Sciprog DS Lab

Enter the Matrix

David Leoni

Sep 22, 2020

Copyright © 2020 by David Leoni.

Sciprog DS Lab 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 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://sciprog.davidleoni.it/

CONTENTS

About . . . 1
Preface . . . 1
Timetable and lecture rooms . . . 1
News . . . 2

1 Overview **3**
1.1 Slides . . . 3
1.2 Office hours . . . 3
1.3 Labs timetable . . . 3
1.4 Tutoring . . . 3
1.5 Exams . . . 4
1.6 Resources . . . 7
1.7 Acknowledgements . . . 10

2 Past Exams **11**
2.1 Data science . . . 11
2.2 2017-18 (QCB) . . . 276
2.3 2016-17 (QCB) . . . 276

3 Slides 2020/21 **277**

4 Commandments **279**

5 **283**

6 Part A **285**
6.1 Introduction solutions . . . 285
6.2 Python basics solutions . . . 301
6.3 Strings solutions . . . 311
6.4 Lists solutions . . . 330
6.5 Tuples solutions . . . 381
6.6 Sets solutions . . . 388
6.7 Dictionaries solutions . . . 393
6.8 Control flow solutions . . . 418
6.9 Functions - solutions . . . 436
6.10 Error handling and testing solutions . . . 456
6.11 Matrices: list of lists solutions . . . 482
6.12 Matrices: Numpy solutions . . . 522
6.13 Data formats solutions . . . 547
6.14 Graph formats solutions . . . 576
6.15 Visualization solutions . . . 625
6.16 Pandas solutions . . . 655

6.17 Binary relations solutions . . . 703

7 Part B **719**
7.1 OOP . . . 719
7.2 Sorting . . . 738
7.3 Linked lists . . . 756
7.4 Stacks . . . 773
7.5 Queues . . . 793
7.6 Trees . . . 821
7.7 Graph algorithms . . . 862
7.8 Changelog . . . 893

8 Index **895**

Data Science Master @University of Trento - AA 2020/21

About

Teaching assistant: David Leoni david.leoni@unitn.it website: davidleoni.it[1]

This work is licensed under a Creative Commons Attribution 4.0 License CC-BY[2]

Preface

> Once men turned their thinking over to machines in the hope that this would set them free. But that only permitted other men with machines to enslave them. — Frank Herbert, Dune, 1965

You can let your smart devices decide what you should see and think, or you can understand how they work.

Make your choice.

Timetable and lecture rooms

Due to the current situation regarding the Covid-19 pandemic, Practicals will take place ONLINE this year. They will be held on Mondays from 14:30 to 16:30 and on Wednesdays from 11:30 to 12:30.

Practicals will use the Zoom platform (https://zoom.us/) and the link for the connection will be published on the practical page available in this site a few minutes before the start of the session.

This first part of the course will tentatively run from Wednesday, September 23rd, 2020 to Monday, November 2nd, 2020.

.1 Moodle

In the Moodle page of the course[3] you can find announcements and videos of the lectures.

.2 Zoom links

The zoom links for the practicals can be found in the Announcements section of the moodle web page. To get you started quickly, I report them here:

CLICK HERE TO SEE ZOOM LINK

[1] http://www.davidleoni.it
[2] https://creativecommons.org/licenses/by/4.0/
[3] https://didatticaonline.unitn.it/dol/course/view.php?id=25445

News

22 September 2020:

> **WARNING: Part A of this website is being gradually improved and transfered to** en.softpython.org[4]
>
> **Keep an eye an news to see what has been transfered so far.**
>
> Whenever a page is moved, I will substitute the old page here with a link.

12 September 2020:

- Moved DS Lab website to sciprog.davidleoni.it[5]
- Simplified exercises structure

Old news

[4] https://en.softpython.org
[5] https://sciprog.davidleoni.it

CHAPTER

ONE

OVERVIEW

1.1 Slides

See *Slides page*

1.2 Office hours

To schedule a meeting, see here[6]

1.3 Labs timetable

For the regular labs timetable please see:

- Part A: Andrea Passerini's course site[7]
- Part B: Luca Bianco's course site[8]

1.4 Tutoring

A tutoring service for Scientific Programming will be set up

Please take advantage of it as much as possible so you don't end up writing random code at the exam!

TBD

[6] http://www.davidleoni.it/office-hours
[7] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/
[8] https://sciproalgo2020.readthedocs.io/

1.5 Exams

1.5.1 Schedule

Taking part to an exam erases *any* vote you had before (except for Midterm B which of course doesn't erase Midterm A taken in the same academic year)

Exams dates:

TBD

1.5.2 Past exams

- *Past exams page*

1.5.3 Exam modalities

Make practice with the lab computers !!

Exam will be in Linux Ubuntu environment (even when online) - so learn how to browse folders there and if in presence also typing with noisy lab keyboards :-)

Sciprog exams are open book. You will only be given access to this documentation (if in presence you can also bring a printed version of the material listed below):

- Sciprog lab website[9]
- Andrea Passerini slides[10] and Luca Bianco slides
- Python 3 documentation[11]
  - In particular, Unittest docs[12]
  - If you need to look up some Python function, please start today learning how to search documentation on Python website.
- Part A: Think Python[13] book
- Part B: Problem Solving with Algorithms and Data Structures using Python[14] book

1.5.4 Expectations

This is a data science master, so you must learn to be a proficient programmer - no matter the background you have.

Exercises proposed during labs are an example of what you will get during the exam, BUT **there is no way you can learn the required level of programming only doing exercises on this website**. Fortunately, since Python is so trendy nowadays there are a zillion good resources to hone your skills - you can find some in *Resources*

To successfully pass the exam, you should be able to quickly solve exercises proposed during labs with difficulty ranging from ⊛ to ⊛⊛⊛ stars. By quickly I mean in half on hour you should be able to solve a three star exercise ⊛⊛⊛. Typically,

[9] https://sciprog.davidleoni.it
[10] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/
[11] https://docs.python.org/3/
[12] https://docs.python.org/3/library/unittest.html
[13] https://runestone.academy/runestone/static/thinkcspy/index.html
[14] https://runestone.academy/runestone/static/pythonds/index.html

an exercise will be divided in two parts, the first easy ⊛⊛ to introduce you to the concept and the second more difficult ⊛⊛⊛ to see if you really grasped the idea.

Before getting scared, keep in mind I'm most interested in your capability to understand the problem and find your way to the solution. In real life, junior programmers are often given by senior colleagues functions to implement based on specifications and possibly tests to make sure what they are implementing meets the specifications. Also, programmers copy code all of the time. This is why during the exam I give you tests for the functions to implement so you can quickly spot errors, and also let you use the course material (see *exam modalities*).

Part A expectations: performance does *not* matters: if you are able to run the required algorithm on your computer and the tests pass, it should be fine. Just be careful when given a 100Mb file, in that case sometimes bad code may lead to very slow execution and/or clog the memory.

In particular, in lab computers the whole system can even hang, so watch out for errors such as:

- infinite `while` which keeps adding new elements to lists - whenever possible, prefer `for` loops
- scanning a big pandas dataframe using a `for in` instead of pandas native transformations

Part B expectations: performance *does* matters (i.e. finding the diagonal of a matrix should take a time linearly proportional to n, not n^2). Also, in this part we will deal with more complex data structures. Here we generally follow the *Do It Yourself* method, reimplementing things from scratch. So please, use the brain:

- if the exercise is about *sorting, do not* call Python `.sort()` method !!!
- if the exercise is about data structures, and you are thinking about converting the whole data structure (or part of it) into python lists, *first,* think about the computational cost of such conversion, and *second,* do ask the instructor for permission.

1.5.5 Grading

Correct implementations: Correct implementations with the required complexity grant you full grade.

Partial implementations: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.

When all tests pass hopefully should get full grade (although tests are never exhaustive!), but if the code is not correct you will still get a percentage. Percentage of course is *subjective,* and may depend on unfathomable factors such as the quantity of jam I found in the morning croissant that particular day. Jokes aside, the amount you get is usually proportional to the amount of time I have to spend to fix your algorithm.

After exams I publish the code with corrections. If all tests pass and you still don't get 100% grade, you may come to my office questioning the grade. If tests *don't* pass I'm less available for debating - I don't like much complaints like 'my colleague did the same error as me and got more points' - even worse is complaining without having read the corrections.

1.5.6 Exams FAQ

As part of the exam, there are some questions you need to know. Luckily, answers are pretty easy.

I did good part A/B, can I only do part B/A on next exam?

No way.

Can I have additional retake just for me?

No way.

Can I have additional oral to increase the grade?

No way.

I have $7 + \sqrt{3}$ INF credits from a Summer School in Applied Calculonics, can I please give only Part B?

I'm not into credits engineering, please ask the administrative office or/and Passerini.

I have another request which does not concern corrections / possibly wrong grading

I'm not the boss, ask Passerini.

I've got 26.99 but this is my last exam and I really need 27 so I can get good master final outcome, could you please raise grade of just that little 0.01?

Preposterous requests like this will be forwarded to our T-800 assistent, *it's very efficient* :

1.5.7 Exams How To

Make sure all exercises at least compile!

Don't forget duplicated code around!

If I see duplicated code, I don't know what to grade, I waste time, and you don't want me angry while grading.

Only implementations of provided function signatures will be evaluated !!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

1.5.8 How to edit and run

Look in *Applications->Programming:*

- Part A: **Jupyter**: open `Terminal` and type `jupyter notebook`
- Part B: open **Visual Studio Code**

If for whatever reason tests don't work in Visual Studio Code, be prepared to run them in the Terminal.

PAY close attention to function comments!

DON'T modify function signatures! Just provide the implementation

DON'T change existing test methods. If you want, you can add tests

DON'T create other files. If you still do it, they won't be evaluated

1.5.9 Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies

Even if print statements are allowed, be careful with prints that might break your function! For example, avoid stuff like this:

```
x = 0
print(1/x)
```

1.6 Resources

- Google colabs[15]: Scratchpads to show python code. During the lesson you can also write on them to share code.
- Source code[16] of these worksheets (download zip[17]), in Jupyter Notebook[18] format.

[15] https://drive.google.com/drive/folders/1aKaEPmqFF7-0EPRohwWKCmrz5FEwwEhq
[16] https://github.com/DavidLeoni/sciprog-ds
[17] https://github.com/DavidLeoni/sciprog-ds/archive/sciprog.davidleoni.it.zip
[18] http://jupyter.org

1.6.1 Part A Resources

Part A Theory slides[19] by Andrea Passerini

Allen Downey, Think Python

- Talks a lot, step by step, good for beginners
- online[20]
- printed[21]
- PDF[22]
- Interactive edition[23]

License: Creative Commons CC BY Non Commercial 3.0[24]as reported in the original page[25]

Tutorials from Nicola Cassetta

- Tutorial step by step, in Italian, good for beginners. They are well done and with solutions - please try them all.
- online[26]

Dive into Python 3

- More practical, contains more focused tutorials (i.e. manage XML files)
- online version[27]
- printed[28]
- zip offline[29]
- PDF[30]

Licence: Creative Commons By Share-alike 3.0[31] as reported at the bottom of book website[32]

LeetCode

Website with collections of exercises sorted by difficulty and acceptance rate. You can generally try sorting by *Acceptance* and *Easy* filters.

leetcode.com[33]

For a selection of exercises from leetcode, see *Further resources* sections at the ends of

- *List chapter*
- *Strings chapter*

[19] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/
[20] http://www.greenteapress.com/thinkpython/html/
[21] https://www.amazon.it/Think-Python-Like-Computer-Scientist/dp/1491939362/ref=sr_1_1?ie=UTF8&qid=1537163819&sr=8-1&keywords=think+python
[22] http://greenteapress.com/thinkpython2/thinkpython2.pdf
[23] https://runestone.academy/runestone/static/thinkcspy/index.html
[24] http://creativecommons.org/licenses/by-nc/3.0/deed.it
[25] http://greenteapress.com/wp/think-python-2e/
[26] http://ncassetta.altervista.org/Tutorial_Python/index.html
[27] http://www.diveintopython3.net/
[28] http://www.amazon.com/gp/product/1430224150?ie=UTF8&tag=diveintomark-20&creativeASIN=1430224150
[29] https://github.com/diveintomark/diveintopython3/zipball/master
[30] https://github.com/downloads/diveintomark/diveintopython3/dive-into-python3.pdf
[31] http://creativecommons.org/licenses/by-sa/3.0/
[32] http://www.diveintopython3.net/
[33] https://leetcode.com

HackerRank

Contains many Python 3 exercises on algorithms and data structures (Needs to login)

hackerrank.com[34]

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[35]

You can select many more topics if you click `more>>` un der Topic Tags:

Material from other courses of mine (in Italian)

- SoftPython - pulizia e analisi dati con Python[36] : contains many practical example of reading files and performing analysis on them.

1.6.2 Part B Resources

- Part B theory slides by Luca Bianco
- Problem Solving with Algorithms and Data Structures using Python[37] online book by Brad Miller and David Ranum
- Theory exercises (complexity, tree visits, graph visits) - by Alberto Montresor[38]
- LeetCode[39] (sort by easy difficulty)

[34] https://www.hackerrank.com
[35] https://practice.geeksforgeeks.org/explore/?category%5B%5D=Strings&difficulty%5B%5D=-2&difficulty%5B%5D=-1&difficulty%5B%5D=0&page=1
[36] http://it.softpython.org
[37] https://runestone.academy/runestone/static/pythonds/index.html
[38] https://drive.google.com/drive/folders/1RwjiSvIq60Z9mj_gCd5K2E6Bj9y1R0CL
[39] https://leetcode.com/problemset/all/

1.6.3 Editors

- Visual Studio Code[40]: the course official editor.
- Spyder[41]: Seems like a fine and simple editor
- PyCharme Community Edition[42]
- Jupyter Notebook[43]: Nice environment to execute Python commands and display results like graphs. Allows to include documentation in Markdown format
- JupyterLab[44] : next and much better version of Jupyter, although as of Sept 2018 is still in beta
- PythonTutor[45], a visual virtual machine (*very useful*! can also be found in examples inside the book!)

1.6.4 Further readings

- Rule based design[46] 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 (not mandatory read, it only gives context and practical applications for some of the material on graphs presented during the course)

1.7 Acknowledgements

- I wish to thank Dr. Luca Bianco for the introductory material on Visual Studio Code and Python
- This site was made with Jupyter using NBSphinx extension[47] and Jupman template[48]

[40] https://code.visualstudio.com/
[41] https://pythonhosted.org/spyder/
[42] https://www.jetbrains.com/pycharm/download/#section=linux
[43] http://jupyter.org
[44] https://github.com/jupyterlab/jupyterlab
[45] http://www.pythontutor.com/visualize.html#mode=edit
[46] https://www.researchgate.net/profile/Stef_Joosten/publication/327022933_Rule_Based_Design/links/5b7321be45851546c903234a/Rule-Based-Design.pdf
[47] http://nbsphinx.readthedocs.io/
[48] http://jupman.softpython.org/

CHAPTER

TWO

PAST EXAMS

2.1 Data science

NOTE: 19-20 exams are very similar to 18-19, the only difference being that **you might also get an exercise on Pandas.**

2.1.1 Midterm Simulation - Tue 13, November 2018 - solutions

Scientific Programming - Data Science Master @ University of Trento

Download exercises

Introduction

- **This simulation gives you NO credit whatsoever, it's just an example**. If you do everything wrong, you lose nothing. If you do everything correct, you gain nothing.

Allowed material

There won't be any internet access. You will only be able to access:

- DS Sciprog Lab worksheets
- Alberto Montresor slides
- Python 3 documentation (in particular, see unittest)
- The course book "Problem Solving with Algorithms and Data Structures using Python"

Grading FACSIMILE - IN THIS SIMULATION TIME YOU GET NO GRADE !!!!

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments
  - write pythonic code[49]

[49] http://docs.python-guide.org/writing/style

- avoid convoluted code like i.e.

```
if x > 5:
    return True
else:
    return False
```

when you could write just

```
return x > 5
```

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use Jupyter (start it from Terminal with `jupyter notebook`), if it doesn't work use an editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2018-11-13-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2018-11-13-FIRSTNAME-LASTNAME-ID
    A1.ipynb
    A2.ipynb
    B1.py
    B1_test.py
    B2.py
    B2_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2018-11-13-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2018-11-12-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

1. matrices

1.1 fill

Difficulty: ⊛⊛

[2]:
```
def fill(lst1, lst2):
    """ 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

    """
    #jupman-raise
    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
    #/jupman-raise

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)
```

(continues on next page)

(continued from previous page)

```
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']
```

1.2 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.

Now implement the algorithm:

[3]:
```
def lab(students, chairs):
    """

        INPUT:
        - students: a list of strings of length <= n*m
        - chairs:   an nxm 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

        Example:

        ss =  ['b', 'd', 'e', 'g', 'c', 'a', 'h', 'f' ]

        mat = [
                    [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

    """
    #jupman-raise

    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
```

(continues on next page)

(continued from previous page)

```
            i += 1
        else:
            j += 1
    #/jupman-raise


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]
              ]


m3 = [
        [None],
        [None],
     ]
lab(['a'], m3) # 1 student 2 chairs in one column
```

(continues on next page)

(continued from previous page)

```
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]
     ]
```

(continues on next page)

(continued from previous page)

```
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']
```

2. phones

A radio station used to gather calls by recording just the name of the caller and the phone number as seen on the phone display. For marketing purposes, the station owner wants now to better understand the places from where listeners where 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 where 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[50] and do read Falsehoods Programmers Believe About Phone Numbers[51]

2.1 canonical

⊛ We first want to canonicalize a phone number 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 !!!!!

[4]:
```
def canonical(phone):
    """ RETURN the canonical version of phone as a string. See above for an␣
↪explanation.
    """
```

(continues on next page)

[50] https://github.com/daviddrysdale/python-phonenumbers
[51] https://github.com/googlei18n/libphonenumber/blob/master/FALSEHOODS.md

(continued from previous page)

```
    #jupman-raise
    p = phone.replace(' ', '')
    if p.startswith('0039'):
        p = p[4:]
    if p.startswith('+39'):
        p = p[3:]
    return p
    #/jupman-raise

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'
```

2.2 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.

[5]:
```
province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']


def prefix(phone):
    """ RETURN the prefix of the phone as a string. Remeber 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 above province_
↪prefixes and mobile_prefixes lists.

        DO USE THE ALREADY DEFINED FUCTION canonical(phone)
    """
    #jupman-raise
    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'
    #/jupman-raise

assert prefix('0461123') == '0461'
assert prefix('+39 0461  4321') == '0461'
```

(continues on next page)

(continued from previous page)

```
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'
```

2.3 hist

Difficulty: ⊛⊛⊛

[6]:
```
province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']


def hist(phones):
    """ 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!

        DO USE THE ALREADY DEFINED FUCTIONS canonical(phone) AND prefix(phone)
    """
    #jupman-raise
    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
    #/jupman-raise

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}
```

2.4 display calls by prefixes

⊛⊛ 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[52]

You should obtain something like this:

[7]:
```
%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
```

[8]:
```
# SOLUTION

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

province_prefixes = ['0461', '02', '011']
province_names = ['Trento', 'Milano', 'Torino']
mobile_prefixes = ['330', '340', '328', '390', '3771']
```

(continues on next page)

[52] https://sciprog.davidleoni.it/visualization/visualization-sol.html

(continued from previous page)

```
phones = ['+39   02 423', '0461123', '02 426', '+39 0461  4321', '0039328 1234567',
↪'02 423', '02 424']

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])

plt.xlabel('prefixes')
plt.ylabel('calls')

plt.savefig('prefixes-count-solution.png')

plt.show()
```

2.1.2 Midterm - Fri 16 November 2018 - solutions

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solution

Introduction

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[53]
  - write pythonic code[54]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

Valid code

> **WARNING**: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

> **WARNING**: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

[53] https://sciprog.davidleoni.it/commandments.html
[54] http://docs.python-guide.org/writing/style

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use Jupyter (start it from Terminal with `jupyter notebook`), if it doesn't work use an editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2018-11-16-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2018-11-16-FIRSTNAME-LASTNAME-ID
    exam-2018-11-16.ipynb
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2018-11-16-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2018-11-16-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise.

4) When done:

if you have unitn login: zip and send to examina.icts.unitn.it[55]

If you don't have unitn login: tell instructors and we will download your work manually

A1 union

⊛⊛ When we talk about the *union* of two graphs, we intend the graph having union of verteces 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:

[2]:
```
ma =  [
            [True, False, False],
            [False, True, False],
            [True, False, False]
      ]
```

[55] http://examina.icts.unitn.it

[3]:
```
draw_mat(ma)
```

And another `mb` like this:

[4]:
```
mb =  [
            [True, True, False],
            [False, False, True],
            [False, True, False]

      ]
```

[5]:
```
draw_mat(mb)
```

The result of calling `union(ma, mb)` will be the following:

[19]:
```
res = [[True, True, False], [False, True, True], [True, True, False]]
```

which will be displayed as

[20]:
```
draw_mat(res)
```

So we get same verteces and edges from both `ma` and `mb`

[6]:
```
def union(mata, matb):
    """ Takes two graphs represented as nxn 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
    """
    #jupman-raise

    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
    #/jupman-raise

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"
```

(continues on next page)

(continued from previous page)

```
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]
```

(continues on next page)

(continued from previous page)

```
       ]

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]

                          ]
```

A2 surjective

⊛⊛ 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)

[7]:
```
G1 = [
        [True, True, False, False],
        [False, False,  False, True],
        [False, True, True, False],
        [False, True, True, True],

     ]
```

[8]:
```
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:

[9]:
```
G2 = [
        [True, True, False, False],
        [False, False,  False, True],
        [False, True, False, False],
        [False, True, False, False],

     ]
```

[10]:
```
draw_mat(G2)
```

[11]:
```
def surjective(mat):
    """ RETURN True if provided graph mat as list of boolean lists is an
        nxn surjective binary relation, otherwise return False
    """
    #jupman-raise
    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
    #/jupman-raise



m1 =  [
          [False]
     ]

assert surjective(m1) == False


m2 =  [
          [True]
     ]

assert surjective(m2) == True

m3 =  [
          [True, False],
          [False, False],
     ]

assert surjective(m3) == False
```

(continues on next page)

(continued from previous page)

```
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],
         [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],
```

(continues on next page)

(continued from previous page)

```
        [False, False,  False, True],
        [False, True, True, False],
        [False, True, True, True],

    ]
assert surjective(m11) == True
```

A3 ediff

⊛⊛⊛ 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 verteces but possibly different edges. This time we will try operate on graphs represented as dictionaries of adjacency lists.

For example, if we have

[12]:
```
da =  {
          'a':['a','c'],
          'b':['b', 'c'],
          'c':['b','c']
        }
```

[13]:
```
draw_adj(da)
```

and

[14]:
```
db =  {
          'a':['c'],
          'b':['a','b', 'c'],
          'c':['a']
        }
```

[15]:
```
draw_adj(db)
```

The result of calling `ediff(da,db)` will be:

[16]:
```
res = {
         'a':['a'],
         'b':[],
         'c':['b','c']
      }
```

Which can be shown as

[17]:
```
draw_adj(res)
```

[18]:
```
def ediff(da,db):
    """  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'],
```

(continues on next page)

(continued from previous page)

```
                        'b':['b', 'c'],
                        'c':['b','c']
                      }

            db =  {
                        'a':['c'],
                        'b':['a','b', 'c'],
                        'c':['a']
                      }

            assert ediff(da, db) == {
                                        'a':['a'],
                                        'b':[],
                                        'c':['b','c']
                                      }

    """
    #jupman-raise

    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
    #/jupman-raise



da1 =  {
          'a': []
       }
db1 =  {
          'a': []
       }


assert ediff(da1, db1) ==   {
                             'a': []
                           }

da2 =  {
          'a': []
       }

db2 =  {
          'a': ['a']
       }

assert ediff(da2, db2) == {
                            'a': []
                          }
```

(continues on next page)

(continued from previous page)

```
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':[]
        }

assert ediff(da6, db6) == {
                            'a':['b'],
                            'b':[]
                          }

da7 =  {
          'a':['a','b'],
          'b':[]
        }
db7 =  {
          'a':['a'],
          'b':[]
```

(continues on next page)

(continued from previous page)

```
        }

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']
                          }
```

2.1.3 Midterm - Thu 10, Jan 2019 - solutions

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solution

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[56]
  - write pythonic code[57]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

[56] https://sciprog.davidleoni.it/commandments.html
[57] http://docs.python-guide.org/writing/style

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-01-10-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-01-10-FIRSTNAME-LASTNAME-ID
    gaps.py
    gaps_test.py
    tasks.py
    tasks_test.py
    exits.py
    exits_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2019-01-10-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-01-10-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

if you have unitn login: zip and send to examina.icts.unitn.it[58]

If you don't have unitn login: tell instructors and we will download your work manually

Introduction

B1 Theory

Please write the solution in the text file `theory.txt`

Given the following function:

```
def fun(N, M):
    S1 = set(N)
    S2 = set(M)
    res = []
    for x in S1:
        if x in S2:
            for i in range(N.count(x)):
                res.append(x)
    return res
```

let `N` and `M` be two lists of length `n` and `m`, respectively. What is the computational complexity of function `fun()` with respect to `n` and `m`?

[58] http://examina.icts.unitn.it

B2 Gaps linked list

Given a linked list of size n which only contains integers, a gap is an **index** `i`, `0<i<n`, such that `L[i-1]<L[i]`. For the purpose of this exercise, we assume an empy list or a list with one element have zero gaps

Example:

```
 data:  9 7 6 8 9 2 2 5
index:  0 1 2 3 4 5 6 7
```

contains three gaps [3,4,7] because:

- number 8 at index 3 is greater than previous number 6 at index 2
- number 9 at index 4 is greater than previous number 8 at index 3
- number 5 at index 7 is greater than previous number 2 at index 6

Open file `gaps.py` and implement this method:

```
def gaps(self):
    """ Assuming all the data in the linked list is made by numbers,
        finds the gaps in the LinkedList and return them as a Python list.

        - we assume empty list and list of one element have zero gaps
        - MUST perform in O(n) where n is the length of the list

        NOTE: gaps to return are *indeces* , *not* data!!!!
    """
```

Testing: `python3 -m unittest gaps_test.GapsTest`

B3 Tasks stack

Very often, you begin to do a task just to discover it requires doing 3 other tasks, so you start carrying them out one at a time and discover one of them actually requires to do yet another two other subtasks….

To represent the fact a task may have subtasks, we will use a dictionary mapping a task label to a list of subtasks, each represented as a label. For example:

[2]:
```
subtasks = {
        'a':['b','g'],
        'b':['c','d','e'],
        'c':['f'],
        'd':['g'],
        'e':[],
        'f':[],
        'g':[]
    }
```

Task `a` requires subtasks `b` and`g` to be carried out (in this order), but task `b` requires subtasks `c`, `d` and `e` to be done. `c` requires `f` to be done, and `d` requires `g`.

You will have to implement a function called `do` and use a Stack data structure, which is already provided and you don't need to implement. Let's see an example of execution.

IMPORTANT: In the execution example, there are many prints just to help you understand what's going on, but the only thing we actually care about is the final list returned by the function!

IMPORTANT: notice subtasks are scheduled in reversed order, so the item on top of the stack will be the first to get executed !

[3]:
```
from tasks_sol import *

do('a', subtasks)
```

```
DEBUG:  Stack:   elements=['a']
DEBUG:  Doing task a, scheduling subtasks ['b', 'g']
DEBUG:          Stack:   elements=['g', 'b']
DEBUG:  Doing task b, scheduling subtasks ['c', 'd', 'e']
DEBUG:          Stack:   elements=['g', 'e', 'd', 'c']
DEBUG:  Doing task c, scheduling subtasks ['f']
DEBUG:          Stack:   elements=['g', 'e', 'd', 'f']
DEBUG:  Doing task f, scheduling subtasks []
DEBUG:          Nothing else to do!
DEBUG:          Stack:   elements=['g', 'e', 'd']
DEBUG:  Doing task d, scheduling subtasks ['g']
DEBUG:          Stack:   elements=['g', 'e', 'g']
DEBUG:  Doing task g, scheduling subtasks []
DEBUG:          Nothing else to do!
DEBUG:          Stack:   elements=['g', 'e']
DEBUG:  Doing task e, scheduling subtasks []
DEBUG:          Nothing else to do!
DEBUG:          Stack:   elements=['g']
DEBUG:  Doing task g, scheduling subtasks []
DEBUG:          Nothing else to do!
DEBUG:          Stack:   elements=[]
```

[3]:
```
['a', 'b', 'c', 'f', 'd', 'g', 'e', 'g']
```

The `Stack` you must use is simple and supports `push`, `pop`, and `is_empty` operations:

[4]:
```
s = Stack()
```

[5]:
```
print(s)
```

```
Stack:   elements=[]
```

[6]:
```
s.is_empty()
```

[6]:
```
True
```

[7]:
```
s.push('a')
```

[8]:
```
print(s)
```

```
Stack:   elements=['a']
```

[9]:
```
s.push('b')
```

[10]:
```
print(s)
```

```
Stack:   elements=['a', 'b']
```

[11]:
```
s.pop()
```

[11]:
```
'b'
```

[12]:
```
print(s)
```

```
Stack:   elements=['a']
```

B3.1 do

Now open `tasks_stack.py` and implement function `do`:

```
def do(task, subtasks):
    """ Takes a task to perform and a dictionary of subtasks,
        and RETURN a list of performed tasks

        - To implement it, inside create a Stack instance and a while cycle.
        - DO *NOT* use a recursive function
        - Inside the function, you can use a print like "I'm doing task a',
          but that is only to help yourself in debugging, only the
          list returned by the function will be considered in the evaluation!
    """
```

Testing: `python3 -m unittest tasks_test.DoTest`

B3.2 do_level

In this exercise, you are asked to implement a slightly more complex version of the previous function where on the `Stack` you push two-valued tuples, containing the task label and the associated level. The first task has level 0, the immediate subtask has level 1, the subtask of the subtask has level 2 and so on and so forth. In the list returned by the function, you will put such tuples.

One possibile use is to display the executed tasks as an indented tree, where the indentation is determined by the level. Here we see an example:

IMPORTANT: Again, the prints are only to let you understand what's going on, and you are *not* required to code them. The only thing that really matters is the list the function must return !

[13]:
```
subtasks = {
        'a':['b','g'],
        'b':['c','d','e'],
        'c':['f'],
        'd':['g'],
        'e':[],
        'f':[],
        'g':[]
    }

do_level('a', subtasks)
```

```
DEBUG:                                                  Stack:   elements=[('a', 0)]
DEBUG:  I'm doing       a               level=0 Stack:   elements=[('g', 1), ('b', 1)]
DEBUG:  I'm doing         b             level=1 Stack:   elements=[('g', 1), ('e', 2),
↪ ('d', 2), ('c', 2)]
DEBUG:  I'm doing           c           level=2 Stack:   elements=[('g', 1), ('e', 2),
↪ ('d', 2), ('f', 3)]
DEBUG:  I'm doing             f         level=3 Stack:   elements=[('g', 1), ('e', 2),
↪ ('d', 2)]
```

(continues on next page)

(continued from previous page)

```
DEBUG:  I'm doing          d          level=2 Stack:   elements=[('g', 1), ('e', 2),
↪ ('g', 3)]
DEBUG:  I'm doing            g        level=3 Stack:   elements=[('g', 1), ('e', 2)]
DEBUG:  I'm doing          e          level=2 Stack:   elements=[('g', 1)]
DEBUG:  I'm doing        g            level=1 Stack:   elements=[]
```

[13]:
```
[('a', 0),
 ('b', 1),
 ('c', 2),
 ('f', 3),
 ('d', 2),
 ('g', 3),
 ('e', 2),
 ('g', 1)]
```

Now implement the function:

```
def do_level(task, subtasks):
    """ Takes a task to perform and a dictionary of subtasks,
        and RETURN a list of performed tasks, as tuples (task label, level)

        - To implement it, use a Stack and a while cycle
        - DO *NOT* use a recursive function
        - Inside the function, you can use a print like "I'm doing task a',
          but that is only to help yourself in debugging, only the
          list returned by the function will be considered in the evaluation
    """
```

Testing: `python3 -m unittest tasks_test.DoLevelTest`

B4 Exits graph

There is a place nearby Trento called Silent Hill, where people always study and do little else. Unfortunately, one day an unethical biotech AI experiment goes wrong and a buggy cyborg is left free to roam in the building. To avoid panic, you are quickly asked to devise an evacuation plan. The place is a well known labyrinth, with endless corridors also looping into cycles. But you know you can model this network as a digraph, and decide to represent crossings as nodes. When a crossing has a door to leave the building, its label starts with letter `e`, while when there is no such door the label starts with letter `n`.

In the example below, there are three exits `e1`, `e2`, and `e3`. Given a node, say `n1`, you want to tell the crowd in that node the **shortest** paths leading to the three exits. To avoid congestion, one third of the crowd may be told to go to `e2`, one third to reach `e1` and the remaining third will go to `e3` even if they are farther than `e2`.

In python terms, we would like to obtain a dictionary of paths like the following, where as keys we have the exits and as values the shortest sequence of nodes from `n1` leading to that exit

```
{
    'e1': ['n1', 'n2', 'e1'],
    'e2': ['n1', 'e2'],
    'e3': ['n1', 'e2', 'n3', 'e3']
}
```

[14]:
```
from sciprog import draw_dig
from exits_sol import *
from exits_test import dig
```

[15]:
```
G = dig({'n1':['n2','e2'],
         'n2':['e1'],
         'e1':['n1'],
         'e2':['n2','n3', 'n4'],
         'n3':['e3'],
         'n4':['n1']})
draw_dig(G)
```

You will solve the exercise in steps, so open `exits_sol.py` and proceed reading the following points.

B4.1 cp

Implement this method

```
def cp(self, source):
    """ Performs a BFS search starting from provided node label source and
        RETURN a dictionary of nodes representing the visit tree in the
        child-to-parent format, that is, each key is a node label and as value
        has the node label from which it was discovered for the first time

        So if node "n2" was discovered for the first time while
        inspecting the neighbors of "n1", then in the output dictionary there
        will be the pair "n2":"n1".

        The source node will have None as parent, so if source is "n1" in the
```

(continues on next page)

(continued from previous page)

```
        output dictionary there will be the pair  "n1": None

        NOTE: This method must *NOT* distinguish between exits
              and normal nodes, in the tests we label them n1, e1 etc just
              because we will reuse in next exercise
        NOTE: You are allowed to put debug prints, but the only thing that
              matters for the evaluation and tests to pass is the returned
              dictionary
    """
```

Testing: `python3 -m unittest exits_test.CpTest`

Example:

[16]:
```
G.cp('n1')
```

```
DEBUG:  Removed from queue: n1
DEBUG:    Found neighbor: n2
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Found neighbor: e2
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['n2', 'e2']
DEBUG:  Removed from queue: n2
DEBUG:    Found neighbor: e1
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['e2', 'e1']
DEBUG:  Removed from queue: e2
DEBUG:    Found neighbor: n2
DEBUG:      already visited
DEBUG:    Found neighbor: n3
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Found neighbor: n4
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['e1', 'n3', 'n4']
DEBUG:  Removed from queue: e1
DEBUG:    Found neighbor: n1
DEBUG:      already visited
DEBUG:    Queue is: ['n3', 'n4']
DEBUG:  Removed from queue: n3
DEBUG:    Found neighbor: e3
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['n4', 'e3']
DEBUG:  Removed from queue: n4
DEBUG:    Found neighbor: n1
DEBUG:      already visited
DEBUG:    Queue is: ['e3']
DEBUG:  Removed from queue: e3
DEBUG:    Queue is: []
```

[16]:
```
{'n1': None,
 'n2': 'n1',
 'e2': 'n1',
 'e1': 'n2',
 'n3': 'e2',
 'n4': 'e2',
 'e3': 'n3'}
```

Basically, the dictionary above represents this visit tree:

```
   n1
  /   \
n2     e2
 \    /  \
 e1   n3  n4
      |
      e3
```

B4.2 exits

Implement this function. **NOTE**: the function is external to class DiGraph.

```
def exits(cp):
    """
        INPUT: a dictionary of nodes representing a visit tree in the
        child-to-parent format, that is, each key is a node label and as value
        has its parent as a node label. The root has associated None as parent.

        OUTPUT: a dictionary mapping node labels of exits to the shortest path
                from the root to the exit (root and exit included)

    """
```

Testing: `python3 -m unittest exits_test.ExitsTest`

Example:

[17]:
```
# as example we can use the same dictionary outputted by the cp call in the previous␣
↪exercise

visit_cp = { 'e1': 'n2',
             'e2': 'n1',
             'e3': 'n3',
             'n1': None,
             'n2': 'n1',
             'n3': 'e2',
             'n4': 'e2'
            }
exits(visit_cp)
```

[17]:
```
{'e1': ['n1', 'n2', 'e1'], 'e2': ['n1', 'e2'], 'e3': ['n1', 'e2', 'n3', 'e3']}
```

[ ]:
```

```

2.1.4 Exam - Wed 23, Jan 2019 - solutions

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solution

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[59]
  - write pythonic code[60]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

[59] https://sciprog.davidleoni.it/commandments.html
[60] http://docs.python-guide.org/writing/style

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-01-23-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-01-23-FIRSTNAME-LASTNAME-ID
    exam-2019-01-23.ipynb
    list.py
    list_test.py
    tree.py
    tree_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2019-01-23-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-01-23-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[61]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A

Open Jupyter and start editing this notebook `exam-2019-01-23.ipynb`

A.1 table_to_adj

Suppose you have a table expressed as a list of lists with headers like this:

[2]:
```
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:

[61] http://examina.icts.unitn.it/studente

[3]:
```
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)]
        }
```

[4]:
```
def table_to_adj(table):
    #jupman-raise
    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
    #/jupman-raise

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]
        ]
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')],
```

(continues on next page)

(continued from previous page)

```
            '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)
```

A.2 bus stops

Today we will analzye intercity bus network in GTFS format taken from dati.trentino.it[62], MITT service.

Original GTFS data was split in several files which we merged into dataset data/network.csv containing the bus stop times of three extra-urban routes. To load it, we provide this function:

[5]:
```
def load_stops():
    "Loads file network.csv and RETURN a list of dictionaries with the stop times"

    import csv
    with open('data/network.csv', newline='', encoding='UTF-8') as csvfile:
        reader = csv.DictReader(csvfile)
        lst = []
        for d in reader:
            lst.append(d)
    return lst
```

[6]:
```
stops = load_stops()

stops[0:2]
```

[6]:
```
[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', '000240274201809122019062 1'),
```

(continues on next page)

[62] https://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs

(continued from previous page)

```
             ('trip_headsign', 'Trento-Autostaz.'),
             ('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', '000240274201809122019062 1'),
             ('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

$velocity_n = \frac{\Delta space_n}{\Delta time_n}$

where

$\Delta time_n = time_n - time_{n-1}$ as the time **in hours** the bus takes between stop n and stop $n-1$.

and

$\Delta space_n = space_n - space_{n-1}$ is the distance the bus has moved between stop n and stop $n-1$.

We also set $velocity_0 = 0$

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:

[7]:
```
secs = 8*60*60+27*60+42
```

and to calculate the time in **float hours** you need to divide `secs` by `60*60=3600`:

[8]:
```
hours_float = secs / (60*60)
hours_float
```

[8]:
```
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 :

[9]:
```
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.

[10]:
```
def to_float_hour(time_string):
    """
        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)
    """
    #jupman-raise
    hours = int(time_string[0:2])
    mins = int(time_string[3:5])
    secs = int(time_string[6:])
    return (hours * 60 * 60 + mins * 60 + secs) / (60*60)
    #/jupman-raise
```

(continues on next page)

(continued from previous page)

```
def plot(route_short_name):
    """ Takes a route_short_name and *PLOTS* with matplotlib a graph of the velocity␣
↪of
        the 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 heigth
        - remember to set axis labels and title
    """
    #jupman-raise
    stops = load_stops()

    %matplotlib inline
    import matplotlib.pyplot as plt
    import numpy as np

    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
        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)
```

(continues on next page)

(continued from previous page)

```
    print('xs = %s' % xs)
    print('ys = %s' % ys)
    print('xticks = %s' % ticks)
    plt.savefig('img/%s.png' % route_short_name)
    plt.show()

    #/jupman-raise
```

```
plot('B202')
```

```
xs = [6.416666666666667, 6.433333333333334, 6.45, 6.466666666666667, 6.
↪516666666666667, 6.55, 6.566666666666666, 6.616666666666666, 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\
↪nAutostaz.\n06:41:00']
```

```
plot('B201')
```

```
xs = [18.25, 18.283333333333335, 18.333333333333332, 18.533333333333335, 18.75, 19.
↪166666666666668]
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']
```

(continues on next page)

(continued from previous page)

```
plot('B301')
```

```
xs = [17.583333333333332, 17.666666666666668, 17.733333333333334, 17.766666666666666,␣
↪17.8, 17.833333333333332, 17.883333333333333, 17.9, 17.916666666666668, 17.
↪933333333333334, 17.983333333333334, 18.0, 18.05, 18.066666666666666, 18.
↪083333333333332, 18.1, 18.133333333333333, 18.15, 18.166666666666668, 18.
↪183333333333334, 18.25, 18.266666666666666, 18.3, 18.316666666666666, 18.35, 18.
↪383333333333333, 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\nC.So\nTre\nNovembre\n17:40:00',
↪'Trento\nViale\nVerona\n17:44:00', 'Trento\nS.Bartolameo\n17:46:00', 'Trento\nViale\
↪nVerona\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\
↪nPizzeria\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\n49\n"Cimitero"\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\nS.Ilario/Via\nStroperi\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']
```

Part B

B.1 Theory

Let `L` a list of size `n`, and `i` and `j` two indeces. Return the computational complexity of function `fun()` with respect to n.

