['Veracruz'],
 'Sinaloa': ['Chiapas', 'Colima', 'Jalisco', 'Mexico', 'Nayarit', 'Nuevo Leon',
 ↪'Sinaloa', 'Sonora', 'Tamaulipas'],
 'Sinaloa_Family': ['Guerrero'],
 'Tijuana': ['Aguascalientes', 'Baja California', 'Chiapas', 'Chihuahua', 'Coahuila',
 ↪'Guerrero', 'Jalisco', 'Mexico', 'Michoacan', 'Nuevo Leon', 'Puebla', 'Sinaloa',
 ↪'Sonora'],
 'Zetas': ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico', 'Nuevo Leon', 'Sinaloa',
 ↪'Tamaulipas', 'Veracruz', 'Yucatan']}

```

&lt;/div&gt;

[8]:

```

import csv
def cartels(filename, mexdb, year):
 raise Exception('TODO IMPLEMENT ME !')

cartels2003 = cartels('CosciaRios2012_DataBase.csv', mexico_db, 2003)
from pprint import pprint
pprint(cartels2003, width=190)

assert cartels2003['Beltran_Leyva'] == ['Colima', 'Morelos', 'Sinaloa']
assert cartels2003['Otros'] == ['Veracruz']
assert cartels2003['Zetas'] == ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico',
 ↪'Nuevo Leon', 'Sinaloa', 'Tamaulipas', 'Veracruz', 'Yucatan']

```

[9]: # further tests

```

expected2003 = {
 'Beltran_Leyva': ['Colima', 'Morelos', 'Sinaloa'],
 'Beltran_Leyva_Family': [],
 'Familia': [],
 'Golfo': ['Campeche', 'Chihuahua', 'Coahuila', 'Durango', 'Mexico', 'Nuevo Leon',
 ↪'San Luis Potosi', 'Tamaulipas', 'Veracruz', 'Yucatan'],
 'Juarez': ['Baja California', 'Chihuahua', 'Coahuila', 'Colima', 'Durango',
 ↪'Guerrero', 'Jalisco', 'Sinaloa', 'Tamaulipas'],
 'Otros': ['Veracruz'],
 'Sinaloa': ['Chiapas', 'Colima', 'Jalisco', 'Mexico', 'Nayarit', 'Nuevo Leon',
 ↪'Sinaloa', 'Sonora', 'Tamaulipas'],
 'Sinaloa_Family': ['Guerrero'],
 'Tijuana': ['Aguascalientes', 'Baja California', 'Chiapas', 'Chihuahua', 'Coahuila',
 ↪'Guerrero', 'Jalisco', 'Mexico', 'Michoacan', 'Nuevo Leon', 'Puebla',
 ↪'Sonora'],
 'Zetas': ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico', 'Nuevo Leon', 'Sinaloa',
 ↪'Tamaulipas', 'Veracruz', 'Yucatan']}

```

(continues on next page)

(continued from previous page)

```
'Zetas': ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico', 'Nuevo Leon', 'Sinaloa',
 ↪'Tamaulipas', 'Veracruz', 'Yucatan']
}

assert cartels2003 == expected2003

expected1999 = {
 'Beltran_Leyva': [],
 'Beltran_Leyva_Family': [],
 'Familia': [],
 'Golfo': ['Baja California', 'Guanajuato', 'Nuevo Leon', 'Puebla'],
 'Juarez': ['Baja California', 'Chihuahua', 'Durango', 'Tamaulipas', 'Veracruz'],
 'Sinaloa': ['Jalisco', 'Veracruz'],
 'Sinaloa_Family': [],
 'Tijuana': ['Baja California', 'Campeche', 'Coahuila', 'Nuevo Leon', 'Sonora',
 ↪'Tamaulipas', 'Yucatan'],
 'Zetas': ['Baja California', 'Mexico', 'Morelos', 'Sinaloa', 'Sonora'],
 'Otros': []
}

cartels1999 = cartels('CosciaRios2012_DataBase.csv', mexico_db, 1999)
assert cartels1999 == expected1999
```

## 10.3.6 Wordnet

### Download worked project

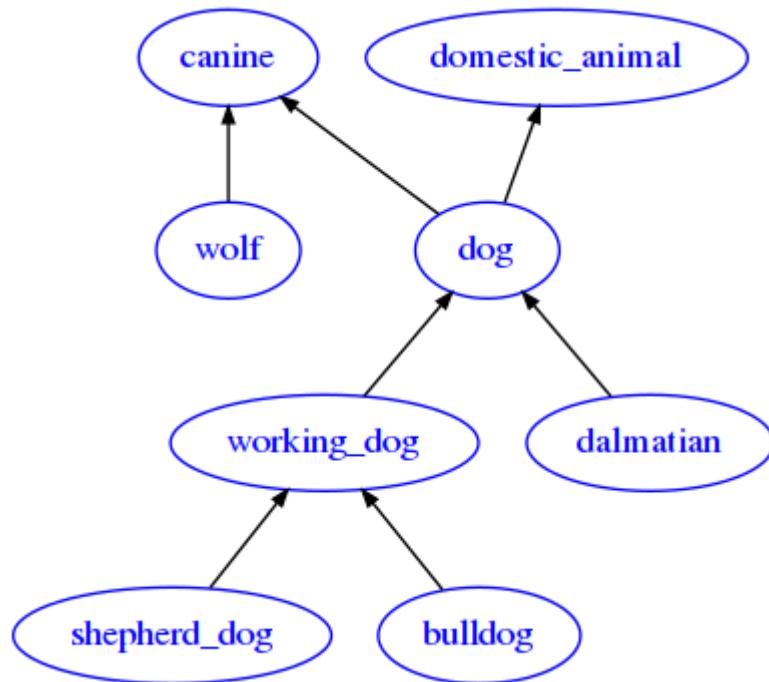
Browse files online<sup>512</sup>

WordNet<sup>513</sup>® is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of semantic relations. The resulting network of related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download, making it a useful tool for computational linguistics and natural language processing. Princeton University “About WordNet.” WordNet<sup>514</sup>. Princeton University. 2010

<sup>512</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/wordnet>

<sup>513</sup> <https://wordnet.princeton.edu/>

<sup>514</sup> <https://wordnet.princeton.edu/>



In Python there are specialized libraries to read WordNet like [NLTK](#)<sup>515</sup>, but for the sake of this worksheet, you will parse the noun database as a text file which can be read line by line.

We will focus on *names* and how they are linked by *IS A* relation, for example, a *dalmatian* *IS A* *dog* (*IS A* is also called *hypernym* relation)

**REQUIREMENTS:** Having read [Relational data tutorial](#)<sup>516</sup>, which contains also instructions for installing required libraries.

## What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
wordnet-prj
 wordnet.ipynb
 wordnet-sol.ipynb
 data.noun
 dogs.noun
 jupman.py
```

**WARNING:** to correctly visualize the notebook, it MUST be in an unzipped folder !

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `wordnet.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

<sup>515</sup> <https://www.nltk.org/howto/wordnet.html>

<sup>516</sup> <https://en.softpython.org/relational/relational1-intro-sol.html#>

- to execute Python code inside a Jupyter cell, press Control + Enter
- to execute Python code inside a Jupyter cell AND select next cell, press Shift + Enter
- to execute Python code inside a Jupyter cell AND create a new cell afterwards, press Alt + Enter
- If the notebooks look stuck, try to select Kernel -> Restart

## 1. parse\_db

First, you will begin with parsing an excerpt of wordnet `dogs.noun` file, which is a noun database shown here in its entirety.

According to documentation<sup>517</sup>, a noun database begins with several lines containing a copyright notice, version number, and license agreement: these lines all begin with **two spaces** and the line number like

```
1 This software and database is being provided to you, the LICENSEE, by
2 Princeton University under the following license. By obtaining, using
3 and/or copying this software and database, you agree that you have
```

Afterwards, each of following lines describe a noun synset, that is, a unique concept identified by a number called `synset_offset`.

- each synset can have many words to represent it - for example, the noun synset 02112993 has 03 (`w_cnt`) words `dalmatian coach_dog, carriage_dog`.
- a synset can be linked to other ones by relations. The `dalmatian` synset is linked to 002 (`p_cnt`) other synsets: to synset 02086723 by the @ relation, and to synset 02113184 by the ~ relation. For our purposes, you can focus on the @ symbol which means *IS A* relation (also called hypernym). If you search for a line starting with 02086723, you will see it is the synset for `dog`, so Wordnet is telling us a `dalmatian` *IS A* `dog`.

**WARNING 1:** lines can be quite long so if they appear to span multiple lines don't be fooled : remember each name definition only occupies one single line with no carriage returns!

**WARNING 2:** there are no empty lines between the synsets, here you see them just to visually separate the text blobs

```
1 This software and database is being provided to you, the LICENSEE, by
2 Princeton University under the following license. By obtaining, using
3 and/or copying this software and database, you agree that you have
4 read, understood, and will comply with these terms and conditions.:
5
6 Permission to use, copy, modify and distribute this software and
7 database and its documentation for any purpose and without fee or
8 royalty is hereby granted, provided that you agree to comply with
9 the following copyright notice and statements, including the disclaimer,
10 and that the same appear on ALL copies of the software, database and
11 documentation, including modifications that you make for internal
12 use or for distribution.
13
14 WordNet 3.1 Copyright 2011 by Princeton University. All rights reserved.
15
16 THIS SOFTWARE AND DATABASE IS PROVIDED "AS IS" AND PRINCETON
17 UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR
```

(continues on next page)

<sup>517</sup> <https://wordnet.princeton.edu/documentation/wndb5wn>

(continued from previous page)

18 IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PRINCETON  
19 UNIVERSITY MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANT-  
20 ABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE  
21 OF THE LICENSED SOFTWARE, DATABASE OR DOCUMENTATION WILL NOT  
22 INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR  
23 OTHER RIGHTS.

24

25 The name of Princeton University or Princeton may not be used in  
26 advertising or publicity pertaining to distribution of the software  
27 and/or database. Title to copyright in this software, database and  
28 any associated documentation shall at all times remain with  
29 Princeton University and LICENSEE agrees to preserve same.

**01320032** 05 n 02 domestic\_animal 0 domesticated\_animal 0 007 @ 00015568 n 0000 ~ 01320304 n 0000 ~ 01320544  
n 0000 ~ 01320872 n 0000 ~ 02086723 n 0000 ~ 02124460 n 0000 ~ 02125232 n 0000 | any of various animals that  
have been tamed and made fit for a human environment

**02085998** 05 n 02 canine 0 canid 0 011 @ 02077948 n 0000 #m 02085690 n 0000 + 02688440 a 0101 ~ 02086324 n  
0000 ~ 02086723 n 0000 ~ 02116752 n 0000 ~ 02117748 n 0000 ~ 02117987 n 0000 ~ 02119787 n 0000 ~ 02120985  
n 0000 %p 02442560 n 0000 | any of various fissiped mammals with nonretractile claws and typically long muzzles

**02086723** 05 n 03 dog 0 domestic\_dog 0 Canis\_familiaris 0 023 @ 02085998 n 0000 @ 01320032 n 0000 #m 02086515  
n 0000 #m 08011383 n 0000 ~ 01325095 n 0000 ~ 02087384 n 0000 ~ 02087513 n 0000 ~ 02087924 n 0000 ~ 02088026  
n 0000 ~ 02089774 n 0000 ~ 02106058 n 0000 ~ 02112993 n 0000 ~ 02113458 n 0000 ~ 02113610 n 0000 ~ 02113781  
n 0000 ~ 02113929 n 0000 ~ 02114152 n 0000 ~ 02114278 n 0000 ~ 02115149 n 0000 ~ 02115478 n 0000 ~ 02115987  
n 0000 ~ 02116630 n 0000 %p 02161498 n 0000 | a member of the genus Canis (probably descended from the common  
wolf) that has been domesticated by man since prehistoric times; occurs in many breeds; "the dog barked all night"

**02106058** 05 n 01 working\_dog 0 016 @ 02086723 n 0000 ~ 02106493 n 0000 ~ 02107175 n 0000 ~ 02109506 n 0000  
~ 02110072 n 0000 ~ 02110741 n 0000 ~ 02110906 n 0000 ~ 02111074 n 0000 ~ 02111324 n 0000 ~ 02111699 n 0000  
~ 02111802 n 0000 ~ 02112043 n 0000 ~ 02112177 n 0000 ~ 02112339 n 0000 ~ 02112463 n 0000 ~ 02112613 n 0000  
| any of several breeds of usually large powerful dogs bred to work as draft animals and guard and guide dogs