```
def fun(L,i,j):
    if i==j:
        return 0
    else:
        m = (i+j)//2
        count = 0
        for x in L[i:m]:
          for y in L[m:j+1]:
             if x==y:
                count = count+1
        left = fun(L,i,m)
        right = fun(L,m+1,j)
        return left+right+count
```

ANSWER: write solution here

$O(n^2)$

B.2 Linked List flatv

Suppose a `LinkedList` only contains integer numbers, say 3,8,8,7,5,8,6,3,9. Implement method `flatv` which scans the list: when it finds the *first* occurence of a node which contains a number which is less then the previous one, and the less than successive one, it inserts after the current one another node with the same data as the current one, and exits.

Example:

for Linked list 3,8,8,7,5,8,6,3,9

calling `flatv` should modify the linked list so that it becomes

Linked list 3,8,8,7,5,5,8,6,3,9

Note that it only modifies the first occurrence found 7,5,8 to 7,5,5,8 and the successive sequence 6,3,9 is not altered

Open `list.py` and implement this method:

```
def flatv(self):
```

B.3 Generic Tree rightmost

In the example above, the rightmost branch of `a` is given by the node sequence `a,d,n`

Open `tree.py` and implement this method:

```
def rightmost(self):
        """ RETURN a list containing the *data* of the nodes
            in the *rightmost* branch of the tree.

            Example:

            a
            ⍰b
```

(continues on next page)

(continued from previous page)

```
            ├c
            │├e
            ├d
             ├f
             ├g
              ├h
              ├i

            should give

            ['a','d','g','i']
        """
```

[ ]:

2.1.5 Exam - Wed 13, Feb 2019 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[63]
  - write pythonic code[64]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

[63] https://sciprog.davidleoni.it/commandments.html
[64] http://docs.python-guide.org/writing/style

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-02-13-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-02-13-FIRSTNAME-LASTNAME-ID
    exam-2019-02-13.ipynb
    company.py
    company_test.py
    tree.py
    tree_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2019-02-13-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-02-13-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[65]
- If you don't have unitn login: tell instructors and we will download your work manually

[65] http://examina.icts.unitn.it/studente

Part A - Bus network visualization

Open Jupyter and start editing this notebook `exam-2019-02-13.ipynb`

Today we will visualize intercity bus network in GTFS format taken from dati.trentino.it[66], MITT service. Original data was split in several files which we merged into dataset data/network-short.csv.

To visualize it, we will use networkx[67] library. Let's first see an example on how to do it:

[2]:
```
import networkx as nx
from sciprog 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)
```

[66] https://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs
[67] https://networkx.github.io/

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']

# 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)
```

```
{'color': '#2ca02c', 'penwidth': 4, 'weight': 5, 'label': '5', '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('data/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]:
```
[{'': '3',
  'agency_id': '12',
  'arrival_time': '06:27:00',
  'departure_time': '06:27:00',
  'direction_id': '0',
  'route_id': '76',
  'route_long_name': 'Trento-Sardagna-Candriai-Vaneze-Vason-Viote',
  'route_short_name': 'B202',
  'route_type': '3',
  'service_id': '22018091220190621',
  'stop_code': '2620VE',
  'stop_desc': '',
  'stop_id': '5025',
  'stop_lat': '46.073125',
  'stop_lon': '11.093579',
  'stop_name': 'Sardagna Civ.20',
  'stop_sequence': '4',
  'trip_headsign': 'Trento-Autostaz.',
  'trip_id': '000240274201809122019062 1',
  'zone_id': '2620.0'},
 {'': '4',
  'agency_id': '12',
  'arrival_time': '06:28:00',
  'departure_time': '06:28:00',
  'direction_id': '0',
  'route_id': '76',
  'route_long_name': 'Trento-Sardagna-Candriai-Vaneze-Vason-Viote',
  'route_short_name': 'B202',
  'route_type': '3',
  'service_id': '22018091220190621',
  'stop_code': '2620MS',
  'stop_desc': '',
  'stop_id': '843',
  'stop_lat': '46.069871',
  'stop_lon': '11.097749',
```

(continues on next page)

(continued from previous page)

```
  'stop_name': 'Sardagna-Maso Scala',
  'stop_sequence': '5',
  'trip_headsign': 'Trento-Autostaz.',
  'trip_id': '000240274201809122019062 1',
  'zone_id': '2620.0'}]
```

A1 extract_routes

Implement `extract_routes` function:

[8]:
```
import networkx as nx
from sciprog import draw_nx

stops = load_stops()

def extract_routes(stops):
    """ Extract all route_short_name from the stops list and RETURN
        an alphabetically sorted list of them, without duplicates
        (see example)

    """
    #jupman-raise
    s = set()
    for diz in stops:
        s.add(diz['route_short_name'])
    ret = list(s)
    ret.sort()
    return ret
    #/jupman-raise
```

Example:

[9]:
```
extract_routes(stops)
```

[9]:
```
['B201', 'B202', 'B211', 'B217', 'B301']
```

A2 to_int_min

Implement this function:

[10]:
```
def to_int_min(time_string):
    """
        Takes a time string in the format like 08:27:42
        and RETURN the time since midnight in minutes, ignoring
        the seconds (es 507)
    """
    #jupman-raise
    hours = int(time_string[0:2])
    mins = int(time_string[3:5])
    return (hours * 60 + mins)
    #/jupman-raise
```

Example:

[11]:
```
to_int_min('08:27:42')
```

[11]:
```
507
```

A3 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:

[12]:
```
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]
```

[13]:
```
get_color(4,5)
```

[13]:
```
'#9467bd'
```

Now implement this function:

[14]:
```
def get_legend_edges():
    """
        RETURN 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.
    """
    #jupman-raise
    legend_edges = []
    i = 0
    routes = extract_routes(stops)

    for route_short_name in routes:
        legend_edges.append({
```

(continues on next page)

(continued from previous page)

```
            'label': route_short_name,
            'color':get_color(i,len(routes))
        })
        i += 1
    return legend_edges
    #/jupman-raise
```

[15]:
```
get_legend_edges()
```

[15]:
```
[{'color': '#1f77b4', 'label': 'B201'},
 {'color': '#ff7f0e', 'label': 'B202'},
 {'color': '#2ca02c', 'label': 'B211'},
 {'color': '#d62728', 'label': 'B217'},
 {'color': '#9467bd', 'label': 'B301'}]
```

A4 calc_nx

Implement this function:

[16]:
```

def calc_nx(stops):
    """
        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

    """
    #jupman-raise

    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
    #/jupman-raise
```

[17]:
```
G = calc_nx(stops)

draw_nx(G, get_legend_edges())
```

A5 color_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'`:

```
[18]: G.node['1']
```

```
[18]: {'color': 'black', 'fontcolor': 'black', 'label': 'Trento\nAutostaz.'}
```

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* :

```
[19]: G.in_edges('1')
```

```
[19]: InEdgeDataView([('723', '1'), ('870', '1')])
```

If you get a View object, if needed you can easily transform to a list:

```
[20]: list(G.in_edges('1'))
```

```
[20]: [('723', '1'), ('870', '1')]
```

```
[21]: G.node['723']
```

```
[21]: {'color': 'black', 'fontcolor': 'black', 'label': 'Trento\nVia\nBrescia\n4'}
```

```
[22]: G.node['870']
```

```
[22]: {'color': 'black', 'fontcolor': 'black', 'label': 'Sarche\nCentro\nComm.'}
```

There is only an outgoing edge toward *Trento Corso 3 Novembre* :

```
[23]: G.out_edges('1')
```

[23]:
```
OutEdgeDataView([('1', '1108')])
```

[24]:
```
G.node['1108']
```

[24]:
```
{'color': 'black',
 'fontcolor': 'black',
 'label': 'Trento\nC.So\nTre\nNovembre'}
```

If, for example, we want to know the `route_id` of this outgoing edge, we can access it this way:

[25]:
```
G['1']['1108']
```

[25]:
```
{'color': '#9467bd',
 'label': '5',
 'penwidth': 4,
 'route_short_name': 'B301',
 'weight': 5}
```

If you want to change the color attribute of the node `'1'`, you can write like this:

[26]:
```
G.node['1']['color'] = 'red'
G.node['1']['fontcolor'] = 'red'
```

Now implement the function `color_hubs`:

[27]:
```
def color_hubs(G):
    """ Print 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

    """
    #jupman-raise

    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'
    #/jupman-raise
    draw_nx(G, legend_edges=get_legend_edges())
```

[28]:
```
color_hubs(G)
```

```
SOLUTION: The hubs are:

stop_id:757
Tione
Autostazione

stop_id:742
Ponte
Arche
Autost.

stop_id:1
Trento
Autostaz.
```

A6 plot_timings

To extract bus times from `G`, use this:

[29]:
```
G.edges()
```

[29]:
```
OutEdgeView([('842', '3974'), ('757', '746'), ('829', '3213'), ('1556', '4392'), (
↪'3974', '841'), ('4391', '4390'), ('857', '742'), ('4392', '4391'), ('5025', '843'),
↪ ('841', '881'), ('723', '1'), ('742', '870'), ('870', '1'), ('3213', '757'), ('1',
↪'1108'), ('843', '842'), ('746', '857'), ('1108', '1109'), ('881', '723'), ('4390',
↪'742')])
```

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:

[30]:
```
G.get_edge_data('1','1108')
```

[30]:
```
{'color': '#9467bd',
 'label': '5',
 'penwidth': 4,
 'route_short_name': 'B301',
 'weight': 5}
```

Now implement the function `plot_timings`:

[31]:
```
def plot_timings(G):
    """
        Given a networkx DiGraph G plots a frequency histogram of the
        time between bus stops.

    """
    #jupman-raise

    import numpy as np
    import matplotlib.pyplot as plt


    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()
    #/jupman-raise
```

[32]:
```
plot_timings(G)
```

Part B

B.1 Theory

Let `L` a list of size `n`, and `i` and `j` two indeces. Return the computational complexity of function `fun()` with respect to `n`.

Write the solution in separate ``theory.txt`` file

```
def fun(L, i, j):
    # j-i+1 is the number of elements
    # between index i and index j (both included)
    if j-i+1 <= 3:
        # Compute their minimum
        return min(L[i:j+1])
    else:
       onethird = (j-i+1)//3
       res1 = fun(L,i, i+onethird)
       res2 = fun(L,i+onethird+1, i+2*onethird)
       res3 = fun(L,i+2*onethird+1, j)
       return min(res1,res2,res3)
```

ANSWER: $\Theta(n)$

B2 Company queues

We can model a company as a list of many employees ordered by their rank, the highest ranking being the first in the list. We assume all employees have different rank. Each employee has a name, a rank, and a queue of tasks to perform (as a Python deque).

When a new employee arrives, it is inserted in the list in the right position according to his rank:

[33]:
```
from company_sol import *
```

(continues on next page)

(continued from previous page)

```
c = Company()
print(c)
```

```
Company:
  name   rank   tasks
```

[34]:
```
c.add_employee('x',9)
```

[35]:
```
print(c)
```

```
Company:
  name   rank   tasks
  x      9      deque([])
```

[36]:
```
c.add_employee('z',2)
```

[37]:
```
print(c)
```

```
Company:
  name   rank   tasks
  x      9      deque([])
  z      2      deque([])
```

[38]:
```
c.add_employee('y',6)
```

[39]:
```
print(c)
```

```
Company:
  name   rank   tasks
  x      9      deque([])
  y      6      deque([])
  z      2      deque([])
```

B2.1 add_employee

Implement this method:

```
def add_employee(self, name, rank):
    """
        Adds employee with name and rank to the company, maintaining
        the _employees list sorted by rank (higher rank comes first)

        Represent the employee as a dictionary with keys 'name', 'rank'
        and 'tasks' (a Python deque)
```

(continues on next page)

(continued from previous page)

```
        - here we don't mind about complexity, feel free to use a
          linear scan and .insert
        - If an employee of the same rank already exists, raise ValueError
        - if an employee of the same name already exists, raise ValueError
    """
```

Testing: `python3 -m unittest company_test.AddEmployeeTest`

B2.2 add_task

Each employee has a queue of tasks to perform. Tasks enter from the right and leave from the left. Each task has associated a required rank to perform it, but when it is assigned to an employee the required rank may exceed the employee rank or be far below the employee rank. Still, when the company receives the task, it is scheduled in the given employee queue, ignoring the task rank.

[40]:
```
c.add_task('a',3,'x')
```

[41]:
```
c
```

[41]:
```
Company:
  name   rank   tasks
  x      9      deque([('a', 3)])
  y      6      deque([])
  z      2      deque([])
```

[42]:
```
c.add_task('b',5,'x')
```

[43]:
```
c
```

[43]:
```
Company:
  name   rank   tasks
  x      9      deque([('a', 3), ('b', 5)])
  y      6      deque([])
  z      2      deque([])
```

[44]:
```
c.add_task('c',12,'x')
c.add_task('d',1,'x')
c.add_task('e',8,'y')
c.add_task('f',2,'y')
c.add_task('g',8,'y')
c.add_task('h',10,'z')
```

[45]:
```
c
```

[45]:
```
Company:
  name   rank   tasks
  x      9      deque([('a', 3), ('b', 5), ('c', 12), ('d', 1)])
  y      6      deque([('e', 8), ('f', 2), ('g', 8)])
  z      2      deque([('h', 10)])
```

Implement this function:

```
def add_task(self, task_name, task_rank, employee_name):
    """ Append the task as a (name, rank) tuple to the tasks of
        given employee

        - If employee does not exist, raise ValueError
    """
```

Testing: `python3 -m unittest company_test.AddTaskTest`

B2.2 work

Work in the company is produced in work steps. Each work step produces a list of all task names executed by the company in that work step.

A work step is done this way:

For each employee, starting from the highest ranking one, dequeue its current task (from the left), and than compare the task required rank with the employee rank according to these rules:

- When an employee discovers a task requires a rank strictly greater than his rank, he will append the task to his supervisor tasks. Note the highest ranking employee may be forced to do tasks that are greater than his rank.
- When an employee discovers he should do a task requiring a rank strictly less than his, he will try to see if the next lower ranking employee can do the task, and if so append the task to that employee tasks.
- When an employee cannot pass the task to the supervisor nor the next lower ranking employee, he will actually execute the task, adding it to the work step list

Example:

[46]:
```
c
```

[46]:
```
Company:
  name  rank  tasks
  x     9     deque([('a', 3), ('b', 5), ('c', 12), ('d', 1)])
  y     6     deque([('e', 8), ('f', 2), ('g', 8)])
  z     2     deque([('h', 10)])
```

[47]:
```
c.work()
```

```
DEBUG: Employee x gives task ('a', 3) to employee y
DEBUG: Employee y gives task ('e', 8) to employee x
DEBUG: Employee z gives task ('h', 10) to employee y
DEBUG: Total performed work this step: []
```

[47]:
```
[]
```

[48]:
```
c
```

[48]:
```
Company:
  name  rank  tasks
  x     9     deque([('b', 5), ('c', 12), ('d', 1), ('e', 8)])
  y     6     deque([('f', 2), ('g', 8), ('a', 3), ('h', 10)])
  z     2     deque([])
```

[49]:
```
c.work()
```

```
DEBUG: Employee x gives task ('b', 5) to employee y
DEBUG: Employee y gives task ('f', 2) to employee z
DEBUG: Employee z executes task ('f', 2)
DEBUG: Total performed work this step: ['f']
```

[49]:
```
['f']
```

[50]:
```
c
```

[50]:
```

Company:
  name  rank  tasks
  x     9     deque([('c', 12), ('d', 1), ('e', 8)])
  y     6     deque([('g', 8), ('a', 3), ('h', 10), ('b', 5)])
  z     2     deque([])
```

[51]:
```
c.work()
```

```
DEBUG: Employee x executes task ('c', 12)
DEBUG: Employee y gives task ('g', 8) to employee x
DEBUG: Total performed work this step: ['c']
```

[51]:
```
['c']
```

[52]:
```
c
```

[52]:
```

Company:
  name  rank  tasks
  x     9     deque([('d', 1), ('e', 8), ('g', 8)])
  y     6     deque([('a', 3), ('h', 10), ('b', 5)])
  z     2     deque([])
```

[53]:
```
c.work()
```

```
DEBUG: Employee x gives task ('d', 1) to employee y
DEBUG: Employee y executes task ('a', 3)
DEBUG: Total performed work this step: ['a']
```

[53]:
```
['a']
```

[54]:
```
c
```

[54]:
```

Company:
  name  rank  tasks
  x     9     deque([('e', 8), ('g', 8)])
  y     6     deque([('h', 10), ('b', 5), ('d', 1)])
  z     2     deque([])
```

[55]:
```
c.work()
```

```
DEBUG: Employee x executes task ('e', 8)
DEBUG: Employee y gives task ('h', 10) to employee x
DEBUG: Total performed work this step: ['e']
```

[55]:
```
['e']
```

[56]:
```
c
```

[56]:
```
Company:
  name  rank  tasks
  x     9     deque([('g', 8), ('h', 10)])
  y     6     deque([('b', 5), ('d', 1)])
  z     2     deque([])
```

[57]:
```
c.work()
```

```
DEBUG: Employee x executes task ('g', 8)
DEBUG: Employee y executes task ('b', 5)
DEBUG: Total performed work this step: ['g', 'b']
```

[57]:
```
['g', 'b']
```

[58]:
```
c
```

[58]:
```
Company:
  name  rank  tasks
  x     9     deque([('h', 10)])
  y     6     deque([('d', 1)])
  z     2     deque([])
```

[59]:
```
c.work()
```

```
DEBUG: Employee x executes task ('h', 10)
DEBUG: Employee y gives task ('d', 1) to employee z
DEBUG: Employee z executes task ('d', 1)
DEBUG: Total performed work this step: ['h', 'd']
```

[59]:
```
['h', 'd']
```

[60]:
```
c
```

[60]:
```
Company:
  name  rank  tasks
  x     9     deque([])
  y     6     deque([])
  z     2     deque([])
```

Now implement this method:

```
def work(self):
    """ Performs a work step and RETURN a list of performed task names.

        For each employee, dequeue its current task from the left and:
        - if the task rank is greater than the rank of the
          current employee, append the task to his supervisor queue
          (the highest ranking employee must execute the task)
        - if the task rank is lower or equal to the rank of the
          next lower ranking employee, append the task to that employee
          queue
        - otherwise, add the task name to the list of
          performed tasks to return
    """
```

Testing: `python3 -m unittest company_test.WorkTest`

B3 GenericTree

B3.1 fill_left

Open `tree.py` and implement `fill_left` method:

```
def fill_left(self, stuff):
    """ MODIFIES the tree by filling the leftmost branch data
        with values from provided array 'stuff'

        - if there aren't enough nodes to fill, raise ValueError
        - root data is not modified
        - *DO NOT* use recursion

    """
```

Testing: `python3 -m unittest tree_test.FillLeftTest`

Example:

[61]:
```
from tree_test import gt
from tree_sol import *
```

[62]:
```
t  = gt('a',
            gt('b',
                    gt('e',
                            gt('f'),
                            gt('g',
                                    gt('i')),
                    gt('h')),
            gt('c'),
            gt('d')))
```

[63]:
```
print(t)
```

```
a
└b
 ├e
 │├f
 │├g
 │└i
 └h
 ├c
 └d
```

[64]:
```
t.fill_left(['x','y'])
```

[65]:
```
print(t)
```

```
a
└x
 ├y
```

(continues on next page)

(continued from previous page)

```
 │├f
 │├g
 │└i
 │└h
 ├c
 └d
```

[66]:
```
t.fill_left(['W','V','T'])
print(t)
```

```
a
└W
 ├V
 │├T
 │├g
 │└i
 │└h
 ├c
 └d
```

B3.2 follow

Open `tree.py` and implement `follow` method:

```
def follow(self, positions):
        """
            RETURN an array of node data, representing a branch from the
            root down to a certain depth.
            The path to follow is determined by given positions, which
            is an array of integer indeces, see example.

            - if provided indeces lead to non-existing nodes, raise ValueError
            - IMPORTANT: *DO NOT* use recursion, use a couple of while instead.
            - IMPORTANT: *DO NOT* attempt to convert siblings to
                         a python list !!!! Doing so will give you less points!

        """
```

Testing: `python3 -m unittest tree_test.FollowTest`

Example:

```
              level  01234

                     a
                     ├b
                     ├c
                     │└e
                     │ ├f
                     │ ├g
                     │ │└i
                     │ └h
                     └d

                    RETURNS
```

(continues on next page)

(continued from previous page)

```
t.follow([])         [a]          root data is always present
t.follow([0])        [a,b]        b is the 0-th child of a
t.follow([2])        [a,d]        d is the 2-nd child of a
t.follow([1,0,2])    [a,c,e,h]    c is the 1-st child of a
                                  e is the 0-th child of c
                                  h is the 2-nd child of e
t.follow([1,0,1,0]) [a,c,e,g,i]  c is the 1-st child of a
                                  e is the 0-th child of c
                                  g is the 1-st child of e
                                  i is the 0-th child of g
```

[ ]:

```

```

2.1.6 Exam - Monday 10, June 2019 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[68]
  - write pythonic code[69]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

[68] https://sciprog.davidleoni.it/commandments.html
[69] http://docs.python-guide.org/writing/style

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-06-10-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-06-10-FIRSTNAME-LASTNAME-ID
    exam-2019-06-10.ipynb
    stack.py
    stack_test.py
    tree.py
    tree_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2019-06-10-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-06-10-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[70]
- If you don't have unitn login: tell instructors and we will download your work manually

[70] http://examina.icts.unitn.it/studente

Part A

Open Jupyter and start editing this notebook `exam-2019-06-10.ipynb`

A1 ITEA real estate

You will now analyze public real estates in Trentino, which are managed by ITEA agency. Every real estate has a type, and we will find the type distribution.

Data provider: ITEA - dati.trentino.it[71]

A function `load_itea` is given to load the dataset (you don't need to implement it):

[2]:
```
def load_itea():
    """Loads file data and RETURN a list of dictionaries with the stop times
    """

    import csv
    with open('data/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

[3]:
```
itea[:5]
```

[3]:
```
[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')]),
 OrderedDict([('Tipologia', 'ALLOGGIO'),
              ('Proprietà', 'ITEA'),
              ('Indirizzo', 'VIALE DELLE ROBINIE 26'),
              ('Frazione', ''),
```

(continues on next page)

[71] https://dati.trentino.it/dataset/patrimonio-immobiliare

(continued from previous page)

```
              ('Comune', 'TRENTO')]),
 OrderedDict([('Tipologia', 'ALLOGGIO'),
              ('Proprietà', 'ITEA'),
              ('Indirizzo', 'VIALE DELLE ROBINIE 26'),
              ('Frazione', ''),
              ('Comune', 'TRENTO')])]
```

A1.1 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. NOTE: Please don't use 5 ifs, try to come up with some generic code to catch all these cases ..)

[4]:
```
def calc_types_hist(db):
    #jupman-raise

    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
    #/jupman-raise

calc_types_hist(itea)
```

[4]:
```
{'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,
```

(continues on next page)

(continued from previous page)

```
 '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}
```

A1.2 calc_types_series

Takes a dictionary histogram and RETURN a list of tuples containing key/value pairs, sorted from most frequent iyems to least frequent.

HINT: if you don't remember how to sort by an element of a tuple, look at this example[72] and also in python documentation about sorting.

[5]:
```
def calc_types_series(hist):
    #jupman-raise
    ret = []

    for key in hist:
        ret.append((key, hist[key]))

    ret.sort(key=lambda c: c[1],reverse=True)
    return ret[:10]
    #/jupman-raise

tipologie = calc_types_series(calc_types_hist(itea))

tipologie
```

[5]:
```
[('ALLOGGIO', 10778),
 ('POSTO MACCHINA', 3147),
 ('GARAGE', 2221),
 ('MAGAZZINO', 143),
 ('NEGOZIO', 139),
 ('ALTRO', 64),
 ('SALA / ATTIVITA SOCIALI', 45),
 ('CABINA ELETTRICA', 41),
 ('CANTINA', 40),
 ('UFFICIO', 29)]
```

[72] https://sciprog.davidleoni.it/visualization/visualization-sol.html#indegree-per-node-sorted

A1.3 Real estates plot

Once you obtained the series as above, plot the first 10 most frequent items, in decreasing order.

- please 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.

[6]:
```
# write here
```

[7]:
```
# SOLUTION

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt


xs = np.arange(len(tipologie))

xs_labels = [t[0].replace('/', '/\n') for t in tipologie]

ys = [t[1] for t in tipologie]

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()
```

A2 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'`, `'Biossido 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: PAT Ag. Provinciale per la protezione dell'Ambiente - dati.trentino.it[73]

A function `load_air_quality` is given to load the dataset (you don't need to implement it):

[8]:
```
def load_air_quality():
    """Loads file data and RETURN a list of dictionaries with the stop times
    """

    import csv
    with open('data/air-quality.csv', 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()
```

IMPORTANT 1: 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

IMPORTANT 2: EVERY field is a STRING, including 'Valore' !

[9]:
```
air_quality[:5]
```

[9]:
```
[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')]),
```

(continues on next page)

[73] https://dati.trentino.it/dataset/qualita-dell-aria-rilevazioni-delle-stazioni-monitoraggio

(continued from previous page)

```
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')])]
```

Now implement the following function:

[10]:
```
def calc_avg_pollution(db):
    """ 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.

    """
    #jupman-raise
    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


    for t in ret:
        ret[t] /= counts[t]
    return ret
    #/jupman-raise

calc_avg_pollution(air_quality)
```

[10]:
```
{('Parco S. Chiara', 'PM10'): 11.385752688172044,
 ('Parco S. Chiara', 'PM2.5'): 7.9471544715447155,
 ('Parco S. Chiara', 'Biossido di Azoto'): 20.828146143437078,
 ('Parco S. Chiara', 'Ozono'): 66.69541778975741,
 ('Parco S. Chiara', 'Biossido Zolfo'): 1.2918918918918918,
 ('Via Bolzano', 'PM10'): 12.526881720430108,
 ('Via Bolzano', 'Biossido di Azoto'): 29.28493894165536,
 ('Via Bolzano', 'Ossido di Carbonio'): 0.5964769647696474,
 ('Piana Rotaliana', 'PM10'): 9.728744939271255,
 ('Piana Rotaliana', 'Biossido di Azoto'): 15.170068027210885,
 ('Piana Rotaliana', 'Ozono'): 67.03633916554509,
 ('Rovereto', 'PM10'): 9.475806451612904,
 ('Rovereto', 'PM2.5'): 7.764784946236559,
```

(continues on next page)

(continued from previous page)

```
 ('Rovereto', 'Biossido di Azoto'): 16.284167794316645,
 ('Rovereto', 'Ozono'): 70.54655870445345,
 ('Borgo Valsugana', 'PM10'): 11.819407008086253,
 ('Borgo Valsugana', 'PM2.5'): 7.413746630727763,
 ('Borgo Valsugana', 'Biossido di Azoto'): 15.73806275579809,
 ('Borgo Valsugana', 'Ozono'): 58.599730458221025,
 ('Riva del Garda', 'PM10'): 9.912398921832883,
 ('Riva del Garda', 'Biossido di Azoto'): 17.125845737483086,
 ('Riva del Garda', 'Ozono'): 68.38159675236807,
 ('A22 (Avio)', 'PM10'): 9.651821862348179,
 ('A22 (Avio)', 'Biossido di Azoto'): 33.0650406504065,
 ('A22 (Avio)', 'Ossido di Carbonio'): 0.4228848821081822,
 ('Monte Gaza', 'PM10'): 7.794520547945205,
 ('Monte Gaza', 'Biossido di Azoto'): 4.34412955465587,
 ('Monte Gaza', 'Ozono'): 99.0858310626703}
```

Part B

B1 Theory

Let `L` be a list containing `n` lists, each of them of size `m`. Return the computational complexity of function `fun()` with respect to `n` and `m`.

Write the solution in separate ``theory.txt`` file

```
def fun(L):
    for r1 in L:
        for r2 in L:
            if r1 != r2 and sum(r1) == sum(r2):
                print("Similar:")
                print(r1)
                print(r2)
```

ANSWER: $\Theta(m \cdot n^2)$

B2 WStack

Using a text editor, open file `stack.py`. You will find a `WStack` class skeleton which represents a simple stack that can only contain integers.

B2.1 implement class WStack

Fill in missing methods in class `WStack` in the order they are presented so to have a `.weight()` method that returns the total sum of integers in the stack in `O(1)` time.

Example:

[11]:
```
from stack_sol import *
```

[12]:
```
s = WStack()
```

[13]:
```
print(s)
```
```
WStack: weight=0 elements=[]
```

[14]:
```
s.push(7)
```

[15]:
```
print(s)
```
```
WStack: weight=7 elements=[7]
```

[16]:
```
s.push(4)
```

[17]:
```
print(s)
```
```
WStack: weight=11 elements=[7, 4]
```

[18]:
```
s.push(2)
```

[19]:
```
s.pop()
```
[19]:
```
2
```

[20]:
```
print(s)
```
```
WStack: weight=11 elements=[7, 4]
```

B2.2 accumulate

Implement function `accumulate`:

```
def accumulate(stack1, stack2, min_amount):
    """ Pushes on stack2 elements taken from stack1 until the weight of
        stack2 is equal or exceeds the given min_amount

        - if the given min_amount cannot possibly be reached because
          stack1 has not enough weight, raises early ValueError without
          changing stack1.
        - DO NOT access internal fields of stacks, only use class methods.
        - MUST perform in O(n) where n is the size of stack1
        - NOTE: this function is defined *outside* the class !
    """
```

Testing: `python -m unittest stacks_test.AccumulateTest`

Example:

[21]:
```
s1 = WStack()

print(s1)
```
```
WStack: weight=0 elements=[]
```

[22]:
```
s1.push(2)
s1.push(9)
s1.push(5)
s1.push(3)
```

[23]:
```
print(s1)
```

```
WStack: weight=19 elements=[2, 9, 5, 3]
```

[24]:
```
s2 = WStack()
print(s2)
```

```
WStack: weight=0 elements=[]
```

[25]:
```
s2.push(1)
s2.push(7)
s2.push(4)
```

[26]:
```
print(s2)
```

```
WStack: weight=12 elements=[1, 7, 4]
```

[27]:
```
# attempts to reach in s2 a weight of at least 17
```

[28]:
```
accumulate(s1,s2,17)
```

[29]:
```
print(s1)
```

```
WStack: weight=11 elements=[2, 9]
```

Two top elements were taken from s1 and now s2 has a weight of 20, which is >= 17

[30]:
```
print(s2)
```

```
WStack: weight=20 elements=[1, 7, 4, 3, 5]
```

B3 GenericTree

Open file `tree.py` in a text editor and read following instructions.

B3.1 is_triangle

A *triangle* is a node which has *exactly* two children.

Let's see some example:

```
      a
    /   \
   /     \
  b ----- c
 /|\     /
```

(continues on next page)

(continued from previous page)

```
d-e-f   g
       / \
      h---i
         /
        l
```

The tree above can also be represented like this:

```
a
├b
│ ├d
│ ├e
│ └f
└c
  └g
   ├h
   └i
    └l
```

- node `a` is a triangle because has *exactly* two children `b` and `c`, note it doesn't matter if `b` or `c` have children)
- `b` is *not* a triangle (has 3 children)
- `c` and `i` are *not* triangles (have only 1 child)
- `g` is a triangle as it has *exactly* two children `h` and `i`
- `d`, `e`, `f`, `h` and `l` are not triangles, because they have zero children

Now implement this method:

```
def is_triangle(self, elems):
    """ RETURN True if this node is a triangle matching the data
        given by list elems.

        In order to match:
        - first list item must be equal to this node data
        - second list item must be equal to this node first child data
        - third list item must be equal to this node second child data

        - if elems has less than three elements, raises ValueError
    """
```

Testing: `python -m unittest tree_test.IsTriangleTest`

Examples:

[31]:
```
from tree_test import gt
```

[32]:
```

# this is the tree from the example above

tb = gt('b', gt('d', gt('e'), gt('f')))
tg = gt('g', gt('h'), gt('i', gt('l')))
ta = gt('a', tb, gt('c', tg))

ta.is_triangle(['a','b','c'])
```

[32]:
```
True
```

[33]:
```
ta.is_triangle(['b','c','a'])
```

[33]:
```
False
```

[34]:
```
tb.is_triangle(['b','d','e'])
```

[34]:
```
False
```

[35]:
```
tg.is_triangle(['g','h','i'])
```

[35]:
```
True
```

[36]:
```
tg.is_triangle(['g','i','h'])
```

[36]:
```
False
```

B3.2 has_triangle

Implement this method:

```
def has_triangle(self, elems):
    """ RETURN True if this node *or one of its descendants* is a triangle
        matching given elems. Otherwise, return False.

        - a recursive solution is acceptable
    """
```

Testing: `python -m unittest tree_test.HasTriangleTest`

Examples:

[37]:
```
# example tree seen at the beginning

tb = gt('b', gt('d', gt('e'), gt('f')))
tg = gt('g', gt('h'), gt('i', gt('l')))
tc = gt('c', tg)
ta = gt('a', tb, tc)


ta.has_triangle(['a','b','c'])
```

[37]:
```
True
```

[38]:
```
ta.has_triangle(['a','c','b'])
```

[38]:
```
False
```

[39]:
```
ta.has_triangle(['b','c','a'])
```

[39]:
```
False
```

[40]:
```
tb.is_triangle(['b','d','e'])
```

[40]:
```
False
```

[41]:
```
tg.has_triangle(['g','h','i'])
```

[41]:
```
True
```

[42]:
```
tc.has_triangle(['g','h','i'])  # check recursion
```

[42]:
```
True
```

[43]:
```
ta.has_triangle(['g','h','i'])  # check recursion
```

[43]:
```
True
```

2.1.7 Exam - Tue 02, July 2019 - solutions

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solution

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[74]
  - write pythonic code[75]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

[74] https://sciprog.davidleoni.it/commandments.html
[75] http://docs.python-guide.org/writing/style

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-07-02-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-07-02-FIRSTNAME-LASTNAME-ID
    exam-2019-07-02.ipynb
    theory.txt
    linked_sort.py
    linked_sort_test.py
    stacktris.py
    stacktris_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2019-07-02-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-07-02-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[76]
- If you don't have unitn login: tell instructors and we will download your work manually

[76] http://examina.icts.unitn.it/studente

Part A

Open Jupyter and start editing this notebook `exam-2019-07-02.ipynb`

A1 Botteghe storiche

You will work on the dataset of _Botteghe storiche del Trentino" (small shops, workshops of Trentino)

Data provider: Provincia Autonoma di Trento - dati.trentino.it[77]

A function `load_botteghe` is given to load the dataset (you don't need to implement it):

[2]:
```
def load_botteghe():
    """Loads file data and RETURN a list of dictionaries with the botteghe dati
    """

    import csv
    with open('data/botteghe.csv', newline='',  encoding='utf-8',) as csvfile:
        reader = csv.DictReader(csvfile,  delimiter=',')
        lst = []
        for d in reader:
            lst.append(d)
    return lst


botteghe = load_botteghe()
```

> **IMPORTANT**: look at the dataset !
>
> Here we show only first 5 rows, but to get a clear picture of the dataset you should explore it further.

[3]:
```
botteghe[:5]
```

[3]:
```
[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.'),
              ('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'),
```

(continues on next page)

[77] https://dati.trentino.it/dataset/botteghe-storiche-del-trentino

(continued from previous page)

```
             ('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.

A1.1 rank_categories

[4]:
```
def rank_categories(db, stopwords):
    #jupman-raise
    ret = {}
    for diz in db:
        parole = diz['Insegna'].split(" ") + diz['Note'].upper().split(" ")
        for parola in parole:
            if parola in ret and not parola in stopwords:
                ret[parola] += 1
            else:
                ret[parola] = 1
    return sorted([(key, val) for key,val in ret.items() if val > 10], key=lambda c:␣
↪c[1], reverse=True)
    #/jupman-raise

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(botteghe, stopwords)

categories
```

[4]:
```
[('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)]
```

A1.2 plot

Now plot the 10 most frequent categories. Please pay attention to plot title, width and height, axis labels. Everything MUST display in a readable way.

[5]:
```
# write here
```

[6]:
```

# SOLUTION
```

(continues on next page)

(continued from previous page)

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

cats = categories[:10]

xs = np.arange(len(cats))

xs_labels = [t[0] for t in cats]

ys = [t[1] for t in cats]

fig = plt.figure(figsize=(15,5))

plt.bar(xs, ys, 0.5, align='center')

plt.title("Categorie botteghe storiche SOLUTION")
plt.xticks(xs, xs_labels)

plt.xlabel('name')
plt.ylabel('frequency')

plt.show()
```

A1.3 enrich

Once you found the categories, implement function `enrich`, which takes the db and previously computed categories, and RETURN a NEW DB where the dictionaries are enriched with a new field `Categorie`, which holds a list of the categories a particular *bottega* belongs to.

[7]:
```
def enrich(db, categories):
    #jupman-raise
    ret = []

    for diz in db:
```

(continues on next page)

(continued from previous page)

```
        new_diz = {key:val for key,val in diz.items()}
        new_diz['Categorie'] = []
        for cat in categories:
            if cat[0] in diz['Insegna'].upper() or cat[0] in diz['Note'].upper():
                new_diz['Categorie'].append(cat[0])
        ret.append(new_diz)
    return ret
    #/jupman-raise


new_db = enrich(botteghe, rank_categories(botteghe, stopwords))

new_db[:6]   #NOTE here we only show a sample
```

[7]:
```
[{'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']},
 {'Numero': '2',
  'Insegna': 'CONFEZIONI MONTIBELLER S.R.L.',
  'Indirizzo': 'Corso Ausugum',
  'Civico': '48',
  'Comune': 'Borgo Valsugana',
  'Cap': '38051',
  'Frazione/Località': '',
  'Note': 'esercizio commerciale',
  'Categorie': []},
 {'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': []},
 {'Numero': '4',
  'Insegna': 'BAR SERAFINI DI MINATI RENZO',
  'Indirizzo': '',
  'Civico': '24',
  'Comune': 'Grigno',
  'Cap': '38055',
  'Frazione/Località': 'Serafini',
  'Note': 'esercizio commerciale',
  'Categorie': ['BAR']},
 {'Numero': '6',
  'Insegna': 'SEMBENINI GINO & FIGLI S.R.L.',
  'Indirizzo': 'Via S. Francesco',
  'Civico': '35',
  'Comune': 'Riva del Garda',
  'Cap': '38066',
  'Frazione/Località': '',
```

(continues on next page)

(continued from previous page)

```
  'Note': '',
  'Categorie': []},
 {'Numero': '7',
  'Insegna': 'HOTEL RISTORANTE PIZZERIA “ALLA NAVE”',
  'Indirizzo': 'Via Nazionale',
  'Civico': '29',
  'Comune': 'Lavis',
  'Cap': '38015',
  'Frazione/Località': 'Nave San Felice',
  'Note': '',
  'Categorie': ['RISTORANTE', 'HOTEL', 'PIZZERIA']}]
```

A2 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 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`.

NOTE: the function does **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.

[8]:
```
def dump(mat, q):
    #jupman-raise
    rem = q
    ret = []

    for riga in mat:
        for j in range(len(riga)):
            cellfill = 7 - riga[j]
            unload = min(cellfill, rem)
            rem -= unload
```

(continues on next page)

(continued from previous page)

```
            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!")
    #/jupman-raise

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]


m6 = [          # tons to dump in each cell
    [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]
```

(continues on next page)

(continued from previous page)

```
try:
    dump ([[5]], 10)
    raise Exception("Should have failed !")
except ValueError:
    pass
```

Part B

B1 Theory

Write the solution in separate ``theory.txt`` file

Let `L1` and `L2` be two lists containing `n` lists, each of them of size `n`. Compute the computational complexity of function `fun()` with respect to `n`.

```
def fun(L1,L2):
    for r1 in L1:
        for val in r1:
            for r2 in L2:
                if val = sum(r2):
                    print(val)
```

ANSWER: $:nbsphinx-math:`Theta`(n^4) $

B2 Linked List sorting

Open a text editor and edit file `linked_sort.py`

B2.1 bubble_sort

You will implement bubble sort on a `LinkedList`.

```
def bubble_sort(self):
    """ Sorts in-place this linked list using the method of bubble sort

        - MUST execute in O(n^2) where n is the length of the linked list
    """
```

Testing: `python3 -m unittest linked_sort_test.BubbleSortTest`

As a reference, you can look at this `example_bubble` implementation below that operates on regular python lists. Basically, you will have to translate the `for` cycles into two suitable `while` and use node pointers.

NOTE: this version of the algorithm is inefficient as we do not use `j` in the inner loop: your linked list implementation can have this inefficiency as well.

[9]:
```
def example_bubble(plist):
    for j in range(len(plist)):
        for i in range(len(plist)):
            if i + 1 < len(plist) and plist[i]>plist[i+1]:
                temp = plist[i]
                plist[i] = plist[i+1]
```

(continues on next page)

(continued from previous page)

```
                plist[i+1] = temp

my_list = [23, 34, 55, 32, 7777, 98, 3, 2, 1]
example_bubble(my_list)
print(my_list)
```

```
[1, 2, 3, 23, 32, 34, 55, 98, 7777]
```

B2.2 merge

Implement this method:

```
def merge(self,l2):
    """ Assumes this linkedlist and l2 linkedlist contain integer numbers
        sorted in ASCENDING order, and  RETURN a NEW LinkedList with
        all the numbers from this and l2 sorted in DESCENDING order

        IMPORTANT 1: *MUST* EXECUTE IN O(n1+n2) TIME where n1 and n2 are
                     the sizes of this and l2 linked_list, respectively

        IMPORTANT 2: *DO NOT* attempt to convert linked lists to
                     python lists!
    """
```

Testing: `python3 -m unittest linked_sort_test.MergeTest`

B3 Stacktris

Open a text editor and edit file `stacktris.py`

A `Stacktris` is a data structure that operates like the famous game Tetris, with some restrictions:

- Falling pieces can be either of length 1 or 2. We call them `1-block` and `2-block` respectively
- The pit has a fixed width of 3 columns
- `2-block`s can only be in horizontal

We print a `Stacktris` like this:

```
\ j 012
i
4  | 11|    # two 1-block
3  | 22|    # one 2-block
2  | 1 |    # one 1-block
1  |22 |    # one 2-block
0  |1 1|    # on the ground there are two 1-block
```

In Python, we model the `Stacktris` as a class holding in the variable `_stack` a list of lists of integers, which models the pit:

```
class Stacktris:

    def __init__(self):
        """ Creates a Stacktris
```

(continues on next page)

(continued from previous page)

```
        """
        self._stack = []
```

So in the situation above the `_stack` variable would look like this (notice row order is inverted with respect to the print)

```
[
    [1,0,1],
    [2,2,0],
    [0,1,0],
    [0,2,2],
    [0,1,1],
]
```

The class has three methods of interest which you will implement, `drop1(j)` , `drop2h(j)` and `_shorten`

Example

Let's see an example:

[10]:
```
from stacktris_sol import *

st = Stacktris()
```

At the beginning the pit is empty:

[11]:
```
st
```

[11]:
```
Stacktris:
EMPTY
```

We can start by dropping from the ceiling a block of dimension 1 into the last column at index `j=2`. By doing so, a new row will be created, and will be a list containing the numbers `[0,0,1]`

IMPORTANT: zeroes are not displayed

[12]:
```
st.drop1(2)
```

```
DEBUG:  Stacktris:
        |  1|
```

[12]:
```
[]
```

Now we drop an horizontal block of dimension 2 (a `2-block`) having the leftmost block at column `j=1`. Since below in the pit there is already the `1` block we previosly put, the new block will fall and stay upon it. Internally, we will add a new row as a python list containing the numbers `[0,2,2]`

[13]:
```
st.drop2h(1)
```

```
DEBUG:  Stacktris:
        | 22|
        |  1|
```

[13]:
```
[]
```

We see the zeroth column is empty, so if we drop there a `1-block` it will fall to the ground. Internally, the zeroth list will become `[1,0,1]`:

[14]:
```
st.drop1(0)
```

```
DEBUG:  Stacktris:
        | 22|
        |1 1|
```

[14]:
```
[]
```

Now we drop again a `2-block` at column `j=2`, on top of the previously laid one. This will add a new row as list `[0,2,2]`.

[15]:
```
st.drop2h(1)
```

```
DEBUG:  Stacktris:
        | 22|
        | 22|
        |1 1|
```

[15]:
```
[]
```

In the game Tetris, when a row becomes completely filled it disappears. So if we drop a `1-block` to the leftmost column, the mid line should be removed.

NOTE: The messages on the console are just debug print, the function `drop1` only returns the extracted line `[1,2,2]`:

[16]:
```
st.drop1(0)
```

```
DEBUG:  Stacktris:
        | 22|
        |122|
        |1 1|

DEBUG:  POPPING [1, 2, 2]
DEBUG:  Stacktris:
        | 22|
        |1 1|
```

[16]:
```
[1, 2, 2]
```

Now we insert another `2-block` starting at `j=0`. It will fall upon the previously laid one:

[17]:
```
st.drop2h(0)
```

```
DEBUG:  Stacktris:
        |22 |
        | 22|
        |1 1|
```

[17]:
```
[]
```

We can complete teh topmost row by dropping a `1-block` to the rightmost column. As a result, the row will be removed from the stack and the row will be returned by the call to `drop1`:

[18]:
```
st.drop1(2)
```

```
DEBUG:  Stacktris:
        |221|
```

(continues on next page)

(continued from previous page)

```
        | 22|
        |1 1|

DEBUG:  POPPING [2, 2, 1]
DEBUG:  Stacktris:
        | 22|
        |1 1|
```

[18]:
```
[2, 2, 1]
```

Another line completion with a `drop1` at column `j=0`:

[19]:
```
st.drop1(0)
```

```
DEBUG:  Stacktris:
        |122|
        |1 1|

DEBUG:  POPPING [1, 2, 2]
DEBUG:  Stacktris:
        |1 1|
```

[19]:
```
[1, 2, 2]
```

We can finally empty the Stacktris by dropping a `1-block` in the mod column:

[20]:
```
st.drop1(1)
```

```
DEBUG:  Stacktris:
        |111|

DEBUG:  POPPING [1, 1, 1]
DEBUG:  Stacktris:
        EMPTY
```

[20]:
```
[1, 1, 1]
```

B3.1 _shorten

Start by implementing this private method:

```
def _shorten(self):
    """ Scans the Stacktris from top to bottom searching for a completely filled line:
        - if found, remove it from the Stacktris and return it as a list.
        - if not found, return an empty list.
    """
```

If you wish, you can add debug prints but they are not mandatory

Testing: `python3 -m unittest stacktris_test.ShortenTest`

B3.2 drop1

Once you are done with the previous function, implement `drop1` method:

NOTE: In the implementation, feel free to call the previously implemented `_shorten` method.

```
def drop1(self, j):
    """ Drops a 1-block on column j.

         - If another block is found,  place the 1-block on top of that block,
           otherwise place it on the ground.

        - If, after the 1-block is placed, a row results completely filled, removes
          the row and RETURN it. Otherwise, RETURN an empty list.

        - if index `j` is outside bounds, raises ValueError
    """
```

Testing: `python3 -m unittest stacktris_test.Drop1Test`

B3.3 drop2h

Once you are done with the previous function, implement `drop2` method:

```
def drop2h(self, j):
    """ Drops a 2-block horizontally with left block on column j,

         - If another block is found,  place the 2-block on top of that block,
           otherwise place it on the ground.

        - If, after the 2-block is placed, a row results completely filled,
          removes the row and RETURN it. Otherwise, RETURN an empty list.

        - if index `j` is outside bounds, raises ValueError
    """
```

Testing: `python3 -m unittest stacktris_test.Drop2hTest`

```
[ ]:
```

2.1.8 Exam - Mon 26, August 2019 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[78]
  - write pythonic code[79]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

[78] https://sciprog.davidleoni.it/commandments.html
[79] http://docs.python-guide.org/writing/style

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-08-26-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-08-26-FIRSTNAME-LASTNAME-ID
    exam-2019-08-26.ipynb
    theory.txt
    backpack.py
    backpack_test.py
    concert.py
    concert_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2019-08-26-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-08-26-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[80]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - University of Trento staff

Open Jupyter and start editing this notebook `exam-2019-08-26.ipynb`

You will work on the dataset of *University of Trento staff*, modified so not to contain names or surnames.

Data provider: University of Trento[81]

A function `load_data` is given to load the dataset (you don't need to implement it):

[80] http://examina.icts.unitn.it/studente
[81] https://dati.trentino.it/dataset/personale-accademico-e-tecnico-amministrativo-dell-universita-di-trento

[1]:
```
import json

def load_data():
    with open('data/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",
```

(continues on next page)

(continued from previous page)

```
        "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'
```

A1 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',
 .
 .
}
```

```
[5]: def calc_uid_to_abbr(db):
    #jupman-raise
    ret = {}
    for person in db:
        for position in person['positions']:
            uid = position['unitIdentifier']
            ret[uid] = abbreviate(position['unitName'])
    return ret
    #/jupman-raise

#calc_uid_to_abbr(unitn)
print(calc_uid_to_abbr(unitn)['STO0008629'])
print(calc_uid_to_abbr(unitn)['STO0012897'])
```

```
DISI
Institutional Relations and Strategic Documents
```

A2.1 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.

[6]:
```
def calc_prof_roles(db):
    #jupman-raise
    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]
    #/jupman-raise

#calc_prof_roles(unitn)
```

[7]:
```
# EXPECTED RESULT
calc_prof_roles(unitn)
```

[7]:
```
[('Humanities', 92),
 ('DICAM', 85),
 ('Law', 84),
 ('Economics', 83),
 ('Sociology', 66),
 ('COGSCI', 61),
 ('Physics', 60),
 ('DISI', 55),
 ('DII', 49),
 ('Mathematics', 47)]
```

A2.2 plot_profs

⊛ Write a funciton to plot a bar chart of data calculated above

[8]:
```
%matplotlib inline
import matplotlib.pyplot as plt


def plot_profs(db):
    #jupman-raise


    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()
    #/jupman-raise

#plot_profs(unitn)
```

[9]:
```
# EXPECTED RESULT
plot_profs(unitn)
```

A3.1 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.

[10]:
```
def calc_roles(db):
    #jupman-raise
    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
    #/jupman-raise

#print(calc_roles(unitn)['STO0000001'])
#print(calc_roles(unitn)['STO0000006'])
#print(calc_roles(unitn)['STO0000012'])
#print(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}
 .
 .
 .
}
```

A3.2 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[82]
- For on example on plotting side by side, see this[83]

[11]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_roles(db, abbrs):
    #jupman-raise
    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,             # rows
                        len(abbrs),    # columns
                        i+1)           # plotting in first cell
            plt.pie(fracs, labels=labels, autopct='%1.1f%%', shadow=True)
            plt.title(abbr )
    #/jupman-raise


#plot_roles(unitn, ['DISI','Sociology', 'COGSCI'])
```

[12]:
```
# EXPECTED RESULT
plot_roles(unitn, ['DISI','Sociology', 'COGSCI'])
```

[82] https://sciprog.davidleoni.it/visualization/visualization-sol.html#Pie-chart
[83] https://sciprog.davidleoni.it/visualization/visualization-sol.html#Showing-plots-side-by-side

A4.1 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

- we do not want to consider Doctoral programmes
- we do not want to consider 'University of Trento' department (STO0000001)
- if your calculations display with swapped names (('COGSCI', 'CIMeC', 23) instead of ('CIMeC', 'COGSCI', 23)) it is not important, as long as they display just once per pair.

To implement this, we provide a 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 possibile 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
         DISI,  COGSCI,  CIMeC,  DICAM
0 DISI    --      --      --      --
1 COGSCI  313     --      --      --
2 CIMeC   231     23      --      --
3 DICAM   12      13      123     --
```

[13]:
```
def calc_shared(db):
    #jupman-raise
    ret = {}
    uid_to_people = {}

    uid_to_abbr = calc_uid_to_abbr(db)
```

(continues on next page)

(continued from previous page)

```
    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 != 'STO0000001') \
              and (uidy != 'STO0000001'):
                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
    #/jupman-raise

#calc_shared(unitn)
```

[14]:
```
# EXPECTED RESULT
calc_shared(unitn)
```

[14]:
```
[('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)]
```

A4.2 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.

[15]:
```
import matplotlib.pyplot as plt

%matplotlib inline

def plot_shared(db):
```

(continues on next page)

(continued from previous page)

```
    #jupman-raise

    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()
    #/jupman-raise

#plot_shared(unitn)
```

[16]:
```
# EXPECTED RESULT

plot_shared(unitn)
```

Part B

B1 Theory

Write the solution in separate ``theory.txt`` file

Let `M` be a square matrix - a list containing n lists, each of them of size `n`. Return the computational complexity of function `fun()` with respect to `n`:

```
def fun(M):
    for row in M:
        for element in row:
            print(sum([x for x in row if x != element]))
```

ANSWER: $O(n^3)$

B2 Backpack

Open a text editor and edit file `backpack_sol.py`

We can model a backpack as stack of elements, each being a tuple with a name and a weight.

A sensible strategy to fill a backpack is to place heaviest elements to the bottom, so our backback will allow pushing an element only if that element weight is equal or lesser than current topmost element weight.

The backpack has also a maximum weight: you can put any number of items you want, as long as its maximum weight is not exceeded.

Example

[17]:
```
from backpack_sol import *

bp = Backpack(30)  # max_weight = 30

bp.push('a',10)   # item 'a' with weight 10
```

```
DEBUG:  Pushing (a,10)
```

[18]:
```
print(bp)
```

```
Backpack: weight=10 max_weight=30
          elements=[('a', 10)]
```

[19]:
```
bp.push('b',8)
```

```
DEBUG:  Pushing (b,8)
```

[20]:
```
print(bp)
```

```
Backpack: weight=18 max_weight=30
          elements=[('a', 10), ('b', 8)]
```

```
>>> bp.push('c', 11)

DEBUG:  Pushing (c,11)

ValueError: ('Pushing weight greater than top element weight! %s > %s', (11, 8))
```

[21]:
```
bp.push('c', 7)
```

```
DEBUG:  Pushing (c,7)
```

[22]:
```
print(bp)
```

```
Backpack: weight=25 max_weight=30
          elements=[('a', 10), ('b', 8), ('c', 7)]
```

```
>>> bp.push('d', 6)

DEBUG:  Pushing (d,6)

ValueError: Can't exceed max_weight ! (31 > 30)
```

B2.1 class

⊛⊛ Implement methods in the class `Backpack`, in the order they are shown. If you want, you can add debug prints by calling the `debug` function

IMPORTANT: the data structure should provide the total current weight in O(1), so make sure to add and update an appropriate field to meet this constraint.

Testing: `python3 -m unittest backpack_test.BackpackTest`

B2.2 remove

⊛⊛ Implement function `remove`:

```
# NOTE: this function is implemented *outside* the class !

def remove(backpack, el):
    """
        Remove topmost occurrence of el found in the backpack,
        and RETURN it (as a tuple name, weight)

        - if el is not found, raises ValueError

        - DO *NOT* ACCESS DIRECTLY FIELDS OF BACKPACK !!!
          Instead, just call methods of the class!

        - MUST perform in O(n), where n is the backpack size

        - HINT: To remove el, you need to call Backpack.pop() until
                the top element is what you are looking for. You need
                to save somewhere the popped items except the one to
                remove, and  then push them back again.

    """
```

Testing: `python3 -m unittest backpack_test.RemoveTest`

Example:

[23]:
```
bp = Backpack(50)

bp.push('a',9)
bp.push('b',8)
bp.push('c',8)
bp.push('b',8)
bp.push('d',7)
bp.push('e',5)
bp.push('f',2)
```

```
DEBUG:  Pushing (a,9)
DEBUG:  Pushing (b,8)
DEBUG:  Pushing (c,8)
DEBUG:  Pushing (b,8)
DEBUG:  Pushing (d,7)
DEBUG:  Pushing (e,5)
DEBUG:  Pushing (f,2)
```

[24]:
```
print(bp)
```

```
Backpack: weight=47 max_weight=50
          elements=[('a', 9), ('b', 8), ('c', 8), ('b', 8), ('d', 7), ('e', 5), ('f',␣
↪2)]
```

[25]:
```
remove(bp, 'b')
```

```
DEBUG:  Popping ('f', 2)
DEBUG:  Popping ('e', 5)
DEBUG:  Popping ('d', 7)
DEBUG:  Popping ('b', 8)
DEBUG:  Pushing (d,7)
DEBUG:  Pushing (e,5)
DEBUG:  Pushing (f,2)
```

[25]:
```
('b', 8)
```

[26]:
```
print(bp)
```

```
Backpack: weight=39 max_weight=50
          elements=[('a', 9), ('b', 8), ('c', 8), ('d', 7), ('e', 5), ('f', 2)]
```

B.3 Concert

Start editing file `concert.py`.

When there are events with lots of potential visitors such as concerts, to speed up check-in there are at least two queues: one for cash where tickets are sold, and one for the actual entrance at the event.

Each visitor may or may not have a ticket. Also, since people usually attend in groups (coupls, families, and so on), in the queue lines each group tends to move as a whole.

In Python, we will model a `Person` as a class you can create like this:

[27]:
```
from concert_sol import *
```

[28]:
```
Person('a', 'x', False)
```

[28]:
```
Person(a,x,False)
```

`a` is the name, `'x'` is the group, and `False` indicates the person doesn't have ticket

To model the two queues, in `Concert` class we have these fields and methods:

```
class Concert:

    def __init__(self):
        self._cash = deque()
        self._entrance = deque()


    def enqc(self, person):
        """ Enqueues at the cash from the right """

        self._cash.append(person)

    def enqe(self, person):
```

(continues on next page)

(continued from previous page)

```
        """ Enqueues at the entrance from the right """

        self._entrance.append(person)
```

B3.1 dequeue

⊛⊛⊛ Implement `dequeue`. If you want, you can add debug prints by calling the `debug` function.

```
def dequeue(self):
    """ RETURN the names of people admitted to concert

        Dequeuing for the whole queue system is done in groups, that is,
        with a _single_ call to dequeue, these steps happen, in order:

        1. entrance queue: all people belonging to the same group at
           the front of entrance queue who have the ticket exit the queue
           and are admitted to concert. People in the group without the
           ticket are sent to cash.
        2. cash queue: all people belonging to the same group at the front
           of cash queue are given a ticket, and are queued at the entrance queue
    """
```

Testing: `python3 -m unittest concert_test.DequeueTest`

Example:

[29]:
```
con = Concert()

con.enqc(Person('a','x',False))  # a,b,c belong to same group x
con.enqc(Person('b','x',False))
con.enqc(Person('c','x',False))
con.enqc(Person('d','y',False))  # d belongs to group y
con.enqc(Person('e','z',False))  # e,f belongs to group z
con.enqc(Person('f','z',False))
con.enqc(Person('g','w',False))  # g belongs to group w
```

[30]:
```
con
```

[30]:
```
Concert:
      cash: deque([Person(a,x,False),
                   Person(b,x,False),
                   Person(c,x,False),
                   Person(d,y,False),
                   Person(e,z,False),
                   Person(f,z,False),
                   Person(g,w,False)])
  entrance: deque([])
```

First time we dequeue, entrance queue is empty so no one enters concert, while at the cash queue people in group `x` are given a ticket and enqueued at the entrance queue

NOTE: The messages on the console are just debug print, the function `dequeue` only return name sof people admitted to concert

[31]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  giving ticket to a (group x)
DEBUG:  giving ticket to b (group x)
DEBUG:  giving ticket to c (group x)
DEBUG:  Concert:
              cash: deque([Person(d,y,False),
                           Person(e,z,False),
                           Person(f,z,False),
                           Person(g,w,False)])
          entrance: deque([Person(a,x,True),
                           Person(b,x,True),
                           Person(c,x,True)])
```

[31]:
```
[]
```

[32]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  a (group x) admitted to concert
DEBUG:  b (group x) admitted to concert
DEBUG:  c (group x) admitted to concert
DEBUG:  giving ticket to d (group y)
DEBUG:  Concert:
              cash: deque([Person(e,z,False),
                           Person(f,z,False),
                           Person(g,w,False)])
          entrance: deque([Person(d,y,True)])
```

[32]:
```
['a', 'b', 'c']
```

[33]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  d (group y) admitted to concert
DEBUG:  giving ticket to e (group z)
DEBUG:  giving ticket to f (group z)
DEBUG:  Concert:
              cash: deque([Person(g,w,False)])
          entrance: deque([Person(e,z,True),
                           Person(f,z,True)])
```

[33]:
```
['d']
```

[34]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  e (group z) admitted to concert
DEBUG:  f (group z) admitted to concert
DEBUG:  giving ticket to g (group w)
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([Person(g,w,True)])
```

[34]:
```
['e', 'f']
```

[35]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  g (group w) admitted to concert
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([])
```

[35]:
```
['g']
```

[36]:
```
# calling dequeue on empty lines gives empty list:
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([])
```

[36]:
```
[]
```

Special dequeue case: broken group

In the special case when there is a group at the entrance with one or more members without a ticket, it is assumed that the group gets broken, so whoever has the ticket enters and the others get enqueued at the cash.

[37]:
```
con = Concert()

con.enqe(Person('a','x',True))
con.enqe(Person('b','x',False))
con.enqe(Person('c','x',True))
con.enqc(Person('f','y',False))

con
```

[37]:
```
Concert:
      cash: deque([Person(f,y,False)])
  entrance: deque([Person(a,x,True),
                   Person(b,x,False),
                   Person(c,x,True)])
```

[38]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  a (group x) admitted to concert
DEBUG:  b (group x) has no ticket! Sending to cash
DEBUG:  c (group x) admitted to concert
DEBUG:  giving ticket to f (group y)
DEBUG:  Concert:
              cash: deque([Person(b,x,False)])
          entrance: deque([Person(f,y,True)])
```

[38]:
```
['a', 'c']
```

[39]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  f (group y) admitted to concert
DEBUG:  giving ticket to b (group x)
DEBUG:  Concert:
```

(continues on next page)

(continued from previous page)

```
              cash: deque([])
          entrance: deque([Person(b,x,True)])
```

[39]:
```
['f']
```

[40]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  b (group x) admitted to concert
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([])
```

[40]:
```
['b']
```

[41]:
```
con
```

[41]:
```
Concert:
      cash: deque([])
  entrance: deque([])
```

[42]:
```
import sys;
sys.path.append('../../../');
import jupman;
import backpack_sol
import backpack_test
backpack_sol.DEBUG = False
jupman.run(backpack_test)

import concert_sol
import concert_test
concert_sol.DEBUG = False
jupman.run(concert_test)
```

```
…
----------------------------------------------------------------------
Ran 18 tests in 0.010s

OK
…
----------------------------------------------------------------------
Ran 7 tests in 0.004s

OK
```

[ ]:
```
```

2.1.9 Midterm sim - Tue 31, October 2019 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

This is only a simulation. By participating to it, you gain nothing, and you lose nothing

Valid code

> **WARNING**: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

> **WARNING**: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-10-31-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-08-26-FIRSTNAME-LASTNAME-ID
    exam-2019-10-31.ipynb
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2019-10-31-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-10-31-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[84]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - offerte lavoro EURES

Open Jupyter and start editing this notebook `exam-2019-10-31.ipynb`

After exiting this university prison, you will look for a job and be shocked to discover in Europe a great variety of languages are spoken. Many job listings are provided by Eures[85] portal, which is easily searchable with many fields on which you can filter. For this exercise we will use a test dataset 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[86]

License: Creative Commons Zero 1.0[87]

> **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 data/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('data/offerte-lavoro.csv', encoding='UTF-8')
offerte.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53 entries, 0 to 52
Data columns (total 8 columns):
RIFER.                 53 non-null object
SEDE LAVORO            53 non-null object
POSTI                  53 non-null int64
IMPIEGO RICHIESTO      53 non-null object
TIPO CONTRATTO         53 non-null object
LINGUA RICHIESTA       51 non-null object
RET. LORDA             53 non-null object
DESCRIZIONE OFFERTA    53 non-null object
dtypes: int64(1), object(7)
memory usage: 3.4+ KB
```

[84] http://examina.icts.unitn.it/studente
[85] https://ec.europa.eu/eures/public/homepage
[86] https://dati.trentino.it/dataset/offerte-di-lavoro-eures-test-odhb2019
[87] http://creativecommons.org/publicdomain/zero/1.0/deed.it

It contains Italian column names, and many string fields:

[2]:
```
offerte.head()
```

[2]:
```
           RIFER.      SEDE LAVORO  POSTI  \
0  18331901000024         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
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
```

[4]:
```
                            Reference  \
0                       18331901000024
1                              083PZMM
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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\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

                                   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
```

(continues on next page)

(continued from previous page)

```
6                Regional Key account manager - Italy
7                                        Receptionist
8           Customer Service Representative in Athens
9                                   Dispatch personnel
10         Mitarbeiter (m/w/d) im Verkaufsinnendienst
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 / Traine…
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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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
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…
```

(continues on next page)

(continued from previous page)

```
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…
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',
```

(continues on next page)

(continued from previous page)

```
    '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',
    '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.

[6]:
```
def countries_to_list(s):
    #jupman-raise
    ret = []
    i = 0
    ns = s.replace('\n',' ')
    for connective in connectives:
```

(continues on next page)

(continued from previous page)

```
        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
    #/jupman-raise

# 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[88]

[7]:
```
# write here
```

[8]:
```
# SOLUTION

offers['Workplace Country'] = offerte['SEDE LAVORO']
offers['Workplace Country'] = offers['Workplace Country'].transform(countries_to_list)
```

[9]:
```
print()
print("            *****************     SOLUTION OUTPUT     ********************")
offers
```

```
            *****************     SOLUTION OUTPUT     ********************
```

[9]:
```
                              Reference  \
0                         18331901000024
```

(continues on next page)

[88] https://sciprog.davidleoni.it/pandas/pandas-sol.html#7.-Transforming

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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\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                                      Svizia          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
```

(continues on next page)

(continued from previous page)

```
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 Verkaufsinnendienst
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 / Traine…
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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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
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]
```

(continues on next page)

(continued from previous page)

```
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 Reisebüroausbildu…            [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 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[89]

For other patterns to check, see asserts.

[10]:
```
months = ['gennaio', 'febbraio', 'marzo'    , 'aprile' , 'maggio'  , 'giugno',
          'luglio' , 'agosto'  , 'settembre', 'ottobre', 'novembre', 'dicembre' ]


def from_to(text):
    #jupman-raise
    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:
        return (np.nan, np.nan)
    #/jupman-raise

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)
```

(continues on next page)

[89] https://sciprog.davidleoni.it/matrices-numpy/matrices-numpy-sol.html#NaNs-and-infinities

(continued from previous page)

```
# 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

⊛ Change `offers` dataframe to so add `From` and `To` columns.

- **HINT 1**: You can call transform, see Transforming section in Pandas worksheet[90]
- **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[91]

[11]:
```
# write here
```

[12]:
```
# SOLUTION

offers['From'] = offers['Contract type'].transform(lambda t: from_to(t)[0])
offers['To'] =  offers['Contract type'].transform(lambda t: from_to(t)[1])
```

[13]:
```
print()
print(" ****************   SOLUTION OUTPUT  ****************")
offers
```

```
 ****************   SOLUTION OUTPUT  ****************
```

[13]:
```
                            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
```

(continues on next page)

[90] https://sciprog.davidleoni.it/pandas/pandas-sol.html#7.-Transforming
[91] https://sciprog.davidleoni.it/functions/functions-sol.html#Lambda-functions

(continued from previous page)

```
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
18                                     Irlanda          1
19                                 Paesi Bassi          5
```

(continues on next page)

(continued from previous page)

```
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

                                     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 Verkaufsinnendienst
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
```

(continues on next page)

(continued from previous page)

```
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 / Traine…
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
22        Da maggio a settembre o da giugno ad agosto
23                                    Non specificato
```

(continues on next page)

(continued from previous page)

```
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
24                       Francese ed italiano fluenti
25                  Inglese fluente; Tedesco discreto
```

(continues on next page)

(continued from previous page)

```
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
26                        Da definire
27                      20000 NOK /mese
```

(continues on next page)

(continued from previous page)

```
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…
28  Ihr Profil :Idealerweise Erfahrung in der Text…
29  Padronanza di una o più lingue tra queste (ita…
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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 NaNs and Infinities section in Numpy Matrices worksheet[92] !

[14]:

```
languages = {
 'italiano':'italian',
 'tedesco':'german',
 'francese':'french',
 'inglese':'english',
 'spagnolo':'spanish',
}

lang_levels = {
    'discreto':'B1',
    'buono':'B2',
    'fluente':'C1',
}

def reqlan(text):
    #jupman-raise

    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
    #/jupman-raise

# 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
```

(continues on next page)

[92] https://sciprog.davidleoni.it/matrices-numpy/matrices-numpy-sol.html#NaNs-and-infinities

(continued from previous page)

```
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
# are constructed by substituing '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:

[15]:
```
# write here

offers['Languages'] = offers['Required languages'].transform(reqlan)
```

[16]:
```
print()
print("         ******************    SOLUTION OUTPUT    **********************")
offers
```

```
         ******************    SOLUTION OUTPUT    **********************
```

[16]:
```
                              Reference  \
0                        18331901000024
1                               083PZMM
2                               4954752
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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\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

                                   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
```

(continues on next page)

(continued from previous page)

```
7                                       Receptionist
8           Customer Service Representative in Athens
9                                 Dispatch personnel
10        Mitarbeiter (m/w/d) im Verkaufsinnendienst
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 / Traine…
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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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
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…
```

(continues on next page)

(continued from previous page)

```
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…
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
```

(continues on next page)

(continued from previous page)

```
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

                                         Languages
0                                  {'english': 'C1'}
1   {'english': 'C1', 'french': 'C1', 'italian': '…
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                 {'english': 'B2', 'italian': 'C1'}
9                                  {'english': 'C1'}
10  {'german': 'C1', 'french': 'B2', 'italian': 'B2'}
11  {'english': 'C1', 'german': 'C1', 'spanish': '…
12                                  {'french': 'B1'}
13                {'english': 'B2', 'italian': 'B2'}
14                 {'english': 'C1', 'german': 'C1'}
15                                 {'english': 'C1'}
16                 {'german': 'C1', 'italian': 'C1'}
17                 {'german': 'B2', 'italian': 'B2'}
18                                 {'english': 'C1'}
```

(continues on next page)

(continued from previous page)

```
                                                {}
                                {'english': 'C1'}
                 {'english': 'C1', 'german': 'C1'}
                                {'english': 'C1'}
                                 {'german': 'C1'}
                 {'french': 'C1', 'italian': 'C1'}
                 {'english': 'C1', 'german': 'B1'}
                                                {}
{'english': 'C1', 'german': 'C1', 'italian': '…
{'english': 'C1', 'german': 'C1', 'spanish': '…
                                {'english': 'B2'}
                {'english': 'C1', 'italian': 'C1'}
                {'english': 'C1', 'italian': 'C1'}
                 {'french': 'C1', 'italian': 'C1'}
                                {'english': 'C1'}
                                {'english': 'C1'}
                 {'german': 'B2', 'italian': 'B2'}
                                {'english': 'B2'}
{'french': 'C1', 'spanish': 'B2', 'italian': '…
                 {'french': 'C1', 'italian': 'C1'}
                {'english': 'C1', 'italian': 'C1'}
                                 {'german': 'B2'}
                 {'english': 'B2', 'german': 'B2'}
                                {'english': 'C1'}
                                {'english': 'C1'}
                 {'german': 'B1', 'italian': 'B1'}
                                 {'german': 'B2'}
                                {'english': 'C1'}
                                {'english': 'C1'}
                 {'english': 'B2', 'german': 'B2'}
                 {'english': 'B2', 'german': 'B2'}
{'english': 'C1', 'french': 'C1', 'italian': '…
{'english': 'C1', 'french': 'C1', 'italian': '…
                                {'english': 'B2'}
```

[1]:
```
#Please execute this cell
import sys;
sys.path.append('../../../');
import jupman;
```

2.1.10 Midterm - Thu 07, Nov 2019 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[93]
  - write pythonic code[94]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

[93] https://sciprog.davidleoni.it/commandments.html
[94] http://docs.python-guide.org/writing/style

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-11-07-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-11-07-FIRSTNAME-LASTNAME-ID
    jupman.py
    sciprog.py
    exam-2019-11-07.ipynb
```

2) Rename `sciprog-ds-2019-11-07-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-11-07-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[95]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A

Open Jupyter and start editing this notebook `exam-2019-11-07.ipynb`

You will work on a dataset of events which occur in the Municipality of Trento, in 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 provider: Comune di Trento[96]

License: Creative Commons Attribution 4.0[97]

> **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 body of functions !! You have to write generic code which works with any input.

[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 !
eventi = pd.read_csv('data/eventi.csv', encoding='UTF-8')
eventi.info()
```

[95] http://examina.icts.unitn.it/studente
[96] https://dati.trentino.it/dataset/eventi-del-comune-di-trento
[97] http://creativecommons.org/licenses/by/4.0/deed.it

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253 entries, 0 to 252
Data columns (total 35 columns):
remoteId                      253 non-null object
published                     253 non-null object
modified                      253 non-null object
Priorità                      253 non-null int64
Evento speciale               0 non-null float64
Titolo                        253 non-null object
Titolo breve                  1 non-null object
Sottotitolo                   227 non-null object
Descrizione                   224 non-null object
Locandina                     16 non-null object
Inizio                        253 non-null object
Termine                       252 non-null object
Quando                        253 non-null object
Orario                        251 non-null object
Durata                        6 non-null object
Dove                          252 non-null object
lat                           253 non-null float64
lon                           253 non-null float64
address                       241 non-null object
Pagina web                    201 non-null object
Contatto email                196 non-null object
Contatto telefonico           196 non-null object
Informazioni                  62 non-null object
Costi                         132 non-null object
Immagine                      252 non-null object
Evento - manifestazione       252 non-null object
Manifestazione cui fa parte   108 non-null object
Tipologia                     252 non-null object
Materia                       252 non-null object
Destinatari                   24 non-null object
Circoscrizione                109 non-null object
Struttura ospitante           220 non-null object
Associazione                  1 non-null object
Ente organizzatore            0 non-null float64
Identificativo                0 non-null float64
dtypes: float64(5), int64(1), object(29)
memory usage: 69.3+ KB
```

We will concentrate on `Quando` (*When*) column:

[3]:
```
eventi['Quando']
```

[3]:
```
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
```

A.1 leap_year

⊛ A leap year has 366 days instead of regular 365. Yor 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 curios about calendars, see this link[98])

[4]:
```
def is_leap(year):
    #jupman-raise
    if year % 4 == 0:
        if year % 100 == 0:
            return year % 400 == 0
        else:
            return True
    else:
        return False
    #/jupman-raise


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
```

[98] https://docs.microsoft.com/en-us/office/troubleshoot/excel/determine-a-leap-year

A.2 full_date

⊛⊛ 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 begininning 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.

[5]:
```
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_phase = ['mattina', 'pomeriggio', 'sera', 'notte']
```

[6]:
```
def full_date(text):
    #jupman-raise
    ntext = text.lower()
    words = ntext.split()
    i = 1
    if words[i] in day_phase:
        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)
    #/jupman-raise

assert full_date("Giovedì 14 novembre 2019") == "2019-11-14"
```

(continues on next page)

(continued from previous page)

```
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"
```

A.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 will 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.

[7]:
```
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']
```

[8]:
```
def partial_date(text):
    #jupman-raise
    if type(text) != str:
        return 'yyyy-mm-dd'

    year = 'yyyy'
    month = 'mm'
    day = 'dd'

    ntext = text.lower()
    ret = []
    words = ntext.split()
```

(continues on next page)

(continued from previous page)

```
    if len(words) > 0:
        if words[0] == connective_from:
            i = 1
        else:
            i = 0
        if words[i] in days:
            i = i + 1
            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)
    #/jupman-raise

# 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"
```

A.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.

[9]:
```
def parse_dates_and(text):
    #jupman-raise
    ntext = text.lower()

    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)

    #/jupman-raise


# 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")
```

(continues on next page)

(continued from previous page)

```
# 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")
```

A.5 Fake news generator

> Functional illiteracy[99] 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 website 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

[10]:
```
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):
```

(continues on next page)

[99] https://en.wikipedia.org/wiki/Functional_illiteracy

(continued from previous page)

```
    #jupman-raise
    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
    #/jupman-raise


#fake_news(db_subjects, db_antecedents, db_consequences)
```

[11]:
```
print()
print("  ******************    EXPECTED OUTPUT    ******************")
print()
fake_news(db_subjects, db_antecedents, db_consequences)
```

```
  ******************    EXPECTED OUTPUT    ******************
```

[11]:
```
['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␣
↪requirements',
 'Government restricted building in natural areas, businesses had to fire many␣
↪employees',
 'Government restricted building in natural areas, businesses are struggling to meet␣
↪law requirements',
 'Government introduced more controls in agrifood production, businesses had to fire␣
↪many employees',
 'Government introduced more controls in agrifood production, businesses are␣
↪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␣
↪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␣
↪requirements',
```

(continues on next page)

(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']
```

2.1.11 Midterm B - Fri 20, Dec 2019

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

You can take this midterm ONLY IF you got grade >= 16 in Part A midterm.

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[100]
  - write pythonic code[101]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

[100] https://sciprog.davidleoni.it/commandments.html
[101] http://docs.python-guide.org/writing/style

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2019-12-20-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2019-12-20-FIRSTNAME-LASTNAME-ID
    exam-2019-12-20.ipynb
    theory.txt
    linked_list.py
    linked_list_test.py
    bin_tree.py
    bin_tree_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2019-12-20-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2019-12-20-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[102]
- If you don't have unitn login: tell instructors and we will download your work manually

[102] http://examina.icts.unitn.it/studente

Part B

B1 Theory

Write the solution in separate ``theory.txt`` file

B1.1 Complexity

Given a list L of n elements, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

```
def my_fun(L):
    R = 0
    for i in range(len(L)):
        for j in range(len(L)-1,0,-1):
            k = 0
            while k < 4:
                R = R + L[j] - L[i]
                k += 1
    return R
```

B1.2 Data structure choice

Given an algorithm that frequently checks the presence of an element in its internal data structure. Please briefly answer the following questions:

a. What data structure would you choose? Why?

b. In case entries are sorted, would you use the same data structures?

B2 LinkedList

Open a text editor and edit file `linkedlist.py`

You are given a `LinkedList` holding pointers `_head`, `_last`, and also `_size` attribute.

> Notice the list also holds `_last` and `_size` attributes !!!

B2.1 rotate

⊛⊛ Implement this method:

```
def rotate(self):
    """ Rotate the list of 1 element, that is, removes last node and
        inserts it as the first one.

       - MUST execute in O(n) where n is the length of the list
       - Remember to also update _last pointer
       - WARNING: DO *NOT* try to convert whole linked list to a python list
       - WARNING: DO *NOT* swap node data or create nodes, I want you to
                  change existing node links !!
    """
```

Testing: `python3 -m unittest linked_list_test.RotateTest`

Example:

[2]:
```
from linked_list_sol import *
```

[3]:
```
ll = LinkedList()
ll.add('d')
ll.add('c')
ll.add('b')
ll.add('a')
print(ll)
```

```
LinkedList: a,b,c,d
```

[4]:
```
ll.rotate()
```

[5]:
```
print(ll)
```

```
LinkedList: d,a,b,c
```

B2.2 rotaten

⊛⊛⊛ Implement this method:

```
def rotaten(self, k):
    """ Rotate k times the linkedlist

        - k can range from 0 to any positive integer number (even greater than list␣
↪size)
        - if k < 0 raise ValueError

        - MUST execute in O( n-(k%n) ) where n is the length of the list
        - WARNING: DO *NOT* call .rotate() k times !!!!
        - WARNING: DO *NOT* try to convert whole linked list to a python list
        - WARNING: DO *NOT* swap node data or create nodes, I want you to
                   change node links !!
    """
```

Testing: `python3 -m unittest linked_list_test.RotatenTest`

> **IMPORTANT HINT**
>
> The line "MUST execute in O(n-(k%n)) where n is the length of the list" means that you have to calculate `m = k%n`, and then only scan first `n-m` nodes!

Example:

[6]:
```
ll = LinkedList()
ll.add('h')
ll.add('g')
ll.add('f')
ll.add('e')
ll.add('d')
```

(continues on next page)

(continued from previous page)

```
ll.add('c')
ll.add('b')
ll.add('a')
print(ll)
```

```
LinkedList: a,b,c,d,e,f,g,h
```

[7]:
```
ll.rotaten(0)  # changes nothing
```

[8]:
```
print(ll)
```

```
LinkedList: a,b,c,d,e,f,g,h
```

[9]:
```
ll.rotaten(3)
```

[10]:
```
print(ll)
```

```
LinkedList: f,g,h,a,b,c,d,e
```

[11]:
```
ll.rotaten(8)  # changes nothing
```

[12]:
```
print(ll)
```

```
LinkedList: f,g,h,a,b,c,d,e
```

[13]:
```
ll.rotaten(5)
```

[14]:
```
print(ll)
```

```
LinkedList: a,b,c,d,e,f,g,h
```

[15]:
```
ll.rotaten(11)  # 11 = 8 + 3 , only rotates 3 nodes
```

[16]:
```
print(ll)
```

```
LinkedList: f,g,h,a,b,c,d,e
```

B3 Binary trees

We will now go looking for leaves, that is, nodes with no children. Open `bin_tree`.

[17]:
```
from bin_tree_test import bt
from bin_tree_sol import *
```

B3.1 sum_leaves_rec

⊛⊛ Implement this method:

```
def sum_leaves_rec(self):
    """ Supposing the tree holds integer numbers in all nodes,
        RETURN the sum of ONLY the numbers in the leaves.