**02112993** 05 n 03 dalmatian 0 coach\_dog 0 carriage\_dog 0 002 @ 02086723 n 0000 ~ 02113184 n 0000 | a large breed  
having a smooth white coat with black or brown spots; originated in Dalmatia

**02107175** 05 n 03 shepherd\_dog 0 sheepdog 0 sheep\_dog 0 012 @ 02106058 n 0000 ~ 02107534 n 0000 ~ 02107903 n  
0000 ~ 02108064 n 0000 ~ 02108157 n 0000 ~ 02108293 n 0000 ~ 02108507 n 0000 ~ 02108682 n 0000 ~ 02108818  
n 0000 ~ 02109034 n 0000 ~ 02109202 n 0000 ~ 02109314 n 0000 | any of various usually long-haired breeds of dog  
reared to herd and guard sheep

**02111324** 05 n 02 bulldog 0 English\_bulldog 0 003 @ 02106058 n 0000 + 01121448 v 0101 ~ 02111567 n 0000 | a  
sturdy thickset short-haired breed with a large head and strong undershot lower jaw; developed originally in England for  
bull baiting

**02116752** 05 n 01 wolf 0 007 @ 02085998 n 0000 #m 02086515 n 0000 ~ 01324999 n 0000 ~ 02117019 n 0000 ~  
02117200 n 0000 ~ 02117364 n 0000 ~ 02117507 n 0000 | any of various predatory carnivorous canine mammals of  
North America and Eurasia that usually hunt in packs

## Field description

While parsing, skip the copyright notice. Then, each name definition follows the following format:

```
synset_offset lex_filenum ss_type w_cnt word lex_id [word lex_id...] p_cnt [ptr...]
→| gloss
```

- **synset\_offset**: Number identifying the synset, for example 02112993. **MUST be converted to a Python int**
- **lex\_filenum**: Two digit decimal integer corresponding to the lexicographer file name containing the synset, for example 03. **MUST be converted to a Python int**
- **ss\_type**: One character code indicating the synset type, store it as a string.
- **w\_cnt**: Two digit **hexadecimal** integer indicating the number of words in the synset, for example b3. **MUST be converted to a Python int**.

**WARNING:** **w\_cnt** is expressed as **hexadecimal**!

To convert an hexadecimal number like b3 to a decimal int you will need to specify the base 16 like in `int('b3', 16)` which produces the decimal integer 179.

- Afterwards, there will be **w\_cnt** words, each represented by two fields (for example, `dalmatian 0`). You **MUST** store these fields into a Python list called `words` containing a dictionary for each word, having these fields:
  - `word`: ASCII form of a word (example: `dalmatian`), with spaces replaced by underscore characters (`_`)
  - `lex_id`: One digit **hexadecimal** integer (example: 0) that **MUST be converted to a Python int**

**WARNING:** **lex\_id** is expressed as **hexadecimal**!

To convert an hexadecimal number like b3 to a decimal int you will need to specify the base 16 like in `int('b3', 16)` which produces the decimal integer 179.

- **p\_cnt**: Three digit **decimal** integer indicating the number of pointers (that is, relations like for example *IS A*) from this synset to other synsets. **MUST be converted to a Python int**

**WARNING:** differently from **w\_cnt**, the value **p\_cnt** is expressed as **decimal**!

- Afterwards, there will be **p\_cnt** pointers, each represented by four fields `pointer_symbol` `synset_offset` `pos` `source/target` (for example, `@ 02086723 n 0000`). **You MUST store these fields into a Python list called** `ptrs` containing a dictionary for each pointer, having these fields:
  - `pointer_symbol`: a symbol indicating the type of relation, for example `@` (which represents *IS A* relation)
  - `synset_offset`: the identifier of the target synset, for example 02086723. **You MUST convert this to a Python int**
  - `pos`: just parse it as a string (we will not use it)
  - `source/target`: just parse it as a string (we will not use it)

**WARNING: DO NOT** assume first pointer is an `@ (IS A) !!`

In the full database, the root synset *entity* can't possibly have a parent synset:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
00001740 03 n 01 entity 0 003 ~ 00001930 n 0000 ~ 00002137 n 0000 ~ 04431553 n ↴
↳0000 | that which is perceived or known or inferred to have its own distinct
↳existence (living or nonliving)
```

- **gloss:** Each synset contains a gloss (that is, a description). A gloss is represented as a vertical bar (|), followed by a text string that continues until the end of the line. For example, a large breed having a smooth white coat with black or brown spots; originated in Dalmatia. **Remove white spaces at the beginning/end.**

## Parsing the db

Implement a function which parses noun database filename as a text file and RETURN a dictionary containing all the synset found. Each key will be a synset\_offset mapping to a dictionary holding the fields of the corresponding synset.

**FULL expected output:** `expected_dogs_db.py`

**Expected output EXCERPT** (showing only two items):

```
>>> parse_db('dogs.noun')
{1320032: {'gloss': 'any of various animals that have been tamed and made fit '
 'for a human environment',
 'lex_filenum': 5,
 'p_cnt': 7,
 'ptrs': [{'pointer_symbol': '@',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 15568},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 1320304},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 1320544},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 1320872},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2086723},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2124460},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2125232}],
 'ss_type': 'n',
 'synset_offset': 1320032,
 'w_cnt': 2,
```

(continues on next page)

(continued from previous page)

```

'words': [{lex_id': 0, word': 'domestic_animal'},
 {'lex_id': 0, word': 'domesticated_animal'}]},
2085998: {'gloss': 'any of various fissiped mammals with nonretractile claws '
 'and typically long muzzles',
 'lex_filenum': 5,
 'p_cnt': 11,
 'ptrs': [{pointer_symbol': '@',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2077948},
 {'pointer_symbol': '#m',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2085690},
 {'pointer_symbol': '+',
 'pos': 'a',
 'source_target': '0101',
 'synset_offset': 2688440},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2086324},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2086723},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2116752},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2117748},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2117987},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2120985},
 {'pointer_symbol': '%p',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2442560}],
 'ss_type': 'n',
 'synset_offset': 2085998,
 'w_cnt': 2,
 'words': [{lex_id': 0, word': 'canine'},
 {'lex_id': 0, word': 'canid'}]}],

```

(continues on next page)

(continued from previous page)

```

.
.
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[1]:

```

def parse_db(filename):

 ret = {}
 with open(filename, encoding='utf-8') as f:
 line=f.readline()
 r = 0
 while line.startswith(' '):
 line=f.readline()
 #print(line)
 r += 1

 while line != "":
 i = 0

 d = {}

 params = line.split('|')[0].split(' ')

 d['synset_offset'] = int(params[0]) # '00001740'
 d['lex_filenum'] = int(params[1]) # '03'
 d['ss_type'] = params[2] # 'n'
 # WARNING: HERE THE STRING REPRESENT A NUMBER IN *HEXADECIMAL* FORMAT,
 # AND WE WANT TO STORE AN *INTEGER*
 # TO DO THE CONVERSION PROPERLY, YOU NEED TO USE int(my_string,_
→16)
 d['w_cnt'] = int(params[3], 16) # 'b3' -> 179
 d['words'] = []
 i = 4
 for j in range(d['w_cnt']):
 wd = {
 'word' : params[i], # 'entity'
 'lex_id': int(params[i + 1],16), # '0'
 }
 d['words'].append(wd)
 i += 2
 #
 # WARNING: HERE THE STRING REPRESENT A NUMBER IN *DECIMAL* FORMAT,
 # AND WE WANT TO STORE AN *INTEGER*
 # TO DO THE CONVERSION PROPERLY, YOU NEED TO USE int(my_string)
 d['p_cnt'] = int(params[i]) # '003' -> 3
 d['ptrs'] = []
 i += 1
 for j in range(d['p_cnt']):
 ptr = {
 'pointer_symbol': params[i], # '~'
```

(continues on next page)

(continued from previous page)

```

'synset_offset': int(params[i + 1]), # '00001930'
'pos': params[i + 2], # 'n'
'source_target': params[i + 3], # '0000'
}
d['ptrs'].append(ptr)
i += 4

d['gloss'] = line.split(' | ')[1].strip()

ret[d['synset_offset']] = d
i += 1
line=f.readline()
return ret

dogs_db = parse_db('dogs.noun')

{1320032: {'gloss': 'any of various animals that have been tamed and made fit '
 'for a human environment',
 'lex_filenum': 5,
 'p_cnt': 7,
 'ptrs': [{'pointer_symbol': '@',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 15568},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 1320304},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 1320544},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 1320872},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2086723},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2124460},
 {'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2125232}],
 'ss_type': 'n',
 'synset_offset': 1320032,
 'w_cnt': 2,
 'words': [{'lex_id': 0, 'word': 'domestic_animal'},
 {'lex_id': 0, 'word': 'domesticated_animal'}]},
2085998: {'gloss': 'any of various fissiped mammals with nonretractile claws '
}

```

(continues on next page)

(continued from previous page)

```

 'and typically long muzzles',
'lex_filenum': 5,
'p_cnt': 11,
'ptrs': [{ 'pointer_symbol': '@',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2077948},
 { 'pointer_symbol': '#m',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2085690},
 { 'pointer_symbol': '+',
 'pos': 'a',
 'source_target': '0101',
 'synset_offset': 2688440},
 { 'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2086324},
 { 'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2086723},
 { 'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2116752},
 { 'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2117748},
 { 'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2117987},
 { 'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2119787},
 { 'pointer_symbol': '~',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2120985},
 { 'pointer_symbol': '%p',
 'pos': 'n',
 'source_target': '0000',
 'synset_offset': 2442560}],
'ss_type': 'n',
'synset_offset': 2085998,
'w_cnt': 2,
'words': [{ 'lex_id': 0, 'word': 'canine' },
 { 'lex_id': 0, 'word': 'canid' }] } }

```

&lt;/div&gt;

[1]:

`def parse_db(filename):`

(continues on next page)

(continued from previous page)

```

raise Exception('TODO IMPLEMENT ME !')

dogs_db = parse_db('dogs.noun')

```

```

[2]: # EXECUTE FOR TESTING
from pprint import pformat; from expected_dogs_db import expected_dogs_db
for soff in expected_dogs_db.keys():
 if soff not in dogs_db: print('\nERROR: MISSING synset', soff); break
 for k in expected_dogs_db[soff]:
 if k not in dogs_db[soff]:
 print('\nERROR at synset', soff, '\n\n MISSING key:', k); break
 if expected_dogs_db[soff][k] != dogs_db[soff][k]:
 print('\nERROR at synset', soff, 'key:', k)
 print(' ACTUAL:\n', pformat(dogs_db[soff][k]))
 print(' EXPECTED:\n', pformat(expected_dogs_db[soff][k]))
 break
for soff in dogs_db:
 if soff not in expected_dogs_db.keys():
 print('\nERROR: found extra synset', soff, 'in dogs_db but not in expected_
→dogs_db!')
assert dogs_db == expected_dogs_db

```

## 2. to\_adj

Implement a function `to_adj` which takes the parsed db and RETURN a graph-like data structure in adjacency list format<sup>518</sup>. Each node represents a synset - as label use the first word of the synset. A node is linked to another one if there is a *IS A* relation among the nodes, so use the @ symbol to filter the hypernyms.

**IMPORTANT:** not all linked synsets are present in the dogs excerpt.

**HINT:** If you couldn't implement the `parse_db` function properly, use `expected_dogs_db.py`

### Expected output:

```

{'bulldog': ['working_dog'],
 'canine': [],
 'dalmatian': ['dog'],
 'dog': ['canine', 'domestic_animal'],
 'domestic_animal': [],
 'shepherd_dog': ['working_dog'],
 'wolf': ['canine'],
 'working_dog': ['dog']}

```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution"
data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```

def to_adj(db):
 ret = {}
 for d in db.values():

```

(continues on next page)

<sup>518</sup> <https://en.softpython.org/relational/relational1-intro-sol.html#Adjacency-lists>

(continued from previous page)

```
targets = []
for ptr in d['ptrs']:
 if ptr['pointer_symbol'] == '@':
 if ptr['synset_offset'] in db:
 targets.append(db[ptr['synset_offset']]['words'][0]['word'])
 else:
 # targets.append(ptr['synset_offset'])
ret[d['words'][0]['word']] = targets
return ret

dogs_graph = to_adj(dogs_db)
from pprint import pprint
pprint(dogs_graph)

{'bulldog': ['working_dog'],
 'canine': [],
 'dalmatian': ['dog'],
 'dog': ['canine', 'domestic_animal'],
 'domestic_animal': [],
 'shepherd_dog': ['working_dog'],
 'wolf': ['canine'],
 'working_dog': ['dog']}
```

&lt;/div&gt;

[3]:

```
def to_adj(db):
 raise Exception('TODO IMPLEMENT ME !')

dogs_graph = to_adj(dogs_db)
from pprint import pprint
pprint(dogs_graph)
```

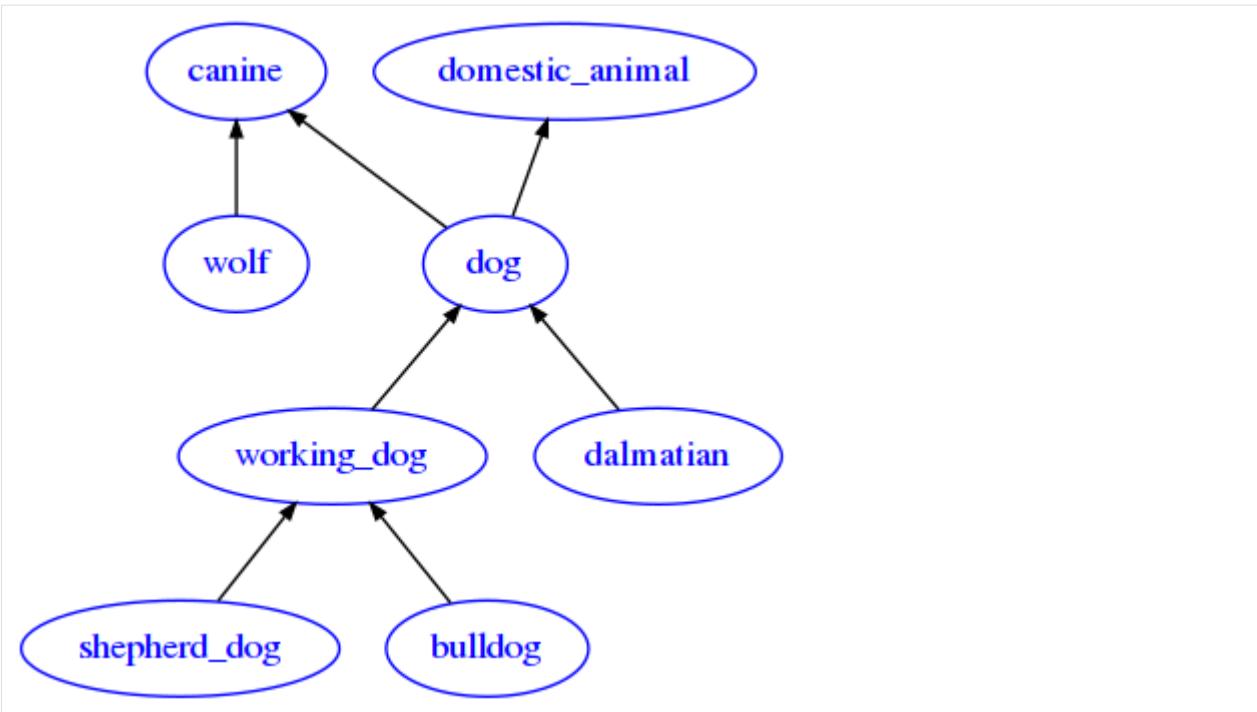
## Check results

If parsing is right, you should get the following graph:

**DO NOT** implement any drawing function, this is just for checking your results

```
[4]: from soft import draw_adj
draw_adj(dogs_graph, options={'graph':{'rankdir':'BT'}})
```

Image saved to file: expected-diagram.png



### 3. hist

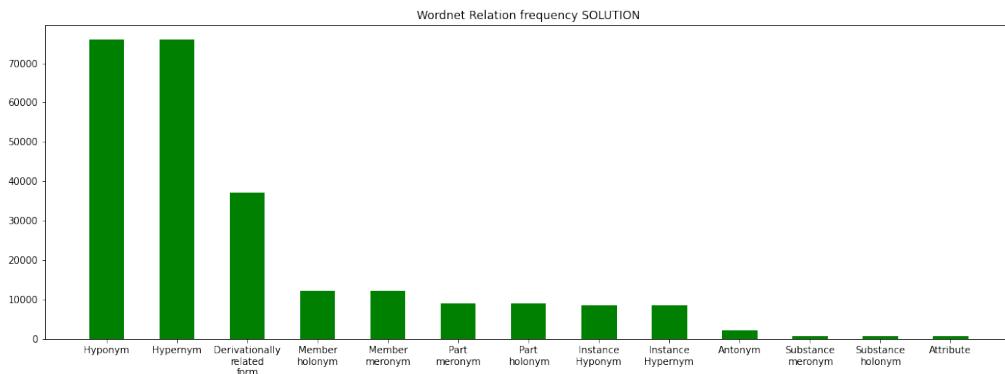
You are given a dictionary mapping each relation symbol (i.e. @) to its description (i.e. Hypernym).

Implement a function to draw the histogram of relation frequencies found in the relation links of the entire Wordnet, which can be loaded from the file [data.noun](#). If you previously implemented `parse_db` in a correct way, you should be able to load the whole db. If for any reasons you can't, try at least to draw the histogram of frequencies found in [expected\\_dogs\\_db.py](#)

- sort the histogram from greatest to lowest frequency
- display the relation names nicely, adding newlines if necessary
- **DO NOT** count the relations containing the word 'domain' inside (upper/lowercase)
- **DO NOT** count the '\` relation

**Expected output:**

```
{
 '!': 2153,
 '#m': 12287,
 '#p': 9110,
 '#s': 796,
 '%m': 12287,
 '%p': 9110,
 '%s': 796,
 '+': 37235,
 '=': 638,
 '@': 75915,
 '@i': 8588,
 '~': 75915,
 '~i': 8588}
```



<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

[5] :

```
relation_names = {
 '!': 'Antonym',
 '@': 'Hypernym',
 '@i': 'Instance Hypernym',
 '~': 'Hyponym',
 '~i': 'Instance Hyponym',
 '#m': 'Member holonym',
 '#s': 'Substance holonym',
 '#p': 'Part holonym',
 '%m': 'Member meronym',
 '%s': 'Substance meronym',
 '%p': 'Part meronym',
 '=': 'Attribute',
 '+': 'Derivationally related form',
 ';c': 'Domain of synset - TOPIC', # DISCARD
 '-c': 'Member of this domain - TOPIC', # DISCARD
 ';r': 'Domain of synset - REGION', # DISCARD
 '-r': 'Member of this domain - REGION', # DISCARD
 ';u': 'Domain of synset - USAGE', # DISCARD
 '-u': 'Member of this domain - USAGE', # DISCARD
 '\\\\': 'Pertainym (pertains to noun)' # DISCARD
}

def draw_hist(db):
 hist = {}
 for d in db.values():
 for ptr in d['ptrs']:
 ps = ptr['pointer_symbol']
 if 'domain' not in relation_names[ps].lower() and ps != '\\\\':
 if ps in hist:
 hist[ps] += 1
 else:
 hist[ps] = 0
 pprint(hist)

import numpy as np
```

(continues on next page)

(continued from previous page)

```

import matplotlib.pyplot as plt

xs = list(range(len(hist.keys())))
coords = [(x, hist[x]) for x in hist.keys()]
coords.sort(key=lambda c: c[1], reverse=True)
ys = [c[1] for c in coords]

fig = plt.figure(figsize=(18, 6))

plt.bar(xs, ys,
 0.5, # the width of the bars
 color='green', # someone suggested the default blue color is depressing,
↳ so let's put green
 align='center') # bars are centered on the xtick

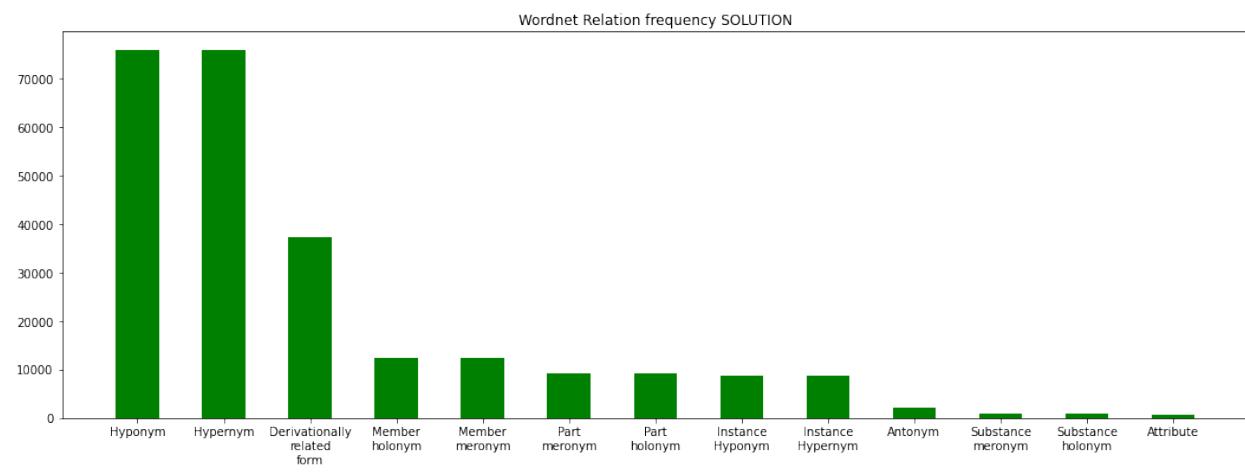
plt.title('Wordnet Relation frequency SOLUTION')
xticks = [relation_names[c[0]].replace(' ', '\n') for c in coords]
plt.xticks(xs, xticks)

plt.show()

wordnet = parse_db('data.noun')
draw_hist(wordnet)

```

{'!': 2153,  
'#m': 12287,  
'#p': 9110,  
'#s': 796,  
'%m': 12287,  
'%p': 9110,  
'%s': 796,  
'+': 37235,  
'=': 638,  
'@': 75915,  
'@i': 8588,  
'~': 75915,  
'~i': 8588}



&lt;/div&gt;

[5]:

```
relation_names = {
 '!': 'Antonym',
 '@': 'Hypernym',
 '@i': 'Instance Hypernym',
 '~': 'Hyponym',
 '~i': 'Instance Hyponym',
 '#m': 'Member holonym',
 '#s': 'Substance holonym',
 '#p': 'Part holonym',
 '%m': 'Member meronym',
 '%s': 'Substance meronym',
 '%p': 'Part meronym',
 '=': 'Attribute',
 '+': 'Derivationally related form',
 ';c': 'Domain of synset - TOPIC', # DISCARD
 '-c': 'Member of this domain - TOPIC', # DISCARD
 ';r': 'Domain of synset - REGION', # DISCARD
 '-r': 'Member of this domain - REGION', # DISCARD
 ';u': 'Domain of synset - USAGE', # DISCARD
 '-u': 'Member of this domain - USAGE', # DISCARD
 '\\\\': 'Pertainym (pertains to noun)' # DISCARD
}

def draw_hist(db):
 raise Exception('TODO IMPLEMENT ME !')

wordnet = parse_db('data.noun')
draw_hist(wordnet)
```

### 10.3.7 Metamath

#### Download worked project

Browse files online<sup>519</sup>

Metamath<sup>520</sup> is a language that can express theorems, accompanied by proofs that can be verified by a computer program. Its website lets you browse from complex theorems<sup>521</sup> up to the most basic axioms<sup>522</sup> they rely on to be proven.

The purpose of this project is to visualize the steps of the proof as a graph, and visualize statement frequencies.

#### DISCLAIMER: No panic !

You **DO NOT** need to understand *any* of the mathematics which follows. Here we are *only* interested in parsing the data and visualize it

<sup>519</sup> <https://github.com/DavidLeoni/softpython-en/tree/master/projects/metamath>

<sup>520</sup> <http://us.metamath.org>

<sup>521</sup> [http://us.metamath.org/mm\\_100.html](http://us.metamath.org/mm_100.html)

<sup>522</sup> <http://us.metamath.org/mpeuni/mmtheorems1.html#mm5>

**REQUIREMENTS:** Having read Relational data tutorial<sup>523</sup>, which contains also instructions for installing required libraries.

## What to do

1. Unzip exercises zip in a folder, you should obtain something like this:

```
metamath-prj
 metamath.ipynb
 metamath-sol.ipynb
 db.mm
 proof.txt
 soft.py
 jupman.py
```

**WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !**

2. open Jupyter Notebook from that folder. Two things should open, first a console and then a browser. The browser should show a file list: navigate the list and open the notebook `metamath.ipynb`
3. Go on reading the notebook, and write in the appropriate cells when asked

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND a create a new cell afterwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

## Metamath db

For this exercise, we have two files to consider:

- `db.mm` contains the description of a simple algebra where you can only add zero to variables
- `proof.txt` contains the awesome proof that... any variable is equal to itself

First you will load `db.mm` and parse text file into Python. here is the full content:

```
$ (Declare the constant symbols we will use $)
 $c 0 + = -> () term wff |- $.
$(Declare the metavariables we will use $)
 $v t r s P Q $.
$(Specify properties of the metavariables $)
 tt $f term t $.
 tr $f term r $.
 ts $f term s $.
 wp $f wff P $.
 wq $f wff Q $.
$(Define "term" and "wff" $)
 tze $a term 0 $.
```

(continues on next page)

<sup>523</sup> <https://en.softpython.org/relational/relational1-intro-sol.html#>

(continued from previous page)

```

tpl $a term (t + r) $.
weq $a wff t = r $.
wim $a wff (P -> Q) $.
$(State the axioms $)
 a1 $a |- (t = r -> (t = s -> r = s)) $.
 a2 $a |- (t + 0) = t $.
$(Define the modus ponens inference rule $)
 ${{
 min $e |- P $.
 maj $e |- (P -> Q) $.
 mp $a |- Q $.
 }}
}

```

Format description:

- Each row is a *statement*
- Words are separated by spaces. Each word that appears in a *statement* is called a *token*
- Tokens starting with dollar \$ are called *keywords*, you may have \$ (, \$), \$c, \$v, \$a,\$f,\${,\$}, \$.
- Statements *may* be identified with a unique arbitrary *label*, which is placed at the beginning of the row. For example, tt, weq, maj are all labels (in the file there are more):
  - tt \$f term t \$.
  - weq \$a wff t = r \$.
  - maj \$e |- ( P -> Q ) \$.
- Some rows have no label, examples:
  - \$c 0 + = -> ( ) term wff |- \$.
  - \$v t r s P Q \$.
  - \$( State the axioms \$)
  - \${
  - \$}
- in each row, after the first dollar *keyword*, you *may* have an arbitrary *sequence* of characters terminated by a dollar followed by a dot \$. **You don't need to care about the sequence meaning!** Examples:
  - tt \$f term t \$. has sequence term t
  - weq \$a wff t = r \$. has sequence wff t = r
  - \$v t r s P Q \$. has sequence t r s P Q

Now implement function `parse_db` which scans the file line by line (it is a text file, so you can use [line files examples](#)<sup>524</sup>), parses ONLY rows with labels, and RETURN a dictionary mapping labels to remaining data in the row represented as a dictionary, formatted like this (showing here only first three labels):

```
{
'a1': {'keyword': '$a',
 'sequence': '|- (t = r -> (t = s -> r = s))'},
'a2': {'keyword': '$a',
 'sequence': '|- (t + 0) = t'},
'maj': {'keyword': '$e',
 'sequence': '|- Q'}
}
```

(continues on next page)

<sup>524</sup> <https://sciprog.davidleoni.it/formats/format-sol.html#1.-line-files>

(continued from previous page)

```

 'sequence': '|- (P -> Q)'},
'min': {'keyword': '$e', 'sequence': '|- P'},
'mp': {'keyword': '$a', 'sequence': '|- Q'},
'tpl': {'keyword': '$a', 'sequence': 'term (t + r)'},
'tr': {'keyword': '$f', 'sequence': 'term r'},
'ts': {'keyword': '$f', 'sequence': 'term s'},
'tt': {'keyword': '$f', 'sequence': 'term t'},
'tze': {'keyword': '$a', 'sequence': 'term 0'},
'weq': {'keyword': '$a', 'sequence': 'wff t = r'},
'wim': {'keyword': '$a', 'sequence': 'wff (P -> Q)'},
'wp': {'keyword': '$f', 'sequence': 'wff P'},
'wq': {'keyword': '$f', 'sequence': 'wff Q'}
}
}
```

## 1. Metamath db

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[3]:

```

def parse_db(filepath):

 ret = {}
 with open(filepath, encoding='utf-8') as f:
 line=f.readline().strip()
 while line != "":
 #print(line)

 if line.startswith('$('):
 label = ''
 keyword = '$('
 sequence = ''
 elif line.split()[0].startswith('${'):
 label = ''
 keyword = '${'
 sequence = ''
 elif line.split()[0].startswith('$}'):
 label = ''
 keyword = '$}'
 sequence = ''
 elif line.split()[0].startswith('$'):
 label = ''
 keyword = line.split()[0]
 sequence = line.split()[1][:-2].strip()
 else:
 label = line.split(' $')[0].strip()
 keyword = line.split()[1]
 if line.endswith('$.'):
 sequence = line.split(keyword)[1][1:-2].strip()

 if label:
 ret[label] = {
 'keyword' : keyword,
 'sequence' : sequence
 }

```

(continues on next page)

(continued from previous page)

```

 #print(' DEBUG: FOUND', label, ':', ret[label])
 else:
 #print(' DEBUG: DISCARDED')
 line=f.readline().strip()
return ret

db_mm = parse_db('db.mm')

assert db_mm['tt'] == {'keyword': '$f', 'sequence': 'term t'}
assert db_mm['maj'] == {'keyword': '$e', 'sequence': '|- (P -> Q)'}
careful 'mp' label shouldn't have spaces inside !
assert 'mp' in db_mm
assert db_mm['mp'] == {'keyword': '$a', 'sequence': '|- Q'}

from pprint import pprint
pprint(db_mm)

{'a1': {'keyword': '$a', 'sequence': '|- (t = r -> (t = s -> r = s))'},
 'a2': {'keyword': '$a', 'sequence': '|- (t + 0) = t'},
 'maj': {'keyword': '$e', 'sequence': '|- (P -> Q)'},
 'min': {'keyword': '$e', 'sequence': '|- P'},
 'mp': {'keyword': '$a', 'sequence': '|- Q'},
 'tpl': {'keyword': '$a', 'sequence': 'term (t + r)'},
 'tr': {'keyword': '$f', 'sequence': 'term r'},
 'ts': {'keyword': '$f', 'sequence': 'term s'},
 'tt': {'keyword': '$f', 'sequence': 'term t'},
 'tze': {'keyword': '$a', 'sequence': 'term 0'},
 'weq': {'keyword': '$a', 'sequence': 'wff t = r'},
 'wim': {'keyword': '$a', 'sequence': 'wff (P -> Q)'},
 'wp': {'keyword': '$f', 'sequence': 'wff P'},
 'wq': {'keyword': '$f', 'sequence': 'wff Q'}}

```

&lt;/div&gt;

[3]:

```

def parse_db(filepath):
 raise Exception('TODO IMPLEMENT ME !')

db_mm = parse_db('db.mm')

assert db_mm['tt'] == {'keyword': '$f', 'sequence': 'term t'}
assert db_mm['maj'] == {'keyword': '$e', 'sequence': '|- (P -> Q)'}
careful 'mp' label shouldn't have spaces inside !
assert 'mp' in db_mm
assert db_mm['mp'] == {'keyword': '$a', 'sequence': '|- Q'}

from pprint import pprint
pprint(db_mm)

```

## 2.1 Metamath proof

A proof file is made of steps, one per row. Each statement, in order to be proven, needs other steps to be proven until very basic facts called axioms are reached, which need no further proof (typically proofs in Metamath are shown in much shorter format, but here we use a more explicit way)

So a proof can be nicely displayed as a tree of the steps it is made of, where the top node is the step to be proven and the axioms are the leaves of the tree.

Complete content of proof.txt:

```

1 tt $f term t
2 tze $a term 0
3 1,2 tpl $a term (t + 0)
4 tt $f term t
5 3,4 weq $a wff (t + 0) = t
6 tt $f term t
7 tt $f term t
8 6,7 weq $a wff t = t
9 tt $f term t
10 9 a2 $a |- (t + 0) = t
11 tt $f term t
12 tze $a term 0
13 11,12 tpl $a term (t + 0)
14 tt $f term t
15 13,14 weq $a wff (t + 0) = t
16 tt $f term t
17 tze $a term 0
18 16,17 tpl $a term (t + 0)
19 tt $f term t
20 18,19 weq $a wff (t + 0) = t
21 tt $f term t
22 tt $f term t
23 21,22 weq $a wff t = t
24 20,23 wim $a wff ((t + 0) = t -> t = t)
25 tt $f term t
26 25 a2 $a |- (t + 0) = t
27 tt $f term t
28 tze $a term 0
29 27,28 tpl $a term (t + 0)
30 tt $f term t
31 tt $f term t
32 29,30,31 a1 $a |- ((t + 0) = t -> ((t + 0) = t -> t = t))
33 15,24,26,32 mp $a |- ((t + 0) = t -> t = t)
34 5,8,10,33 mp $a |- t = t

```

Each line represents a step of the proof. Last line is the final goal of the proof.

Each line contains, in order:

- a step number at the beginning, starting from 1 (step\_id)
- possibly a list of other step\_ids, separated by commas, like 29,30,31 - they are references to previous rows
- label of the db\_mm statement referenced by the step, like tt, tze, weq - that label must have been defined somewhere in db.mm file
- statement type: a token starting with a dollar, like \$a, \$f
- a sequence of characters, like (for you they are just characters, **don't care about the meaning !**):
  - term ( t + 0 )

- |- ( ( t + 0 ) = t -> ( ( t + 0 ) = t -> t = t ) )

Implement function `parse_proof`, which takes a `filepath` to the proof and RETURN a list of steps expressed as a dictionary, in this format (showing here only first 5 items):

**NOTE:** referenced `step_ids` are **integer** numbers and they are the original ones from the file, meaning they start **from one**.

```
[{'keyword': '$f',
 'label': 'tt',
 'sequence': 'term t',
 'step_ids': []},
 {'keyword': '$a',
 'label': 'tze',
 'sequence': 'term 0',
 'step_ids': []},
 {'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [1, 2]},
 {'keyword': '$f',
 'label': 'tt',
 'sequence': 'term t',
 'step_ids': []},
 {'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff (t + 0) = t',
 'step_ids': [3, 4]},
 .
 .
 .]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);"

data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

```
[5]: def parse_proof(filepath):

 ret = []

 with open(filepath, encoding='utf-8') as f:
 line=f.readline().strip()

 while line != "":

 step_id = int(line.split(' ')[0])
 label = line.split('$')[0].strip().split(' ')[-1]
 keyword = '$' + line.split('$')[1][1:]
 sequence = line.split('$')[1][2:]
 candidate_step_ids = line.split(' ')[1]

 if candidate_step_ids != label:
 step_ids = [int(x) for x in line.split(' ')[1].split(',')]

 else:
 step_ids = []
 #print('deps =', deps)
```

(continues on next page)

(continued from previous page)

```

 ret.append({
 'step_ids': step_ids,
 'sequence': sequence,
 'label': label,
 'keyword': keyword
 })

 line=f.readline().strip()
return ret

proof = parse_proof('proof.txt')

assert proof[0] == {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids':[]}
assert proof[1] == {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids':[]}
assert proof[2] == {'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [1, 2]}
assert proof[4] == {'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff (t + 0) = t',
 'step_ids': [3, 4]}
assert proof[33] == { 'keyword': '$a',
 'label': 'mp',
 'sequence': '|- t = t',
 'step_ids': [5, 8, 10, 33]}

pprint(proof)
[{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [1, 2]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff (t + 0) = t',
 'step_ids': [3, 4]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'weq', 'sequence': 'wff t = t', 'step_ids': [6, 7]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
 'label': 'a2',
 'sequence': '|- (t + 0) = t',
 'step_ids': [9]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [11, 12]},
```