        - a root with no children is considered a leaf
        - implement it as a recursive Depth First Search (DFS) traversal
          NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.SumLeavesRecTest`

Example:

[18]:
```
t = bt(3,
       bt(10,
```

(continues on next page)

(continued from previous page)

```
                bt(1),
                bt(7,
                    bt(5))),
        bt(9,
                bt(6,
                    bt(2,
                            None,
                            bt(4)),
                    bt(8))))

t.sum_leaves_rec()  #  1 + 5 + 4 + 8
```

[18]: 18

B3.2 leaves_stack

⊛⊛⊛ Implement this method:

```
def leaves_stack(self):
    """ RETURN a list holding the *data* of all the leaves  of the tree,
        in left to right order.

        - a root with no children is considered a leaf
        - DO *NOT* use recursion
        - implement it with a while and a stack (as a Python list)
    """
```

Testing: `python3 -m unittest bin_tree_test.LeavesStackTest`

Example:

[19]:
```
t = bt('a',
            bt('b',
                    bt('c'),
                    bt('d',
                            None,
                            bt('e'))),
            bt('f',
                    bt('g',
                            bt('h')),
                    bt('i')))
t.leaves_stack()
```

[19]: ['c', 'e', 'h', 'i']

[ ]:

2.1.12 Exam - Thu 23, Jan 2020 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[103]
  - write pythonic code[104]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

[103] https://sciprog.davidleoni.it/commandments.html
[104] http://docs.python-guide.org/writing/style

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2020-01-23-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2020-01-23-FIRSTNAME-LASTNAME-ID
   data
       db.mm
       proof.txt

   exam-2020-01-23.ipynb
   digi_list.py
   digi_list_test.py
   bin_tree.py
   bin_tree_test.py
   jupman.py
   sciprog.py
```

2) Rename `sciprog-ds-2020-01-23-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2020-01-23-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[105]
- If you don't have unitn login: tell instructors and we will download your work manually

[105] http://examina.icts.unitn.it/studente

Part A

Open Jupyter and start editing this notebook `exam-2020-01-23.ipynb`

Metamath

Metamath[106] 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[107] up to the most basic axioms[108] they rely on to be proven .

For this exercise, we have two files to consider, `db.mm` and `proof.txt`.

- `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

The purpose of this exercise 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

Metamath db

First you will load `data/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 $.
    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:

[106] http://us.metamath.org
[107] http://us.metamath.org/mm_100.html
[108] http://us.metamath.org/mpeuni/mmtheorems1.html#mm5s

- 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 $)`
  - `${`
  - `$}`
- in each row, after the first dollar *keyword*, you *may* have an arbitratry *sequence* of characters terminated by a dollar followed by a dot `$.`. **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[109]), 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': '|- ( P -> Q )'
        },
 .
 .
 .
}
```

[109] https://sciprog.davidleoni.it/formats/formats-sol.html#1.-line-files

A.1 Metamath db

[2]:
```
def parse_db(filepath):
    #jupman-raise
    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
                }
                #print('   DEBUG: FOUND', label, ':', ret[label])
            #else:
                #print('   DEBUG: DISCARDED')
            line=f.readline().strip()
    return ret
    #/jupman-raise

db_mm = parse_db('data/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)
```

[3]:
```
from pprint import pprint
```

(continues on next page)

(continued from previous page)

```
print("************   EXPECTED OUTPUT:  ****************")
pprint(db_mm)
```

```
************   EXPECTED OUTPUT:  ****************
{'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'}}
```

A.2 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 `data/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
```

(continues on next page)

(continued from previous page)

```
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]},
     .
     .
     .
]
```

[4]:
```
def parse_proof(filepath):
    #jupman-raise
    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)

            ret.append( {
                'step_ids': step_ids,
                'sequence': sequence,
                'label': label,
                'keyword': keyword
            })

            line=f.readline().strip()
        return ret
    #/jupman-raise

proof = parse_proof('data/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 )',
```

(continues on next page)

(continued from previous page)

```
 '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': '$a',
 'label': 'tpl',
 'sequence': 'term ( t + 0 )',
 'step_ids': [16, 17]},
{'keyword': '$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]},
```

(continues on next page)

(continued from previous page)

```
 {'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]}]
```

Checking proof

If you've done everything properly, by executing following cells you should be 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

[5]:
```
from sciprog import draw_proof
# uncomment and check
#draw_proof(proof, db_mm, only_ids=True)  # all graph, only numbers
```

[6]:
```
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.

[7]:
```
# uncomment and check
#draw_proof(proof, db_mm, step_id=24)
```

[8]:
```
print()
print('************************* EXPECTED DETAIL GRAPH ␣
↪******************************')
draw_proof(proof, db_mm, step_id=24)
```

```
************************* EXPECTED DETAIL GRAPH  ******************************
```

A.3 Metamath top statements

We can measure the importance of theorems and definitions (in general, *statements*) by counting how many times they are referenced in proofs.

A3.1 histogram

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

A3.2 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`

[9]:
```
# write here
```

[10]:
```
# SOLUTION

import numpy as np
import matplotlib.pyplot as plt


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


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
```

[ ]:

Part B

B1 Theory

Write the solution in separate ``theory.txt`` file

B1.1 my_fun

Given a list `L` of `n` elements, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

```
def my_fun(L):
    n = len(L)
    if n <= 1:
        return 1
    else:
        L1 = L[0:n//2]
        L2 = L[n//2:]
        a = my_fun(L1) + max(L1)
        b = my_fun(L2) + max(L2)
        return a + b
```

B1.2 differences

Briefly describe the main differences between the stack and queue data structures. Please provide an example of where you would use one or the other.

B2 plus_one

Open a text editor and edit file `digi_lists.py`

You are given this class:

```
class DigiList:
    """
        This is a stripped down version of the LinkedList as previously seen,
        which can only hold integer digits 0-9

        NOTE: there is also a _last pointer

    """
```

Implement this method:

```
def plus_one(self):
    """ MODIFIES the digi list by summing one to the integer number it represents
        - you are allowed to perform multiple scans of the linked list
        - remember the list has a _last pointer

        - MUST execute in O(N) where N is the size of the list
        - DO *NOT* create new nodes EXCEPT for special cases:
            a. empty list ( [] -> [5] )
            b. all nines ( [9,9,9] -> [1,0,0,0] )
        - DO *NOT* convert the digi list to a python int
        - DO *NOT* convert the digi list to a python list
        - DO *NOT* reverse the digi list
    """
```

Test: `python3 -m unittest digi_list_test.PlusOneTest`

Example:

[11]:
```
from digi_list_sol import *

dl = DigiList()

dl.add(9)
dl.add(9)
dl.add(7)
dl.add(3)
dl.add(9)
dl.add(2)

print(dl)
```

```
DigiList: 2,9,3,7,9,9
```

[12]:
```
dl.last()
```

[12]:
```
9
```

[13]:
```
dl.plus_one()
```

[14]:
```
print(dl)
```

```
DigiList: 2,9,3,8,0,0
```

B3 add_row

Open a text editor and edit file `bin_tree.py`.

Now implement this method:

```
def add_row(self, elems):
    """ Takes as input a list of data and MODIFIES the tree by adding
        a row of new leaves, each having as data one element of elems,
        in order.

        - elems size can be less than 2*|leaves|
        - if elems size is more than 2*|leaves|, raises ValueError
        - for simplicity, you can assume assume self is a perfect
          binary tree, that is a binary tree in which all interior nodes
          have two children and all leaves have the same depth
        - MUST execute in O(n+|elems|)  where n is the size of the tree
        - DO *NOT* use recursion
        - implement it with a while and a stack (as a Python list)
    """
```

Test: `python3 -m unittest bin_tree_test.AddRowTest`

Example:

[15]:
```
from bin_tree_sol import *
from bin_tree_test import bt

t = bt('a',
            bt('b',
```

(continues on next page)

(continued from previous page)

```
                    bt('d'),
                    bt('e')),
            bt('c',
                    bt('f'),
                    bt('g')))

print(t)
```

```
a
├b
│├d
│└e
└c
 ├f
 └g
```

[16]:
```
t.add_row(['h','i','j','k','l'])
```

[17]:
```
print(t)
```

```
a
├b
│├d
││├h
││└i
│└e
│ ├j
│ └k
└c
 ├f
 │├l
 │└
 └g
```

2.1.13 Exam - Monday 10, February 2020 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.
- **Bonus point**: One bonus point can be earned by writing stylish code. You got style if you:
  - do not infringe the Commandments[110]
  - write pythonic code[111]
  - avoid convoluted code like i.e.

    ```
    if x > 5:
        return True
    else:
        return False
    ```

    when you could write just

    ```
    return x > 5
    ```

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

[110] https://sciprog.davidleoni.it/commandments.html
[111] http://docs.python-guide.org/writing/style

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2020-02-10-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2020-02-10-FIRSTNAME-LASTNAME-ID
    exam-2020-02-10.ipynb
    B1-theory.txt
    B2_italian_queue_v2.py
    B2_italian_queue_v2_test.py
    jupman.py
    sciprog.py
```

2) Rename `sciprog-ds-2020-02-10-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2020-02-10-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[112]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A

Open Jupyter and start editing this notebook `exam-2020-02-10.ipynb`

> WordNet[113]® 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[114]. Princeton University. 2010

In Python there are specialized libraries to read WordNet like NLTK[115], but for the sake of this exercise, 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)

[112] http://examina.icts.unitn.it/studente
[113] https://wordnet.princeton.edu/
[114] https://wordnet.princeton.edu/
[115] https://www.nltk.org/howto/wordnet.html

A1 parse_db

First, you will begin with parsing an excerpt of wordnet `data/dogs.noun`, which is a noun database shown here in its entirety.

According to documentation[116], 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
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
```

(continues on next page)

[116] https://wordnet.princeton.edu/documentation/wndb5wn

(continued from previous page)

```
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`

implement parse_db

[2]:
```
def parse_db(filename):
    """ 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 correspoing synset. See next printout for an␣
↪example.
    """
    #jupman-raise

    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 ],     # '~'
                         'synset_offset': int(params[i + 1]),  # '00001930'
                         'pos': params[i + 2],            # 'n'
```

(continues on next page)

(continued from previous page)

```
                        'source_target':params[i + 3],   # '0000'
                    }
                d['ptrs'].append(ptr)
                i += 4


            d['gloss'] = line.split('|')[1]

            ret[d['synset_offset']] = d
            i += 1
            line=f.readline()
        return ret
    #/jupman-raise
```

[3]:
```
dogs_db = parse_db('data/dogs.noun')

from pprint import pprint
pprint(dogs_db)
```

```
{1320032: {'gloss': ' any of various animals that have been tamed and made fit '
                    'for a human environment\n',
           '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  \n',
          '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'}]},
2086723: {'gloss': ' 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" \n',
          'lex_filenum': 5,
```

(continues on next page)

(continued from previous page)


```
        'p_cnt': 23,
        'ptrs': [{'pointer_symbol': '@',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2085998},
                 {'pointer_symbol': '@',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 1320032},
                 {'pointer_symbol': '#m',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2086515},
                 {'pointer_symbol': '#m',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 8011383},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 1325095},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2087384},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2087513},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2087924},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2088026},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2089774},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2106058},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2112993},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2113458},
                 {'pointer_symbol': '~',
                  'pos': 'n',
                  'source_target': '0000',
                  'synset_offset': 2113610},
```


(continues on next page)

(continued from previous page)

```
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2113781},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2113929},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2114152},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2114278},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2115149},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2115478},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2115987},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2116630},
                    {'pointer_symbol': '%p',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2161498}],
          'ss_type': 'n',
          'synset_offset': 2086723,
          'w_cnt': 3,
          'words': [{'lex_id': 0, 'word': 'dog'},
                    {'lex_id': 0, 'word': 'domestic_dog'},
                    {'lex_id': 0, 'word': 'Canis_familiaris'}]},
2106058: {'gloss': ' any of several breeds of usually large powerful dogs '
                   'bred to work as draft animals and guard and guide '
                   'dogs  \n',
          'lex_filenum': 5,
          'p_cnt': 16,
          'ptrs': [{'pointer_symbol': '@',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2086723},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2106493},
                   {'pointer_symbol': '~',
                    'pos': 'n',
```

(continues on next page)

(continued from previous page)

```
                 'source_target': '0000',
                 'synset_offset': 2107175},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2109506},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2110072},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2110741},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2110906},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2111074},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2111324},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2111699},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2111802},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2112043},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2112177},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2112339},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2112463},
                {'pointer_symbol': '~',
                 'pos': 'n',
                 'source_target': '0000',
                 'synset_offset': 2112613}],
        'ss_type': 'n',
        'synset_offset': 2106058,
        'w_cnt': 1,
```

(continues on next page)

(continued from previous page)

```
          'words': [{'lex_id': 0, 'word': 'working_dog'}]},
2107175: {'gloss': ' any of various usually long-haired breeds of dog reared '
                   'to herd and guard sheep\n',
          'lex_filenum': 5,
          'p_cnt': 12,
          'ptrs': [{'pointer_symbol': '@',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2106058},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2107534},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2107903},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108064},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108157},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108293},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108507},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108682},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108818},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2109034},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2109202},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2109314}],
          'ss_type': 'n',
          'synset_offset': 2107175,
          'w_cnt': 3,
          'words': [{'lex_id': 0, 'word': 'shepherd_dog'},
```

(continues on next page)

(continued from previous page)

```
                    {'lex_id': 0, 'word': 'sheepdog'},
                    {'lex_id': 0, 'word': 'sheep_dog'}]},
2111324: {'gloss': ' a sturdy thickset short-haired breed with a large head '
                   'and strong undershot lower jaw; developed originally in '
                   'England for bull baiting  \n',
          'lex_filenum': 5,
          'p_cnt': 3,
          'ptrs': [{'pointer_symbol': '@',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2106058},
                   {'pointer_symbol': '+',
                    'pos': 'v',
                    'source_target': '0101',
                    'synset_offset': 1121448},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2111567}],
          'ss_type': 'n',
          'synset_offset': 2111324,
          'w_cnt': 2,
          'words': [{'lex_id': 0, 'word': 'bulldog'},
                    {'lex_id': 0, 'word': 'English_bulldog'}]},
2112993: {'gloss': ' a large breed having a smooth white coat with black or '
                   'brown spots; originated in Dalmatia  \n',
          'lex_filenum': 5,
          'p_cnt': 2,
          'ptrs': [{'pointer_symbol': '@',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2086723},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2113184}],
          'ss_type': 'n',
          'synset_offset': 2112993,
          'w_cnt': 3,
          'words': [{'lex_id': 0, 'word': 'dalmatian'},
                    {'lex_id': 0, 'word': 'coach_dog'},
                    {'lex_id': 0, 'word': 'carriage_dog'}]},
2116752: {'gloss': ' any of various predatory carnivorous canine mammals of '
                   'North America and Eurasia that usually hunt in packs  \n',
          'lex_filenum': 5,
          'p_cnt': 7,
          'ptrs': [{'pointer_symbol': '@',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2085998},
                   {'pointer_symbol': '#m',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2086515},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
```

(continues on next page)

(continued from previous page)

```
                     'synset_offset': 1324999},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2117019},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2117200},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2117364},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2117507}],
            'ss_type': 'n',
            'synset_offset': 2116752,
            'w_cnt': 1,
            'words': [{'lex_id': 0, 'word': 'wolf'}]}}
```

A2 to_adj

Implement a function `to_adj` which takes the parsed db and RETURN a graph-like data structure in adjacency list format. Each node represent 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 as data the result of the previous print.

[4]:
```python
def to_adj(db):
    #jupman-raise
    ret = {}

    for d in db.values():
        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
    #/jupman-raise

dogs_graph = to_adj(dogs_db)
from pprint import pprint
pprint(dogs_graph)
```

```
{'bulldog': ['working_dog'],
 'canine': [],
 'dalmatian': ['dog'],
 'dog': ['canine', 'domestic_animal'],
```

(continues on next page)

(continued from previous page)

```
 'domestic_animal': [],
 'shepherd_dog': ['working_dog'],
 'wolf': ['canine'],
 'working_dog': ['dog']}
```

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

[5]:
```
from sciprog import draw_adj
draw_adj(dogs_graph, options={'graph':{'rankdir':'BT'}})
```

A.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/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 `dogs_db`

- sort the histogram from greatest to lowest frequency
- do not count the relations containing the word 'domain' inside (upper/lowercase)
- do not count the '' relation

- display the relation names nicely, adding newlines if necessary

[6]:

```
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):
    #jupman-raise
    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
    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')
```

(continues on next page)

(continued from previous page)

```
    xticks = [relation_names[c[0]].replace(' ', '\n') for c in coords]
    plt.xticks(xs,xticks)

    plt.show()
    #/jupman-raise
```

[ ]:

```

```

[7]:

```
wordnet = parse_db('data/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}
```

Part B

B1 Theory

Write the solution in separate ``theory.txt`` file

B1.1 complexity

Given a list L of n elements, please compute the asymptotic computational complexity of the following function, explaining your reasoning. Any ideas on how to improve the complexity of this code?

```
def my_fun(L):
    n = len(L)
    out = []
    for i in range(n-2):
        out.insert(0,L[i] + L[i+1] + L[i+2])
    return out
```

B1.2 graph visits

Briefly describe the two classic ways of visiting the nodes of a graph.

B2 ItalianQueue v2

Open a text editor and have a look at file `italian_queue_v2.py`

In the original v1 implementation of the ItalianQueue we've already seen in class[117], `enqueue` can take $O(n)$: you will improve it by adding further indexing so it runs in $O(1)$

An `ItalianQueue` is modelled as a LinkedList with two pointers, a `_head` and a `_tail`:

- an element is enqueued scanning from `_head` until a matching group is found, in which case the element is inserted after (that is, at the right) of the matching group, otherwise the element is appended at the very end marked by `_tail`
- an element is dequeued from the `_head`

For this improved v2 version, you will use an additional dictionary `_tails` which associates to each group present in the queue the node at the tail of that group sequence. This way, instead of scanning you will be able to directly jump to insertion point.

```
class ItalianQueue:

    def __init__(self):
        """ Initializes the queue.

            - Complexity: O(1)
        """
        self._head = None
        self._tail = None
        self._tails = {}    #  <---- NEW  !
        self._size = 0
```

Example:

If we have the following situation:

```
data  :  a -> b -> c -> d -> e -> f -> g -> h
group :  x    x    y    y    y    z    z    z
         ^    ^              ^              ^
```

(continues on next page)

[117] https://sciprog.davidleoni.it/queues/queues.html#3.-ItalianQueue

(continued from previous page)

```
         |    |              |              |
         | _tails[x]      _tails[y]      _tails[z]
         |                                  |
       _head                              _tail
```

By calling

```
q.enqueue('i','y')
```

We get:

```
data  :  a -> b -> c -> d -> e -> i -> f -> g -> h
group :  x    x    y    y    y    y    z    z    z
         ^    ^                   ^              ^
         |    |                   |              |
         | _tails[x]           _tails[y]      _tails[z]
         |                                       |
       _head                                   _tail
```

We can see here the complete run:

[8]:
```
from italian_queue_v2_sol import *

q = ItalianQueue()
print(q)
```

```
ItalianQueue:

       _head: None
       _tail: None
      _tails: {}
```

[9]:
```
q.enqueue('a','x')    # 'a' is the element,'x' is the group
```

[10]:
```
print(q)
```

```
ItalianQueue: a
              x
       _head: Node(a,x)
       _tail: Node(a,x)
      _tails: {'x': Node(a,x),}
```

[11]:
```
q.enqueue('c','y')     # 'c' belongs to new group 'y', goes to the end of the queue
```

[12]:
```
print(q)
```

```
ItalianQueue: a->c
              x  y
       _head: Node(a,x)
       _tail: Node(c,y)
      _tails: {'x': Node(a,x),
               'y': Node(c,y),}
```

[13]:
```
q.enqueue('d','y')     # 'd' belongs to existing group 'y', goes to the end of the␣
↪group
```

[14]:
```
print(q)
```

```
ItalianQueue: a->c->d
              x  y  y
       _head: Node(a,x)
       _tail: Node(d,y)
      _tails: {'x': Node(a,x),
               'y': Node(d,y),}
```

[15]:
```
q.enqueue('b','x')    # 'b' belongs to existing group 'x', goes to the end of the␣
↪group
```

[16]:
```
print(q)
```

```
ItalianQueue: a->b->c->d
              x  x  y  y
       _head: Node(a,x)
       _tail: Node(d,y)
      _tails: {'x': Node(b,x),
               'y': Node(d,y),}
```

[17]:
```
q.enqueue('f','z')    # 'f' belongs to new group, goes at the end of the queue
```

[18]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->f
              x  x  y  y  z
       _head: Node(a,x)
       _tail: Node(f,z)
      _tails: {'x': Node(b,x),
               'y': Node(d,y),
               'z': Node(f,z),}
```

[19]:
```
q.enqueue('e','y')   # 'e' belongs to an existing group 'y', goes at the end of the␣
↪group
```

[20]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->e->f
              x  x  y  y  y  z
       _head: Node(a,x)
       _tail: Node(f,z)
      _tails: {'x': Node(b,x),
               'y': Node(e,y),
               'z': Node(f,z),}
```

[21]:
```
q.enqueue('g','z')   # 'g' belongs to an existing group 'z', goes at the end of the␣
↪group
```

[22]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->e->f->g
              x  x  y  y  y  z  z
       _head: Node(a,x)
       _tail: Node(g,z)
      _tails: {'x': Node(b,x),
```

(continues on next page)

(continued from previous page)

```
                'y': Node(e,y),
                'z': Node(g,z),}
```

[23]:
```
q.enqueue('h','z')  # 'h' belongs to an existing group 'z', goes at the end of the
↪group
```

[24]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->e->f->g->h
              x  x  y  y  y  z  z  z
       _head: Node(a,x)
       _tail: Node(h,z)
      _tails: {'x': Node(b,x),
               'y': Node(e,y),
               'z': Node(h,z),}
```

[25]:
```
q.enqueue('h','z')  # 'h' belongs to an existing group 'z', goes at the end of the
↪group
```

[26]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->e->f->g->h->h
              x  x  y  y  y  z  z  z  z
       _head: Node(a,x)
       _tail: Node(h,z)
      _tails: {'x': Node(b,x),
               'y': Node(e,y),
               'z': Node(h,z),}
```

[27]:
```
q.enqueue('i','y')  # 'i' belongs to an existing group 'y', goes at the end of the
↪group
```

[28]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->e->i->f->g->h->h
              x  x  y  y  y  y  z  z  z  z
       _head: Node(a,x)
       _tail: Node(h,z)
      _tails: {'x': Node(b,x),
               'y': Node(i,y),
               'z': Node(h,z),}
```

Dequeue is always from the head, without taking in consideration the group:

[29]:
```
q.dequeue()
```

[29]:
```
'a'
```

[30]:
```
print(q)
```

```
ItalianQueue: b->c->d->e->i->f->g->h->h
              x  y  y  y  y  z  z  z  z
       _head: Node(b,x)
       _tail: Node(h,z)
      _tails: {'x': Node(b,x),
               'y': Node(i,y),
               'z': Node(h,z),}
```

[31]:
```
q.dequeue()    # removed last member of group 'x', key 'x' disappears from _tails['x']
```

[31]:
```
'b'
```

[32]:
```
print(q)
```

```
ItalianQueue: c->d->e->i->f->g->h->h
              y  y  y  y  z  z  z  z
       _head: Node(c,y)
       _tail: Node(h,z)
      _tails: {'y': Node(i,y),
               'z': Node(h,z),}
```

[33]:
```
q.dequeue()
```

[33]:
```
'c'
```

[34]:
```
print(q)
```

```
ItalianQueue: d->e->i->f->g->h->h
              y  y  y  z  z  z  z
       _head: Node(d,y)
       _tail: Node(h,z)
      _tails: {'y': Node(i,y),
               'z': Node(h,z),}
```

B2.1 enqueue

Implement `enqueue`:

```
def enqueue(self, v, g):
    """ Enqueues provided element v having group g, with the following
        criteria:

        Queue is scanned from head to find if there is another element
        with a matching group:
            - if there is, v is inserted after the last element in the
              same group sequence (so to the right of the group)
            - otherwise v is inserted at the end of the queue

        - MUST run in O(1)
    """
```

Testing: `python3 -m unittest italian_queue_test.EnqueueTest`

B2.2 dequeue

Implement `dequeue`:

```
def dequeue(self):
        """ Removes head element and returns it.

            - If the queue is empty, raises a LookupError.
            - MUST perform in O(1)
            - REMEMBER to clean unused _tails keys
        """
```

IMPORTANT: you can test ``dequeue`` even if you didn't implement ``enqueue`` correctly

Testing: `python3 -m unittest italian_queue_test.DequeueTest`

[ ]:

2.1.14 Exam - Tuesday 16, June 2020 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solutions

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation

def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2020-06-16-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2020-06-16-FIRSTNAME-LASTNAME-ID
   exam-2020-06-16.ipynb
   theory.txt
   linked_list.py
   linked_list_test.py
   bin_tree.py
   bin_tree_test.py
```

2) Rename `sciprog-ds-2020-06-16-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2020-06-16-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[118]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - Zoom surveillance

A training center holds online courses with Zoom software[119]. Participants attendance is mandatory, and teachers want to determine who left, when and for what reason. Zoom allows to save a meeting log in a sort of CSV format which holds the timings of joins and leaves of each student. You will clean the file content and show relevant data in charts.

Basically, you are going to build a surveillance system to monitor YOU. Welcome to digital age.

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 (if you want, you may use LibreOffice Calc to help yourself)

Here we only show the few fields we are actually interested in, and examples of trasformations you should apply:

From general meeting information section:

[118] http://examina.icts.unitn.it/studente
[119] https://zoom.us/

- `Meeting ID:` `123 4567 8901`
- `Topic:` `Hydraulics Exam`
- `Start Time:` `"Apr 17, 2020 02:00 PM"` should become `Apr 17, 2020`

From participant sessions section:

- `Participant:` `Luigi`
- `Join Time:` `01:54 PM` should become `13:54`
- `Leave Time:` `03:10 PM(Luigi 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):

- `(Luigi got disconnected from the meeting.Reason: Network connection error. )` should become `Network connection error`
- `(Bowser left the meeting.Reason: Host closed the meeting. )` should become `Host closed the meeting`
- `(Princess Toadstool 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:

```
[['meeting_id', 'topic', 'date', 'participant', 'join_time', 'leave_time', 'reason'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Luigi','13:54','15:10','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Luigi','15:12','15:54','left the␣
↪meeting'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','14:02','14:16','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','14:19','15:02','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','15:04','15:50','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','15:52','15:55','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','15:56','16:00','Host␣
↪closed the meeting'],
 ...
]
```

To fix the times, you will first need to implement the following function.

Open Jupyter and start editing this notebook `exam-2020-06-16.ipynb`

A1 time24

[1]:
```
def time24(t):
    """ Takes a time string like '06:27 PM' and outputs a string like 18:27
    """
    #jupman_raise
    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','')
    #/jupman_raise

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'
```

A2 load

Implement a function which loads the file `UserQos_12345678901.csv` and RETURN a list of lists.

To parse the file, you can use simple CSV parsing[120] as seen in class (there is no need to use pandas)

[2]:
```
import csv

def load(filepath):
    #jupman-raise
    ret = []
    with open(filepath, encoding='utf-8', newline='') as f:

        lettore = csv.reader(f, delimiter=',')
        next(lettore)
        riga_meeting = next(lettore)
        meeting_id = riga_meeting[0]
        topic = riga_meeting[1]
        meeting_date = riga_meeting[7]
        next(lettore) # riga vuota
        next(lettore) # secondo header
        ret.append(['meeting_id', 'topic','date', 'participant','join_time','leave_
↪time','reason'])
        for riga in lettore:
            session = {}
            if len(riga) > 0:
```

(continues on next page)

[120] https://sciprog.davidleoni.it/formats/formats-sol.html#2.-File-CSV

(continued from previous page)

```
                ret.append([meeting_id,
                            topic,
                            meeting_date[:12],
                            riga[0],
                            time24(riga[10]),
                            time24(riga[11].split('(')[0]),
                            riga[11].split('Reason: ')[1].split('.')[0]])
    return ret
    #/jupman-raise

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', 'Hydraulics Exam', 'Apr 17, 2020', 'Luigi', '13:54', '15:10',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Luigi', '15:12', '15:54',
↪'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '14:02', '14:16',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '14:19', '15:02',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '15:04', '15:50',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '15:52', '15:55',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '15:56', '16:00',
↪'Host closed the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '14:15', '14:30',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '14:54', '15:03',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '15:12', '15:40',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '15:45', '16:00',
↪'Host closed the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Princess Toadstool', '13:56',
↪'15:33', 'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:05', '14:10',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:15', '14:29',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:33', '15:10',
↪'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '15:25', '15:54',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '15:55', '16:00',
↪'Host closed the meeting']]
```

[3]:
```
EXPECTED_MEETING_LOG = \
[['meeting_id', 'topic', 'date', 'participant', 'join_time', 'leave_time', 'reason'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Luigi', '13:54', '15:10',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Luigi', '15:12', '15:54',
↪'left the meeting'],
```

(continues on next page)

(continued from previous page)

```
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '14:02', '14:16',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '14:19', '15:02',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '15:04', '15:50',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '15:52', '15:55',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '15:56', '16:00',
↪'Host closed the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '14:15', '14:30',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '14:54', '15:03',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '15:12', '15:40',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '15:45', '16:00',
↪'Host closed the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Princess Toadstool', '13:56',
↪'15:33', 'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:05', '14:10',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:15', '14:29',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:33', '15:10',
↪'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '15:25', '15:54',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '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 Luigi row
assert meeting_log[1:3] == EXPECTED_MEETING_LOG[1:3]  # Luigi rows
assert meeting_log[:4]  == EXPECTED_MEETING_LOG[:4]   # until first Mario row included
assert meeting_log      == EXPECTED_MEETING_LOG       # all table
```

A3.1 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**

[4]:
```
def duration(a, b):
    #jupman-raise
    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
    #/jupman-raise

assert duration('15:00','15:34') == 34
assert duration('15:00','17:34') == 120 + 34
assert duration('15:50','16:12') == 22
```

(continues on next page)

(continued from previous page)

```
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 !!
```

A3.2 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 partecipant 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 happend 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

[5]:
```
def calc_stats(log):
    #jupman-raise
    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
                               }

        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
```

(continues on next page)

(continued from previous page)

```
        last_sessions[participant] = session
    return ret
    #/jupman-raise


stats = calc_stats(meeting_log)

# in case you had trouble implementing load function, use this:
#stats = calc_stats(EXPECTED_MEETING_LOG)

stats
```

[5]:
```
{'Bowser': {'gaps': 3, 'max_gap': 24, 'time_away': 38},
 'Luigi': {'gaps': 1, 'max_gap': 2, 'time_away': 2},
 'Mario': {'gaps': 4, 'max_gap': 3, 'time_away': 8},
 'Princess Toadstool': {'gaps': 0, 'max_gap': 0, 'time_away': 0},
 'Wario': {'gaps': 4, 'max_gap': 15, 'time_away': 25}}
```

[6]:
```
EXPECTED_STATS = {            'Bowser': {'gaps': 3, 'max_gap': 24, 'time_away': 38},
                               'Luigi': {'gaps': 1, 'max_gap': 2,  'time_away': 2},
                               'Mario': {'gaps': 4, 'max_gap': 3,  'time_away': 8},
                  'Princess Toadstool': {'gaps': 0, 'max_gap': 0,  'time_away': 0},
                               'Wario': {'gaps': 4, 'max_gap': 15, 'time_away': 25}}

assert stats == EXPECTED_STATS
```

A4 viz

Produce a bar chart of the statistics you calculated before. For how to do it, see examples in Visualiation tutorial[121]

- 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

[7]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def viz(stats):
    #jupman-raise

    xs = np.arange(len(stats))
    ys_max_gap = []
    ys_time_away = []

    labels = list(sorted(stats.keys()))

    for participant in sorted(stats):
```

(continues on next page)

[121] https://sciprog.davidleoni.it/visualization/visualization-sol.html

(continued from previous page)

```
        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)

    ax.set_title('Disconnections SOLUTION')
    ax.legend()


    plt.ylabel('minutes')
    plt.savefig('surveillance.png')
    plt.show()
    #/jupman-raise

viz(stats)

# in case you had trouble implementing calc_stats, use this:
#viz(EXPECTED_STATS)
```

Part B

B1 Theory

Write the solution in separate `theory.txt` **file**

B1.1 complexity

Given a list `L` of `n` positive integers, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

```
def my_max(L):
    M = -1
    for e in L:
        if e > M:
            M = e
    return M

def my_fun(L):
    n = len(L)
    out = 0
    for i in range(5):
        out = out + my_max(L[i:])
    return out
```

B1.2 describe

Briefly describe what a bidirectional linked list is. How does it differ from a queue?

B2 - LinkedList slice

Open a text editor and edit file `linked_list.py`

Implement the method `slice`:

```
def slice(self, start, end):
    """ RETURN a NEW LinkedList created by copying nodes of this list
        from index start INCLUDED to index end EXCLUDED

        - if start is greater or equal than end, returns an empty LinkedList
        - if start is greater than available nodes, returns an empty LinkedList
        - if end is greater than the available nodes, copies all items until the tail␣
↪without errors
        - if start index is negative, raises ValueError
        - if end index is negative, raises ValueError

        - IMPORTANT: All nodes in the returned LinkedList MUST be NEW
        - DO *NOT* modify original linked list
        - DO *NOT* add an extra size field
        - MUST execute in O(n), where n is the size of the list

    """
```

Testing: `python3 -m unittest linked_list_test.SliceTest`

Example:

[8]:
```
from linked_list_sol import *
```

[9]:
```
la = LinkedList()
la.add('g')
la.add('f')
la.add('e')
la.add('d')
la.add('c')
la.add('b')
la.add('a')
```

[10]:
```
print(la)
```

```
LinkedList: a,b,c,d,e,f,g
```

Creates a **NEW** `LinkedList` copying nodes from index `2` INCLUDED up to index `5` EXCLUDED:

[11]:
```
lb = la.slice(2,5)
```

[12]:
```
print(lb)
```

```
LinkedList: c,d,e
```

Note original `LinkedList` is still intact:

[13]:
```
print(la)
```

```
LinkedList: a,b,c,d,e,f,g
```

Special cases

If `start` is greater or equal then `end`, you get an empty `LinkedList`:

[14]:
```
print(la.slice(5,3))
```

```
LinkedList:
```

If `start` is greater than available nodes, you get an empty `LinkedList`:

[15]:
```
print(la.slice(10,15))
```

```
LinkedList:
```

If `end` is greater than the available nodes, you get a copy of all the nodes until the tail without errors:

[16]:
```
print(la.slice(3,10))
```

```
LinkedList: d,e,f,g
```

Using negative indexes for either `start` , `end` or both raises `ValueError`:

```
la.slice(-3,4)
```

(continues on next page)

(continued from previous page)

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-184-e3380bb66e77> in <module>()
----> 1 la.slice(-3,4)

~/Da/prj/sciprog-ds/prj/exams/2020-06-16/linked_list_sol.py in slice(self, start, end)
     63
     64         if start < 0:
---> 65             raise ValueError('Negative values for start are not supported! %s
↪' % start)
     66         if end < 0:
     67             raise ValueError('Negative values for end are not supported: %s'
↪% end)

ValueError: Negative values for start are not supported! -3
```

```
la.slice(1,-2)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-185-8e09ec468c30> in <module>()
----> 1 la.slice(1,-2)

~/Da/prj/sciprog-ds/prj/exams/2020-06-16/linked_list_sol.py in slice(self, start, end)
     65             raise ValueError('Negative values for start are not supported! %s
↪' % start)
     66         if end < 0:
---> 67             raise ValueError('Negative values for end are not supported: %s'
↪% end)
     68
     69         ret = LinkedList()

ValueError: Negative values for end are not supported: -2
```

B3 BinaryTree prune_rec

Implement the method `prune_rec`:

```
def prune_rec(self, el):
    """ MODIFIES the tree by cutting all the subtrees that have their
        root node data equal to el. By 'cutting' we mean they are no longer linked
        by the tree on which prune is called.

        - if prune is called on a node having data equal to el, raises ValueError

        - MUST execute in O(n) where n is the number of nodes of the tree
        - NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.PruneRecTest`

Example:

[17]:
```
from bin_tree_sol import *
from bin_tree_test import bt
```

[18]:
```
t = bt('a',
            bt('b',
                   bt('z'),
                   bt('c',
                         bt('d'),
                         bt('z',
                                 None,
                                 bt('e'))))),
            bt('z',
                   bt('f'),
                   bt('z',
                           None,
                           bt('g'))))
```

[19]:
```
print(t)
```

```
a
├b
│├z
│└c
│ ├d
│ └z
│  ├
│  └e
└z
 ├f
 └z
  ├
  └g
```

[20]:
```
t.prune_rec('z')
```

[21]:
```
print(t)
```

```
a
├b
│├
│└c
│ ├d
│ └
└
```

[22]:
```
t.prune_rec('c')
```

[23]:
```
print(t)
```

```
a
├b
└
```

Trying to prune the root will throw a `ValueError`:

```
t.prune_rec('a')

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-f8e8fa8a97dd> in <module>()
----> 1 t.prune_rec('a')

ValueError: Tried to prune the tree root !
```

[ ]:

```
```

2.1.15 Exam - Friday 17, July 2020 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solutions

Introduction

- **Taking part to this exam erases any vote you had before**

Grading

- **Correct implementations**: Correct implementations with the required complexity grant you full grade.
- **Partial implementations**: Partial implementations *might* still give you a few points. If you just can't solve an exercise, try to solve it at least for some subcase (i.e. array of fixed size 2) commenting why you did so.

Valid code

WARNING: MAKE SURE ALL EXERCISE FILES AT LEAST COMPILE !!! 10 MINS BEFORE THE END OF THE EXAM I WILL ASK YOU TO DO A FINAL CLEAN UP OF THE CODE

WARNING: *ONLY* IMPLEMENTATIONS OF THE PROVIDED FUNCTION SIGNATURES WILL BE EVALUATED !!!!!!!!!

For example, if you are given to implement:

```
def f(x):
    raise Exception("TODO implement me")
```

and you ship this code:

```
def my_f(x):
    # a super fast, correct and stylish implementation
```

(continues on next page)

(continued from previous page)

```
def f(x):
    raise Exception("TODO implement me")
```

We will assess only the latter one `f(x)`, and conclude it doesn't work at all :P !!!!!!!

Helper functions

Still, you are allowed to define any extra helper function you might need. If your `f(x)` implementation calls some other function you defined like `my_f` here, it is ok:

```
# Not called by f, will get ignored:
def my_g(x):
    # bla

# Called by f, will be graded:
def my_f(y,z):
    # bla

def f(x):
    my_f(x,5)
```

How to edit and run

To edit the files, you can use any editor of your choice, you can find them under *Applications->Programming*:

- **Visual Studio Code**
- Editra is easy to use, you can find it under *Applications->Programming->Editra*.
- Others could be *GEdit* (simpler), or *PyCharm* (more complex).

To run the tests, use **the Terminal** which can be found in *Accessories -> Terminal*

IMPORTANT: Pay close attention to the comments of the functions.

WARNING: *DON'T* modify function signatures! Just provide the implementation.

WARNING: DON'T change the existing test methods, just add new ones !!! You can add as many as you want.

WARNING: *DON'T* create other files. If you still do it, they won't be evaluated.

Debugging

If you need to print some debugging information, you are allowed to put extra print statements in the function bodies.

WARNING: even if print statements are allowed, be careful with prints that might break your function!

For example, avoid stuff like this:

```
x = 0
print(1/x)
```

What to do

1) Download `sciprog-ds-2020-07-17-exam.zip` and extract it on your desktop. Folder content should be like this:

```
sciprog-ds-2020-07-17-FIRSTNAME-LASTNAME-ID
   exam-2020-07-17.ipynb
   theory.txt
   office_queue_exercise.py
   office_queue_test.py
```

2) Rename `sciprog-ds-2020-07-17-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2020-07-17-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[122]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - NACE codes

https://ec.europa.eu/eurostat/ramon/nomenclatures/index.cfm?TargetUrl=LST_CLS_DLD&StrNom=NACE_REV2&StrLanguageCode=EN&StrLayoutCode=HIERARCHIC#

So you want to be a data scientist. Good, plenty of oopportunities 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 burocratic 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[123], which is then further specialized by various states (for example Italy's catalog is called ATECO[124])

[122] http://examina.icts.unitn.it/studente
[123] https://ec.europa.eu/eurostat/ramon/nomenclatures/index.cfm?TargetUrl=LST_NOM_DTL&StrNom=NACE_REV2&StrLanguageCode=EN&IntPcKey=&StrLayoutCode=HIERARCHIC
[124] https://www.istat.it/it/archivio/17888

Sections

A NACE code is subdivided in a hierarchical, four-level structure. The categories at the highest level are called *sections*, here they are:

Detail	
+ A	AGRICULTURE, FORESTRY AND FISHING Detail
+ B	MINING AND QUARRYING Detail
+ C	MANUFACTURING Detail
+ D	ELECTRICITY, GAS, STEAM AND AIR CONDITIONING SUPPLY Detail
+ E	WATER SUPPLY; SEWERAGE, WASTE MANAGEMENT AND REMEDIATION ACTIVITIES Detail
+ F	CONSTRUCTION Detail
+ G	WHOLESALE AND RETAIL TRADE; REPAIR OF MOTOR VEHICLES AND MOTORCYCLES Detail
+ H	TRANSPORTATION AND STORAGE Detail
+ I	ACCOMMODATION AND FOOD SERVICE ACTIVITIES Detail
+ J	INFORMATION AND COMMUNICATION Detail
+ K	FINANCIAL AND INSURANCE ACTIVITIES Detail
+ L	REAL ESTATE ACTIVITIES Detail
+ M	PROFESSIONAL, SCIENTIFIC AND TECHNICAL ACTIVITIES Detail
+ N	ADMINISTRATIVE AND SUPPORT SERVICE ACTIVITIES Detail
+ O	PUBLIC ADMINISTRATION AND DEFENCE; COMPULSORY SOCIAL SECURITY Detail
+ P	EDUCATION Detail
+ Q	HUMAN HEALTH AND SOCIAL WORK ACTIVITIES Detail
+ R	ARTS, ENTERTAINMENT AND RECREATION Detail
+ S	OTHER SERVICE ACTIVITIES Detail
+ T	ACTIVITIES OF HOUSEHOLDS AS EMPLOYERS; UNDIFFERENTIATED GOODS- AND SERVICES-PRODUCING ACTIVITIES OF HOUSEHOLDS FOR OWN USE Detail
+ U	ACTIVITIES OF EXTRATERRITORIAL ORGANISATIONS AND BODIES Detail

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	**Section**	**M**	a single alphabetic char	PROFESSIONAL, SCIENTIFIC AND TECHNICAL ACTIVITIES
2	**Division**	**73**	two-digits	Advertising and market research
3	**Group**	**73.1**	three-digits, with dot after first two	Advertising
4	**Class**	**73.12**	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**: for part A you will **NOT** 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', -1)
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):
Order                       996 non-null int64
Level                       996 non-null int64
Code                        996 non-null object
Parent                      975 non-null object
Description                 996 non-null object
This item includes          778 non-null object
This item also includes     202 non-null object
Rulings                     134 non-null object
This item excludes          507 non-null object
Reference to ISIC Rev. 4    996 non-null object
dtypes: int64(2), object(8)
memory usage: 77.9+ KB
```

[2]:
```
df.head(5)
```

[2]:
```
    Order  Level   Code Parent  \
0  398481  1      A      NaN
1  398482  2      01     A
2  398483  3      01.1   01
3  398484  4      01.11  01.1
4  398485  4      01.12  01.1

                                                        Description  \
0  AGRICULTURE, FORESTRY AND FISHING
1  Crop and animal production, hunting and related service activities
2  Growing of non-perennial crops
3  Growing of cereals (except rice), leguminous crops and oil seeds
4  Growing of rice

                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                
↪                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                This item includes  \
0  This section includes the exploitation of vegetal and animal natural resources,
↪comprising the activities of growing of crops, raising and breeding of animals,
↪harvesting of timber and other plants, animals or animal products from a farm or
↪their natural habitats.
1  This division includes two basic activities, namely the production of crop
↪products and production of animal products, covering also the forms of organic
↪agriculture, the growing of genetically modified crops and the raising of
↪genetically modified animals. This division includes growing of crops in open
↪fields as well in greenhouses.\n \nGroup 01.5 (Mixed farming) breaks with the usual
↪principles for identifying main activity. It accepts that many agricultural
↪holdings have reasonably balanced crop and animal production, and that it would be
↪arbitrary to classify them in one category or the other.
2  This group includes the growing of non-perennial crops, i.e. plants that do not
↪last for more than two growing seasons. Included is the growing of these plants for
↪the purpose of seed production.
3  This class includes all forms of growing of cereals, leguminous crops and oil
↪seeds in open fields. The growing of these crops is often combined within
↪agricultural units.\n\nThis class includes:\n- growing of cereals such as:\n  .
↪wheat\n  . grain maize\n  . sorghum\n  . barley\n  . rye\n  . oats\n  . millets\n  .
↪ other cereals n.e.c.\n- growing of leguminous crops such as:\n  . beans\n  . broad
↪beans\n  . chick peas\n  . cow peas\n  . lentils\n  . lupines\n  . peas\n  . pigeon
↪peas\n  . other leguminous crops\n- growing of oil seeds such as:\n  . soya beans\n
↪ . groundnuts\n  . castor bean\n  . linseed\n  . mustard seed\n  . niger seed\n  .
↪rapeseed\n  . safflower seed\n  . sesame seed\n  . sunflower seed\n  . other oil
↪seeds
4  This class includes:\n- growing of rice (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.
```

(continues on next page)

(continued from previous page)

```
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.\n\nThe 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.
2  NaN
3  This class excludes:\n- growing of rice, see 01.12\n- growing of sweet corn, see␣
↪01.13\n- growing of maize for fodder, see 01.19\n- growing of oleaginous fruits,␣
↪see 01.26
4  NaN

  Reference to ISIC Rev. 4
0  A
1  01
2  011
3  0111
4  0112
```

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',}))
```

```
<pandas.io.formats.style.Styler at 0x7f2cb7c8e588>
```

A1 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.

A1.1 is_nace

Implement following function. NOTE: it was not explicitly required in the original exam but could help detecting words.

[4]:
```
def is_nace(word):
    """Given a word, RETURN True if the word is a NACE code, else otherwise"""
    #jupman-raise
    # 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:].isdigit()
    else:
        return False
    #/jupman-raise

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
```

A1.2 extract_codes

Implement following function which extracts codes from `This item excludes` column cells. For examples, see asserts.

[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
    """
    #jupman-raise
    ret = []

    words = [word.strip(';,.:()"\'') for word in text.replace('-',' ').split()]
```

(continues on next page)

(continued from previous page)

```
    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
    #/jupman-raise

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
```

(continues on next page)

(continued from previous page)

```
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)
"""
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']
```

A2 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

```
[23]: def build_db(filepath):
          #jupman-raise
```

(continues on next page)

(continued from previous page)

```
    ret = {}
    import csv
    with open(filepath, encoding='utf-8', newline='') as f:
        my_reader = csv.DictReader(f, delimiter=',')
        for d in my_reader:
            diz = {'description' : d['Description'],
                   'exclusions' : extract_codes(d['This item excludes'])}
            ret[d['Code']] = diz
    return ret
    #/jupman-raise

activities_db = build_db('NACE_REV2_20200628_213139.csv')
#activities_db
```

A3 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 file expected_db.py, to import it write: `from expected_db import activities_db`

[8]:
```
%matplotlib inline
def plot(db, level):

    import matplotlib.pyplot as plt
    #jupman-raise

    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','␣
↪of\n')

    plt.xticks(xs, ['NACE ' + c[0] + '\n' + fix_label(db[c[0]]['description']) for c␣
↪in coords])
```

(continues on next page)

(continued from previous page)

```
    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.savefig('division-exclusions-solution.png')
    plt.show()

    #/jupman-raise

#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)
```

Part B

B1 Theory

Write the solution in separate `theory.txt` **file**

B1.1 complexity

Given a list `L` of `n` elements, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

```
def my_fun(L):
    n = len(L)
    if n <= 1:
        return 1
    else:
        L1 = L[0:n//2]
        L2 = L[n//2:]
        a = my_fun(L1) + min(L1) - n
        b = my_fun(L2) + min(L2) - n
        return a + b
```

B1.2 describe

Briefly describe what a hash table is and provide an example of its usage.

B2 - OfficeQueue

An office offers services `'x'`, `'y'` and `'z'`. When people arrive at the office, they state which service they need, get a ticket and enqueue. Suppose at the beginning of the day we are considering there is only one queue.

The office knows on average how much time each service requires:

[9]:
```
SERVICES = { 'x':5,    # minutes
             'y':20,
             'z':30
           }
```

With this information it is able to inform new clients approximately how long they will need to wait.

`OfficeQueue` is implemented as a linked list, where people enter the queue from the tail and leave from the head. We can represent it like this (**NOTE**: 'cumulative wait' is not actually stored in the queue):

```
wait time: 155 minutes

cumulative wait:  5    10   15   45   50   55   85   105  110  130  150  155
wait times:       5    5    5    30   5    5    30   20   5    20   20   5
                  x    x    x    z    x    x    z    y    x    y    y    x
                  a -> b -> c -> d -> e -> f -> g -> h -> i -> l -> m -> n
                  ^                                                      ^
                  |                                                      |
                 head                                                   tail
```

Each node holds the client identifier `'a'`, `'b'`, `'c'`, and the service label (like `'x'`) requested by the client:

```
class Node:
    def __init__(self, initdata, service):
        self._data = initdata
        self._service = service
        self._next = None
```

`OfficeQueue` keeps fields `_services`, `_size` and a field `_wait_time` which holds the total wait time of the queue:

```
class OfficeQueue:
    def __init__(self, services):
        self._head = None
        self._tail = None
        self._size = 0
        self._wait_time = 0
        self._services = dict(services)
```

[10]:
```
from office_queue_sol import *
SERVICES = { 'x':5,    # minutes
             'y':20,
             'z':30
           }


oq = OfficeQueue(SERVICES)
print(oq)
```

```
OfficeQueue:
```

[11]:
```
oq.enqueue('a','x')
oq.enqueue('b','x')
oq.enqueue('c','x')
oq.enqueue('d','z')
oq.enqueue('e','x')
oq.enqueue('f','x')
oq.enqueue('g','z')
oq.enqueue('h','y')
oq.enqueue('i','x')
oq.enqueue('l','y')
oq.enqueue('m','y')
oq.enqueue('n','x')
```

[12]:
```
print(oq)
```

```
OfficeQueue:
  x    x    x    z    x    x    z    y    x    y    y    x
  a -> b -> c -> d -> e -> f -> g -> h -> i -> l -> m -> n
```

[13]:
```
oq.size()
```

[13]:
```
12
```

Total wait time can be accessed from outside with the method `wait_time()`:

[14]:
```
oq.wait_time()
```

[14]:
```
155
```

> **ATTENTION**: you only need to implement the methods `time_to_service` and `split`
>
> **DO NOT** touch other methods.

B2.1 - time_to_service

Open file `office_queue_exercise.py` with and start editing.

In order to schedule work and pauses, for each service office employees want to know after how long they will have to process the first client requiring that particular service.

First service encountered will always have a zero time interval (in this example it's `x`):

```
wait time: 155

cumulative wait:  5    10   15   45   50   55   85   105  110  130  150  155
wait times:       5    5    5    30   5    5    30   20   5    20   20   5
                  x    x    x    z    x    x    z    y    x    y    y    x
                  a -> b -> c -> d -> e -> f -> g -> h -> i -> l -> m -> n
                 ||              |                   |
                 x : 0           |                   |
                 |               |                   |
                 |---------------|                   |
                 |     z : 15                        |
                 |                                   |
                 |-----------------------------------|
                                  y : 85
```

[15]:
```
SERVICES = { 'x':5,    # minutes
             'y':20,
             'z':30
           }

oq = OfficeQueue(SERVICES)
print(oq)
```

```
OfficeQueue:
```

[16]:
```
oq.enqueue('a','x')
oq.enqueue('b','x')
oq.enqueue('c','x')
oq.enqueue('d','z')
oq.enqueue('e','x')
oq.enqueue('f','x')
oq.enqueue('g','z')
oq.enqueue('h','y')
oq.enqueue('i','x')
oq.enqueue('l','y')
oq.enqueue('m','y')
oq.enqueue('n','x')

print(oq)
```

```
OfficeQueue:
  x    x    x    z    x    x    z    y    x    y    y    x
  a -> b -> c -> d -> e -> f -> g -> h -> i -> l -> m -> n
```

Method to implement will return a dictionary mapping each service to the time interval after which the service is first required:

[17]:
```
oq.time_to_service()
```

[17]:
```
{'x': 0, 'y': 85, 'z': 15}
```

Services not required by any client

As a special case, if a service is not required by any client, its time interval is set to the queue total wait time (because a client requiring that service might still show up in the future and get enqueued)

[18]:
```
oq = OfficeQueue(SERVICES)
oq.enqueue('a','x')    # completed after 5 mins
oq.enqueue('b','y')    # completed after 5 + 20 mins
print(oq)
```

```
OfficeQueue:
  x    y
  a -> b
```

[19]:
```
print(oq.wait_time())
```

```
25
```

[20]:
```
oq.time_to_service()    # note z is set to total wait time
```

[20]:
```
{'x': 0, 'y': 5, 'z': 25}
```

Now implement this:

```
def time_to_service(self):
    """ RETURN a dictionary mapping each service to the time interval after which
        the service is first required.

        - the first service encountered will always have a zero time interval
        - If a service is not required by any client, time interval is set to
          the queue total wait time
        - MUST run in O(n) where n is the size of the queue.
    """
```

Testing: `python3 -m unittest office_queue_test.TestTimeToService`

B2.2 split

Suppose a new desk is opened: to reduce waiting times the office will comunicate on a screen to some people in the current queue to move to the new desk, thereby creating a new queue. The current queue will be split in two according to this criteria: after the cut, the total waiting time of the current queue should be the same or slightly bigger than the waiting time in the new queue:

> **ATTENTION**: This example is **different** from previous one (total wait time is 150 instead of 155)

```
ORIGINAL QUEUE:

wait time = 150 minutes
wait time / 2 = 75 minutes


cumulative wait:  30   50   80   110  115  120  140  145  150
wait times:       30   20   30   30   5    5    20   5    5
                  z    y    z    z    x    x    y    x    x
                  a -> b -> c -> d -> e -> f -> g -> h -> i
                  ^            ^                         ^
                  |            |                         |
                 head       cut here                    tail


MODIFIED QUEUE:

wait time: 80 minutes

wait times:       30   20   30
cumulative wait:  30   50   80
                  z    y    z
                  a -> b -> c
                  ^         ^
                  |         |
                 head      tail
```

(continues on next page)

(continued from previous page)

```
NEW QUEUE:

wait time: 75 minutes

wait times:       30   5    5    20   5    5
cumulative wait:  30   35   40   60   65   70
                  z    x    x    y    x    x
                  d -> e -> f -> g -> h -> i
                  ^                        ^
                  |                        |
                 head                     tail
```

Implement this method:

```
def split(self):
    """ Perform two operations:
        - MODIFY the queue by cutting it so that the wait time of this cut
          will be half (or slightly more) of wait time for the whole original queue
        - RETURN a NEW queue holding remaining nodes after the cut - the wait time of
          new queue will be half (or slightly less) than original wait time

        - If queue to split is empty or has only one element, modify nothing
          and RETURN a NEW empty queue
        - After the call, present queue wait time should be equal or slightly bigger
          than returned queue.
        - DO *NOT* create new nodes, just reuse existing ones
        - REMEMBER to set _size, _wait_time, _tail in both original and new queue
        - MUST execute in O(n) where n is the size of the queue
    """
```

Testing: `python3 -m unittest office_queue_test.SplitTest`

```
[ ]:
```

2.1.16 Exam - Monday 24, August 2020 - solutions

Scientific Programming - Data Science @ University of Trento

Download exercises and solutions

Introduction

- **Taking part to this exam erases any vote you had before**

What to do

1) Download `sciprog-ds-2020-08-24-exam.zip` and extract it on your desktop. Folder content should be like this:

2) Rename `sciprog-ds-2020-08-24-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2020-08-24-john-doe-432432`

From now on, you will be editing the files in that folder. At the end of the exam, that is what will be evaluated.

3) Edit the files following the instructions in this worksheet for each exercise. Every exercise should take max 25 mins. If it takes longer, leave it and try another exercise.

4) When done:

- if you have unitn login: zip and send to examina.icts.unitn.it/studente[125]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - Prezzario

Open Jupyter and start editing this notebook `exam-2020-08-24.ipynb`

You are going to analyze the dataset `EPPAT-2018-new-compact.csv`, which is the price list for all products and services the Autonomous Province of Trento may require. Source: dati.trentino.it[126]

> **DO NOT WASTE TIME LOOKING AT THE WHOLE DATASET!**
>
> The dataset is quite complex, please focus on the few examples we provide

We will show examples with pandas, but it is not required to solve the exercises.

[1]:
```
import pandas as pd
import numpy as np

pd.set_option('display.max_colwidth', -1)

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]
```

[2]:
```
     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 ELETTROGENO
```

(continues on next page)

[125] http://examina.icts.unitn.it/studente
[126] https://dati.trentino.it/dataset/prezziario-dei-lavori-pubblici-della-provincia-autonoma-di-trento

(continued from previous page)

```
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
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 like the following (rows 3 to 8 included):

[3]:
```
df[3:12]
```

[3]:
```
     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

[22]:
```
import math


#use this list to rmeove unneeded stuff
PUNCTUATION=[',','-','.','%']
UNTIL = 'fino'
BEYOND = 'oltre'

def extract_bounds(text):
    #jupman-raise

    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
```

(continues on next page)

(continued from previous page)

```
            left = 0

    return (left,right)
    #/jupman-raise

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**
- **implement the parsing with a** `csv.DictReader`, see example[127]
- **as encoding, use** `latin-1`

[5]:
```
# Suppose we want to get all info about A.02.40.0010 prefix:
df[3:12]
```

[5]:

	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

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'},
```

(continues on next page)

[127] https://sciprog.davidleoni.it/formats/formats-sol.html#Reading-as-dictionaries

(continued from previous page)

```
            {'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.

[6]:
```
import csv
from pprint import pprint

def extract_product(filename, code, measure_unit):
    #jupman-raise

    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
    #/jupman-raise

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'},
```

(continues on next page)

(continued from previous page)

```
            {'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'}]}
```

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**.

[7]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def plot_product(product):
    #jupman-raise

    models = product['models']
    xs = np.arange(len(models))
    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.savefig('pompa-a-motore-solution.png')
    plt.show()
    #/jupman-raise


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', '%')
```

(continues on next page)

(continued from previous page)

```
plot_product(product)
```

Part B

B1 Theory

Write the solution in separate ``theory.txt`` file

B1.1 complexity

Given a list `L` of `n` elements, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

```
def my_fun(L):
    n = len(L)
    tmp = []
    for i in range(int(n)):
        tmp.insert(0,L[i]-L[int(n/3)])
    return sum(tmp)
```

B1.2 describe

Briefly describe what a graph is and the two classic ways that can be used to represent it as a data structure.

B2 couple_sort

Open a text editor and edit file `linked_list.py`. Implement this method:

```
def couple_sort(self):
        """MODIFIES the linked list by considering couples of nodes at *even* indexes
           and their successors: if a node data is lower than its successor data,␣
↪swaps
           the nodes *data*.

           - ONLY swap *data*, DO NOT change node links.
           - if linked list has odd size, simply ignore the exceeding node.
           - MUST execute in O(n), where n is the size of the list
        """
```

Testing: `python3 -m unittest linked_list_Test.CoupleSortTest`

Example:

[8]:
```
from linked_list_sol import *
from linked_list_test import to_ll
```

[9]:
```

ll = to_ll([4,3,5,2,6,7,6,3,2,4,5,3,2])
```

[10]:
```
print(ll)
```

```
LinkedList: 4,3,5,2,6,7,6,3,2,4,5,3,2
```

[11]:
```
ll.couple_sort()
```

[12]:
```
print(ll)
```

```
LinkedList: 3,4,2,5,6,7,3,6,2,4,3,5,2
```

Notice it sorted each couple at even positions. This particular linked list has odd size (13 items), so last item 2 was not considered.

B3 schedule_rec

Suppose the nodes of a binary tree represent tasks (nodes data is the task label). Each task may have up to two subtasks, represented by its children. To be declared as completed, each task requires first the completion of all of its subtasks.

We want to create a schedule of tasks, so that to declare completed the task at the root of the tree, before all tasks below it must be completed, specifically first the tasks on the left side, and then the tasks on the right side. If you apply this reasoning recursively, you can obtain a schedule of tasks to be executed.

Open `bin_tree.py` and implement this method:

```
def schedule_rec(self):
    """ RETURN a list of task labels in the order they will be completed.

        - Implement it with recursive calls.
        - MUST run in O(n) where n is the size of the tree

        NOTE: with big trees a recursive solution would surely
              exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.ScheduleRecTest`

Example:

For this tree, it should return the schedule `['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']`

Here we show code execution with the same tree:

[13]:
```
from bin_tree_sol import *
from bin_tree_test import bt
```

[17]:
```
tasks = bt('i',
                bt('d',
                        bt('b',
                                bt('a')),
                        bt('c')),
                bt('h',
                        bt('f',
                                None,
                                bt('e')),
                        bt('g')))
```

[18]:
```
print(tasks)
```

```
i
├d
│├b
││├a
││└
│└c
└h
 ├f
 │├
 │└e
 └g
```

[15]:
```
tasks.schedule_rec()
```

[15]:
```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

2.2 2017-18 (QCB)

See QCB master past exams on sciprolab2.readthedocs.io[128]

NOTE: Those exams are useful, but for you there will be:

- no biological examples
- less dynamic programming
- more exercises on graphs & matrices
- exercise on pandas
- custom DiGraph won't have Visit and VertexLog classes

2.3 2016-17 (QCB)

See davidleoni.github.io/algolab/past-exams.html[129]

WARNING: keep in mind that 2016-17 exams are for Python 2 - in this course we use Python 3

[ ]:
```

```

[128] https://sciprolab2.readthedocs.io/intro.html#Past-exams
[129] https://davidleoni.github.io/algolab/past-exams.html

CHAPTER THREE

SLIDES 2020/21

old 2019/20 slides

TBD

```
[ ]:
```

CHAPTER

FOUR

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, bad things shall happen.

COMMANDMENT 1: You shall test!

To run tests, enter the following command in the terminal:

Windows Anaconda:

```
python -m unittest my-file
```

Linux/Mac: remember the three after python command:

```
python3 -m unittest my-file
```

WARNING: In the call above, DON'T append the extension .py to my-file

WARNING: Still, on the hard-disk the file MUST be named with a *.py* at the end, like *my-file.py*

WARNING: If strange errors occur, make sure to be using python version **3**. Just run the interpreter and it will display the current version.

COMMANDMENT 2: You shall also write on paper!

If staring at the monitor doesn't work, help yourself and draw a representation of the state sof the program. Tables, nodes, arrows, all can help figuring out a solution for the problem.

COMMANDMENT 3: You shall copy exactly the same function definitions as in the exercises!

For example don't write :

```
def MY_selection_sort(A):
```

COMMANDMENT 4: You shall never ever reassign function parameters

```
def myfun(i, s, L, D):

    # You shall not do any of such evil, no matter what the type of the parameter is:
    i = 666             # basic types (int, float, ...)
    s = "evil"           # strings
    L = [666]           # containers
    D = {"evil":666}    # dictionaries

    # For the sole case of composite parameters like lists or dictionaries,
    # you can write stuff like this IF AND ONLY IF the function specification
    # requires you to modify the parameter internal elements (i.e. sorting a list
    # or changing a dictionary field):

    L[4] = 2              # list
    D["my field"] = 5     # dictionary
    C.my_field = 7        # class
```

COMMANDMENT 5: You shall never ever reassign self:

Never ever write horrors such as:

```
class MyClass
    def my_method(self, x, y):
        self = {a:666}  # since self is a kind of dictionary, you might be tempted to␣
↪do like this
                        # but to the outside world this will bring no effect.
                        # For example, let's say somebody from outside makes a call␣
↪like this:
                        #    mc = MyClass()
                        #    mc.my_method()
                        # after the call mc will not point to {a:666}
        self = ['evil']  # self is only supposed to be a sort of dictionary and␣
↪passed from outside
        self = 6        # self is only supposed to be a sort of dictionary and passed␣
↪from outside
```

COMMANDMENT 6: You shall never ever assign values to function nor method calls

WRONG WRONG:

```
my_fun() = 666
my_fun() = 'evil'
my_fun() = [666]
```

CORRECT:

With the assignment operator we want to store in the left side a value from the right side, so all of these are valid operations:

```
x = 5
y = my_fun()
z = []
z[0] = 7
d = dict()
d["a"] = 6
```

Function calls such as `my_fun()` return instead results of calculations in a box that is created just for the purpose of the call and Python will just not allow us to reuse it as a variable. So whenever you see 'name()' at the left side, it *can't* be possibly followed by one equality = sign (but it can be followed by two equality signs == if you are performing a comparison).

COMMANDMENT 7: You shall use return command only if you see written "return" in the 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.

COMMANDMENT 8: You shall never ever redefine system functions

Python has system defined function, 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:

```
[1]: list("ciao")
```

```
[1]: ['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 you own miserable purposes:

```
[2]: 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")

TypeError: 'list' object is not callable
```

COMMANDMENT 9: Whenever you introduce a variable in a cycle, such variable must be new

If you read carefully Commandment 4 you should not need to be reminded of this Commandment, nevertheless it is always worth restating the Right Way.

If you defined a variable before, you shall not reintroduce it in a `for`, since it is as confusing as reassigning function parameters.

So avoid these sins:

[3]:
```
i = 7
for i in range(3):  # sin, you lose i variable
    print(i)
```

```
0
1
2
```

[4]:
```
def f(i):
    for i in range(3): # sin again, you lose i parameter
        print(i)
```

[5]:
```
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
```

CHAPTER

FIVE

CHAPTER

SIX

PART A

6.1 Introduction solutions

6.1.1 Download exercises zip

Browse files online[130]

In this practical we will set up a working Python3 development environment and will start familiarizing a bit with Python.

There are many ways to write and execute Python code:

- Python tutor (online, visual debugger)
- Python interpreter (command line)
- Visual Studio Code (editor, good debugger)
- Jupyter (notebook)
- Google Colab (online, collaborative)

During this lab we see all of them and familiarize with the exercises format. For now ignore the exercises zip and proceed reading.

6.1.2 Installation

You will need to install several pieces of software to get a working programming environment. In this section we will install everything that we are going to need in the next few weeks.

Python3 is available for Windows, Mac and Linux. Python3 alone is often not enough, and you will need to install extra system-specific libraries + editors like Visual Studio Code and Jupyter.

Windows/Mac installation

To avoid hassles, especially on Win / Mac you should install some so called *package manager* (Linux distributions already come with a package manager). Among the many options for this course we use the package manager Anaconda for Python 3.7.

1. Install Anaconda for Python 3.7[131] (anaconda installer will ask you to install also visual studio code, so accept the kind offer)

[130] https://github.com/DavidLeoni/sciprog-ds/tree/master/introduction
[131] https://www.anaconda.com/download/

2. If you didn't in the previous point, install now Visual Studio Code, which is available for all platforms. You can read about it here[132]. Downloads for all platforms can be found here[133]

Linux installation

Although you can install Anaconda on Linux, it is usually better to use the system package manager that comes with your distribution.

1. Check the Python interpreter - most probably you already have one in your distribution, but you have to check it is the right version. In this course we will use python version 3.x. Open a terminal and try typing in:

```
python3
```

if you get an error like "python3 command not found" , try typing

```
python
```

if you get something like this (mind the version 3):

```
biancol@bluhp:~$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

you are already sorted, just type `Ctrl-D` to exit. If it doesn't work, try typing `exit()` and hit Enter

Otherwise you need to install Python 3.

Linux, debian-like(e.g. Ubuntu)

Issue the following commands on a terminal:

```
sudo apt-get update
```

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

Linux Fedora:

Issue the following commands on a terminal:

```
sudo dnf install python3
```

2. Install now the package manager `pip`, which is a very convenient tool to install python packages, with the following command (**on Fedora the command above should have already installed it**):

   ```
   sudo apt-get install python3-pip
   ```

   > **Note:**
   >
   > If pip is already installed in your system you will get a message like: python3-pip is already the newest version (3.x.y)

3. Install Jupyter notebook:

[132] https://code.visualstudio.com/
[133] https://code.visualstudio.com/Download

Open the system console and copy and paste this command:

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

It will install jupyter in your user home.

4. Finally, install Visual Studio Code. You can read about it here[134]. Downloads for all platforms can be found here[135].

6.1.3 Python tutor

Let's meet Python on the web with Python Tutor[136] is a great way to visualize Python code.

Use it as much as possible! . It really provides great guidance about how things are working under the hood.

By default works for standard Python code. If you want to use it also with code from modules (i.e. numpy) you have to select `Write code in Python3 with Anaconda (experimental)`

6.1.4 System console

Let's look at the operating system console. In Anaconda installations you *must* open it with Anaconda Prompt (if you have a Mac but not Anaconda, open the Terminal). We assume Linux users can get around their way.

> **WARNING**: In the *system* console we are entering commands *for the operating system*, using the *system command language* which varies for each operating system. So following commands are not Python !

- to see files of the folder you are in you can type `dir` in windows and `ls` in Mac/Linux
- to enter a folder: `cd MYFOLDER`
- to leave a folder: `cd ..`
  - mind the space between cd and two dots

6.1.5 Python interpreter

To start the Python interpreter, from system console run (to open it see previous paragraph)

```
python
```

You will see the python interpreter (the one with >>>), where you can directly issue commands and see the output. If for some reason it doesn't work, try running

```
python3
```

> **WARNING**: you must be running Python 3, in this course we only use that version ! Please check you are indeed using version 3 by looking at the interpreter banner, it should read something similar to this:

[134] https://code.visualstudio.com/
[135] https://code.visualstudio.com/Download
[136] http://pythontutor.com/visualize.html

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

WARNING: if you take random code from the internet, be sure it is for Python **3**

WARNING: the >>> is there just to tell you you are looking at python interpreter. It is *not* python code ! If you find written >>> in some code example , do not copy it !

Now we are all set to start interacting with the Python interpreter. First make sure you are inside the interpreter (you should see a >>> in the console, if not see previous paragraph), then type in the following instructions:

[2]:
```
5 + 3
```

[2]:
```
8
```

All as expected. The "In [1]" line is the input, while the "Out [1]" reports the output of the interpreter. Let's challenge python with some other operations:

[3]:
```
12 / 5
```

[3]:
```
2.4
```

[4]:
```
1/133
```

[4]:
```
0.007518796992481203
```

[5]:
```
2**1000
```

[5]:
```
107150860718626732094842504906000181056140481170553360744375038837035105112493612249319837881569585812
```

And some assignments:

[6]:
```
a = 10
b = 7
s = a + b
d = a / b

print("sum is:",s, " division is:",d)
```

```
sum is: 17  division is: 1.4285714285714286
```

In the first four lines, values have been assigned to variables through the = operator. In the last line, the print function is used to display the output. For the time being, we will skip all the details and just notice that the print function somehow managed to get text and variables in input and coherently merged them in an output text. Although quite useful in some occasions, the console is quite limited therefore you can close it for now. To exit type Ctrl-D or exit().

6.1.6 Visual Studio Code

Visual Studio Code[137] is an Integrated Development Editor (IDE) for text files. It can handle many languages, Python included (python programs are text files ending in `.py`).

Features:

- open source
- lightweight
- used by many developers
- Python plugin is not the best, but works enough for us

Once you open the IDE Visual Studio Code you will see the welcome screen:

You can find useful information on this tool here[138]. Please spend some time having a look at that page.

Once you are done with it you can close this window pressing on the "x". First thing to do is to set the python interpreter to use. Click on View –> Command Palette and type "Python" in the text search space. Select **Python: Select Workspace Interpreter** as shown in the picture below.

[137] https://code.visualstudio.com/
[138] https://code.visualstudio.com/docs#vscode

Finally, select the python version you want to use (e.g. Python3).

Now you can click on **Open Folder** to create a new folder to place all the scripts you are going to create. You can call it something like "exercises". Next you can create a new file, *example1.py* (*.py* extension stands for python).

Visual Studio Code will understand that you are writing Python code and will help you with valid syntax for your program.

Warning:

If you get the following error message:

click on **Install Pylint** which is a useful tool to help your coding experience.

Add the following text to your *example1.py* file.

[7]:
```
"""
This is the first example of Python script.
"""
a = 10 # variable a
b = 33 # variable b
c = a / b # variable c holds the ratio

# Let's print the result to screen.
print("a:", a, " b:", b, " a/b=", c)
```

```
a: 10  b: 33  a/b= 0.30303030303030304
```

A couple of things worth nothing. The first three lines opened and closed by `"""` are some text describing the content of the script. Moreover, comments are proceeded by the hash key (#) and they are just ignored by the python interpreter. **Please remember to comment your code, as it helps readability and will make your life easier when you have to modify or just understand the code you wrote some time in the past.**

Please notice that Visual Studio Code will help you writing your Python scripts. For example, when you start writing

the **print** line it will complete the code for you (**if the Pylint extension mentioned above is installed**), suggesting the functions that match the letters written. This useful feature is called *code completion* and, alongside suggesting possible matches, it also visualizes a description of the function and parameters it needs. Here is an example:

Save the file (Ctrl+S as shortcut). It is convenient to ask the IDE to highlight potential *syntactic* problems found in the code. You can toggle this function on/off by clicking on **View –> Problems**. The *Problems* panel should look like this

Visual Studio Code is warning us that the variable names *a,b,c* at lines 4,5,6 do not follow Python naming conventions for constants. This is because they have been defined at the top level (there is no structure to our script yet) and therefore are interpreted as constants. The naming convention for constants states that they should be in capital letters. To amend the code, you can just replace all the names with the corresponding capitalized name (i.e. A,B,C). If you do that, and you save the file again (Ctrl+S), you will see all these problems disappearing as well as the green underlining of the variable names. If your code does not have an empty line before the end, you might get another warning "*Final new line missing*". Note that these were just warnings and the interpreter **in this case** will happily and correctly execute the code anyway, but it is always good practice to understand what the warnings are telling us before deciding to ignore them!

Had we by mistake mispelled the *print* function name (something that should not happen with the code completion tool that suggests functions names!) writing *printt* (note the double t), upon saving the file, the IDE would have underlined in red the function name and flagged it up as a problem.

This is because the builtin function *printt* does not exist and the python interpreter does not know what to do when it reads it. Note that *printt* is actually underlined in red, meaning that there is an error which will cause the interpreter to stop the execution with a failure. **Please remember that before running any piece of code all errors must be fixed**.

Now it is time to execute the code. By right-clicking in the code panel and selecting **Run Python File in Terminal** (see picture below) you can execute the code you have just written.

Upon clicking on *Run Python File in Terminal* a terminal panel should pop up in the lower section of the coding panel and the result shown above should be reported.

Saving script files like the *example1.py* above is also handy because they can be invoked several times (later on we will learn how to get inputs from the command line to make them more useful…). To do so, you just need to call the python intepreter passing the script file as parameter. From the folder containing the *example1.py* script:

```
python3 example1.py
```

will in fact return:

a: 10 b: 33 a/b= 0.30303030303030304

Before ending this section, let me add another note on errors. The IDE will diligently point you out *syntactic* warnings and errors (i.e. errors/warnings concerning the structure of the written code like name of functions, number and type of parameters, etc.) but it will not detect *semantic* or *runtime* errors (i.e. connected to the meaning of your code or to the value of your variables). These sort of errors will most probably make your code crash or may result in unexpected results/behaviours. In the next section we will introduce the debugger, which is a useful tool to help detecting these errors.

Before getting into that, consider the following lines of code (do not focus on the *import* line, this is only to load the mathematics module and use its method *sqrt*):

[8]:
```
"""
Runtime error example, compute square root of numbers
"""
import math

A = 16
B = math.sqrt(A)
C = 5*B
print("A:", A, " B:", B, " C:", C)

#D = math.sqrt(A-C) # whoops, A-C is now -4!!!
#print(D)
```

```
A: 16  B: 4.0  C: 20.0
```

If you add that code to a python file (e.g. sqrt_example.py), you save it and you try to execute it, you should get an error message as reported above. You can see that the interpreter has happily printed off the vaule of A,B and C but then stumbled into an error at line 9 (math domain error) when trying to compute $\sqrt{A-C} = \sqrt{-4}$, because the sqrt method of the math module cannot be applied to negative values (i.e. it works in the domain of real numbers).

Please take some time to familiarize with Visual Studio Code (creating files, saving files etc.) as in the next practicals we will take this ability for granted.

6.1.7 The debugger

Another important feature of advanced Integrated Development Environments (IDEs) is their debugging capabilities. Visual Studio Code comes with a debugging tool that can help you trace the execution of your code and understand where possible errors hide.

Write the following code on a new file (let's call it *integer_sum.py*) and execute it to get the result.

[9]:
```
""" integer_sum.py is a script to
 compute the sum of the first 1200 integers. """

S = 0
for i in range(0, 1201):
    S = S + i
```

(continues on next page)

(continued from previous page)

```
print("The sum of the first 1200 integers is: ", S)
```

```
The sum of the first 1200 integers is:  720600
```

Without getting into too many details, the code you just wrote starts initializing a variable S to zero, and then loops from 0 to 1200 assigning each time the value to a variable i, accumulating the sum of S + i in the variable S. **A final thing to notice is indentation.** In Python it is important to indent the code properly as this provides the right scope for variables (e.g. see that the line *S = S + 1* starts more to the right than the previous and following line – this is because it is inside the for loop). You do not have to worry about this for the time being, we will get to this in a later practical…

How does this code work? How does the value of S and i change as the code is executed? These are questions that can be answered by the debugger.

To start the debugger, click on **Debug –> Start Debugging** (shortcut F5). The following small panel should pop up:

We will use it shortly, but before that, let's focus on what we want to track. On the left hand side of the main panel, a *Watch* panel appeared. This is where we need to add the things we want to monitor as the execution of the program goes. With respect to the code written above, we are interested in keeping an eye on the variables S, i and also of the expression S+i (that will give us the value of S of the next iteration). Add these three expressions in the watch panel (click on **+** to add new expressions). The watch panel should look like this:

do not worry about the message "*name X is not defined*", this is normal as no execution has taken place yet and the interpreter still does not know the value of these expressions.

The final thing before starting to debug is to set some breakpoints, places where the execution will stop so that we can check the value of the watched expressions. This can be done by hovering with the mouse on the left of the line number. A small reddish dot should appear, place the mouse over the correct line (e.g. the line corresponding to *S = S + 1* and click to add the breakpoint (a red dot should appear once you click).

Now we are ready to start debugging the code. Click on the green triangle on the small debug panel and you will see that the yellow arrow moved to the breakpoint and that the watch panel updated the value of all our expressions.

The value of all expressions is zero because the debugger stopped **before** executing the code specified at the breakpoint line (recall that S is initialized to 0 and that i will range from 0 to 1200). If you click again on the green arrow, execution will continue until the next breakpoint (we are in a for loop, so this will be again the same line - trust me for the time being).

Now i has been increased to 1, S is still 0 (remember that the execution stopped **before** executing the code at the breakpoint) and therefore S + i is now 1. Click one more time on the green arrow and values should update accordingly (i.e. S to 1, i to 2 and S + i to 3), another round of execution should update S to 3, i to 3 and S + i to 6. Got how this works? Variable i is increased by one each time, while S increases by i. You can go on for a few more iterations and see if this makes any sense to you, once you are done with debugging you can stop the execution by pressing the red square on the small debug panel.

Please take some more time to familiarize with Visual Studio Code (creating files, saving files, interacting with the debugger etc.) as in the next practicals we will take this ability for granted. Once you are done you can move on and do the following exercises.

6.1.8 Jupyter

Jupyter is a handy program to write notebooks organized in cells (files with `.ipynb` extension), where there is both code, output of running that code and text. The code by default is Python, but can also be other languages like R). The text is formatted with Markdown language[139] - see cheatsheet[140]. It's becoming the de-facto standard for writing technical documentation (you can find everywhere, i.e. blogs).

Run Jupyter

Jupyter is a web server, so when you run it, a Jupyter server starts and you should see a system console opening (on Anaconda system you might see it for a very short time), afterwards an internet browser should open. Since Jupyter is a server, what you see in the browser is just the UI which is connecting to the server.

If you have Anaconda :

Launch Anaconda Navigator, and then search and run Jupyter.`

If you don't have Anaconda:

From system console try to run

```
jupyter notebook
```

or, as alternative if the previous doesn't work:

```
python3 -m notebook
```

Editing notebooks

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 aftwerwards, press `Alt + Enter`

Some tips:

- when something seem wrong in computations, try to clean memory by running `Kernel->Restart and Run all`
- when you see an asterisk to the side of a cell, maybe the computationg has hanged (an infinite while?). To solve the problem, run `Kernel->shutdown` and then `Kernel -> restart

[139] https://en.wikipedia.org/wiki/Markdown
[140] https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Browsing notebooks

(Optional) To improve your browsing experience, you might wish to install some Jupyter extension , like `toc2` which shows paragraphs headers on the sidebar. To install it:

Install the Jupyter contrib extensions[141] package:

If you have Anaconda:

Open Anaconda Prompt, and type:

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

If you don't have Anaconda:

1. Open a Terminal and type:

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

2. Install it in Jupyter:

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

3. Enable extensions

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

Once you installed: To see tocs when in a document you will need to press a list button at the right-end of the toolbar.

6.1.9 Course exercise formats

In this course, you will find the solutions to the exercises on the website. At the top page of each solution, you will find the link to download a zip like this:

Download exercises zip

Browse files online[142]

- now unzip exercises in a folder, you should get something like this:

```
-jupman.py
-sciprog.py
-other stuff ...
-exercises
     |- introduction
         |- introduction.ipynb
         |- introduction-sol.ipynb
         |- other stuff ..
```

WARNING 1: to correctly visualize the notebook, it MUST be in an unzipped folder !

[141] https://github.com/ipython-contrib/jupyter_contrib_nbextensions
[142] https://github.com/DavidLeoni/sciprog-ds/tree/master/introduction

Each zip contains both the exercises to do as files to edit, along with their solution in a separate file.

Some exercises will need to be done in Jupyter notebooks (`.ipynb` files), while others in plain `.py` Python files.

- 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 `introduction/introduction.ipynb`

> **WARNING 2**: DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

- now look into the exercise notebook, it should begin with a cell like this:

```
#Please execute this cell
import sys;
sys.path.append('../');
import jupman;
import sciprog;
```

This is because some code is common to all exercises. In particular:

- in `jupman.py` there is code for special cell outputs in Jupyter notebooks (like Python tutor or unit tests display)
- in `sciprog.py` there are common algorithms and data structures used in the course

A notebook always looks for modules in the current directory of the notebook. Since `jupman.py` stays a parent directory in the zip, with the lines

```
import sys;
sys.path.append('../');
```

we tell Python to also look modules (= python .py files) in a directory which is two parent folders above the current one.

It is not the most elegant way to locate modules but gets around the quirks of Jupyter fine enough for our purposes.

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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

Python Tutor inside Jupyter

We implemented a command `jupman.pytut()` to show a Python tutor debugger in a Python notebook. Let's see how it works.

You can put a call to `jupman.pytut()` at the end of a cell, and the cell code will magically appear in python tutor in the output (except the call to pytut() of course).

> **ATTENTION**: To see Python tutor you need to be online!
>
> For this to work you need to be online both when you execute the cell and when visiting the built website.

[10]:
```
x = 5
y= 7
z = x + y

jupman.pytut()
```

[10]:
```
<IPython.lib.display.IFrame at 0x7f1ed40a1860>
```

Beware of variables which were initialized in previous cells which won't be available in Python Tutor, like `w` in this case:

[11]:
```
w = 8
```

[12]:
```
x =  w + 5
jupman.pytut()
```

[12]:
```
<IPython.lib.display.IFrame at 0x7f1ecd71a1d0>
```

6.1.10 Exercises

Try to familiarize yourself with Jupyter and Visual Studio Code by doing these exercises in both of them.

1. Compute the area of a triangle having base 120 units (B) and height 33 (H). Assign the result to a variable named area and print it.

[13]:
```
# SOLUTION

B = 120
H = 33
Area = B*H/H
print("Triangle area is:", Area)
```

```
Triangle area is: 120.0
```

2. Compute the area of a square having side (S) equal to 145 units. Assign the result to a variable named area and print it.

[14]:
```
S = 145
Area = S**2
print("Square area is:",Area)
```

```
Square area is: 21025
```

3. Modify the program at point 2. to acquire the side S from the user at runtime. (Hint: *use the input function and remember to convert the acquired value into an int*).

ANSWER:

```
print("Insert size: ")
S = int(input())
Area = S**2
print("Square area is:",Area)
```

4. If you have not done so already, put the two previous scripts in two separate files (e.g. `triangle_area.py` and `square_area.py` and execute them from the terminal).

5. Write a small script (`trapezoid.py`) that computes the area of a trapezoid having major base (MB) equal to 30 units, minor base (mb) equal to 12 and height (H) equal to 17. Print the resulting area. Try executing the script from inside Visual Studio Code and from the terminal.