(continues on next page)

(continued from previous page)

```
{
 "keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$a",
 "label": "weq",
 "sequence": "wff (t + 0) = t",
 "step_ids": [13, 14]},
 {"keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$a", "label": "tze", "sequence": "term 0", "step_ids": []},
 {"keyword": "$a",
 "label": "tpl",
 "sequence": "term (t + 0)",
 "step_ids": [16, 17]},
 {"keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$a",
 "label": "weq",
 "sequence": "wff (t + 0) = t",
 "step_ids": [18, 19]},
 {"keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$a",
 "label": "weq",
 "sequence": "wff t = t",
 "step_ids": [21, 22]},
 {"keyword": "$a",
 "label": "wim",
 "sequence": "wff ((t + 0) = t -> t = t)",
 "step_ids": [20, 23]},
 {"keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$a",
 "label": "a2",
 "sequence": "|- (t + 0) = t",
 "step_ids": [25]},
 {"keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$a", "label": "tze", "sequence": "term 0", "step_ids": []},
 {"keyword": "$a",
 "label": "tpl",
 "sequence": "term (t + 0)",
 "step_ids": [27, 28]},
 {"keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$f", "label": "tt", "sequence": "term t", "step_ids": []},
 {"keyword": "$a",
 "label": "a1",
 "sequence": "|- ((t + 0) = t -> ((t + 0) = t -> t = t))",
 "step_ids": [29, 30, 31]},
 {"keyword": "$a",
 "label": "mp",
 "sequence": "|- ((t + 0) = t -> t = t)",
 "step_ids": [15, 24, 26, 32]},
 {"keyword": "$a",
 "label": "mp",
 "sequence": "|- t = t",
 "step_ids": [5, 8, 10, 33]}]
```

&lt;/div&gt;

```
[5]: def parse_proof(filepath):
 raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```

proof = parse_proof('proof.txt')

assert proof[0] == {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []
 ↳ []}
assert proof[1] == {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []
 ↳ []}
assert proof[2] == {'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [1, 2]}
assert proof[4] == {'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff (t + 0) = t',
 'step_ids': [3, 4]}
assert proof[33] == {'keyword': '$a',
 'label': 'mp',
 'sequence': '|- t = t',
 'step_ids': [5, 8, 10, 33]}

pprint(proof)

[{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [1, 2]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff (t + 0) = t',
 'step_ids': [3, 4]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'weq', 'sequence': 'wff t = t', 'step_ids': [6, 7]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
 'label': 'a2',
 'sequence': '|- (t + 0) = t',
 'step_ids': [9]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [11, 12]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff (t + 0) = t',
 'step_ids': [13, 14]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [16, 17]},
```

(continues on next page)

(continued from previous page)

```
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff (t + 0) = t',
 'step_ids': [18, 19]},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff t = t',
 'step_ids': [21, 22]},
{'keyword': '$a',
 'label': 'wim',
 'sequence': 'wff ((t + 0) = t -> t = t)',
 'step_ids': [20, 23]},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a',
 'label': 'a2',
 'sequence': '|- (t + 0) = t',
 'step_ids': [25]},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
{'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term (t + 0)',
 'step_ids': [27, 28]},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a',
 'label': 'a1',
 'sequence': '|- ((t + 0) = t -> ((t + 0) = t -> t = t))',
 'step_ids': [29, 30, 31]},
{'keyword': '$a',
 'label': 'mp',
 'sequence': '|- ((t + 0) = t -> t = t)',
 'step_ids': [15, 24, 26, 32]},
{'keyword': '$a',
 'label': 'mp',
 'sequence': '|- t = t',
 'step_ids': [5, 8, 10, 33]}]
```

## 2. Checking proof

If you've done everything properly, by executing following cells you should be able to see nice graphs.

**IMPORTANT: You do not need to implement anything!**

Just look if results match expected graphs

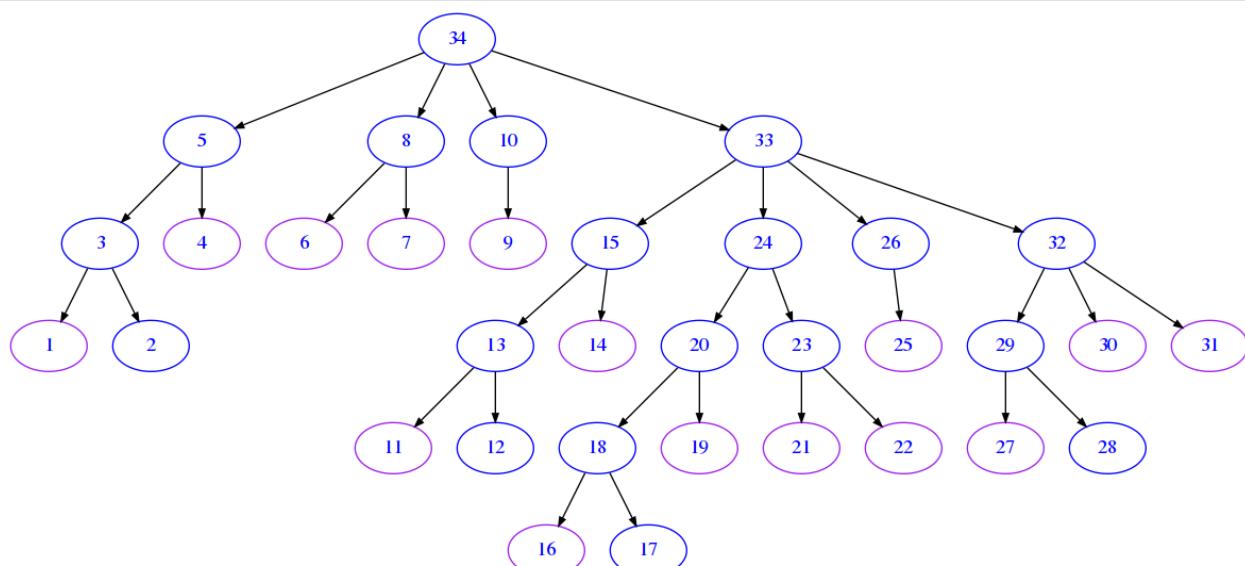
## Overview plot

Here we only show step numbers using function `draw_proof` defined in `sciprog` library

```
[6]: from soft import draw_proof
uncomment and check
#draw_proof(proof, db_mm, only_ids=True) # all graph, only numbers
```

```
[7]: print()
print('***** EXPECTED COMPLETE GRAPH *****')
draw_proof(proof, db_mm, only_ids=True)
```

\*\*\*\*\* EXPECTED COMPLETE GRAPH \*\*\*\*\*



## Detail plot

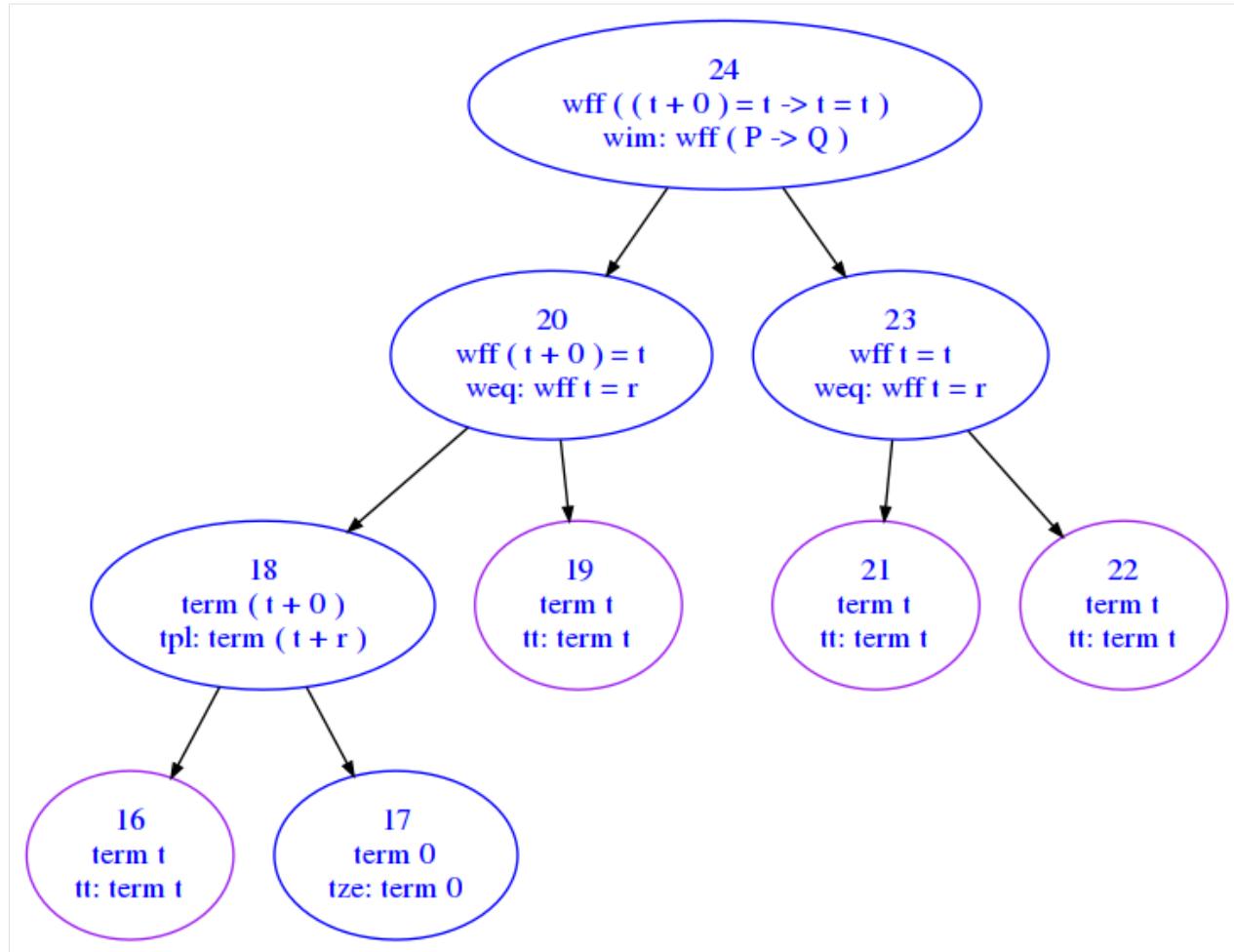
Here we show data from both the `proof` and the `db_mm` we calculated earlier. To avoid having a huge graph we only focus on subtree starting from `step_id=24`.

To understand what is shown, look at node 20: - first line contains statement `wff ( t + 0 ) = t` taken from line 20 of `proof` file - second line `weq: wff t = r` is taken from `db_mm`, and means rule labeled `weq` was used to derive the statement in the first line.

```
[8]: # uncomment and check
#draw_proof(proof, db_mm, step_id=24)
```

```
[9]: print()
print('***** EXPECTED DETAIL GRAPH *****')
draw_proof(proof, db_mm, step_id=24)
```

\*\*\*\*\* EXPECTED DETAIL GRAPH \*\*\*\*\*



### 3. Top statements

We can measure the importance of theorems and definitions (in general, *statements*) by counting how many times they are referenced in proofs.

**3.1 plot:** Write some code to plot the histogram of *statement* labels referenced by steps in `proof`, from most to least frequently referenced.

A label gets a count each time a step references another step with that label.