[15]:
```
# SOLUTION

"""trapezoid.py"""
MB = 30
mb = 12
H = 17
Area = (MB + mb)*H/2
print("Trapezoid area is: ", Area)
```

```
Trapezoid area is:  357.0
```

6. Rewrite the example of the sum of the first 1200 integers by using the following equation: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

[16]:
```
# SOLUTION

N = 1200

print("Sum of first 1200 integers: ", N*(N+1)/2)
```

```
Sum of first 1200 integers:  720600.0
```

7. Modify the program at point 6. to make it acquire the number of integers to sum N from the user at runtime.

ANSWER:

```
print("Input number N:")
N = int(input())
print("Sum of first ", N, " integers: ", N*(N+1)/2)
```

8. Write a small script to compute the length of the hypotenuse (c) of a right triangle having sides a=133 and b=72 units (see picture below). Hint: *remember the Pythagorean theorem and use math.sqrt*).

[17]:
```
# SOLUTION

import math

a = 133
b = 72

c = math.sqrt(a**2 + b**2)

print("Hypotenuse: ", c)
```

```
Hypotenuse:  151.23822268196622
```

6.2 Python basics solutions

6.2.1 Download exercises zip

Browse files online[143]

References

- Andrea Passerini, Lecture A01[144]

In this practical we will start interacting more with Python, practicing on how to handle data, functions and methods. We will see some built-in data types (integers, floats, booleans - we will reserve strings for later)

6.2.2 Modules

Python modules are simply text files having the extension **.py** (e.g. `exercise.py`). When you were writing the code in the IDE in the previous practical, you were in fact implementing the corresponding module.

As said in the previous practical, once you implemented and saved the code of the module, you can execute it by typing

```
python3 exercise1.py
```

or, in Visual Studio Code, by right clicking on the code panel and selecting **Run Python File in Terminal**.

A Module `A` can be loaded from another module `B` so that `B` can use the functions defined in `A`. Remember when we used the `sqrt` function? It is defined in the module `math`. To import it and use it we indeed wrote something like:

[2]:
```
import math

x = math.sqrt(4)
print(x)
```

```
2.0
```

When importing modules we do not need to specify the extension `.py` of the file.

6.2.3 Objects

Python understands very well objects, and in fact everything is an object in Python. Objects have **properties** (characteristic features) and **methods** (things they can do). For example, an object *car* has the *properties* model, make, color, number of doors etc., and the *methods* steer right, steer left, accelerate, break, stop, change gear,… According to Python's official documentation:

"Objects are Python's abstraction for data. All data in a Python program is represented by objects or by relations between objects."

All you need to know for now is that in Python objects have an **identifier (ID)** (i.e. their name), a **type** (numbers, text, collections,…) and a **value** (the actual data represented by the objects). Once an object has been created the *identifier* and the *type* never change, while its *value* can either change (**mutable objects**) or stay constant (**immutable objects**).

Python provides these built-in data types:

[143] https://github.com/DavidLeoni/sciprog-ds/tree/master/basics
[144] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A01-introduction.pdf

Type	Meaning	Domain	Mutable?
`bool`	Condition	`True`, `False`	No
`int`	Integer	~~$\{-2^{-63}, \ldots, 2^{63}-1\}$~~ $\mathbb{Z}$	No
~~`long`~~	~~Integer~~	~~$\mathbb{Z}$~~	~~No~~
`long`	Integer	$\mathbb{Z}$	No
`float`	Rational	$\mathbb{Q}$ (more or less)	No
`str`	Text	Text	No
`list`	Sequence	Collections of things	Yes
`tuple`	Sequence	Collections of things	No
`dict`	Map	Maps between things	Yes

We will stick with the simplest ones for now, but later on we will dive deeper into the all of them.

6.2.4 Variables

Variables are just references to objects, in other words they are the **name** given to an object. Variables can be **assigned** to objects by using the assignment operator =.

The instruction

[3]:
```
sides = 4
```

might represent the number of sides of a square. What happens when we execute it in Python? An object is created, it is given an identifier, its type is set to "int" (an integer number), it value to 4 and a name *sides* is placed in the current namespace to point to that object, so that after that instruction we can access that object through its name. The type of an object can be accessed with the function **type**() and the identifier with the function **id**():

[4]:
```
sides = 4
print( type(sides) )
print( id(sides) )
```

```
<class 'int'>
94241937814656
```

Consider now the following code:

[5]:
```
sides = 4  # a square
print ("value:", sides, " type:", type(sides), " id:", id(sides))
sides = 5  # a pentagon
print ("value:", sides, " type:", type(sides), " id:", id(sides))
```

```
value: 4  type: <class 'int'>  id: 94241937814656
value: 5  type: <class 'int'>  id: 94241937814688
```

The value of the variable sides has been changed from 4 to 5, but as stated in the table above, the type `int` is **immutable**. Luckily, this did not prevent us to change the value of sides from 4 to 5. What happened behind the scenes when we executed the instruction sides = 5 is that a new object has been created of type int (5 is still an integer) and it has been made accessible with the same name *sides*, but since it is a different object (i.e. the integer 5) you can see that the identifier is

actually different. Note: you do not have to really worry about what happens behind the scenes, as the Python interpreter will take care of these aspects for you, but it is nice to know what it does.

You can even change the type of a variable during execution but that is normally a **bad idea** as it makes understanding the code more complicated.

You can do (**but, please, refrain!**):

[6]:
```
sides = 4 # a square
print ("value:", sides, " type:", type(sides), " id:", id(sides))
sides = "four" #the sides in text format
print ("value:", sides, " type:", type(sides), " id:", id(sides))
```

```
value: 4  type: <class 'int'>  id: 94241937814656
value: four  type: <class 'str'>  id: 140613404719232
```

IMPORTANT NOTE: You can chose the name that you like for your variables (I 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. Names cannot be equal to reserved keywords:
4. variable names should start with a lowercase letter

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		

Exercise: variable names

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: DO UNDERSTAND THE *VERY IMPORTANT DIFFERENCE* BETWEEN THIS AND FOLLOWING TWOs !!!
15. `asino`
16. `As`
17. `lista` PLEASE: DO UNDERSTAND THE *VERY IMPORTANT DIFFERENCE* BETWEEN THIS AND FOLLOWING TWOs !!!
18. `list` DO NOT EVEN TRY TO ASSIGN THIS ONE IN THE INTERPRETER (like `list = 5`), IF YOU DO YOU 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`
27. `aren'tyouboredyet`
28. `<angular>`

[7]:
```
# write here
```

6.2.5 Numeric types

We already mentioned that numbers are **immutable objects**. Python provides different numeric types: integers, reals (floats), booleans and even complex numbers and fractions (but we will not get into those).

Integers

Their range of values is limited only by the memory available. As we have already seen, python provides also a set of standard operators to work with numbers:

[8]:
```
a = 7
b = 4

a + b  # 11
a - b  # 3
a // b # integer division: 1
a * b  # 28
a ** b # power: 2401
```

(continues on next page)

(continued from previous page)

```
a / b  # division 0.8333333333333334
type(a / b)
```

[8]:
```
float
```

Note that in the latter case the result is no more an integer, but a float (we will get to that later).

6.2.6 Booleans

These objects are used for the boolean algebra and have type `bool`.

Truth values are represented with the keywords `True` and `False` in Python, a boolean object can only have value `True` or `False`.

[9]:
```
x = True
```

[10]:
```
x
```

[10]:
```
True
```

[11]:
```
type(x)
```

[11]:
```
bool
```

[12]:
```
y = False
```

[13]:
```
type(y)
```

[13]:
```
bool
```

Boolean operators

We can operate on boolean values with the boolean operators `not`, `and`, `or`. Recall boolean algebra for their use:

[14]:
```
print("not True: ", not True)    # False
print("not False: ", not False) # True
print()
print("False and False: ", False and False) # False
print("False and True:  ", False and True )  # False
print("True  and False: ", True and False)   # False
print("True  and True:  ", True and True)     # True
print()
print("False or  False: ", False or False)     # False
print("False or  True:  ", False or True)     # True
print("True  or  False: ", True or False)     # True
print("True  or  True:  ", True or True)       # True
```

```
not True:  False
not False:  True

False and False:  False
False and True:   False
True  and False:  False
```

(continues on next page)

(continued from previous page)

```
True  and True:   True

False or  False:  False
False or  True:   True
True  or  False:  True
True  or  True:   True
```

Booleans exercise: constants

Try to guess the result of these boolean expressions (*first* guess, and *then* try it out !!)

```
not (True and False)
(not True) or (not (True or False))
not (not True)
not (True and (False or True))
not (not (not False))
True and (not (not((not False) and True)))
False or (False or ((True and True) and (True and False)))
```

Booleans exercise: variables

For which values of `x` and `y` these expressions give `True` ? Try to think the answer before trying it !!!!

NOTE: there can be more combinations that produce `True`, try to find all of them.

```
x or (not x)
(not x) and (not y)
x and (y or y)
x and (not y)
(not x) or  y
y or not (y and x)
x and ((not x) or not(y))
(not (not x)) and not (x and y)
x and (x or (not(x) or not(not(x or not (x)))))
```

For which values of `x`, `y` and `z` these expressions give `False` ?

NOTE: there can be more combinations that produce `False`, try to find all of them.

```
x or ((not y) or z)
x or (not y) or (not z)
not (x and y and (not z))
not (x and (not y) and (x or z))
y or ((x or y) and (not z))
```

Boolean conversion

We can convert booleans into integers with the builtin function `int`. Any integer can be converted into a boolean (and vice-versa) with `bool`:

[15]:
```
a = bool(1)
b = bool(0)
c = bool(72)
d = bool(-5)
t = int(True)
f = int(False)

print("a: ", a)
print("b: ", b)
print("c: ", c)
print("d: ", d)
print("t: ", t)
print("f: ", f)
```

```
a:  True
b:  False
c:  True
d:  True
t:  1
f:  0
```

Any integer is evaluated to `True`, except `0`. Note that, the truth values `True` and `False` respectively behave like the integers `1` and `0`.

Booleans exercise: what is a boolean?

Read carefully previous description of booleans, and try to guess the result of following expressions.

```
bool(True)
bool(False)
bool(2 + 4)
bool(4-3-1)
int(4-3-1)
True + True
True + False
True - True
True * True
```

6.2.7 Numeric operators

Numeric comparators are operators that return a boolean value. Here are some examples (from the lecture):

a == b	True if and only if $a = b$
a != b	True if and only if $a \neq b$
a < b	True if and only if $a < b$
a > b	True if and only if $a > b$
a <= b	True if and only if $a \leq b$
a >= b	True if and only if $a \geq b$

Example: Given a variable `a = 10` and a variable `b = 77`, let's swap their values (i.e. at the end `a` will be equal to `77` and `b` to `10`). Let's also check the values at the beginning and at the end.

[16]:
```
a = 10
b = 77
print("a: ", a, " b:", b)
print("is a equal to 10?", a == 10)
print("is b equal to 77?", b == 77)

TMP = b  # we need to store the value of b safely
b = a    # ok, the old value of b is gone... is it?
a = TMP  # a gets the old value of b... :-)

print()
print("a: ", a, " b:", b)
print("is a equal to 10?", a == 10)
print("is a equal to 77?", a == 77)
print("is b equal to 10?", b == 10)
print("is b equal to 77?", b == 77)
```

```
a:  10  b: 77
is a equal to 10? True
is b equal to 77? True

a:  77  b: 10
is a equal to 10? False
is a equal to 77? True
is b equal to 10? True
is b equal to 77? False
```

Numeric operators exercise: cycling

Write a program that given three variables with numebers `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
#
#
#
```

[17]:

```
a = 4
b = 7
c = 9

# write code here


print(a)
print(b)
print(c)
```

```
4
7
9
```

[18]:

```
# 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
```

(continues on next page)

(continued from previous page)

```
c = t

print(a)
print(b)
print(c)
```

```
9
4
7
```

[19]:
```
# SOLUTION
```

6.2.8 Real numbers

Python stores real numbers (floating point numbers) in 64 bits of information divided in sign, exponent and mantissa.

Exercise: Let's calculate the area of the center circle of a football pitch (radius = 9.15m) recalling that $area = Pi * R^2$ (as power operator, use `**`):

[20]:
```
# SOLUTION

R = 9.15
Pi = 3.1415926536
Area = Pi*(R**2)
print(Area)
```

```
263.02199094102605
```

Note that the parenthesis around the `R**2` are not necessary as operator `**` has the precedence, but I personally think it helps readability.

Here is a reminder of the precedence of operators:

`**`	Power (Highest precedence)
`+,-`	Unary plus and minus
`* / // %`	Multiply, divide, floor division, modulo
`+ -`	Addition and subtraction
`<= < > >=`	Comparison operators
`== !=`	Equality operators
`not or and`	Logical operators (Lowest precedence)

Example: Let's compute the GC content of a DNA sequence 33 base pairs long, having 12 As, 9 Ts, 5 Cs and 7Gs. The GC content can be expressed by the formula: $gc = \frac{G+C}{A+T+C+G}$ where A,T,C,G represent the number of nucleotides of each kind. What is the AT content? Is the GC content higher than the AT content ?

[21]:
```
A = 12
T = 9
C = 5
```

(continues on next page)

(continued from previous page)

```
G = 7

gc = (G+C)/(A+T+C+G)

print("The GC content is: ", gc)

at = 1 - gc

print("Is the GC content higher than AT content? ", gc > at)
```

```
The GC content is:  0.36363636363636365
Is the GC content higher than AT content?  False
```

Real numbers exercise: quadratic equation

Calculate the zeros of the equation $ax^2 - b = 0$ where `a = 10` and `b = 1`. Hint: use `math.sqrt` or `** 0.5`. Finally check that substituting the obtained value of `x` in the equation gives zero.

[22]:
```
# SOLUTION

import math

A = 10
B = 1

x = math.sqrt(B/A)

print("10x**2 - 1 = 0 for x:", x)
print("Is x a solution?", 10*x**2 -1 == 0)
```

```
10x**2 - 1 = 0 for x: 0.31622776601683794
Is x a solution? True
```

6.3 Strings solutions

6.3.1 Download exercises zip

Browse files online[145]

6.3.2 What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-my_lib.py
-other stuff ...
-exercises
     |- lists
```

(continues on next page)

[145] https://github.com/DavidLeoni/sciprog-ds/tree/master/strings

(continued from previous page)

```
        |- strings.ipynb
        |- strings-sol.ipynb
        |- other stuff ..
```

WARNING 1: 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 `strings/strings.ipynb`

WARNING 2: DO NOT use the *Upload* button in Jupyter, instead navigate to the unzipped folder while in Jupyter browser!

- Go on reading that notebook, and follow instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.3.3 Introduction

References:

- Andrea Passerini slides A02 - datastructures[146]
- Think Python, Chapter 8, Strings[147]
- Think Python, Chapter 9, word play[148]
- *extra* if you want to do *text mining*:
  - Read before first part on Unicode encoding from chapter Strings in book Dive into Python[149]
  - Look at NLTK library[150]

Strings are **immutable objects** (note the actual type is **str**) used by python to handle text data. Strings are sequences of *unicode code points* that can represent characters, but also formatting information (e.g. '\n' for new line). Unlike other programming languages, python does not have the data type character, which is represented as a string of length 1.

There are several ways to define a string:

[1]:
```
S = "my first string, in double quotes"

S1 = 'my second string, in single quotes'

S2 = '''my third string is
```

(continues on next page)

[146] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A02-datastructures.pdf
[147] http://greenteapress.com/thinkpython2/html/thinkpython2009.html
[148] http://greenteapress.com/thinkpython2/html/thinkpython2010.html
[149] https://diveintopython3.problemsolving.io/strings.html
[150] https://www.nltk.org/

(continued from previous page)

```
in triple quotes
therefore it can span several lines'''

S3 = """my fourth string, in triple double-quotes
can also span
several lines"""

print(S, '\n') #let's add a new line at the end of the string with \n
print(S1,'\n')
print(S2, '\n')
print(S3, '\n')
```

```
my first string, in double quotes

my second string, in single quotes

my third string is
in triple quotes
therefore it can span several lines

my fourth string, in triple double-quotes
can also span
several lines
```

To put special characters like ‘,” and so on you need to “escape them” (i.e. write them following a back-slash).

\\	Backslash
\n	ASCII linefeed (also known as newline)
\t	ASCII tab character
\’	Single quote
\"	Double quote
\xxxx	Unicode character xxxx (hexadecimal)

Example: Let’s print a string containing a quote and double quote (i.e. ‘ and “).

[2]:
```
myString = "This is how I \'quote\' and \"double quote\" things in strings"
print(myString)
```

```
This is how I 'quote' and "double quote" things in strings
```

Strings can be converted to and from numbers with the functions `str()`, `int()` or `float()`.

Example: Let’s define a string *myString* with the value “47001” and convert it into an `int`. Try adding one and print the result.

[3]:
```
my_string = "47001"
print(my_string, " has type ", type(my_string))

my_int = int(my_string)

print(my_int, " has type ", type(my_int))

my_int = my_int + 7   #adds seven
```

(continues on next page)

(continued from previous page)

```
my_string = my_string + "7" # cannot add 7 (we need to use a string).
                            # This will append 7 at the end of the string

#my_string = my_string + 7 # CANNOT DO THIS, python will complain about concatenating␣
↪a stirng to a different type,
                           #                in this case an int

my_string = my_string + str(7) # this works, I have to force before the conversion of␣
↪inter to string.


print(my_int)
print(my_string)
```

```
47001  has type  <class 'str'>
47001  has type  <class 'int'>
47008
4700177
```

Python defines some operators to work with strings. Recall the slides shown during the lecture:

Result	Operator	Meaning
int	len(str)	Return the length of the string
str	str + str	Concatenate two strings
str	str * int	Replicate the string
bool	str in str	Check if a string is present in another string
str	str[int]	Read the character at specified index
str	str[int:int]	Extract a sub-string

Exercise: many hello

Look at the table above. Given the string `x = "hello"`, print a string with `"hello"` repeated 5 times: `"hellohellohellohellohello"`. Your code must work with any string stored in the variable `x`

[4]:
```
x = "hello"

# write here

print("hello"*5)
```

```
hellohellohellohellohello
```

Exercise: interleave terns

Given two strings which both have length `3`, print a string which interleaves characters from both strings. Your code should work for any string of such lenght.

Example:

Given

```
x="say"
y="hi!"
```

should print

```
shaiy!
```

[5]:
```
# write here

x="say"
y="hi!"
print(x[0] + y[0] + x[1] + y[1] + x[2] + y[2])
```

```
shaiy!
```

Exercise: print length

Write some code that given a string `x`, prints the content of the string followed by its length. Your code should work for any content of the variable `x`.

Example:

Given

```
x = 'howdy'
```

should print

```
howdy5
```

[6]:
```
# write here

x = 'howdy'
print(x + str(len(x)))
```

```
howdy5
```

Exercise: both contained

You are given two strings `x` and `y`, and a third one `z`. Write some code that prints `True` if `x` and `y` are contained in `z`.

For example,

Given

```
x = 'cad'
y = 'ra'
z = 'abracadabra'
```

it should print

```
True
```

```
x = 'zam'
y = 'ra'
z = 'abracadabra'
```

it should print

```
False
```

[7]:
```
# write here

x = 'cad'
y = 'ra'
z = 'abracadabra'


print((x in z) and (y in z))
```

```
True
```

Slicing

We can access strings at specific positions (indexing) or get a substring starting from a position S to a position E. The only thing to remember is that numbering starts from 0. The `i`-th character of a string can be accessed as `str[i-1]`. Substrings can be accessed as `str[S:E]`, optionally a third parameter can be specified to set the step (i.e. `str[S:E:STEP]`).

> **Important note.** Remember that when you do str[S:E], **S is inclusive, while E is exclusive** (see S[0:6] below).

Let's see these aspects in action with an example:

[8]:
```
S = "Luther College"

print(S) #print the whole string
print(S == S[:]) #a fancy way of making a copy of the original string
print(S[0]) #first character
print(S[3]) #fourth character
print(S[-1]) #last character
print(S[0:6]) #first six characters
print(S[-7:]) #final seven characters
print(S[0:len(S):2]) #every other character starting from the first
print(S[1:len(S):2]) #every other character starting from the second
```

```
Luther College
True
L
h
e
Luther
College
Lte olg
uhrClee
```

Exercise: garalampog

Write some code to extract and print `alam` from the string `"garalampog"`. Try to correctly guess indeces.

[9]:
```
x = "garalampog"
# write here

#       0123456789
print(x[3:7])
```

```
alam
```

Exercise: ifE:nbsphinx-math:*te*:nbsphinx-math:`nfav `lkD lkWe

Write some code to extract and print `kS` from the string `"ifE\te\nfav lkD lkWe"`. Mind the spaces and special characters (you might want to print `x` first). Try to correctly guess indeces.

[10]:
```
x = "ifE\te\nfav  lkD lkWe"

# write here

#      0123 45 67890123456789
#x = "ifE\te\nfav  lkD lkWe"

print(x[12:14])
```

```
kD
```

Exercise: javarnanda

Given a string `x`, write some code to extract and print its last 3 characters and join them to the first 3. Code should work for any string of length at least 3.

Example:

Given

```
x = "javarnanda"
```

it should print

```
javnda
```

Given

```
x = "abcd"
```

it should print

```
abcbcd
```

[11]:
```
# write here

x = "abcd"
print(x[:3] + x[-3:])
```

```
abcbcd
```

Methods for the str object

The object `str` has some methods that can be applied to it (remember methods are things you can do on objects). Recall from the lecture that the main methods are:

Result	Method	Meaning
str	str.upper()	Return the string in upper case
str	str.lower()	Return the string in lower case
str	str.strip(str)	Remove strings from the sides
str	str.lstrip(str)	Remove strings from the left
str	str.rstrip(str)	Remove strings from the right
str	str.replace(str, str)	Replace substrings
bool	str.startswith(str)	Check if the string starts with another
bool	str.endswith(str)	Check if the string ends with another
int	str.find(str)	Return the first position of a substring starting from the left
int	str.rfind(str)	Return the position of a substring starting from the right
int	str.count(str)	Count the number of occurrences of a substring

ATTENTION: Since Strings are immutable, every operation that changes the string actually produces a new *str* object having the modified string as value.

Example

[12]:
```
my_string = "ciao"

anotherstring = my_string.upper()

print(anotherstring)
```

```
CIAO
```

[13]:
```
print(my_string)  # didn't change
```

```
ciao
```

If you are unsure about a method (for example `strip`), you can ask python help like this:

NOTICE there are no round parenthesis after the method !!!

[14]:
```
help("ciao".strip)
```

```
Help on built-in function strip:

strip(…) method of builtins.str instance
    S.strip([chars]) -> str
```

(continues on next page)

(continued from previous page)

```
    Return a copy of the string S with leading and trailing
    whitespace removed.
    If chars is given and not None, remove characters in chars instead.
```

Exercise substitute

Given a string `x`, write some code to print a string like `x` but with all occurrences of `bab` substituted by `dada`

Example:

Given

```
x = 'kljsfsdbabòkkrbabej'
```

it should print

```
kljsfsddadaòkkrdadaej
```

[15]:
```
# write here

x = 'kljsfsdbabòkkrbabej'
print(x.replace('bab', 'dada'))
```

```
kljsfsddadaòkkrdadaej
```

Exercise hatespace

Given a string `x` which may contain blanks (spaces, special controls characters such as `\t` and `n`, …) at the beginning and end, write some code that prints the string without the blanks and the strings `START` and `END` at the extremities.

Example:

Given

```
x = ' \t  \n \n hatespace\n   \t \n'
```

prints

```
STARThatespaceEND
```

[16]:
```
# write here

x = ' \t  \n \n hatespace\n   \t \n'

print('START' + x.strip() + 'END')
```

```
STARThatespaceEND
```

6.3.4 Exercises with functions

ATTENTION

Following exercises require you to know:

Complex statements: Andrea Passerini slides A03[151]

Functions: Andrea Passerini slides A04[152]

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
```

[17]:
```
# write here

# version with len, faster because python with a string always mantains 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
```

contains

⊛ Write the function `contains(word, character)`, which RETURN `True` is the string contains the given character, otherwise RETURN `False`

- Use `in` operator

```
>>> x = contains('ciao', 'a')
>>> x
True
```

(continues on next page)

[151] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[152] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf

(continued from previous page)

```
>>> y = contains('ciao', 'z')
>>> y
False
```

[18]:
```
# write here

def contains(word, character):
    return character in word
```

invertilet

⊛ Write the function `invertilet(first, second)` which takes in input two strings of length greater than 3, and RETURN a nnew string in which the words are concataned and separated by a space, the last two characters in the words are inverted. For example, if you pass in input `'ciao'` and `'world'`, the function should RETURN `'ciad worlo'`

If the two strings are not of adequate length, the program PRINTS *error!*

HINT: use *slices*

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 not actually good practice, here we use it as 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[153]

```
>>> x = invertilet("ciao", "world")
>>> x
'ciad worlo'
>>> x = invertilet('hi','mondo')
'errore!'
>>> x
None
>>> x = invertilet('cirippo', 'bla')
'errore!'
>>> x
None
```

[19]:
```
# write here

def invertilet(first,second):
    if len(first) <= 3 or len(second) <=3:
        print("errore!")
    else:
        return first[:-1] + second[-1] + " " + second[:-1] + first[-1]
```

[153] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Unforeseen-situations

nspace

⊛ Write the function `nspace` that given a string `s` in input, RETURN a new string in which the `n`-character is a space.

For example, given the string `'largamente'` and the index `5`, the program should RETURN the string `'larga ente'`. NOTE: if the number is too big (for example, the word has 6 characters and you pass the number 9), the program PRINTS *error!*.

NOTE 1: if the number is too big (for example, the word has 6 character and you pass the number 9), the program PRINTS *error!*.

NOTE 2: PRINTing is different from RETURNing !!! Whatever gets printed is shown to the user but Python cannot reuse it for other calculations.

NOTE 3: Resorting to prints on error conditions is not actually a good practice, here we use it as 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[154]

```
>>> x = nspazio('largamente', 5)
>>> x
'larga ente'

>>> x = nspazio('ciao', 9)
errore!
>>> x
None

>>> x = nspazio('ciao', 4)
errore!
>>> x
None
```

[20]:
```
# write here

def nspace(word, index):
    if index >= len(word):
        print("error!")
    return word[:index] + ' ' + word[index+1:]

#nspace("largamente", 5)
```

startend

⊛ Write a Python program which takes a string `s`, and if it has a length greater than 4, the program PRINTS the first and last two characters, otherwise, PRINTS `I want at least 4 characters`. For example, by passing `"ciaomondo"`, the function should print `"cido"`. By passing `"ciao"` it should print `ciao` and by passing `"hi"` it should print `I want at least 4 characters`.

```
>>> startend('ciaomondo')
cido

>>> startend('hi')
Voglio almeno 4 caratteri
```

[154] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Unforeseen-situations

[21]:
```
# write here

def startend(s):
    if len(s) >= 4:
        print(s[:2] + s[-2:])
    else:
        print("I want at least 4 characters")
```

swap

Write a function that given a string, swaps the first and last character and PRINTS the result.

For example, given the string `"world"`, the program will PRINT `'oondm'`

```
>>> swap('mondo')
oondm
```

[22]:
```
# write here

def swap(s):
    print(s[-1] + s[1:-1] + s[0])
```

6.3.5 Verify comprehension

ATTENTION

Following exercises require you to know:

Complex statements: Andrea Passerini slides A03[155]

Functions: Andrea Passerini slides A04[156]

Tests with asserts[157]: Following exercises contain automated tests to help you spot errors. To understand how to do them, read before Error handling and testing[158]

has_char

⊛ RETURN `True` if `word` contains `char`, `False` otherwise

- use `while` cycle (just for didactical purposes, using `in` would certainly be faster & shorter)

[23]:
```
def has_char(word, char):
    #jupman-raise
    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:
```

(continues on next page)

[155] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[156] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf
[157] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-asserts
[158] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
    # we found nothing, so we have to return False
    return False
    #/jupman-raise


# 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
```

count

⊛ RETURN the number of occurrences of `char` in `word`

NOTE: I DO NOT WANT A PRINT, IT MUST *RETURN* THE VALUE !

- Use the cycle for in (just for didactical purposes, strings already provide a method to do it fast - which one?)

[24]:

```
def count(word, char):
    #jupman-raise
    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 !!
    #/jupman-raise

# 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
```

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
```

[25]:

```
def dialect(word):
    #jupman-raise
    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
    #/jupman-raise

# 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 !
```

[26]:

```
def countvoc(word):
    #jupman-raise
    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:
        return n_vocals
    else:
        raise ValueError("Odd vocals !")
    #/jupman-raise

# 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
```

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('scatola')
>>> 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 ..).

[27]:

```
def palindrome(word):
    #jupman-raise
    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
    #/jupman-raise

# 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('scatola') == False
```

[ ]:

extract_email

**COMMANDMENT 4 (adapted for strings): You shall never ever reassign function parameters **

```
def myfun(s):

    # You shall not do any of such evil, no matter what the type of the parameter is:
    s = "evil"          # strings
```

[28]:
```
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
    """
    #jupman-raise
    stripped = s.strip()
    i = stripped.find('<')
    return stripped[i+1:len(stripped)-1]
    #/jupman-raise

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 !!!!!

[29]:
```
def phone_canon(phone):
    #jupman-raise
    p = phone.replace(' ', '')
    if p.startswith('0039'):
        p = p[4:]
    if p.startswith('+39'):
        p = p[3:]
    return p
    #/jupman-raise

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'
```

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)`

[30]:
```
province_prefixes = ['0461', '02', '011']
mobile_prefixes = ['330', '340', '328', '390', '3771']


def phone_prefix(phone):
    #jupman-raise
    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'
```

(continues on next page)

(continued from previous page)

```
    #/jupman-raise

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'
```

6.3.6 Further resources

Have a look at leetcode string problems[159] sorting by *Acceptance* and *Easy*.

In particular, you may check:

- Detect Capital[160]
- Unique email addresses[161]
- Robot return to origin[162]
- Reverse Words in a String III[163]
- Unique Morse codes[164]
- Goat Latin[165]
- Count Binary Substrings[166]

```
[ ]:
```

6.4 Lists solutions

6.4.1 Download exercises zip

Browse files online[167]

[159] https://leetcode.com/tag/string/
[160] https://leetcode.com/problems/detect-capital/description/
[161] https://leetcode.com/problems/unique-email-addresses/description/
[162] https://leetcode.com/problems/robot-return-to-origin/description/
[163] https://leetcode.com/problems/reverse-words-in-a-string-iii/description/
[164] https://leetcode.com/problems/unique-morse-code-words/description/
[165] https://leetcode.com/problems/goat-latin/description/
[166] https://leetcode.com/problems/count-binary-substrings/description/
[167] https://github.com/DavidLeoni/sciprog-ds/tree/master/lists

6.4.2 What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-sciprog.py
-other stuff ...
-exercises
     |- lists
         |- lists.ipynb
         |- lists-sol.ipynb
         |- other stuff ..
```

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 `lists/lists.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 instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.4.3 Introduction

References

- Andrea Passerini, A02 slides (lists and tuples)[168]
- Think Python, Chapter 10, Lists[169]
- Think Python, Chapter 12, Tuples[170]

Python lists are **ordered** collections of (homogeneous) objects, but they can hold also non-homogeneous data. List are **mutable objects**. Elements of the collection are specified within two square brackets **[]** and are comma separated.

We can use the function print to print the content of lists. Some examples of list definitions follow:

[2]:
```
my_first_list = [1,2,3]
print("first:" , my_first_list)

my_second_list = [1,2,3,1,3] #elements can appear several times
print("second: ", my_second_list)
```

(continues on next page)

[168] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A02-datastructures.pdf
[169] http://greenteapress.com/thinkpython2/html/thinkpython2011.html
[170] http://greenteapress.com/thinkpython2/html/thinkpython2013.html

(continued from previous page)

```
fruits = ["apple", "pear", "peach", "strawberry", "cherry"] #elements can be strings
print("fruits:", fruits)

an_empty_list = []
print("empty:" , an_empty_list)

another_empty_list = list()
print("another empty:", another_empty_list)

a_list_containing_other_lists = [[1,2], [3,4,5,6]] #elements can be other lists
print("list of lists:", a_list_containing_other_lists)

my_final_example = [my_first_list, a_list_containing_other_lists]
print("a list of lists of lists:", my_final_example)
```

```
first: [1, 2, 3]
second:  [1, 2, 3, 1, 3]
fruits: ['apple', 'pear', 'peach', 'strawberry', 'cherry']
empty: []
another empty: []
list of lists: [[1, 2], [3, 4, 5, 6]]
a list of lists of lists: [[1, 2, 3], [[1, 2], [3, 4, 5, 6]]]
```

Operators for lists

Python provides several operators to handle lists. The following behave like on strings (**remember that as in strings, the first position is 0!**):

Result	Operator	Meaning
bool	=, !=	Check if two lists are equal or different
int	len(list)	Return the length of the list
list	list + list	Concatenate two lists (returns a new list)
list	list * int	Replicate the list (returns a new list)
list	list[int:int]	Extract a sub-list

While this requires that the whole tested obj is present in the list

Result	Operator	Meaning
bool	obj in list	Check if an element is present in a list

and

Result	Operator	Meaning
obj	list[int]	Read/write an element at a specified index

can also change the corresponding value of the list (**lists are mutable objects**).

Some examples follow.

[3]:
```
A = [1, 2, 3 ]
B = [1, 2, 3, 1, 2]

print("A is a ", type(A))
```

```
A is a  <class 'list'>
```

[4]:
```
print(A, " has length: ", len(A))
```

```
[1, 2, 3]  has length:  3
```

[5]:
```
print("A[0]: ", A[0], " A[1]:", A[1], " A[-1]:", A[-1])
```

```
A[0]:  1  A[1]: 2  A[-1]: 3
```

[6]:
```
print(B, " has length: ", len(B))
```

```
[1, 2, 3, 1, 2]  has length:  5
```

[7]:
```
print("Is A equal to B?", A == B)
```

```
Is A equal to B? False
```

[8]:
```
C = A + [1, 2]
print(C)
```

```
[1, 2, 3, 1, 2]
```

[9]:
```
print("Is C equal to B?", B == C)
```

```
Is C equal to B? True
```

[10]:
```
D = [1, 2, 3]*8
```

[11]:
```
E = D[12:18] #slicing
print(E)
```

```
[1, 2, 3, 1, 2, 3]
```

[12]:
```
print("Is A*2 equal to E?", A*2 == E)
```

```
Is A*2 equal to E? True
```

[13]:
```
A = [1, 2, 3, 4, 5, 6]
B = [1, 3, 5]
print("A:", A)
print("B:", B)
```

```
A: [1, 2, 3, 4, 5, 6]
B: [1, 3, 5]
```

[14]:
```
print("Is B in A?", B in A)
```

```
Is B in A? False
```

[15]:
```
print("A\'s ID:", id(A))
```

```
A's ID: 140585721605768
```

[16]:
```
A[5] = [1,3,5] #we can add elements
print(A)
```

```
[1, 2, 3, 4, 5, [1, 3, 5]]
```

[17]:
```
print("A\'s ID:", id(A))
```

```
A's ID: 140585721605768
```

[18]:
```
print("A has length:", len(A))
```

```
A has length: 6
```

[19]:
```
print("Is now B in A?", B in A)
```

```
Is now B in A? True
```

> **Note:** When slicing do not exceed the list boundaries (or you will be prompted a `list index out of range` error).

Consider the following example:

[20]:
```
A = [1, 2, 3, 4, 5, 6]
print("A has length:", len(A))
```

```
A has length: 6
```

[21]:
```
print("First element:", A[0])
```

```
First element: 1
```

```
print("7th-element: ", A[6])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-67-98687c36d491> in <module>
----> 1 print("7th-element: ", A[6])

IndexError: list index out of range
```

Example: Consider the matrix $M = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \end{bmatrix}$ and the vector $v = [10, 5, 10]^T$. What is the matrix-vector product $M * v$?

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \end{bmatrix} * [10, 5, 10]^T = [50, 30, 45]^T$$

[22]:
```
M = [[1, 2, 3], [1, 2, 1], [1, 1, 3]]
v = [10, 5, 10]
prod = [0, 0 ,0] #at the beginning the product is the null vector

prod[0]=M[0][0]*v[0] + M[0][1]*v[1] + M[0][2]*v[2]
prod[1]=M[1][0]*v[0] + M[1][1]*v[1] + M[1][2]*v[2]
prod[2]=M[2][0]*v[0] + M[2][1]*v[1] + M[2][2]*v[2]

print("M: ", M)
```

```
M:  [[1, 2, 3], [1, 2, 1], [1, 1, 3]]
```

[23]:
```
print("v: ", v)
```

```
v:  [10, 5, 10]
```

[24]:
```
print("M*v: ", prod)
```

```
M*v:  [50, 30, 45]
```

Methods of the class list

The class list has some methods to operate on it. Recall from the lecture the following methods:

Return	Method	Meaning
None	`list.append(obj)`	Add a new element at the end of the list
None	`list.extend(list)`	Add several new elements at the end of the list
None	`list.insert(int,obj)`	Add a new element at some given position
None	`list.remove(obj)`	Remove the first occurrence of an element
None	`list.reverse()`	Invert the order of the elements
None	`list.sort()`	Sort the elements
int	`list.count(obj)`	Count the occurrences of an element

Note: Lists are **mutable objects** and therefore virtually all the previous methods (except *count*) do not have an output value, but they **modify** the list.

Some usage examples follow:

[25]:
```
#A numeric list
A = [1, 2, 3]
print(A)
```

```
[1, 2, 3]
```

[26]:
```
print("A has id:", id(A))
```

```
A has id: 140585712305608
```

[27]:
```
A.append(72) # appends one and only one object.
             # NOTE: does not return anything !!!!
```

[28]:
```
print(A)
```

```
[1, 2, 3, 72]
```

[29]:
```
print("A has id:", id(A))
```

```
A has id: 140585712305608
```

[30]:
```
A.extend([1, 5, 124, 99]) # adds all these objects, one after the other.
                          # NOTE: does not return anything !!!
```

[31]:
```
print(A)
```

```
[1, 2, 3, 72, 1, 5, 124, 99]
```

[32]:
```
print("A has id:", id(A))  # same id as before
```

```
A has id: 140585712305608
```

[33]:
```
D = [9,6,4]

A = A + D   # beware: + between lists generates an entirely *new* list !!!!
print(A)
```

```
[1, 2, 3, 72, 1, 5, 124, 99, 9, 6, 4]
```

[34]:
```
print("A has now id:", id(A))  # id is different from before !!!
```

```
A has now id: 140585822899400
```

[35]:
```
A.reverse()  # Does not return anything !!!
```

[36]:
```
print(A)
```

```
[4, 6, 9, 99, 124, 5, 1, 72, 3, 2, 1]
```

[37]:
```
A.sort()
print(A)
```

```
[1, 1, 2, 3, 4, 5, 6, 9, 72, 99, 124]
```

[38]:
```
print("Min value: ", A[0]) # In this simple case, could have used min(A)
```

```
Min value:  1
```

[39]:
```
print("Max value: ", A[-1]) #In this simple case, could have used max(A)
```

```
Max value:  124
```

[40]:
```
print("Number 1 appears:", A.count(1), " times")
```

```
Number 1 appears: 2  times
```

[41]:
```
print("While number 837: ", A.count(837))
```

```
While number 837:  0
```

Exercise: growing list 1

Given a list `la` of *fixed size 7*, write some code to grow an empty list `lb` so that it contains *only* the elements from `la` at even indeces (0, 2, 4, …).

- Your code should work for any list `la` of fixed size 7.

```
#   0 1 2 3 4 5 6  indeces
la=[8,4,3,5,7,3,5]
lb=[]
```

After your code, you should get:

```
>>> print(lb)
[8,3,7,5]
```

[42]:
```
#   0 1 2 3 4 5 6  indeces
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]
```

Exercise: growing list 2

Given two *lists* `la` and `lb`, write some code that MODIFIES `la` such that `la` contains at the end also all elements of `lb`.

- **NOTE 1**: your code should work with any `la` and `lb`
- **NOTE 2**: If you try to print `id(la)` *before* modifying `la` and `id(la)` afterwords, you should get *exactly* the same id. If you get a different one, it means you generated an entirely new list. In any case, check how it works in python tutor.

```
la = [5,9,2,4]
lb = [9,1,2]
```

You should obtain:

```
>>> print(la)
[5,9,2,4,9,1,2]
>>> print(lb)
[9,1,2]
```

[43]:
```
la = [5,9,2,4]
lb = [9,1,2]

# write here
la.extend(lb)
print(la)
print(lb)
```

```
[5, 9, 2, 4, 9, 1, 2]
[9, 1, 2]
```

List of strings

Let's now try a list with strings, we will try to obtain a a reverse lexicographic order:

[44]:
```
#A string list
fruits = ["apple", "banana", "pineapple", "cherry","pear", "almond", "orange"]

print(fruits)
```

```
['apple', 'banana', 'pineapple', 'cherry', 'pear', 'almond', 'orange']
```

[45]:
```
fruits.sort()  # does not return anything. Modifies list!
```

[46]:
```
print(fruits)
```

```
['almond', 'apple', 'banana', 'cherry', 'orange', 'pear', 'pineapple']
```

[47]:
```
fruits.reverse()
print(fruits)
```

```
['pineapple', 'pear', 'orange', 'cherry', 'banana', 'apple', 'almond']
```

[48]:
```
fruits.remove("banana")   # NOTE: does not return anything !!!
```

[49]:
```
print(fruits)
```

```
['pineapple', 'pear', 'orange', 'cherry', 'apple', 'almond']
```

[50]:
```
fruits.insert(5, "wild apple") # put wild apple after apple.
                               # NOTE: does not return anything !!!
```

[51]:
```
print(fruits)
```

```
['pineapple', 'pear', 'orange', 'cherry', 'apple', 'wild apple', 'almond']
```

Let's finally obtain the sorted fruits:

[52]:
```
fruits.sort() # does not return anything. Modifies list!
```

[53]:
```
print(fruits)
```

```
['almond', 'apple', 'cherry', 'orange', 'pear', 'pineapple', 'wild apple']
```

Some things to remember

1. append and extend work quite differently:

[54]:
```
A = [1, 2, 3]

A.extend([4, 5])
```

[55]:
```
print(A)
```

```
[1, 2, 3, 4, 5]
```

[56]:
```
B = [1, 2, 3]
B.append([4,5])    # NOTE: append does not return anything !
```

[57]:
```
print(B)
```

```
[1, 2, 3, [4, 5]]
```

2. To remove an object it must exist:

[58]:
```
A = [1,2,3, [[4],[5,6]], 8]
print(A)
```

```
[1, 2, 3, [[4], [5, 6]], 8]
```

[59]:
```
A.remove(2)    # NOTE: remove does not return anything !!
```

[60]:
```
print(A)
```

```
[1, 3, [[4], [5, 6]], 8]
```

[61]:
```
A.remove([[4],[5,6]])    # NOTE: remove does not return anything !!
```

[62]:
```
print(A)
```

```
[1, 3, 8]
```

```
A.remove(7)     # 7 is not present in list, python will complain during execution
                # NOTE: remove does not return anything !!
```


```
A.remove(7)     # 7 is not present in list, python will complain during execution
                # NOTE: remove does not return anything !!
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-61-6cfd75f76650> in <module>
----> 1 A.remove(7)     # 7 is not present in list, python will complain during␣
↪execution
      2                 # NOTE: remove does not return anything !!

ValueError: list.remove(x): x not in list
```

3. To sort a list, its elements must be sortable (i.e. homogeneous)!

[63]:
```
A = [4,3, 1,7, 2]
print(A)
```

```
[4, 3, 1, 7, 2]
```

[64]:
```
A.sort()     # NOTE: sort does not return anything !!
```

[65]:
```
print(A)
```

```
[1, 2, 3, 4, 7]
```

[66]:
```
A.append("banana")    # NOTE: append does not return anything !!
```

[67]:
```
print(A)
```

```
[1, 2, 3, 4, 7, 'banana']
```

```
A.sort()    # Python will complain, list contains uncomparable elements
            # like ints and strings
            # NOTE: sort does not return anything !!
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-acf26fcfe0bf> in <module>
----> 1 A.sort()    # Python will complain, list contains uncomparable elements like␣
↪ints and strings
      2             # NOTE: sort does not return anything !!

TypeError: '<' not supported between instances of 'str' and 'int'
```

Lists hold references

Important to remember:

Lists are **mutable objects** and this has some consequences! Since lists are mutable objects, they hold references to objects rather than objects.

Take a look at the following examples:

[68]:
```
l1 = [1, 2]
print("l1:", l1)
```

```
l1: [1, 2]
```

[69]:
```
l2 = [4, 3]
print("l2:",l2)
```

```
l2: [4, 3]
```

[70]:
```
LL = [l1, l2]
print("LL:", LL)
```

```
LL: [[1, 2], [4, 3]]
```

[71]:
```
l1.append(7)     # NOTE: does not return anything !!
print("\nAppending 7 to l1...")
print("l1:", l1)
print("LL now: ", LL)
```

```

Appending 7 to l1…
l1: [1, 2, 7]
LL now:  [[1, 2, 7], [4, 3]]
```

[72]:
```
LL[0][1] = -1
print("\nSetting LL[0][1]=-1...")
print("LL now:" , LL)
print("l1 now", l1)
```

```

Setting LL[0][1]=-1…
LL now: [[1, -1, 7], [4, 3]]
l1 now [1, -1, 7]
```

[73]:
```
# but the list can point also to a different object,
# without affecting the original list.
LL[0] = 100
print("\nSetting LL[0] = 100")
print("LL now:", LL)
print("l1 now", l1)
```

```

Setting LL[0] = 100
LL now: [100, [4, 3]]
l1 now [1, -1, 7]
```

Making copies

[74]:
```
A = ["hi", "there"]
print("A:", A)
```

```
A: ['hi', 'there']
```

[75]:
```
B = A
print("B:", B)
```

```
B: ['hi', 'there']
```

[76]:
```
A.extend(["from", "python"])  # NOTE: extend does not return anything !
```

[77]:
```
print("A now: ", A)
print("B now: ", B)
```

```
A now:  ['hi', 'there', 'from', 'python']
B now:  ['hi', 'there', 'from', 'python']
```

Copy example

Let's make a distinct copy of `A`

[78]:
```
C = A[:] # all the elements of A have been copied in C
print("C:", C)
```

```
C: ['hi', 'there', 'from', 'python']
```

[79]:
```
A[3] = "java"
print("A now:", A)
print("C now:", C)
```

```
A now: ['hi', 'there', 'from', 'java']
C now: ['hi', 'there', 'from', 'python']
```

Be careful though:

[80]:
```
D = [A, A]
print("D:", D)
```

```
D: [['hi', 'there', 'from', 'java'], ['hi', 'there', 'from', 'java']]
```

[81]:
```
E = D[:]
print("E:", E)
```

```
E: [['hi', 'there', 'from', 'java'], ['hi', 'there', 'from', 'java']]
```

[82]:
```
D[0][0] = "hello"
print("\nD now:", D)
print("E now:", E)
print("A now:", A)
```

```

D now: [['hello', 'there', 'from', 'java'], ['hello', 'there', 'from', 'java']]
E now: [['hello', 'there', 'from', 'java'], ['hello', 'there', 'from', 'java']]
A now: ['hello', 'there', 'from', 'java']
```

Equality and identity

[83]:
```
A = [1, 2, 3]
B = A
C = [1, 2, 3]
```

[84]:
```
print("Is A equal to B?", A == B)
```

```
Is A equal to B? True
```

[85]:
```
print("Is A actually B?", A is B)
```

```
Is A actually B? True
```

[86]:
```
print("Is A equal to C?", A == C)
```

```
Is A equal to C? True
```

[87]:
```
print("Is A actually C?", A is C)
```

```
Is A actually C? False
```

In fact:

[88]:
```
print("\nA's id:", id(A))
print("B's id:", id(B))
print("C's id:", id(C))
```

```

A's id: 140585712271432
B's id: 140585712271432
C's id: 140585711965896
```

[89]:
```
#just to confirm that:
A.append(4)     # NOTE: append does not return anything !
```

[90]:
```
B.append(5)     # NOTE: append does not return anything !
```

[91]:
```
print("\nA now: ", A)
print("B now: ", A)
```

```

A now:  [1, 2, 3, 4, 5]
B now:  [1, 2, 3, 4, 5]
```

From strings to lists, the `split` method

Strings have a method *split* that can literally split the string at specific characters.

Example Suppose we have a protein encoded as a multiline-string. How can we split it into several lines?

[92]:
```
chain_a = """SSSVPSQKTYQGSYGFRLGFLHSGTAKSVTCTYSPALNKM
FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVV
RRCPHHERCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFR
HSVVVPYEPPEVGSDCTTIHYNYMCNSSCMGGMNRRPILT
IITLEDSSGNLLGRNSFEVRVCACPGRDRRTEEENLRKKG
```

(continues on next page)

(continued from previous page)

```
EPHHELPPGSTKRALPNNT"""

lines = chain_a.split('\n')
print("Original sequence:")
print( chain_a, "\n") #some spacing to keep things clear
print("line by line:")
print("1st line:" ,lines[0])
print("2nd line:" ,lines[1])
print("3rd line:" ,lines[2])
print("4th line:" ,lines[3])
print("5th line:" ,lines[4])
print("6th line:" ,lines[5])

print("\nSplit the 1st line in correspondence to FRL:\n",lines[0].split("FRL"))
```

```
Original sequence:
SSSVPSQKTYQGSYGFRLGFLHSGTAKSVTCTYSPALNKM
FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVV
RRCPHHERCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFR
HSVVVPYEPPEVGSDCTTIHYNYMCNSSCMGGMNRRPILT
IITLEDSSGNLLGRNSFEVRVCACPGRDRRTEEENLRKKG
EPHHELPPGSTKRALPNNT

line by line:
1st line: SSSVPSQKTYQGSYGFRLGFLHSGTAKSVTCTYSPALNKM
2nd line: FCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVV
3rd line: RRCPHHERCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFR
4th line: HSVVVPYEPPEVGSDCTTIHYNYMCNSSCMGGMNRRPILT
5th line: IITLEDSSGNLLGRNSFEVRVCACPGRDRRTEEENLRKKG
6th line: EPHHELPPGSTKRALPNNT

Split the 1st line in correspondence to FRL:
 ['SSSVPSQKTYQGSYG', 'GFLHSGTAKSVTCTYSPALNKM']
```

Note that in the last instruction, the substring FRL is disappeared (as happened to the newline).

And back to strings with the `join` method

Given a list, one can join the elements of the list together into a string by using the `join` method of the class string. The syntax is the following: *str.join(list)* which joins together all the elements in the list in a string separating them with the string *str*.

Example Given the list `['Oct', '5', '2018', '15:30']`, let's combine all its elements in a string joining the elements with a dash ("-") and print them. Let's finally join them with a tab (`"\t"`) and print them.

[93]:
```
vals = ['Oct', '5th', '2018', '15:30']
print(vals)
myStr = "-".join(vals)
print("\n" + myStr)
myStr = "\t".join(vals)
print("\n" + myStr)
```

```
['Oct', '5th', '2018', '15:30']

Oct-5th-2018-15:30

Oct     5th     2018    15:30
```

Exercise: manylines

Given the following text string:

```
"""this is a text
string on
several lines that does not say anything."""
```

1. print it
2. print how many lines, words and characters it contains.
3. sort the words alphabetically and print the first and the last in lexicographic order.

You should obtain:

```
this is a text
string on
several lines that does not say anything.

Lines: 3 words: 13 chars: 66

['t', 'h', 'i', 's', ' ', 'i', 's', ' ', 'a', ' ', 't', 'e', 'x', 't', '\n', 's', 't',
↪ 'r', 'i', 'n', 'g', ' ', 'o', 'n', '\n', 's', 'e', 'v', 'e', 'r', 'a', 'l', ' ', 'l
↪', 'i', 'n', 'e', 's', ' ', 't', 'h', 'a', 't', ' ', 'd', 'o', 'e', 's', ' ', 'n',
↪'o', 't', ' ', 's', 'a', 'y', ' ', 'a', 'n', 'y', 't', 'h', 'i', 'n', 'g', '.']
66

First word:  a
Last word: this
['a', 'anything.', 'does', 'is', 'lines', 'not', 'on', 'say', 'several', 'string',
↪'text', 'that', 'this']
```

[94]:
```
s = """this is a text
string on
several lines that does not say anything."""

# write here

# 1) print it
print(s)
print("")

# 2) print the lines, words and characters
lines = s.split('\n')

# NOTE: words are split 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 way for number of characters:
print("")
characters = list(s)
num_chars2 = len(characters)
print(characters)
print(num_chars2)
```

(continues on next page)

(continued from previous page)

```
# 3. sort the words alphabetically and print the first and the last in lexicographic␣
↪order.
words.sort() # NOTE: it does not return ANYTHING!!!
print("")
print("First word: ", words[0])
print("Last word:", words[-1])
print(words)
```

```
this is a text
string on
several lines that does not say anything.

Lines: 3 words: 13 chars: 66

['t', 'h', 'i', 's', ' ', 'i', 's', ' ', 'a', ' ', 't', 'e', 'x', 't', '\n', 's', 't',
↪ 'r', 'i', 'n', 'g', ' ', 'o', 'n', '\n', 's', 'e', 'v', 'e', 'r', 'a', 'l', ' ', 'l
↪', 'i', 'n', 'e', 's', ' ', 't', 'h', 'a', 't', ' ', 'd', 'o', 'e', 's', ' ', 'n',
↪'o', 't', ' ', 's', 'a', 'y', ' ', 'a', 'n', 'y', 't', 'h', 'i', 'n', 'g', '.']
66

First word:  a
Last word: this
['a', 'anything.', 'does', 'is', 'lines', 'not', 'on', 'say', 'several', 'string',
↪'text', 'that', 'this']
```

Exercise: welldone

Given the list

```
L = ["walnut", "eggplant", "lemon", "lime", "date", "onion", "nectarine", "endive" ]:
```

1. Create another list (called `newList`) containing the first letter of each element of `L` (e.g `newList =["w", "e", ...]`).
2. Add a space to newList at position 4 and append an exclamation mark (`!`) at the end.
3. Print the list.
4. Print the content of the list joining all the elements with an empty space (i.e. use the method `join`: `"".join(newList)`)

You should obtain:

```
['w', 'e', 'l', 'l', ' ', 'd', 'o', 'n', 'e', '!']

 well done!
```

[95]:
```
L = ["walnut", "eggplant", "lemon", "lime", "date", "onion", "nectarine", "endive" ]

# write here

newList = []
newList.append(L[0][0])
newList.append(L[1][0])
newList.append(L[2][0])
```

(continues on next page)

(continued from previous page)

```
newList.append(L[3][0])
newList.append(L[4][0])
newList.append(L[5][0])
newList.append(L[6][0])
newList.append(L[7][0])

newList.insert(4," ")
newList.append("!")

print(newList)
print("\n", "".join(newList))
```

```
['w', 'e', 'l', 'l', ' ', 'd', 'o', 'n', 'e', '!']

 well done!
```

Exercise: numlist

Given the list `lst = [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]`

1. find the min, max and median value (hint: sort it and extract the right values).
2. Create a list with only the elements at even indexes (i.e. [10, 72, 11, ..], note that the ".." means that the list is not complete!) and re-compute min, max and median values.
3. re-do the same for the elements located at odd indexes (i.e. [60, 118,..]).

You should obtain:

```
lst: [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]
lst : Min:  10  Max. 175  Median:  72
even: Min:  10  Max. 120  Median:  56
odd: Min:  60  Max. 175  Median:  89
```

[2]:
```
lst = [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]

# write here

even = L[0::2] #get only even-indexed elements
odd = L[1::2] #get only odd-indexed elements

print("lst:" , lst)
print("Leven:", even)
print("Lodd:", odd)
lst.sort()
even.sort()
odd.sort()


print("sorted:  " , lst)
print("sorted even:  " , even)
```

(continues on next page)

(continued from previous page)

```
print("sorted odd:  " , odd)

print("lst: Min: ", lst[0], " Max." , lst[-1], " Median: ", lst[len(lst) // 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])
```

```
lst: [10, 60, 72, 118, 11, 71, 56, 89, 120, 175]
Leven: [10, 56, 71, 89, 120]
Lodd: [11, 60, 72, 118, 175]
sorted:   [10, 11, 56, 60, 71, 72, 89, 118, 120, 175]
sorted even:   [10, 56, 71, 89, 120]
sorted odd:   [11, 60, 72, 118, 175]
lst: Min:  10  Max. 175  Median:  72
even: Min:  10  Max. 120  Median:  71
odd: Min:  11  Max. 175  Median:  72
```

List comprehension

List comprehension is a quick way of creating a list. The resulting list is normally obtained by applying a function or a method to the elements of another list that **remains unchanged**.

The basic syntax is:

```
new_list = [ some_function (x) for x in start_list]
```

or

```
new_list = [ x.some_method() for x in start_list]
```

List comprehension can also be used to filter elements of a list and produce another list as sublist of the first one (**remember that the original list is not changed**).

In this case the syntax is:

```
new_list = [ some_function (x) for x in start_list if condition]
```

or

```
new_list = [ x.some_method() for x in start_list if condition]
```

where the element x in start_list becomes part of new_list if and only if the condition holds True.

Let's see some examples:

Example: Given a list of strings ["hi", "there", "from", "python"] create a list with the length of the corresponding element (i.e. the one with the same index).

[97]:
```
elems = ["hi", "there", "from", "python"]

newList = [len(x) for x in elems]

for i in range(0,len(elems)):
    print(elems[i], " has length ", newList[i])
```

```
hi  has length  2
there  has length  5
from  has length  4
python  has length  6
```

Example: Given a list of strings ["dog", "cat", "rabbit", "guinea pig", "hamster", "canary", "goldfish"] create a list with the elements starting with a "c" or "g".

[98]:
```
pets = ["dog", "cat", "rabbit", "guinea pig", "hamster", "canary", "goldfish"]

cg_pets = [x for x in pets if x.startswith("c") or x.startswith("g")]

print("Original:")
print(pets)
print("Filtered:")
print(cg_pets)
```

```
Original:
['dog', 'cat', 'rabbit', 'guinea pig', 'hamster', 'canary', 'goldfish']
Filtered:
['cat', 'guinea pig', 'canary', 'goldfish']
```

Example: Create a list with all the numbers divisible by 17 from 1 to 200.

[99]:
```
values = [ x for x in range(1,200) if x % 17 == 0]
print(values)
```

```
[17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187]
```

Example: Transpose the matrix $\begin{bmatrix} 1 & 10 \\ 2 & 20 \\ 3 & 30 \\ 4 & 40 \end{bmatrix}$ stored as a list of lists (i.e. matrix = [[1, 10], [2,20], [3,30], [4,40]]). The output matrix should be: $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 10 & 20 & 30 & 40 \end{bmatrix}$, represented as [[1, 2, 3, 4], [10, 20, 30, 40]]

[100]:
```
matrix = [[1, 10], [2,20], [3,30], [4,40]]
print(matrix)
transpose = [[row[i] for row in matrix] for i in range(2)]
print (transpose)
```

```
[[1, 10], [2, 20], [3, 30], [4, 40]]
[[1, 2, 3, 4], [10, 20, 30, 40]]
```

Example: Given the list:

```
["Hotel", "Icon"," Bus","Train", "Hotel", "Eye", "Rain", "Elephant"]
```

create a list with all the first letters.

[101]:
```
myList = ["Hotel", "Icon"," Bus","Train", "Hotel", "Eye", "Rain", "Elephant"]
initials = [x[0] for x in myList]

print(myList)
print(initials)
print("".join(initials))
```

```
['Hotel', 'Icon', ' Bus', 'Train', 'Hotel', 'Eye', 'Rain', 'Elephant']
['H', 'I', ' ', 'T', 'H', 'E', 'R', 'E']
HI THERE
```

6.4.4 Exercises with functions

ATTENTION

Following exercises require you to know:

Complex statements: Andrea Passerini slides A03[171]

Functions: Andrea Passerini slides A04[172]

printwords

⊛ Write a function `printwords` that PRINTS all the words in a phrase

```
>>> printwords("ciao come stai?")
```

```
ciao
come
stai?
```

[102]:
```
# 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?
```

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
```

[103]:
```
# write here

def printeven(xs):

    for x in xs:
        if x % 2 == 0:
```

(continues on next page)

[171] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[172] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf

(continued from previous page)

```
            print(x)

numbers = [1,2,3,4,5,6]
printeven(numbers)
```

```
2
4
6
```

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
```

[104]:
```
# write here

def find26(xs):
    return (26 in xs)

numbers = [1,26,143,431,53,6]
find26(numbers)
```

[104]:
```
True
```

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
```

[105]:
```
# 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
```

threeven

⊛ 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"`

```
>>> threeven([6,4,8,4,5])
yes
>>> threeven([2,5,6,3,4,5])
no
>>> threeven([4])
not good
```

[106]:
```
# write here

def threeven(xs):
    if len(xs) >= 3:
        if xs[0] % 2 == 0 and xs[1] % 2 == 0 and xs[2] % 2 == 0:
            print("yes")
        else:
            print("no")
    else:
        print("not good")

threeven([6,4,8,4,5])
threeven([2,5,6,3,4,5])
threeven([4])
```

```
yes
no
not good
```

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
```

[107]:
```
# 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
```

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
```

[108]:
```
# 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
```

[108]:
```
26.75
```

6.4.5 Verify comprehension

ATTENTION

Following exercises require you to know:

Complex statements: Andrea Passerini slides A03[173]

Functions: Andrea Passerini slides A04[174]

Tests with asserts[175]: Following exercises contain automated tests to help you spot errors. To understand how to do them, read before Error handling and testing[176]

We will discuss differences between *modifying* a list and *returning a new one*, and look into basic operations like transform, filter, mapping.

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.

newdoublefor

Difficulty: ⊛

[109]:
```
def newdoublefor(lst):
    """ Takes a list of integers in input and RETURN a NEW one with all
        the numbers of lst doubled. Implement it with a for.

        Example:

                newdouble([3,7,1])

        returns:

                [6,14,2]
    """
    #jupman-raise
    ret = []
    for x in lst:
        ret.append(x*2)
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert newdoublefor([]) == []
assert newdoublefor([3]) == [6]
assert newdoublefor([3,7,1]) == [6,14,2]

l = [3,7,1]
assert newdoublefor(l) == [6,14,2]
assert l == [3,7,1]
# TEST END
```

[173] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[174] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf
[175] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-asserts
[176] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html

double

Difficulty: ⊛⊛

[110]:
```
def double(lst):
    """ Takes a list of integers in input and MODIFIES it by doubling all the numbers
    """

    #jupman-raise
    for i in range(len(lst)):
        lst[i] = lst[i] * 2
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
l = []
double(l)
assert l == []

l = [3]
double(l)
assert l == [6]


l = [3,7,1]
double(l)
assert l == [6,14,2]
# TEST END
```

newdoublecomp

Difficulty: ⊛

[111]:
```
def newdoublecomp(lst):
    """ Takes a list of integers in input and RETURN a NEW one with all
        the numbers of lst doubled. Implement it as a list comprehnsion

        Example:

                newdouble([3,7,1])

        returns:

                [6,14,2]
    """
    #jupman-raise
    return [x*2 for x in lst]
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert newdoublecomp([]) == []
assert newdoublecomp([3]) == [6]
assert newdoublecomp([3,7,1]) == [6,14,2]
```

(continues on next page)

(continued from previous page)

```
l = [3,7,1]
assert newdoublecomp(l) == [6,14,2]
assert l == [3,7,1]
# TEST END
```

up

Difficulty: ⊛

[112]:
```
def up(lst):
    """ Takes a list of strings and RETURN a NEW list having all the strings in lst␣
↪in capital
        (use .upper() method and a list comprehension )
    """
    #jupman-raise
    return [x.upper() for x in lst]
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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']
# TEST END
```

Filter

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 give back a NEW list or MODIFY the existing list.

remall

Difficulty: ⊛⊛

[113]:
```
def remall(list1, list2):
    """ RETURN a NEW list which has the elements from list2 except the elements in␣
↪list1
    """
    #jupman-raise
    list3 = list2[:]
    for x in list1:
        if x in list3:
            list3.remove(x)

    return list3
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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']

assert remall(['a','d'], ['a','c','d','b']) == ['c', 'b']
# TEST END
```

only_capital_for

Difficulty: ⊛

[114]:
```
def only_capital_for(lst):
    """ 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')

        Implement it with a for
    """
    #jupman-raise
    ret = []
    for el in lst:
        if el.isupper():
            ret.append(el)
    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert only_capital_for(["CD"]) == [ "CD"]
assert only_capital_for(["ab"]) == []
```

(continues on next page)

(continued from previous page)

```
assert only_capital_for(["dE"]) == []
assert only_capital_for(["De"]) == []
assert only_capital_for(["ab","DE"]) == ["DE"]
assert only_capital_for(["ab", "CD", "Hb", "EF"]) == [ "CD", "EF"]
# TEST END
```

only_capital_comp

Difficulty: ⊛

[115]:
```
def only_capital_comp(lst):
    """ 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')

        Implement it with a list comprehension
    """
    #jupman-raise
    return [el for el in lst if el.isupper() ]
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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"]
assert only_capital_comp(["ab", "CD", "Hb", "EF"]) == [ "CD", "EF"]
# END
```

Reduce

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 give back a NEW list or MODIFY the existing list.

sum_all

Difficulty: ⊛

[116]:
```
def sum_all(lst):
    """ RETURN the sum of all elements in lst

        Implement it as you like.
    """
    #jupman-raise
    return sum(lst)
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
```

(continues on next page)

(continued from previous page)

```
assert sum_all([]) == 0
assert sum_all([7,5]) == 12
assert sum_all([9,5,8]) == 22

# TEST END
```

sum_all_even_for

Difficulty: ⊛

[117]:
```
def sum_all_even_for(lst):
    """ RETURN the sum of all even elements in lst

        Implement it with a for
    """
    #jupman-raise
    ret = 0
    for el in lst:
        if el % 2 == 0:
            ret += el
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert sum_all_even_for([]) == 0
assert sum_all_even_for([9]) == 0
assert sum_all_even_for([4]) == 4
assert sum_all_even_for([7,2,5,8]) == 10
# END
```

sum_all_even_comp

Difficulty: ⊛

[118]:
```
def sum_all_even_comp(lst):
    """ RETURN the sum of all even elements in lst

        Implement it in one line as an operation on a list comprehension
    """
    #jupman-raise
    return sum([el for el in lst if el % 2 == 0])
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert sum_all_even_comp([]) == 0
assert sum_all_even_comp([9]) == 0
assert sum_all_even_comp([4]) == 4
assert sum_all_even_comp([7,2,5,8]) == 10
# END
```

Other exercises

contains

⊛ RETURN `True` if `elem` is present in list, otherwise RETURN `False`

[119]:
```
def contains(xs, x):
    #jupman-raise
    return x in xs
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert contains([],'a') == False
assert contains(['a'],'a') == True
assert contains(['a','b','c'],'b') == True
assert contains(['a','b','c'],'z') == False
# END TEST
```

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` < 0, RETURN an empty list

Ingredients:

- variable list to return
- variable counter
- cycle `while` (there also other ways)
- `return`

[120]:
```

def firstn(n):
    #jupman-raise
    ret = []
    counter = 0
    while counter < n:
        ret.append(counter)
        counter += 1
    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert firstn(-1) == []
assert firstn(-2) == []
assert firstn(0) == []
assert firstn(1) == [0]
```

(continues on next page)

(continued from previous page)

```
assert firstn(2) == [0,1]
assert firstn(3) == [0,1,2]
# TEST END
```

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.

[121]:
```
def firstlast(xs):
    #jupman-raise
    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
    #       so we can directly return the result of the expression

    #/jupman-raise



# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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
# TEST END
```

dup

⊛ RETURN a NEW list, in which each list element in input is duplicated. For example,

```
dup(['ciao','mondo','python'])
```

must RETURN

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

Ingredients: - variable for a new list - for cycle - return

[122]:
```
def dup(xs):
    #jupman-raise

    ret = []
```

(continues on next page)

(continued from previous page)

```
    for x in xs:
        ret.append(x)
        ret.append(x)
    return ret

    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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']
# TEST END
```

hasdup

⊛⊛ RETURN `True` if `xs` contains element `x` more than once, otherwise RETURN `False`.

[123]:
```
def hasdup(x, xs):
    #jupman-raise

    counter = 0

    for y in xs:
        if y == x:
            counter += 1
            if counter > 1:
                return True
    return False
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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
# TEST END
```

ord3

⊛⊛ RETURN `True` if provided list has first elements increasingly ordered, `False` otherwise

- if `xs` has less than three elements, RETURN `False`

[124]:
```
def ord3(xs):
    #jupman-raise
    if len(xs) >= 3:
        return xs[0] <= xs[1] and xs[1] <= xs[2]
    else:
        return False
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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
# TEST END
```

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'])
```

must return

```
['a','b','a','a','b']
```

[125]:
```
def filterab(xs):
    #jupman-raise
    ret = []
    for x in xs:
        if x == 'a' or x == 'b':
            ret.append(x)
    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert filterab([]) == []
assert filterab(['a']) == ['a']
assert filterab(['b']) == ['b']
assert filterab(['a','b']) == ['a','b']
```

(continues on next page)

(continued from previous page)

```
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

# TEST END
```

hill

⊛⊛ RETURN a list having as with first elements the numbers from one to `n` increasing, and after `n` the decrease until `1` included. NOTE: `n` is contained only once.

Example:

```
hill(4)
```

must return

```
[1,2,3,4,3,2,1]
```

Ingredients: - variable for the list to return - two for cycles one after the other and `range` functions or two `while` one after the other

[126]:
```
def hill(n):

    #jupman-raise
    ret = []
    for i in range(1,n):
        ret.append(i)
    for i in range(n,0,-1):
        ret.append(i)
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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]
# TEST END
```

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
```

[127]:

```python
def peak(xs):
    #jupman-raise
    if len(xs) < 3:
        raise ValueError("Empty list !")
    if len(xs) == 1:
        return xs[0]

    for i in range(len(xs)):
        if xs[i] > xs[i+1]:
            return xs[i]

    return xs[-i]  # road without way down

    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
try:
    peak([])         # with this anomalous list we expect the excpetion 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
```

even

⊛⊛ RETURN a list containing the elements at even position, starting from zero which is considered even

- you can assume the input list always contains an even number of elements
- HINT: remember that `range` can take three parameters

[128]:
```
def even(xs):
    #jupman-raise
    ret = []
    for i in range(0,len(xs),2):
        ret.append(xs[i])
    return ret
    #/jupman-raise



# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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']
# TEST END
```

mix

⊛⊛ RETURN a NEW list in which the elements are taken in alternation from `lista` and `listb`

- you can assume that `lista` and `listb` contain the same number of elements

Example:

```
mix(['a', 'b','c'], ['x', 'y','z'])
```

must give

```
['a', 'x', 'b','y', 'c','z']
```

[129]:
```

def mix(lista, listb):
    #jupman-raise
    ret = []
    for i in range(len(lista)):
        ret.append(lista[i])
        ret.append(listb[i])
    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert mix([], []) == []
```

(continues on next page)

(continued from previous page)

```
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']
# TEST END
```

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

[130]:

```
def fill(lst1, lst2):

    #jupman-raise
    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
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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)
```

(continues on next page)

(continued from previous page)

```
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']
# TEST END
```

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 ny 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 rimove 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)`

[131]:
```
def nostop(s, stopwords):
    #jupman-raise
    words = s.split(" ")
    for s in stopwords:
        if s in words:
            words.remove(s)
    return " ".join(words)
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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"
# TEST END
```

threes

⊛⊛ To check if an integer is divisible for a number `n`, you can check the reminder of the integer division by `x` and `n` is equal to zero using the operator `%`:

[132]:
```
0 % 3
```

[132]:
```
0
```

[133]:
```
1 % 3
```

[133]:
```
1
```

[134]:
```
2 % 3
```

[134]:
```
2
```

[135]:
```
3 % 3
```

[135]:
```
0
```

[136]:
```
4 % 3
```

[136]:
```
1
```

[137]:
```
5 % 3
```

[137]:
```
2
```

[138]:
```
6 % 3
```

[138]:
```
0
```

Now implement the following function:

[139]:
```
def threes(lst):
    """ RETURN a NEW lst with the same elements of lst, except the ones at indeces␣
↪which are divisible by 3.
        In such cases, the output list will contain a the string 'z'
    """
    #jupman-raise
    ret = []
    for i in range(len(lst)):
        if i % 3 == 0:
            ret.append('z')
        else:
            ret.append(lst[i])
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert threes([]) == []
assert threes(['a']) == ['z']
assert threes(['a','b']) == ['z','b']
assert threes(['a','b','c']) == ['z','b','c']
assert threes(['a','b','c','d']) == ['z','b','c','z']
assert threes(['f','c','s','g','a','w','a','b']) == ['z','c','s','z','a','w','z','b']
# TEST END
```

list_to_int

Given a non-empty array 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 array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

Example:

```
Input:  [3,7,5]
Output: 375

Input:  [2,0]
Output: 20

Input:  [0]
Output: 0
```

list_to_int_dirty

⊛⊛ This is the totally dirty approach, but may be fun (never do this in real life - prefer instead the next `list_to_int_proper` approach).

1. convert the list to a string `'[5,7,4]'` using the function `str()`
2. remove from the string `[ , ']` and the commas `,` using the method `.replace(str1, str2)` which returns a NEW string with `str1` replaced for `str2`
3. convert the string to an integer using the special function `int()` and return it

[140]:
```
def list_to_int_dirty(lst):
    #jupman-raise
    s = str(lst)
    stripped = s.replace('[', '').replace(']','').replace(',','').replace(' ', '')
    n = int(stripped)
    return n
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert list_to_int_dirty([0]) == 0
assert list_to_int_dirty([1]) == 1
assert list_to_int_dirty([2]) == 2
assert list_to_int_dirty([92]) == 92
assert list_to_int_dirty([5,7,4]) == 574
# TEST END
```

list_to_int

⊛⊛ The proper way is to follow rules of math. To do it, keep 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 progressivly bigger power of 10, starting from $10^0 = 1$ up to $10^4 = 1000$.

To understand how it could work in Python, we might progressivly 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
```

(continues on next page)

(continued from previous page)

```
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!)

[141]:
```
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`:

[142]:
```
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 !

---

[143]:
```
def list_to_int(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.

        Example:

        Input:  [3,7,5]
        Output: 375

        Input:  [2,0]
        Output: 20

        Input:  [0]
        Output: 0

    """
    #jupman-raise
    power = 1
```

(continues on next page)

(continued from previous page)

```
    num = 0
    for digit in reversed(lst):
        num += power * digit
        power = power * 10
    return num
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert list_to_int([0]) == 0
assert list_to_int([1]) == 1
assert list_to_int([2]) == 2
assert list_to_int([92]) == 92
assert list_to_int([90]) == 90
assert list_to_int([5,7,4]) == 574
# END
```

int_to_list

⊛⊛ Let's now try the inverse operation, that is, going from a proper Python number like `574` to a list `[5,7,4]`

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 reminder of integer division between `547` and `10`:

[144]:
```
574 % 10
```

[144]:
```
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 `//`:

[145]:
```
574 // 10
```

[145]:
```
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`.

[146]:
```
def int_to_list(num):
    """ Takes an integer number >= 0 and RETURN a list of digits representing the␣
↪number in base 10.

        Example:

            Input:  375
            Output: [3,7,5]

            Input:  20
            Output: [2,0]
```

(continues on next page)

(continued from previous page)

```
            Input:  0
            Output: [0]

    """
    #jupman-raise
    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))
        #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert int_to_list(0) == [0]
assert int_to_list(1) == [1]
assert int_to_list(2) == [2]
assert int_to_list(92) == [9,2]
assert int_to_list(90) == [9,0]
assert int_to_list(574) == [5,7,4]
# TEST END
```

add one

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:

```
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`

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

[147]:
```
def add_one_conv(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.

        Implement by calling already used implemented functions.
    """
    #jupman-raise
    power = 1
    num = list_to_int(lst)

    return int_to_list(num + 1)
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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]
# TEST END
```

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:

```
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]
```

To implement it, directly operate on the list, using a `carry` variable (*riporto* in italian).

Just follow addition as done in elementary school. Start from the last digit and sum one:

If you get a number <= 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
```

Credits: inspiration taken from leetcode.com[177]

[148]:
```
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.

        Implement it using the carry method explained before.
    """

    #jupman-raise
    ret = []
    carry = 1
    for digit in reversed(lst):
```

(continues on next page)

[177] https://leetcode.com/tag/array/

(continued from previous page)

```
        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
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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]
# TEST END
```

collatz

Difficulty: ⊛⊛⊛

More challenging, implement this function from Montresor slides (the Collatz conjecture says that starting from any n you end up to 1):

The 3n +1 sequence is defined like this: given a number n , compute a new value for n as follow: if n is even, divide n by 2 . If n is odd, multiply it by 3 and add 1 . Stop when 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 D , such that for each value n between 1 and 50 , D [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[178]

[149]:
```
def collatz():
    """ Return D"""
    raise Exception("TODO IMPLEMENT ME !")
```

[ ]:
```

```

[178] https://www.dcode.fr/collatz-conjecture

6.4.6 Recursive operations

Here we deal with recursion. Before doing this, you might wait until doing Montresor class on recursion theory[179]

When we have a problem, we try to solve it by splitting its dimension in half (or more), look for solutions in each of the halves and then decide what to do with the found solutions, if any.

Several cases may occur:

1. No solution is found
2. One solution is found
3. Two solutions are found

case 1): we can only give up.

case 2): we have only one solution, so we can just return that one.

case 3): we have two solutions, so we need to decide what is the purpose of the algorithm.

Is the purpose to …

- find all possible solutions? Then we return both of them.
- find the *best* solution, according to some measure of 'goodness'? Then we measure each of the solutions and give back the highest scoring one.
- always provide a *combination* of existing solutions, according to some combination method? Then we combine the found solutions and give them back

gap_rec

⊛⊛ In a list L containing n≥2 integers, a gap is an index i, $0 < i < n$, such that $L[i-1] < L[i]$

If n≥2 and $L[0] < L[n-1]$, L contains at least one gap

Design an algorithm that, given a list L containing n≥2 integers such that $L[0] < L[n-1]$, finds a gap in the list.

Try to code and test the `gap` function. To avoid displaying directly Python, here we wrote it as pseudocode:

```
gap(int[] L, int i, int j)
if j == i + 1 then
    return j
m = ⌊(i + j)/2⌋
if L[m] ≤ L[i] then
    return gap(L, m, j)
else
    return gap(L, i, m)
```

Use the following skeleton to code it and add some test to the provided testcase class.To understand what's going on, try copy pasting in Python tutor[180]

Notice that

- We created a function `gap_rec` to differentiate it from the iterative one

[179] http://disi.unitn.it/~montreso/sp/handouts/A06-recursion.pdf
[180] http://pythontutor.com/visualize.html#mode=edit

- Users of `gap_rec` function might want to call it by passing just a list, in order to find any gap in the whole list. So for convenience the new function `gap_rec(L)` only accepts a list, without indexes `i` and `j`. This function just calls the other function `gap_rec_helper` that will actually contain the recursive calls. So your task is to translate the pseudocode of `gap` into the Python code of `gap_rec_helper`, which takes as input the array and the indexes as `gap` does. Adding a helper function is a frequent pattern you can find when programming recursive functions.

WARNING: The specification of gap_rec assumes the input is always a list of at least two elements, and that the first element is less or equal than the last one. If these conditions are not met, function behaviour could be completely erroneus!

When preconditions are not met, execution could stop because of an error like index out of bounds, or, even worse, we might get back some wrong index as a gap! To prevent misuse of the function, a good idea can be putting a check at the beginning of the `gap_rec` function. Such check should immediately stop the execution and raise an error if the parameters don't satisfy the preconditions. One way to do this could be to to some assertion[181] like this:

```
def gap_rec(L, i , j):
    assert len(L) >= 2
    assert L[0] <= L[len(L)-1]
```

- These commands will make python interrupt execution and throw an error as soon it detects list `L` is too small or with wrong values
- This kind of behaviour is also called *fail fast*, which is better than returning wrong values!
- You can put any condition you want after `assert`, but ideally they should be fast to execute.
- asserts might be better here than `raise Exception` constructs because asserts can be disabled with a flag passed to the interpreter. So, when you debug you can take advantage of them, and when the code is production quality and supposed to be bug free you can disable all assertions at once to gain in execution speed.

GOOD PRACTICE: Notice I wrote as a comment what the helper function is expected to receive. Writing down specs often helps understanding what the function is supposed to do, and helps users of your code as well!

COMMANDMENT 2: You shall also write on paper!

To get an idea of how `gap_rec` is working, draw histograms on paper like the following, with different heights at index `m`:

[181] https://sciprog.davidleoni.it/testing/testing.html#Assertions

Notice how at each recursive call, we end up with a histogram that is similar to the inital one, that is, it respects the same preconditions (a list of size >= 2 where first element is smaller or equal than the last one)

[150]:
```
def gap_rec(L, i, j):
    #jupman-raise
    if j == i+1:
        return j
    else:
        m = (i+j) // 2
        if L[m] < L[j]:
            return gaprec(L,m,j)
        else:
            return gaprec(L,i,m)
    #/jupman-raise

def gap(L):
    #jupman-raise
    return gap_rec(L, 0, len(L) - 1)
    #/jupman-raise

# try also to write asserts
```

6.4.7 Further resources

Have a look at leetcode array problems[182] sorting by *Acceptance* and *Easy*.