For example, in the subgraph above:

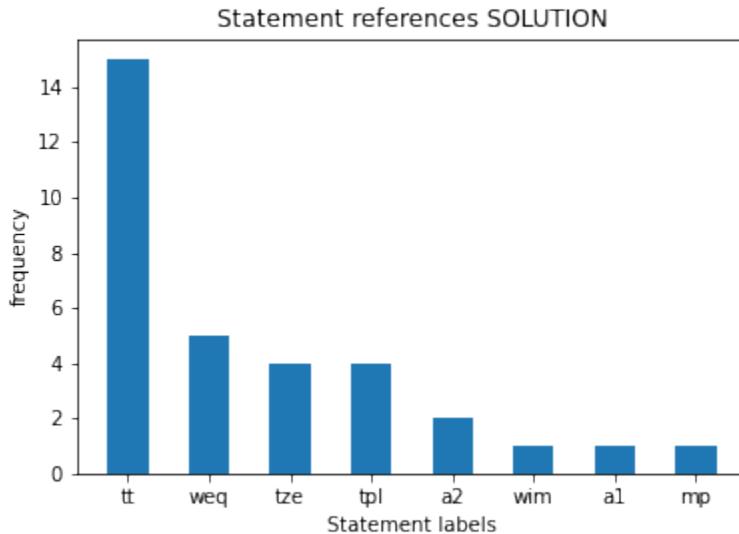
- `tt` is referenced 4 times, that is, there are 4 steps referencing other steps which contain the label `tt`
- `weq` is referenced 2 times
- `tpl` and `tze` are referenced 1 time each
- `wim` is referenced 0 times (it is only present in the last node, which being the root node cannot be referenced by any step)

**NOTE:** the previous counts are just for the subgraph example.

In your exercise, you will need to consider all the steps

**3.2 print list:** Below the graph, print the list of labels from most to least frequent, associating them to corresponding statement sequence taken from db\_mm

**Expected output:**



```
tt : term t
weq : wff t = r
tze : term 0
tpl : term (t + r)
a2 : |- (t + 0) = t
wim : wff (P -> Q)
a1 : |- (t = r -> (t = s -> r = s))
mp : |- Q
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);;" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a><div class="jupman-sol jupman-sol-code" style="display:none">

[10]:

```
import numpy as np
import matplotlib.pyplot as plt

write here

freqs = {}
for step in proof:
 for step_id in step['step_ids']:
 label = proof[step_id-1]['label']
 if label not in freqs:
 freqs[label] = 1
 else:
 freqs[label] += 1
```

(continues on next page)

(continued from previous page)

```

xs = np.arange(len(freqs.keys()))

coords = [(k, freqs[k]) for k in freqs]

coords.sort(key=lambda c: c[1], reverse=True)

ys_in = [c[1] for c in coords]

plt.bar(xs, ys_in, 0.5, align='center')

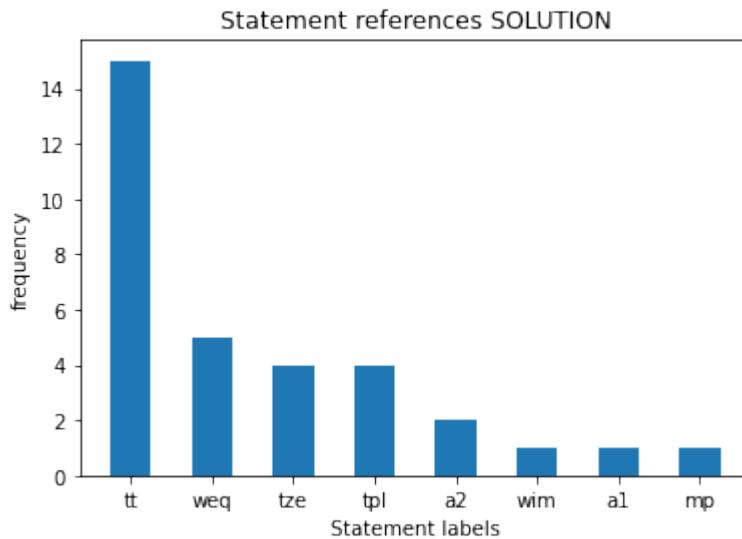
plt.title("Statement references SOLUTION")
plt.xticks(xs, [c[0] for c in coords])

plt.xlabel('Statement labels')
plt.ylabel('frequency')

plt.show()

for c in coords:
 print(c[0], ':', '\t', db_mm[c[0]]['sequence'])

```



```

tt : term t
weq : wff t = r
tze : term 0
tpl : term (t + r)
a2 : |- (t + 0) = t
wim : wff (P -> Q)
a1 : |- (t = r -> (t = s -> r = s))
mp : |- Q

```

&lt;/div&gt;

[10]:

(continues on next page)

(continued from previous page)

```
import numpy as np
import matplotlib.pyplot as plt

write here
```

[ ]:



## E - APPENDIX

### 11.1 Commandments

The Supreme Committee for the Doctrine of Coding has ruled important Commandments you shall follow.  
If you accept their wise words, you shall become a true Python Jedi.

**WARNING:** if you don't follow the Commandments, you will end up in *Debugging Hell!*

#### 11.1.1 I COMMANDMENT

---

##### You shall write Python code

---

Who does not writes Python code, does not learn Python

#### 11.1.2 II COMMANDMENT

---

##### Whenever you insert a variable in a `for` cycle, such variables must be new

---

If you defined the variable before, you shall not reintroduce it in a `for`, because doing so might bring confusion in the minds of the readers.

So avoid such sins:

```
[1]: i = 7
for i in range(3): # sin, you lose variable i
 print(i)

print(i) # prints 2 and not 7 !!

0
1
2
2
```

```
[2]: for i in range(2):

 for i in range(5): # debugging hell, you lose the i of external cycle
 print(i)

 print(i) # prints 4 !!

0
1
2
3
4
4
0
1
2
3
4
4
```

```
[3]: def f(i):
 for i in range(3): # sin, you lose parameter i
 print(i)

 print(i) # prints 2, not the 7 we passed!

f(7)

0
1
2
2
```

### 11.1.3 III COMMANDMENT

---

#### You shall never ever reassign function parameters

---

Never perform any of these assignments, as you risk losing the parameter passed during function call:

```
[4]: def sin(my_int):
 my_int = 666 # you lost the 5 passed from external call!
 print(my_int) # prints 666

x = 5
sin(x)

666
```

Same reasoning can be applied to all other types:

```
[5]: def evil(my_string):
 my_string = "666"
```

```
[6]: def disgrace(my_list):
 my_list = [666]
```

```
[7]: def delirium(my_dict):
 my_dict = {"evil":666}
```

For the sole case when you have composite parameters like lists or dictionaries, you can write like below IF AND ONLY IF the function description requires to MODIFY the internal elements of the parameter (like for example sorting a list in-place or changing the field of a dictionary).

```
[8]: # MODIFY my_list in some way
def allowed(my_list):
 my_list[2] = 9 # OK, function text requires it

outside = [8,5,7]
allowed(outside)
print(outside)

[8, 5, 9]
```

```
[9]: # MODIFY dictionary in some way
def ok(dictionary):
 dictionary["my field"] = 5 # OK, function text requires it
```

```
[10]: # MODIFY instance in some way
def fine(class_instance):
 class_instance.my_field = 7 # OK, function text requires it
```

On the other hand, if the function requires to RETURN a NEW object, you shall not fall into the temptation of modifying the input:

```
[11]: # RETURN a NEW sorted list
def pain(my_list):
 my_list.sort() # BAD, you are modifying the input list instead of creating a
 ↪new one!
 return my_list
```

```
[12]: # RETURN a NEW list
def crisis(my_list):
 my_list[0] = 5 # BAD, as above
 return my_list
```

```
[13]: # RETURN a NEW dictionary
def torment(my_dict):
 my_dict['a'] = 6 # BAD, you are modifying the input dictionary instead of
 ↪creating a new one!
 return my_dict
```

```
[14]: # RETURN a NEW class instance
def desperation(my_instance):
 my_instance.my_field = 6 # BAD, you are modifying the input object
 # instead of creating a new one!
 return my_instance
```

## 11.1.4 IV COMMANDMENT

---

You shall never ever reassign values to function calls or methods

---

WRONG:

```
my_function() = 666
my_function() = 'evil'
my_function() = [666]
```

CORRECT:

```
x = 5
y = my_fun()
z = []
z[0] = 7
d = dict()
d["a"] = 6
```

Function calls like `my_function()` return calculations results and store them in a box in memory which is only created for the purposes of the call, and Python will not allow us to reuse it like it were a variable.

Whenever you see `name()` in the left part, it *cannot* be followed by the equality sign `=` (but it can be followed by two equals sign `==` if you are doing a comparison).

## 11.1.5 V COMMANDMENT

---

You shall never ever redefine system functions

---

Python has several system defined functions. For example `list` is a Python type: as such, you can use it for example as a function to convert some type to a list:

```
[15]: list("ciao")
[15]: ['c', 'i', 'a', 'o']
```

When you allow the forces of evil to take the best of you, you might be tempted to use reserved words like `list` as a variable for your own miserable purposes:

```
[16]: list = ['my', 'pitiful', 'list']
```

Python allows you to do so, but we do **not**, for the consequences are disastrous.

For example, if you now attempt to use `list` for its intended purpose like casting to list, it won't work anymore:

```
list("ciao")
```

```

TypeError Traceback (most recent call last)
<ipython-input-4-c63add832213> in <module>()
 1 list("ciao")
----> 1 list("ciao")

TypeError: 'list' object is not callable
```

In particular, we recommend to **not redefine** these precious functions:

- bool, int, float, tuple, str, list, set, dict
- max, min, sum
- next, iter
- id, dir, vars, help

## 11.1.6 VI COMMANDMENT

---

**You shall use `return` command only if you see written RETURN in function description!**

---

If there is no `return` in function description, the function is intended to return `None`. In this case you don't even need to write `return None`, as Python will do it implicitly for you.

## 11.1.7 VII COMMANDMENT

---

**You shall also write on paper!**

---

If staring at the monitor doesn't work, help yourself and draw a representation of the state of the program. Tables, nodes, arrows, all can help figuring out a solution for the problem.

## 11.1.8 VIII COMMANDMENT

---

**You shall never ever reassing `self` !**

---

Never write horrors such as this:

```
[17]: class MyClass:
 def my_method(self):
 self = {'my_field':666} # SIN
```

Since `self` is a kind of a dictionary, you might be tempted to write like above, but to external world it will bring no effect.

For example, let's suppose somebody from outside makes a call like this:

```
[18]: mc = MyClass()
mc.my_method()
```

After the call `mc` will not point to `{'my_field':666}`

```
[19]: mc
[19]: <__main__.MyClass at 0x7f4d9423b210>
```

and will not have `my_field`:

```
mc.my_field

AttributeError Traceback (most recent call last)
<ipython-input-26-5c4e6630908d> in <module>()
----> 1 mc.my_field

AttributeError: 'MyClass' object has no attribute 'my_field'
```

Following the same reasoning, you shall never reassign `self` to lists or others things:

```
[20]: class MyClass:
 def my_method(self):
 self = ['evil'] # YET ANOTHER SIN
 self = 666 # NO NO NO
```

### 11.1.9 IX COMMANDMENT

---

#### You shall test!

---

Untested code *does not work* by definition. For ideas on how to test it, have a look at [Errors and testing](#)<sup>525</sup>

### 11.1.10 X COMMANDMENT

---

#### You shall never ever add nor remove elements from a sequence you are iterating with a `for`!

---

Falling into such temptations **would produce totally unpredictable behaviours** (do you know the expression *pulling the rug out from under your feet* ? )

**Do not add**, because you risk walking on a tapis roulant that never turns off:

```
my_list = ['a','b','c','d','e']
for el in my_list:
 my_list.append(el) # YOU ARE CLOGGING COMPUTER MEMORY
```

**Do not remove**, because you risk corrupting the natural order of things:

```
[21]: my_list = ['a','b','c','d','e']

for el in my_list:
 my_list.remove(el) # VERY BAD IDEA
```

Look at the code. You think we removed everything, uh?

```
[22]: my_list
[22]: ['b', 'd']
```

O\_o ' Do not even try to make sense of such sorcery - nobody can, because it is related to Python internal implementation.

Our version of Python gives this absurd result, yours may give another. Same applies for iteration on sets and dictionaries.

**You are warned.**

---

<sup>525</sup> <https://en.softpython.org/errors-and-testing/errors-and-testing-sol.ipynb>

If you really need to remove stuff from the sequence you are iterating on, use a `while` cycle<sup>526</sup> or first make a copy of the original sequence.

[ ]:

## 11.2 Revisions

SoftPython English

<https://en.softpython.org>

**NOTE:** Latest news are published *at the top of book home*

**August 11, 2022:** added *Applications: Database tutorial*, fixed some visualization bug in Python Tutor

**April 16, 2022:** added a lot of *worked projects* (ported from sciprog course exams)

---

**January 2022: English version is mostly complete and usable as introductory course material**

---

See [Github issues](#)<sup>527</sup> for missing parts.

**November 4, 2021:** restructured *relational data*, added challenges

**October 29, 2021:** restructured *pandas intro page*

**October 28, 2021:** substituted graph stuff in *visualization intro* with other exercises

**October 15, 2021:** added *formats challenges*, moved graph formats and binary relations to *relational data* section

**October 14, 2021** added *functions*, *matrix lists challenges* and *mixed structures challenges*

**October 8, 2021** added *for*, *while*, *sequences* challenges

**October 7, 2021:** added *sets*, *dictionary*, *if* challenges

**October 1, 2021:** added *lists* and *tuples* challenges

**September 30, 2021:** added *string challenges*

**September 22, 2021:**

- major update, added new exercises and pages
- added *worked projects* section

**October 3, 2020:** updated *References* page

---

<sup>526</sup> <https://en.softpython.org/while/while1-sol.html>

<sup>527</sup> <https://github.com/DavidLeoni/softpython-en/issues>

## 11.2.1 July 2020

Site is online

## 11.3 References

### 11.3.1 Foundations of Python Programming

Runestone Academy [FOPP<sup>528</sup>](#) is a practical free online book with many projects and related ‘hands on’ theory, definitely recommended!

Note on graphics: to make activities more interesting, the book often asks to visualize data with the following libraries:

- `turtle` is a Python module which was designed really only for didactical purposes. While fun, you will most probably want to try doing the same exercises using a more ‘serious’ library like `matplotlib`
- `cimage`: this is a simple image manipulation library, made mostly for didactical purposes: you might want to try `numpy` and `matplotlib` instead
- `altair` is a ‘pro’ library for cool interactive visualizations: we don’t treat `altair` in this book, you can try it or stick with the good old `matplotlib`

### 11.3.2 W3Resources website

Contains many simple exercises on Python basics, do them!

- [Basic1<sup>529</sup>](#), [Basic2<sup>530</sup>](#), [String<sup>531</sup>](#), [List<sup>532</sup>](#), [Dictionary<sup>533</sup>](#), [Tuple<sup>534</sup>](#), [Sets<sup>535</sup>](#), [Condition Statements and Loops<sup>536</sup>](#), [Functions<sup>537</sup>](#), [Lambda<sup>538</sup>](#), [CSV Read Write<sup>539</sup>](#)

### 11.3.3 Software Carpentry

Software Carpentry<sup>540</sup> is a website full of free educational resources, there is definitely a lot of good stuff to discover. We highlight these exercises (in tutorial format):

- [Programming with Python<sup>541</sup>](#): Nice tutorial with many exercises about processing a csv with topics: python basics, `numpy`, `csv`
- [Plotting and programming with Python<sup>542</sup>](#) More advanced, uses `pandas`

<sup>528</sup> <https://runestone.academy/runestone/books/published/fopp/index.html>

<sup>529</sup> <https://www.w3resource.com/python-exercises/>

<sup>530</sup> <https://www.w3resource.com/python-exercises/basic/>

<sup>531</sup> <https://www.w3resource.com/python-exercises/string/>

<sup>532</sup> <https://www.w3resource.com/python-exercises/list/>

<sup>533</sup> <https://www.w3resource.com/python-exercises/dictionary/>

<sup>534</sup> <https://www.w3resource.com/python-exercises/tuple/>

<sup>535</sup> <https://www.w3resource.com/python-exercises/sets/>

<sup>536</sup> <https://www.w3resource.com/python-exercises/python-conditional-statements-and-loop-exercises.php>

<sup>537</sup> <https://www.w3resource.com/python-exercises/python-functions-exercises.php>

<sup>538</sup> <https://www.w3resource.com/python-exercises/lambda/index.php>

<sup>539</sup> <https://www.w3resource.com/python-exercises/csv/index.php>

<sup>540</sup> <https://software-carpentry.org/lessons/>

<sup>541</sup> <https://swcarpentry.github.io/python-novice-inflammation/>

<sup>542</sup> <https://swcarpentry.github.io/python-novice-gapminder/>

You may find other stuff in Community Developed Lessons for Jupyter<sup>543</sup> and Python<sup>544</sup>

### 11.3.4 Edabit

Contains many python exercises<sup>545</sup> with solutions. Here we put a small selection, for others you may look at ‘Very hard’ level, they are not so hard after all.

#### Edabit - Basics

- Calculated Bonus<sup>546</sup>

#### Edabit - Strings

- First Before Second Letter<sup>547</sup>
- Wrap Around<sup>548</sup>
- C\*ns\*r\*d Str\*ngs<sup>549</sup>
- Valid Rondo Form<sup>550</sup>
- Parenthesis Clusters<sup>551</sup>
- Count Missing Numbers<sup>552</sup>
- Math Making<sup>553</sup>
- To Adjust the Time<sup>554</sup>

#### Edabit - Lists

- Combined Consecutive Sequence<sup>555</sup>
- Prison break<sup>556</sup>
- Water Balloon<sup>557</sup>
- Fulcrum<sup>558</sup>
- Beginning and End Pairs<sup>559</sup>

<sup>543</sup> <https://carpentries.org/community-lessons/#jupyter-notebook>

<sup>544</sup> <https://carpentries.org/community-lessons/#python>

<sup>545</sup> <https://edabit.com/challenges/python3>

<sup>546</sup> <https://edabit.com/challenge/ksiA6Q34iXgTcMeZF>

<sup>547</sup> <https://edabit.com/challenge/D6XfxhRobdQvbKX4v>

<sup>548</sup> <https://edabit.com/challenge/Q9EkExy6BYLnqBCQB>

<sup>549</sup> <https://edabit.com/challenge/ehyZvt6AJF4rKFFXT>

<sup>550</sup> <https://edabit.com/challenge/stXWY2iufNhBo9sTW>

<sup>551</sup> <https://edabit.com/challenge/Fpymv2HieqEd7ptAq>

<sup>552</sup> <https://edabit.com/challenge/vBwRuR4mF5yQ4cNuc>

<sup>553</sup> <https://edabit.com/challenge/3r7z6pkGnd4u7eZAd>

<sup>554</sup> <https://edabit.com/challenge/YsD3af7LgaH6JRSCH>

<sup>555</sup> <https://edabit.com/challenge/mHLAmj4vmRuXrT8Nb>

<sup>556</sup> <https://edabit.com/challenge/SHdu4GwBQehhDm4xT>

<sup>557</sup> <https://edabit.com/challenge/3y2FmfjhbiQPPYbcn>

<sup>558</sup> <https://edabit.com/challenge/pn7QpvW2fW9grvYYE>

<sup>559</sup> <https://edabit.com/challenge/HrCuzAKE6skEYgDmf>

- Sort by the Letters<sup>560</sup>
- Anonymous Name<sup>561</sup>
- Almost Palindrome<sup>562</sup>
- Number of Two or More Consecutive Ones<sup>563</sup>
- Rearrange the Number<sup>564</sup>

### Edabit - Dictionaries

- How Many Unique Styles?<sup>565</sup>
- People Sort<sup>566</sup>
- Encoded String Parse<sup>567</sup>
- Generating Words from Names<sup>568</sup>

### Edabit - Matrices

- Tallest Skyscraper<sup>569</sup>
- Majority vote<sup>570</sup>
- Make a Box<sup>571</sup>
- Advanced List Sort<sup>572</sup>
- Layers in a Rug<sup>573</sup>, a bit convoluted, but interesting
- Leaderbord Sort<sup>574</sup>
- Concert seats<sup>575</sup>
- Word Nests - Part 2<sup>576</sup>
- Tic Tac Toe<sup>577</sup>
- Cleaning Project Files<sup>578</sup>

<sup>560</sup> <https://edabit.com/challenge/LhMkMu46rG8EweYf7>

<sup>561</sup> <https://edabit.com/challenge/MKP8QxzuDaqYAJ6sZ>

<sup>562</sup> <https://edabit.com/challenge/APNhiaMCuRSwALN63>

<sup>563</sup> <https://edabit.com/challenge/u4rHyBDs5RM2PfNxy>

<sup>564</sup> <https://edabit.com/challenge/jwzAdBnJnBxCe4AXP>

<sup>565</sup> <https://edabit.com/challenge/AvP94XqJvjoMk5PT>

<sup>566</sup> <https://edabit.com/challenge/hDT4TR9JAoQ3BPuCH>

<sup>567</sup> <https://edabit.com/challenge/7vN8ZRw43yuWNoy3Y>

<sup>568</sup> <https://edabit.com/challenge/sDvjdcBrbHoXKvDsZ>

<sup>569</sup> <https://edabit.com/challenge/76ibd8jZxvhAwDskb>

<sup>570</sup> <https://edabit.com/challenge/pQavNkBbdmvSMmx5x>

<sup>571</sup> <https://edabit.com/challenge/dy3WWJr34gSGRPLee>

<sup>572</sup> <https://edabit.com/challenge/6vSZmN66xhMRDX8YT>

<sup>573</sup> <https://edabit.com/challenge/LaBMjgbMjf5BajczX>

<sup>574</sup> <https://edabit.com/challenge/ZsBPGxuBsbbHfPSkk>

<sup>575</sup> <https://edabit.com/challenge/xbjDMxzpFcsAWKp97>

<sup>576</sup> <https://edabit.com/challenge/ZwmfETsazpvBTWoQT>

<sup>577</sup> <https://edabit.com/challenge/A8gEGRXqMwRWQJvBf>

<sup>578</sup> <https://edabit.com/challenge/NC888jKPkquSDqaaH>

### 11.3.5 LeetCode

Website with collections of exercises sorted by difficulty and acceptance rate, quite performance-oriented. You can generally try sorting by *Acceptance* and *Easy* filters.

- [leetcode.com](https://leetcode.com)<sup>579</sup>

We put here a selection.

#### LeetCode - Strings

Check string problems<sup>580</sup> sorted by *Acceptance* and *Easy*. In particular:

- [Shuffle Strings](https://leetcode.com/problems/shuffle-string/)<sup>581</sup>
- [Increasing Decreasing String](https://leetcode.com/problems/increasing-decreasing-string/)<sup>582</sup>
- [Detect Capital](https://leetcode.com/problems/detect-capital/)<sup>583</sup>
- [Unique email addresses](https://leetcode.com/problems/unique-email-addresses/)<sup>584</sup>
- [Robot return to origin](https://leetcode.com/problems/robot-return-to-origin/)<sup>585</sup>
- [String matching in an Array](https://leetcode.com/problems/string-matching-in-an-array/)<sup>586</sup>
- [Reverse Words in a String III](https://leetcode.com/problems/reverse-words-in-a-string-iii/)<sup>587</sup>
- [Unique Morse codes](https://leetcode.com/problems/unique-morse-codes/)<sup>588</sup>
- [Goat Latin](https://leetcode.com/problems/goat-latin/)<sup>589</sup>
- [Count Binary Substrings](https://leetcode.com/problems/count-binary-substrings/)<sup>590</sup>

#### LeetCode - Lists

Check array problems<sup>591</sup> sorted by *Acceptance* and *Easy*. In particular:

- [Average Salary Excluding the Minimum and Maximum Salary](https://leetcode.com/problems/average-salary-excluding-the-minimum-and-maximum-salary/)<sup>592</sup>
- [Contains Duplicate](https://leetcode.com/problems/contains-duplicate/)<sup>593</sup>
- [Majority Element](https://leetcode.com/problems/majority-element/)<sup>594</sup>
- [Maximum Gap](https://leetcode.com/problems/maximum-gap/)<sup>595</sup>

---

<sup>579</sup> <https://leetcode.com>

<sup>580</sup> <https://leetcode.com/tag/string/>

<sup>581</sup> <https://leetcode.com/problems/shuffle-string/>

<sup>582</sup> <https://leetcode.com/problems/increasing-decreasing-string/>

<sup>583</sup> <https://leetcode.com/problems/detect-capital/>

<sup>584</sup> <https://leetcode.com/problems/unique-email-addresses/>

<sup>585</sup> <https://leetcode.com/problems/robot-return-to-origin/>

<sup>586</sup> <https://leetcode.com/problems/string-matching-in-an-array/>

<sup>587</sup> <https://leetcode.com/problems/reverse-words-in-a-string-iii/>

<sup>588</sup> <https://leetcode.com/problems/unique-morse-code-words/>

<sup>589</sup> <https://leetcode.com/problems/goat-latin/>

<sup>590</sup> <https://leetcode.com/problems/count-binary-substrings/>

<sup>591</sup> <https://leetcode.com/tag/array/>

<sup>592</sup> <https://leetcode.com/problems/average-salary-excluding-the-minimum-and-maximum-salary/>

<sup>593</sup> <https://leetcode.com/problems/contains-duplicate/>

<sup>594</sup> <https://leetcode.com/problems/majority-element/>

<sup>595</sup> <https://leetcode.com/problems/maximum-gap/>

- Can Make Arithmetic Progression From Sequence<sup>596</sup>
- Max consecutive ones<sup>597</sup>
- Missing number<sup>598</sup> - has many possible solutions
- Move Zeros<sup>599</sup>
- K Closest Points to Origin<sup>600</sup> (use lambda functions<sup>601</sup>)
- Rotated Digits<sup>602</sup>
- Filter Restaurants by Vegan-Friendly, Price and Distance<sup>603</sup> (to sort use lambda functions<sup>604</sup>)
- Largest Perimeter Triangle<sup>605</sup> hint: you don't actually need to try many combinations ...
- H-Index<sup>606</sup>
- Sort array by parity 1<sup>607</sup>
- Sort array by parity 2<sup>608</sup>
- Relative sort array<sup>609</sup>
- Insert Intervals<sup>610</sup> (use lambda functions<sup>611</sup>)
- Merge Intervals<sup>612</sup> (use lambda functions<sup>613</sup>)
- Sort colors<sup>614</sup>
- Find all numbers disappeared in an array<sup>615</sup>
- Degree of an array<sup>616</sup>
- The k Strongest Values in an Array<sup>617</sup> a bit convoluted but doable
- Array partition 1<sup>618</sup> actually a bit hard but makes you think
- Distant Barcodes<sup>619</sup>
- Reorganize String<sup>620</sup> think first when the task is *not* possible, for the rest is like previous one

<sup>596</sup> <https://leetcode.com/problems/can-make-arithmetic-progression-from-sequence/>

<sup>597</sup> <https://leetcode.com/problems/max-consecutive-ones/>

<sup>598</sup> <https://leetcode.com/problems/missing-number/>

<sup>599</sup> <https://leetcode.com/problems/move-zeroes/>

<sup>600</sup> <https://leetcode.com/problems/k-closest-points-to-origin/>

<sup>601</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>

<sup>602</sup> <https://leetcode.com/problems/rotated-digits/>

<sup>603</sup> <https://leetcode.com/problems/filter-restaurants-by-vegan-friendly-price-and-distance/>

<sup>604</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>

<sup>605</sup> <https://leetcode.com/problems/largest-perimeter-triangle/>

<sup>606</sup> <https://leetcode.com/problems/h-index/>

<sup>607</sup> <https://leetcode.com/problems/sort-array-by-parity/>

<sup>608</sup> <https://leetcode.com/problems/sort-array-by-parity-ii/>

<sup>609</sup> <https://leetcode.com/problems/relative-sort-array/>

<sup>610</sup> <https://leetcode.com/problems/insert-interval/>

<sup>611</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>

<sup>612</sup> <https://leetcode.com/problems/merge-intervals/>

<sup>613</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>

<sup>614</sup> <https://leetcode.com/problems/sort-colors/>

<sup>615</sup> <https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/>

<sup>616</sup> <https://leetcode.com/problems/degree-of-an-array/>

<sup>617</sup> <https://leetcode.com/problems/the-k-strongest-values-in-an-array/>

<sup>618</sup> <https://leetcode.com/problems/array-partition-i/>

<sup>619</sup> <https://leetcode.com/problems/distant-barcodes/>

<sup>620</sup> <https://leetcode.com/problems/reorganize-string/>

## LeetCode - Sets and Dictionaries

Check dictionary problems<sup>621</sup> sorted by *Acceptance* and *Easy*.

Note: Keep in mind these problems are in section *dictionaries* for good reason: in order to execute fast they often require you to preprocess the data by indexing in it in some way, like i.e. putting strings in a set or as keys in a dictonary so you can later look them up very fast.

**WARNING:** if you feel the need to use nested cycles, or search methods on lists/strings like `.index`, `.find`, `in` operator, `.count`, `.replace` on strings, try thinking first whether it is really necessary or you might use the above mentioned preprocessing instead.

Check in particular:

- Replace words<sup>622</sup>
- Word break<sup>623</sup>
- Fair candy swap<sup>624</sup>
- Verifying an alien dictionary<sup>625</sup> Note: you can use lambda functions<sup>626</sup>, but it is not strictly necessary
- Least Number of Unique Integers after K Removals<sup>627</sup>
- People Whose List of Favorite Companies Is Not a Subset of Another List<sup>628</sup>

## 11.3.6 LeetCode - Matrices

- Matrix Diagonal Sum<sup>629</sup>
- Cells with odd values in a matrix<sup>630</sup>
- Count negative numbers in Sorted matrix<sup>631</sup>
- Lucky Numbers in a Matrix<sup>632</sup>
- The k-weakest rows in a Matrix<sup>633</sup> (use lambda functions<sup>634</sup>)
- Matrix Cells in Distance Order<sup>635</sup>
- Toepliz Matrix<sup>636</sup>
- Special Positions in a Binary Matrix<sup>637</sup>
- Reshape the Matrix<sup>638</sup>

<sup>621</sup> <https://leetcode.com/problemset/all/?search=dictionaries>

<sup>622</sup> <https://leetcode.com/problems/replace-words/>

<sup>623</sup> <https://leetcode.com/problems/word-break/>

<sup>624</sup> <https://leetcode.com/problems/fair-candy-swap/>

<sup>625</sup> <https://leetcode.com/problems/verifying-an-alien-dictionary/>

<sup>626</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>

<sup>627</sup> <https://leetcode.com/problems/least-number-of-unique-integers-after-k-removals/>

<sup>628</sup> <https://leetcode.com/problems/people-whose-list-of-favorite-companies-is-not-a-subset-of-another-list/>

<sup>629</sup> <https://leetcode.com/problems/matrix-diagonal-sum/>

<sup>630</sup> <https://leetcode.com/problems/cells-with-odd-values-in-a-matrix/>

<sup>631</sup> <https://leetcode.com/problems/count-negative-numbers-in-a-sorted-matrix/>

<sup>632</sup> <https://leetcode.com/problems/lucky-numbers-in-a-matrix/>

<sup>633</sup> <https://leetcode.com/problems/the-k-weakest-rows-in-a-matrix>

<sup>634</sup> <https://docs.python.org/3/howto/sorting.html#key-functions>

<sup>635</sup> <https://leetcode.com/problems/matrix-cells-in-distance-order/>

<sup>636</sup> <https://leetcode.com/problems/toeplitz-matrix/>

<sup>637</sup> <https://leetcode.com/problems/special-positions-in-a-binary-matrix/>

<sup>638</sup> <https://leetcode.com/problems/reshape-the-matrix/>

- Kth Smallest Element in a Sorted Matrix<sup>639</sup> - there are many possible optimizations, you can make a first version using `sort` on everything, and then think about improving the algorithm
- Set Matrix Zeroes<sup>640</sup> interesting, try avoiding duplicating the matrix
- Search a 2D Matrix<sup>641</sup>
- Search a 2D Matrix ii<sup>642</sup>
- Spiral Matrix<sup>643</sup>
- Spiral Matrix ii<sup>644</sup>
- Matrix Block Sum<sup>645</sup>
- Sort the Matrix Diagonally<sup>646</sup> not fun, but doable

### Leet code - Graphs

Note: here on softpython we do not put links to exercises about visiting graphs, so for these you do not need stuff like breadth first search, depth first search, etc.

- Find the Town Judge<sup>647</sup>
- Maximal Network Rank<sup>648</sup>

### 11.3.7 HackerRank

Contains many Python 3 exercises on algorithms and data structures (Needs to login)

- [hackerrank.com](https://www.hackerrank.com)<sup>649</sup>

### 11.3.8 Geeks for Geeks

Contains many exercises - doesn't have solutions nor explicit asserts but if you login and submit solutions, the system will run some tests serverside and give you a response.

In general for Part A you can filter difficulty by school+basic+easy and if you need to do part B also include medium.

- Example: Filter difficulty by school+basic+easy and topic String<sup>650</sup>

You can select many more topics if you click `more>>` under Topic Tags:

<sup>639</sup> <https://leetcode.com/problems/kth-smallest-element-in-a-sorted-matrix/>

<sup>640</sup> <https://leetcode.com/problems/set-matrix-zeroes/>

<sup>641</sup> <https://leetcode.com/problems/search-a-2d-matrix/>

<sup>642</sup> <https://leetcode.com/problems/search-a-2d-matrix-ii/>

<sup>643</sup> <https://leetcode.com/problems/spiral-matrix/>

<sup>644</sup> <https://leetcode.com/problems/spiral-matrix-ii/>

<sup>645</sup> <https://leetcode.com/problems/matrix-block-sum/>

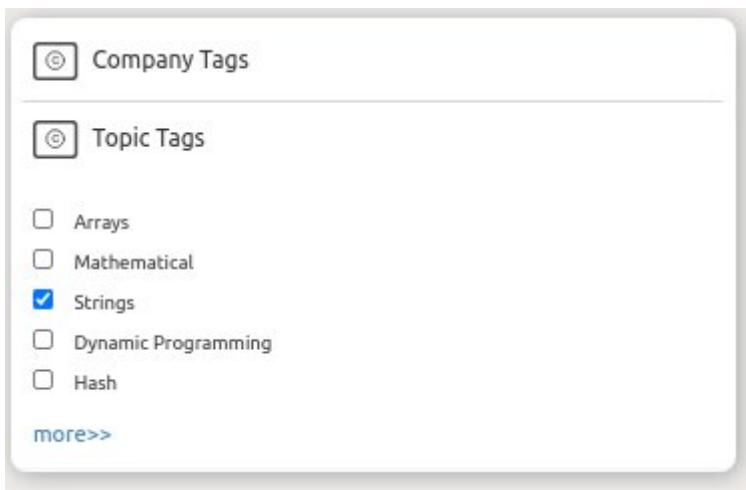
<sup>646</sup> <https://leetcode.com/problems/sort-the-matrix-diagonally/>

<sup>647</sup> <https://leetcode.com/problems/find-the-town-judge/>

<sup>648</sup> <https://leetcode.com/problems/maximal-network-rank>

<sup>649</sup> <https://www.hackerrank.com>

<sup>650</sup> <https://practice.geeksforgeeks.org/explore/?category%5B%5D=Strings&difficulty%5B%5D=-2&difficulty%5B%5D=-1&difficulty%5B%5D=0&page=1>



### 11.3.9 Dive into Python 3

More practical, contains more focused tutorials (i.e. manage XML files)

- [online version<sup>651</sup>](#)
- [printed<sup>652</sup>](#)
- [zip offline<sup>653</sup>](#)
- [PDF<sup>654</sup>](#)

Licence: Creative Commons By Share-alike 3.0<sup>655</sup> as reported at the bottom of book website<sup>656</sup>

### 11.3.10 Introduction to Scientific Programming with Python

Focuses on numerical calculations, you can check first 7 chapters until dictionaries.

By Joakim Sundnes.

- [PDF<sup>657</sup>](#) for Python (only theory)
- [Exercises<sup>658</sup>](#) – a LOT of stuff, although some exercises are too much into engineering / maths compared to this book
- EXTRA: if you like, it also contains chapters on classes which are certainly useful.

[ ] :

<sup>651</sup> <http://www.diveintopython3.net/>

<sup>652</sup> <http://www.amazon.com/gp/product/1430224150?ie=UTF8&tag=diveintomark-20&creativeASIN=1430224150>

<sup>653</sup> <https://github.com/diveintomark/diveintopython3/zipball/master>

<sup>654</sup> <https://github.com/downloads/diveintomark/diveintopython3/dive-into-python3.pdf>

<sup>655</sup> <http://creativecommons.org/licenses/by-sa/3.0/>

<sup>656</sup> <http://www.diveintopython3.net/>

<sup>657</sup> <https://link.springer.com/content/pdf/10.1007%2F978-3-030-50356-7.pdf>

<sup>658</sup> [https://www.uio.no/studier/emner/matnat/ifi/INF1100/h16/ressurser/INF1100\\_exercises\\_5th\\_ed.pdf](https://www.uio.no/studier/emner/matnat/ifi/INF1100/h16/ressurser/INF1100_exercises_5th_ed.pdf)