In particular, you may check:

- Contains Duplicate[183]
- Sort array by parity[184]
- Max consecutive ones[185]
- Fair Candy Swap[186]
- Move Zeros[187]
- Rotated Digits[188]
- Missing number[189] - has many possible solutions
- Find all numbers disappeared in an array[190]
- Majority Element[191]
- Degree of an array[192]
- Array partition 1[193] actually a bit hard but makes you think

[182] https://leetcode.com/tag/array/
[183] https://leetcode.com/problems/contains-duplicate/description/
[184] https://leetcode.com/problems/sort-array-by-parity/description/
[185] https://leetcode.com/problems/max-consecutive-ones/description/
[186] https://leetcode.com/problems/fair-candy-swap/description/
[187] https://leetcode.com/problems/move-zeroes/description/
[188] https://leetcode.com/problems/rotated-digits/description/
[189] https://leetcode.com/problems/missing-number/description/
[190] https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/description/
[191] https://leetcode.com/problems/majority-element/description/
[192] https://leetcode.com/problems/degree-of-an-array/description/
[193] https://leetcode.com/problems/array-partition-i/description/

```
[ ]:
```

6.5 Tuples solutions

6.5.1 Download exercises zip

Browse files online[194]

6.5.2 What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-sciprog.py
-exercises
     |- lists
         |- tuples.ipynb
         |- tuples-sol.ipynb
```

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 `tuples/tuples.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 instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

[194] https://github.com/DavidLeoni/sciprog-ds/tree/master/tuples

6.5.3 Introduction

References

- Andrea Passerini, A02 slides (lists and tuples)[195]
- Think Python, Chapter 10, Lists[196]
- Think Python, Chapter 12, Tuples[197]

Tuples are **immutable** sequences, so it is not possible to change their content without actually changing the object. They are sequential collections of objects, and elements of tuples are assumed to be in a particular order.

- Duplicates are allowed
- They can hold heterogeneous information.

Building tuples

Tuples are created with round brackets ()

Some examples:

[2]:
```
first_tuple = (1,2,3)
print(first_tuple)
```

```
(1, 2, 3)
```

[3]:
```
second_tuple = (1,) # this contains one element only, but we need the comma!
print(second_tuple, " type:", type(second_tuple))
```

```
(1,)  type: <class 'tuple'>
```

[4]:
```
var = (1) # This is not a tuple!!!
print(var, " type:", type(var))
```

```
1  type: <class 'int'>
```

[5]:
```
empty_tuple = () # fairly useless
print(empty_tuple, "\n")
```

```
()

```

[6]:
```
third_tuple = ("January", 1 ,2007) # heterogeneous info
print(third_tuple)
```

```
('January', 1, 2007)
```

[7]:
```
days = (third_tuple,("February",2,1998), ("March",2,1978),("June",12,1978))
print(days, "\n")
```

```
(('January', 1, 2007), ('February', 2, 1998), ('March', 2, 1978), ('June', 12, 1978))

```

Remember tuples are immutable objects…

[195] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A02-datastructures.pdf
[196] http://greenteapress.com/thinkpython2/html/thinkpython2011.html
[197] http://greenteapress.com/thinkpython2/html/thinkpython2013.html

[8]:
```
print("Days has id: ", id(days))
days = ("Mon","Tue","Wed","Thu","Fri","Sat","Sun")
```

```
Days has id:  140632243535944
```

…hence reassignment creates a new object

[9]:
```
print("Days now has id: ", id(days))
```

```
Days now has id:  140632252392016
```

Building from sequences

You can build a tuple from any sequence:

[10]:
```
tuple([8,2,5])
```

[10]:
```
(8, 2, 5)
```

[11]:
```
tuple("abc")
```

[11]:
```
('a', 'b', 'c')
```

Tuple operators

The following operators work on tuples and they behave exactly as on lists:

Result	Operator	Meaning
bool	=, !=	Check if two tuples are equal or different
int	len(tuple)	Return the length of the tuple
tuple	tuple + tuple	Concatenate two tuples (returns a new tuple)
tuple	tuple * int	Replicate the tuple (returns a tuple)
tuple	tuple[int]	Read an element of the tuple
tuple	tuple[int:int]	Extract a sub-tuple

[12]:
```
practical1 = ("Friday", "28/09/2018")
practical2 = ("Tuesday", "02/10/2018")
practical3 = ("Friday", "05/10/2018")

# A tuple containing 3 tuples
lectures = (practical1, practical2, practical3)
print("The first three lectures:\n", lectures, "\n")
```

```
The first three lectures:
 (('Friday', '28/09/2018'), ('Tuesday', '02/10/2018'), ('Friday', '05/10/2018'))
```

[13]:
```
# One tuple only
mergedLectures = practical1 + practical2 + practical3
print("mergedLectures:\n", mergedLectures)
```

```
mergedLectures:
 ('Friday', '28/09/2018', 'Tuesday', '02/10/2018', 'Friday', '05/10/2018')
```

[14]:
```
# This returns the whole tuple
print("1st lecture was on: ", lectures[0], "\n")
```

```
1st lecture was on:  ('Friday', '28/09/2018')
```

[15]:
```
# 2 elements from the same tuple
print("1st lecture was on ", mergedLectures[0], ", ", mergedLectures[1], "\n")
```

```
1st lecture was on  Friday ,  28/09/2018
```

[16]:
```
# Return type is tuple!
print("3rd lecture was on: ", lectures[2])
```

```
3rd lecture was on:  ('Friday', '05/10/2018')
```

[17]:
```
# 2 elements from the same tuple returned in tuple
print("3rd lecture was on ", mergedLectures[4:], "\n")
```

```
3rd lecture was on  ('Friday', '05/10/2018')
```

The following methods are available for tuples:

Return	Method	Meaning
int	list.count(obj)	Count the occurrences of an element
int	list.index(obj)	Return the index of the first occurrence of an object

[18]:
```
practical1 = ("Friday", "28/09/2018")
practical2 = ("Tuesday", "02/10/2018")
practical3 = ("Friday", "05/10/2018")


mergedLectures = practical1 + practical2 + practical3  # One tuple only
print(mergedLectures.count("Friday"), " lectures were on Friday")
print(mergedLectures.count("Tuesday"), " lecture was on Tuesday")

print("Index:", practical2.index("Tuesday"))
```

```
2  lectures were on Friday
1  lecture was on Tuesday
Index: 0
```

```
# not present in tuple, python will complain
print("Index:", practical2.index("Wednesday"))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-125-f7ecc5f7f5d6> in <module>
----> 1 print("Index:", practical2.index("Wednesday"))

ValueError: tuple.index(x): x not in tuple
```

Exercise: pet tuples

Given the string `pets = "siamese cat,dog,songbird,guinea pig,rabbit,hampster"`

1. convert it into a list.
2. create then a tuple of tuples where each tuple has two information: the name of the pet and the length of the name. E.g. (("dog",3), ("hampster",8)).
3. print the tuple

You should obtain:

```
['cat', 'dog', 'bird', 'guinea pig', 'rabbit', 'hampster']
(('cat', 3), ('dog', 3), ('bird', 4), ('guinea pig', 10), ('rabbit', 6), ('hampster',␣
↪8))
```

[19]:
```
pets = "cat,dog,bird,guinea pig,rabbit,hampster"

# write here
pet_list = pets.split(',')

print(pet_list)

pet_tuples = ((pet_list[0], len(pet_list[0])),
              (pet_list[1], len(pet_list[1])),
              (pet_list[2], len(pet_list[2])),
              (pet_list[3], len(pet_list[3])),
              (pet_list[4], len(pet_list[4])),
              (pet_list[5], len(pet_list[5])))

print(pet_tuples)
```

```
['cat', 'dog', 'bird', 'guinea pig', 'rabbit', 'hampster']
(('cat', 3), ('dog', 3), ('bird', 4), ('guinea pig', 10), ('rabbit', 6), ('hampster',␣
↪8))
```

Exercise: fruits

Given the string `S="apple|pear|apple|cherry|pear|apple|pear|pear|cherry|pear|strawberry"`. Store the elements separated by the `"|"` in a list.

1. How many elements does the list have?
2. Knowing that the list created at the previous point has only four distinct elements (i.e. `"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 how many times it appears in the original list). Ex. list_of_tuples = [("apple", 3), ("pear", "5"),…]. Here you can and should write code that only works with the given constant string, so there is no need for cycles.
3. Print the content of each tuple in a separate line (ex. first line: apple is present 3 times)

You should obtain:

```
['apple', 'pear', 'apple', 'cherry', 'pear', 'apple', 'pear', 'pear', 'cherry', 'pear
↪', 'strawberry']
[('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
```

[20]:
```
S="apple|pear|apple|cherry|pear|apple|pear|pear|cherry|pear|strawberry"

# write here

Slist = S.split("|")
print(Slist)

appleT = ("apple", Slist.count("apple"))
pearT = ("pear", Slist.count("pear"))
cherryT = ("cherry", Slist.count("cherry"))
strawberryT = ("strawberry", Slist.count("strawberry"))
list_of_tuples =[appleT, pearT, cherryT, strawberryT]

print(list_of_tuples, "\n") #adding newline to separate elements

print(appleT[0], " is present ", appleT[1], " times")
print(pearT[0], " is present ", pearT[1], " times")
print(cherryT[0], " is present ", cherryT[1], " times")
print(strawberryT[0], " is present ", strawberryT[1], " times")
```

```
['apple', 'pear', 'apple', 'cherry', 'pear', 'apple', 'pear', 'pear', 'cherry', 'pear
↪', 'strawberry']
[('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
```

Exercise: build a tuple

Given a tuple `x`, store in variable `y` another tuple containing the same elements as `x` *except* the last one_, and also the elements `d` and `e` appended at the end. Your code should work with *any* input `x`.

Example:

```
x = ('a','b','c')
```

after your code, you should get printed:

```
x = ('a', 'b', 'c')
y = ('a', 'b', 'd', 'e')
```

[21]:
```
x = ('a','b','c')

# write here
y = tuple(x[:-1]) + ('d','e')

print('x=',x)
print('y=',y)
```

```
x= ('a', 'b', 'c')
y= ('a', 'b', 'd', 'e')
```

6.5.4 Verify comprehension

ATTENTION

Following exercises require you to know:

Complex statements: Andrea Passerini slides A03[198]

Functions: Andrea Passerini slides A04[199]

Tests with asserts[200]: Following exercises contain automated tests to help you spot errors. To understand how to do them, read before Error handling and testing[201]

doubletup

⊛⊛ Takes as input a list with `n` integer numbers, and RETURN a NEW list which contains `n` tuples each with two elements. Each tuple contains a number taken from the corresponding position from original list, and its double

Example:

```
>>> doubletup([ 5, 3, 8])
[(5,10), (3,6), (8,16)]
```

[22]:
```
def doubletup(xs):
    #jupman-raise
    ret = []
```

(continues on next page)

[198] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[199] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf
[200] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-asserts
[201] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
    for x in xs:
        ret.append((x, x * 2))
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert doubletup([]) == []
assert doubletup([3]) == [(3,6)]
assert doubletup([2,7]) == [(2,4),(7,14)]
assert doubletup([5,3,8]) == [(5,10), (3,6), (8,16)]

# verify original list has not changed
la = [6]
lb = doubletup(la)
assert la == [6]
assert lb == [(6,12)]
# END TEST
```

6.6 Sets solutions

6.6.1 Download exercises zip

Browse files online[202]

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-sciprog.py
-exercises
     |-sets
         |- sets.ipynb
         |- sets-sol.ipynb
```

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), you will edit the files ending in `.py` files
- Go on reading this notebook, and follow instuctions inside.

[202] https://github.com/DavidLeoni/sciprog-ds/tree/master/sets

6.6.2 introduction

A set is an *unordered* collection of *distinct* elements, so no duplicates are allowed.

Creating a set

In Python you can create a set with a call to `set()`

[2]:
```
s = set()
```

[3]:
```
s
```

[3]:
```
set()
```

To add elements, use `.add()` method:

[4]:
```
s.add('hello')
s.add('world')
```

Notice Python represents a set with curly brackets, but differently from a dictionary you won't see colons `:` nor key/value couples:

[5]:
```
s
```

[5]:
```
{'hello', 'world'}
```

set from a sequence

You can create a set from any sequence, like a list. Doing so will eliminate duplicates present:

[6]:
```
set(['a','b','c','b','a','d'])
```

[6]:
```
{'a', 'b', 'c', 'd'}
```

Empty sets

WARNING: `{}` means empty dictionary, not empty set !

Since a set print out representation starts and ends with curly brackets as dictionaries, when you see written `{}` you might wonder whether that is the empty set or the empty dictionary.

The empty set is represented with `set()`

[7]:
```
s = set()
```

[8]:
```
s
```

[8]:
```
set()
```

[9]:
```
type(s)
```

[9]:
```
set
```

Instead, the empty dictionary is represented as a curly bracket:

[10]:
```
d = {}
```

[11]:
```
d
```

[11]:
```
{}
```

[12]:
```
type(d)
```

[12]:
```
dict
```

Iterating a set

You can iterate in a set with the `for in` construct:

[13]:
```
for el in s:
    print(el)
```

From the print out you notice sets, like dictionaries keys, are not necessarily iterated in same order as the insertion one. This also means they do not support access by index:

```
s[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-61-f8bb2b116405> in <module>()
----> 1 s[0]

TypeError: 'set' object does not support indexing
```

Adding twice

Since sets must contain distinct elements, if we add the same element twice the same remains unmodified with no complaints from Python:

[14]:
```
s.add('hello')
```

[15]:
```
s
```

[15]:
```
{'hello'}
```

[16]:
```
s.add('world')
```

[17]:
```
s
```

[17]:
```
{'hello', 'world'}
```

In a set we add eterogenous elements, like a numer here:

[18]:
```
s.add(7)
```

[19]:
```
s
```

[19]:
```
{7, 'hello', 'world'}
```

To remove an element, use `.remove()` method:

[20]:
```
s.remove('world')
```

[21]:
```
s
```

[21]:
```
{7, 'hello'}
```

Belonging to a set

To determine if an item belongs to a set you can use the usual 'in' operator as for any other sequence:

[22]:
```
'b' in set(['a','b','c','d'])
```

[22]:
```
True
```

[23]:
```
'z' in set(['a','b','c','d'])
```

[23]:
```
False
```

There is an important difference with other sequences such as lists, though: searching for an item in a set is always very fast, while searching in a list in the worst case requires Python to search the whole list.

There is a catch though: to get such performance you are obliged to only put in the set immutable data, such as numbers, strings, etc. If you try to add a mutable type like i.e. a list, you will get an error:

```
s = set()
s.add( ['a','b','c'] )
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-b345c7f28446> in <module>
----> 1 s.add( ['a','b','c'] )

TypeError: unhashable type: 'list'
```

Operations

You can perform set `.union(s2)`, `.intersection(s2)`, `.difference(s2)` ...

NOTE: set operations which don't have 'update' in the name create a NEW set each time!!!

[24]:
```
s1 = set(['a','b','c','d','e'])
print(s1)
```

```
{'a', 'e', 'c', 'b', 'd'}
```

[25]:
```
s2 = set(['b','c','f'])
```

[26]:
```
s3 = s1.intersection(s2)  # NOTE: it returns a NEW set !!!
print(s3)
```

```
{'c', 'b'}
```

[27]:
```
print(s1)  # did not change
```

```
{'a', 'e', 'c', 'b', 'd'}
```

updating sets

If you do want to change the original, you have to use `intersection_update`:

[28]:
```
s4 = set(['a','b','c','d','e'])
s5 = set(['b','c','f'])
res = s4.intersection_update(s5)  #NOTE: this MODIFIES s4 and thus return None !!!!
print(res)
```

```
None
```

[29]:
```
print(s4)
```

```
{'c', 'b'}
```

Exercise: set operators

Write some code that 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* `s1`,`s2`,`s3`.

With

```
s1 = set(['a','b','c','d','e'])
s2 = set(['b','c','f','g'])
s3 = set(['b','f'])
```

After you code you should get

```
{'d', 'a', 'c', 'g', 'e'}
```

[30]:
```
s1 = set(['a','b','c','d','e'])
s2 = set(['b','c','f','g'])
s3 = set(['b','f'])

# write here
s4 = s1.union(s2).difference(s3)
print(s4)
```

```
{'g', 'a', 'e', 'c', 'd'}
```

Exercise: dedup

Write some short code to create a `listb` which contains all elements from `lista` without duplicates and sorted alphabetically.

- MUST NOT change original lista
- no cycles allowed !
- your code should work with any `lista`

```
lista = ['c','a','b','c','d','b','e']
```

after your code, you should get

```
lista = ['c', 'a', 'b', 'c', 'd', 'b', 'e']
listb = ['a', 'b', 'c', 'd', 'e']
```

[31]:
```
lista = ['c','a','b','c','d','b','e']

# write here
s = set(lista)
listb = list(sorted(s))  # NOTE: sorted generates a NEW sequence
print("lista =",lista)
print("listb =",listb)
```

```
lista = ['c', 'a', 'b', 'c', 'd', 'b', 'e']
listb = ['a', 'b', 'c', 'd', 'e']
```

[ ]:
```

```

6.7 Dictionaries solutions

6.7.1 Download exercises zip

Browse files online[203]

6.7.2 What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-my_lib.py
-other stuff ...
-exercises
     |- lists
         |- dictionaries.ipynb
         |- dictionaries-sol.ipynb
         |- other stuff ..
```

[203] https://github.com/DavidLeoni/sciprog-ds/tree/master/dictionaries

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 `dictionaries/dictionaries.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 instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.7.3 Introduction

We will review dictionaries, discuss ordering issues for keys, and finally deal with nested dictionaries

Dict

First let's review Python dictionaries:

Dictionaries map keys to values. Keys must be *immutable* types such as numbers, strings, tuples (so i.e. no lists are allowed as keys), while values can be anything. In the following example, we create a dictionary `d` that initially maps from strings to numbers:

[2]:
```
# create empty dict:

d = dict()
d
```

[2]:
```
{}
```

[3]:
```
type( dict() )
```

[3]:
```
dict
```

Alternatively, to create a dictionary you can type `{}` :

[4]:
```
{}
```

[4]:
```
{}
```

[5]:
```
type( {} )
```

[5]:
```
dict
```

[6]:
```
# associate string "some key" to number 4
d['some key'] = 4
d
```

[6]:
```
{'some key': 4}
```

To access a value corresponding to a key, write this:

[7]:
```
d['some key']
```

[7]:
```
4
```

You can't associate mutable objects like lists:

```
d[ ['a', 'mutable', 'list', 'as key']  ] = 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-204-fb9d60c4e88a> in <module>()
----> 1 d[ ['a', 'mutable', 'list', 'as key']  ] = 3

TypeError: unhashable type: 'list'
```

But you can associate tuples:

[8]:
```
d[ ('an', 'immutable', 'tuple', 'as key')  ] = 3
d
```

[8]:
```
{('an', 'immutable', 'tuple', 'as key'): 3, 'some key': 4}
```

[9]:
```
# associate string "some other key" to number 7
d['some other key'] = 7
d
```

[9]:
```
{('an', 'immutable', 'tuple', 'as key'): 3, 'some key': 4, 'some other key': 7}
```

[10]:
```
# Dictionary is mutable, so you can reassign a key to a different value:
d['some key'] = 5
d
```

[10]:
```
{('an', 'immutable', 'tuple', 'as key'): 3, 'some key': 5, 'some other key': 7}
```

[11]:
```
# Dictionares are eterogenous, so values can be of different types:

d['yet another key'] = 'now a string!'
d
```

[11]:
```
{('an', 'immutable', 'tuple', 'as key'): 3,
 'some key': 5,
 'some other key': 7,
 'yet another key': 'now a string!'}
```

[12]:
```
# Keys also can be of eterogeneous types, but they *must* be of immutable types:
```

[13]:
```
d[123] = 'hello'
d
```

[13]:
```
{('an', 'immutable', 'tuple', 'as key'): 3,
 123: 'hello',
 'some key': 5,
 'some other key': 7,
 'yet another key': 'now a string!'}
```

To iterate through keys, use a 'for in' construct :

WARNING: iteration order most often is NOT the same as insertion order!!

[14]:
```
for k in d:
    print(k)
```

```
123
some key
some other key
('an', 'immutable', 'tuple', 'as key')
yet another key
```

Get all keys:

[15]:
```
d.keys()
```

[15]:
```
dict_keys([123, 'some key', 'some other key', ('an', 'immutable', 'tuple', 'as key'),
↪'yet another key'])
```

Get all values:

[16]:
```
d.values()
```

[16]:
```
dict_values(['hello', 5, 7, 3, 'now a string!'])
```

[17]:
```
# delete a key:

del d['some key']
d
```

[17]:
```
{('an', 'immutable', 'tuple', 'as key'): 3,
 123: 'hello',
 'some other key': 7,
 'yet another key': 'now a string!'}
```

6.7.4 Dictionary methods

Recall what seen in the lecture, the following methods are available for dictionaries:

Return	Method	Meaning
`list`	`dict.keys()`	Returns the list of the keys that are present in the dictionary
`list`	`dict.values()`	Returns the list of the values that are present in the dictionary
`list of tuples`	`dict.items()`	Returns the list of pairs (key, value) that are present in the dictionary

These methods are new to dictionaries and can be used to loop through the elements in them.

ATTENTION: `dict.keys()` returns a `dict_keys` object not a list. To cast it to list, we need to call `list(dict.keys())`.

Functions working on dictionaries

As for the other data types, python provides several operators that can be applied to dictionaries. The following operators are available and they basically work as in lists. The only exception being that the operator **in** checks whether the specified object is present among the **keys**.

Result	Operator	Meaning
`bool`	`obj in dict`	Return `True` if a key is present in the dictionary
`int`	`len(dict)`	Return the number of elements in the dictionary
`obj`	`dict[obj]`	Read the value associate with a key
	`dict[obj] = obj`	Add or modify the value associated with a key

Exercise print key

⊛ PRINT the value of key `'b'`, that is, `2`

[18]:
```
d = {'a':6, 'b':2,'c':5}

# write here

d['b']
```

[18]:
```
2
```

Exercise modify dictionary

⊛ MODIFY the dictionary, by substituting the key `c` with `8`. Then PRINT the dictionary

NOTE: the order in which couples key/value are printed is NOT relevant!

[19]:
```
d = {'a':6, 'b':2, 'c':5}

# write here

d['c'] = 8
print(d)
```

```
{'c': 8, 'a': 6, 'b': 2}
```

Exercise print keys

⊛ PRINT a sequence with all the keys, using the appropriate method of dictionaries

[20]:
```
d = {'a':6, 'b':2,'c':5}

# write here

d.keys()
```

[20]:
```
dict_keys(['c', 'a', 'b'])
```

Exercise print dimension

⊛ PRINT the number of couples key/value in the dictionary

[21]:
```
d = {'a':6, 'b':2, 'c':5}

# write here

print(len(d))
```

```
3
```

Exercise print keys as list

⊛ PRINT a LIST with all the keys in the dictionary

- **NOTE 1**: it is NOT necessary that the list is ordered
- **NOTE 2**: to convert any sequence to a list, use the predefined function `list`

[22]:
```
d = {'a':6, 'b':2,'c':5}

# write here

list(d.keys())
```

[22]:
```
['c', 'a', 'b']
```

Exercise ordered keys

⊛ PRINT an ordered LIST holding all dictionary keys

- **NOTE 1**: now it is necessary for the list to be ordered
- **NOTE 2**: to convert any sequence to a list, use the predefined function `list`

[23]:
```
d = {'a':6, 'c':2,'b':5}

# write here

my_list = list(d.keys())
my_list.sort()  # REMEMBER: sort does NOT return anything !!!
print(my_list)
```

```
['a', 'b', 'c']
```

OrderedDict

As we said before, when you scan the keys of a dictionary, the order most often is **not** the same as the insertion order. To have it predictable, you need to use an `OrderedDict`

[24]:
```
# first you need to import it from collections module
from collections import OrderedDict

od = OrderedDict()

# OrderedDict looks and feels exactly as regular dictionaries. Here we reproduce the␣
↪previous example:

od['some key'] = 5

od['some other key'] = 7
od[('an', 'immutable', 'tuple','as key')] = 3
od['yet another key'] = 'now a string!'
od[123] = 'hello'
od
```

[24]:
```
OrderedDict([('some key', 5),
             ('some other key', 7),
             (('an', 'immutable', 'tuple', 'as key'), 3),
             ('yet another key', 'now a string!'),
             (123, 'hello')])
```

Now you will see that if you iterate with the `for in` construct, you get exactly the same insertion sequence:

[25]:
```
for key in od:
    print("%s  :  %s" %(key, od[key]))
```

```
some key  :  5
some other key  :  7
('an', 'immutable', 'tuple', 'as key')  :  3
yet another key  :  now a string!
123  :  hello
```

To create it all at once, since you want to be sure of the order, you can pass a list of tuples representing key/value pairs. Here we reproduce the previous example:

[26]:
```
od = OrderedDict(
        [
            ('some key', 5),
            ('some other key', 7),
            (('an', 'immutable', 'tuple','as key'), 3),
            ('yet another key', 'now a string!'),
            (123, 'hello')
        ]
)

od
```

[26]:
```
OrderedDict([('some key', 5),
             ('some other key', 7),
             (('an', 'immutable', 'tuple', 'as key'), 3),
             ('yet another key', 'now a string!'),
             (123, 'hello')])
```

Again you will see that if you iterate with the `for in` construct, you get exactly the same insertion sequence:

[27]:
```
for key in od:
    print("%s  :  %s" % (key, od[key]))
```

```
some key  :  5
some other key  :  7
('an', 'immutable', 'tuple', 'as key')  :  3
yet another key  :  now a string!
123  :  hello
```

Exercise: OrderedDict phonebook

Write some short code that given three tuples, like the following, prints an OrderedDict which associates names to phone numbers, in the order they are proposed above.

- Your code should work with *any* tuples.
- Don't forget to import the OrderedDict from collections

Example:

```
t1 = ('Alice', '143242903')
t2 = ('Bob', '417483437')
t3 = ('Charles', '423413213')
```

after your code should give:

```
OrderedDict([('Alice', '143242903'), ('Bob', '417483437'), ('Charles', '423413213')])
```

[28]:
```
# first you need to import it from collections module
from collections import OrderedDict

t1 = ('Alice', '143242903')
t2 = ('Bob', '417483437')
t3 = ('Charles', '423413213')

# write here
od = OrderedDict([t1, t2, t3])
print(od)
```

```
OrderedDict([('Alice', '143242903'), ('Bob', '417483437'), ('Charles', '423413213')])
```

Exercise: OrderedDict copy

Given an OrderedDict `od1` containing translations English -> Italian, create a NEW OrderedDict called `od2` which contains the same translations as the input one PLUS the translation `'water':'acqua'`.

- **NOTE 1**: your code should work with any input ordered dict
- **NOTE 2**: `od2` MUST hold a NEW OrderedDict !!

Example:

With

```
od1 = OrderedDict()
od1['dog'] = 'cane'
od1['home'] = 'casa'
od1['table'] = 'tavolo'
```

after your code you should get:

```
>>> print(od1)
OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo')])
>>> print(od2)
OrderedDict([('dog', 'cane'), ('home', 'casa'), ('table', 'tavolo'), ('water', 'acqua
↪')])
```

[29]:
```
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')])
```

List of nested dictionaries

Suppose you have a list of dictionaries which represents a database of employees. Each employee is represented by a dictionary:

```
{
    "name":"Mario",
    "surname": "Rossi",
    "age": 34,
    "company": {
                    "name": "Candy Apples Inc.",
                    "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": "Candy Apples Inc.",
        "sector":"Food"
    }
```

[30]:
```
employees_db = [
  {
    "name":"Mario",
    "surname": "Rossi",
    "age": 34,
    "company": {
                    "name": "Candy Apples Inc.",
                    "sector":"Food"
                }
  },
  {
    "name":"Pippo",
    "surname": "Rossi",
    "age": 20,
    "company": {
                     "name": "Batworks",
                         "sector":"Clothing"
                }
  },
  {
    "name":"Paolo",
    "surname": "Bianchi",
    "age": 25,
    "company": {
                     "name": "Candy Apples Inc.",
                     "sector":"Food"
                }
  }

]
```

Exercise: print employees

Write some code to print all employee names and surnames from the above `employees_db`

You can assume `employees_db` has exactly 3 employees (so `for` cycle is not even needed)

You should obtain:

```
Mario Rossi
Pippo Rossi
Paolo Bianchi
```

[31]:
```
# write here
print(employees_db[0]["name"], employees_db[0]["surname"])
print(employees_db[1]["name"], employees_db[1]["surname"])
print(employees_db[2]["name"], employees_db[2]["surname"])
```

```
Mario Rossi
Pippo Rossi
Paolo Bianchi
```

Exercise: print company names

Write some code to print all company names and sector from the above `employees_db`, without duplicating them. Pay attention to sector lowercase name.

You can assume `employees_db` has exactly 3 employees (so `for` cycle is not even needed)

[32]:
```
# write here
print(employees_db[0]["company"]["name"], "is a", employees_db[0]["company"]["sector
↪"].lower(), "company")
print(employees_db[1]["company"]["name"],  "is a", employees_db[1]["company"]["sector
↪"].lower(), "company")
```

```
Candy Apples Inc. is a food company
Batworks is a clothing company
```

6.7.5 Exercises with functions

ATTENTION

Following exercises require you to know:

Complex statements: Andrea Passerini slides A03[204]

Functions: Andrea Passerini slides A04[205]

print_val

⊛ Write the function `print_val(d, key)` which RETURN the value associated to `key`

```
>>> x = print_val({'a':5,'b':2}, 'a')
>>> x
5
>>> y = print_val({'a':5,'b':2}, 'b')
>>> y
2
```

[33]:
```
# write here

def print_val(d, key):
    return d[key]

#x = print_val({'a':5,'b':2}, 'a')
#x
```

[204] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[205] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf

has_key

Write a function `has_key(d,key)` which PRINTS `"found"` if `diz` contains the key `key`, otherwise PRINTS `"not found"`

```
>>> has_key({'a':5,'b':2}, 'a')
found
>>> has_key({'a':5,'b':2}, 'z')
not found
```

[34]:
```
# write here

def has_key(d, key):
    if key in d:
        print("found")
    else:
        print("not found")


#has_key({'a':5,'b':2}, 'a')
#has_key({'a':5,'b':2}, 'b')
#has_key({'a':5,'b':2}, 'z')
```

dim

⊛ Write a function `dim(d)` which RETURN the associations key-value present in the dictionary

```
>>> x = dim({'a':5,'b':2,'c':9})
>>> x
3
```

[35]:
```
# write here

def dim(d):
    return len(d)

#x = dim({'a':5,'b':2,'c':9})
#x
```

keyring

⊛ Given a dictionary, write a function `keyring` which RETURN an ORDERED LIST with all the keys, una at a time

NOTE: the order of keys in this list IS important !

```
>>> x = keyring({'a':5,'c':2,'b':9})
>>> x
['a','b','c']
```

[36]:
```
# write here

def keyring(d):
```

(continues on next page)

(continued from previous page)

```
    my_list = list(d.keys())
    my_list.sort()  # REMEMBER: .sort() does NOT return anything !!
    return my_list


#x = keyring({'a':5,'c':2,'b':9})
#x
```

couples

⊛ Given a dictionary, write a function `couples` which PRINTS all key/value couples, one per row

NOTE: the order of the print is *NOT* important, it si enough to print all couples !

```
>>> couples({'a':5,'b':2,'c':9})
a 5
c 9
b 2
```

[37]:
```
# write here

def couples(d):
    for key in d:
        print(key,d[key])

#couples({'a':5,'b':2,'c':9})
```

6.7.6 Verify comprehension

ATTENTION

Following exercises require you to know:

Complex statements: Andrea Passerini slides A03[206]

Functions: Andrea Passerini slides A04[207]

Tests with asserts[208]: Following exercises contain automated tests to help you spot errors. To understand how to do them, read before Error handling and testing[209]

histogram

⊛⊛ RETURN a dictionary that for each character in string contains the number of occurrences. The keys are the caracthers and the values are to occurrences

[38]:
```

def histogram(string):
```

(continues on next page)

[206] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[207] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf
[208] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-asserts
[209] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
    #jupman-raise
    ret = dict()
    for c in string:
        if c in ret:
            ret[c] += 1
        else:
            ret[c] = 1
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

assert histogram("babbo") == {'b': 3, 'a':1, 'o':1}
assert histogram("") == {}
assert histogram("cc") == {'c': 2}
assert histogram("aacc") == {'a': 2, 'c':2}
# TEST END
```

listify

⊛⊛ Takes a dictionary `d` as input and RETURN a LIST with only the values from the dict (so no keys)

To have a predictable order, the function also takes as input a list `order` where there are the keys from first dictionary ordered as we would like in the resulting list

[39]:
```
def listify(d, order):
    #jupman-raise
    ret = list()
    for element in order:
        ret.append (d[element])
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

assert listify({}, []) == []
assert listify({'ciao':123}, ['ciao']) == [123]
assert listify({'a':'x','b':'y'}, ['a','b']) == ['x','y']
assert listify({'a':'x','b':'y'}, ['b','a']) == ['y','x']
assert listify({'a':'x','b':'y','c':'x'}, ['c','a','b']) == ['x','x','y']
assert listify({'a':'x','b':'y','c':'x'}, ['b','c','a']) == ['y','x','x']
assert listify({'a':5,'b':2,'c':9}, ['b','c','a']) == [2,9,5]
assert listify({6:'x',8:'y',3:'x'}, [6,3,8]) == ['x','x','y']
# TEST END
```

tcounts

⊛⊛ Takes a list of tuples. Each tuple has two values, the first is an immutable object and the second one is an integer number (the counts of that object). RETURN a dictionary that for each immutable object found in the tuples, associate the total count found for it.

See asserts for examples

[40]:
```
def tcounts(lst):
    ret = {}
    for c in lst:
        if c[0] in ret:
            ret[c[0]] += c[1]
        else:
            ret[c[0]] = c[1]
    return ret

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert tcounts([]) == {}
assert tcounts([('a',3)]) == {'a':3}
assert tcounts([('a',3),('a',4)]) == {'a':7}
assert tcounts([('a',3),('b',8), ('a',4)]) == {'a':7, 'b':8}
assert tcounts([('a',5), ('c',8), ('b',7), ('a',2), ('a',1), ('c',4)]) == {'a':5+2+1,
↪'b':7, 'c': 8 + 4}
# TEST END
```

inter

⊛⊛ Write a function `inter(d1,d2)` which takes two dictionaries and RETURN a SET of keys for which the couple is the same in both dictionaries

Example

```
>>> a = {'key1': 1, 'key2': 2 , 'key3': 3}
>>> b = {'key1': 1 ,'key2': 3 , 'key3': 3}
>>> inter(a,b)
{'key1','key3'}
```

[41]:
```

def inter(d1, d2):
    #jupman-raise
    res = set()
    for key in d1:
        if key in d2:
            if d1[key] == d2[key]:
                res.add(key)
    return res
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert inter({'key1': 1, 'key2': 2 , 'key3': 3}, {'key1':1 ,'key2':3 , 'key3':3}) == {
↪'key1', 'key3'}
```

(continues on next page)

(continued from previous page)

```
assert inter(dict(), {'key1':1 ,'key2':3 , 'key3':3}) == set()
assert inter({'key1':1 ,'key2':3 , 'key3':3}, dict()) == set()
assert inter(dict(),dict()) == set()
# TEST END
```

unique_vals

⊛⊛ Write a function `unique_vals(d)` which RETURN a list of unique values from the dictionary. The list MUST be ordered alphanumerically

Question: We need it ordered for testing purposes. Why?

- to order the list, use method .sort()

Example:

```
>>> unique_vals({'a':'y','b':'x','c':'x'})

['x','y']
```

[42]:
```
def unique_vals(d):
    #jupman-raise
    s = set(d.values())
    ret = list(s)  # we can only sort lists (sets have no order)
    ret.sort()
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert unique_vals({}) == []
assert unique_vals({'a':'y','b':'x','c':'x'}) == ['x','y']
assert unique_vals({'a':4,'b':6,'c':4,'d':8}) == [4,6,8]
# TEST END
```

uppers

⊛⊛ Takes a list and RETURN a dictionary which associates to each string in the list the same string but with all characters uppercase

Example:

```
>>> uppers(["ciao", "mondo", "come va?"])
{"ciao":"CIAO", "mondo":"MONDO", "come va?":"COME VA?"}
```

Ingredients:

- for cycle
- `.upper()` method

[43]:
```

def uppers(xs):
    #jupman-raise
```

(continues on next page)

(continued from previous page)

```
    d = {}
    for s in xs:
        d[s] = s.upper()
    return d
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert uppers([]) == {}
assert uppers(["ciao"]) == {"ciao":"CIAO"}
assert uppers(["ciao", "mondo"]) == {"ciao":"CIAO", "mondo":"MONDO"}
assert uppers(["ciao", "mondo", "ciao"]) == {"ciao":"CIAO", "mondo":"MONDO"}
assert uppers(["ciao", "mondo", "come va?"]) == {"ciao":"CIAO", "mondo":"MONDO",
↪"come va?":"COME VA?"}
# TEST END
```

filtraz

⊛⊛ RETURN a NEW dictionary, which contains only the keys key/value of the dictionary `d` in input in which in the key is present the character `'z'`

Example:

```
filtraz({'zibibbo':'to drink',
         'mc donald': 'to avoid',
         'liquirizia': 'ze best',
         'burger king': 'zozzerie'
})
```

must RETURN the NEW dictionary

```
{
'zibibbo':'da bere',
'liquirizia': 'ze best'
}
```

In other words, we only kept those keys which contained at least a `z`. We do not care about `z` in values.

Ingredients:

To check if `z` is in the key, use the operator `in`, for example

```
'z' in 'zibibbo' == True
'z' in 'mc donald' == False
```

[44]:
```
def filtraz(diz):
    #jupman-raise

    ret = {}
    for chiave in diz:
        if 'z' in chiave:
            ret[chiave] = diz[chiave]
    return ret
```

(continues on next page)

(continued from previous page)

```
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert filtraz({}) == {}
assert filtraz({'az':'t'}) == {'az':'t'}
assert filtraz({'zc':'w'}) == {'zc':'w'}
assert filtraz({'b':'h'}) == {}
assert filtraz({'b':'hz'}) == {}
assert filtraz({'az':'t','b':'hz'}) == {'az':'t'}
assert filtraz({'az':'t','b':'hz','zc':'w'}) == {'az':'t', 'zc':'w'}
# TEST END
```

powers

⊛⊛ RETURN a dictionary in which keys are integer numbers from `1` to `n` included, and respective values are the sqaures of the keys.

Example:

```
powers(3)
```

should return:

```
{
 1:1,
 2:4,
 3:9
}
```

[45]:

```
def powers(n):
    #jupman-raise
    d=dict()
    for i in range(1,n+1):
        d[i]=i**2
    return d
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert powers(1) == {1:1}
assert powers(2) == {
                        1:1,
                        2:4
                    }
assert powers(3) == {
                        1:1,
                        2:4,
                        3:9
```

(continues on next page)

(continued from previous page)

```
                    }

assert powers(4) == {
                        1:1,
                        2:4,
                        3:9,
                        4:16
                    }
# TEST END
```

dilist

⊛⊛ RETURN a dictionary with `n` couples key-value, where the keys are integer numbers from `1` to `n` included, and to each key `i` is associated a list of numbers from `1` to `i`.

NOTE: the keys are *integer numbers*, NOT strings !!!!

Example

```
>>> dilist(3)
{
    1:[1],
    2:[1,2],
    3:[1,2,3]
}
```

[46]:

```
def dilist(n):
    #jupman-raise
    ret = dict()
    for i in range(1,n+1):
        lista = []
        for j in range(1,i+1):
            lista.append(j)
        ret[i] = lista
    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert dilist(0) == dict()
assert dilist(1) == {
                        1:[1]
                    }
assert dilist(2) == {
                        1:[1],
                        2:[1,2]
                    }
assert dilist(3) == {
                        1:[1],
                        2:[1,2],
                        3:[1,2,3]
```

(continues on next page)

(continued from previous page)

```
                        }
# TEST END
```

prefixes

⊛⊛ Write a function`prefixes` which given

a dictionary `d` having as keys italian provincies and as values their phone numbers (note: prefixes are also strings !) - a list `provinces` with the italian provinces

RETURN a list of prefixes corresponding to provinces of given list.

Example:

```
>>> prefissi({
                'tn':'0461',
                'bz':'0471',
                'mi':'02',
                'to':'011',
                'bo':'051'
              },
              ['tn','to', 'mi'])

['0461', '011', '02']
```

HINTS:

- intialize an empty list to return
- go through provinces list and take corresponding prefixes from the dictionary

[47]:
```
def prefixes(d, provinces):

    #jupman-raise
    ret = []
    for province in provinces:
        ret.append(d[province])

    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert prefixes({'tn':'0461'}, []) == []
assert prefixes({'tn':'0461'}, ['tn']) == ['0461']
assert prefixes({'tn':'0461', 'bz':'0471'}, ['tn']) == ['0461']
assert prefixes({'tn':'0461', 'bz':'0471'}, ['bz']) == ['0471']
assert prefixes({'tn':'0461', 'bz':'0471'}, ['tn','bz']) == ['0461', '0471']
assert prefixes({'tn':'0461', 'bz':'0471'}, ['bz','tn']) == ['0471', '0461']
assert prefixes({'tn':'0461',
                 'bz':'0471',
                 'mi':'02',
                 'to':'011',
                 'bo':'051'
                },
```

(continues on next page)

(continued from previous page)

```
                ['tn','to', 'mi']) == ['0461', '011', '02']
# TEST END
```

Managers

Let's look at this `managers_db` data structure. It is a list of dictionaries of managers.

- Each manager supervises a department, which is also represented as a dictionary.
- Each department can stay either in building `"A"` or building `"B"`

[48]:
```
managers_db = [
  {
    "name":"Diego",
    "surname": "Zorzi",
    "age": 34,
    "department": {
                    "name": "Accounting",
                    "budget":20000,
                    "building":"A"
                  }
  },
  {
    "name":"Giovanni",
    "surname": "Tapparelli",
    "age": 45,
    "department": {
                    "name": "IT",
                    "budget":10000,
                    "building":"B"
                  }
  },
  {
    "name":"Sara",
    "surname": "Tomasi",
    "age": 25,
    "department": {
                    "name": "Human resources",
                    "budget":30000,
                    "building":"A"
                  }
  },
  {
    "name":"Giorgia",
    "surname": "Tamanin",
    "age": 28,
    "department": {
                    "name": "R&D",
                    "budget":15000,
                    "building":"A"
                  }
  },
  {
    "name":"Paola",
    "surname": "Guadagnini",
    "age": 30,
```

(continues on next page)

(continued from previous page)

```
    "department": {
                    "name": "Public relations",
                    "budget":40000,
                    "building":"B"
                  }
  }

]
```

managers: extract_managers

⊛⊛ RETURN the names of the managers in a list

[49]:
```
def extract_managers(db):
    #jupman-raise
    ret = []
    for d in db:
        ret.append(d["name"])
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
assert extract_managers([]) == []

# if it doesn't find managers_db, remember to executre the cell above which defins it␣
↪!
assert extract_managers(managers_db) == ['Diego', 'Giovanni', 'Sara', 'Giorgia',
↪'Paola']
# TEST END
```

managers: extract_departments

⊛⊛ RETURN the names of departments in a list.

[50]:
```
def extract_departments(db):
    #jupman-raise
    ret = []
    for d in db:
        ret.append(d["department"]["name"])

    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

assert extract_departments([]) == []
```

(continues on next page)

(continued from previous page)

```
# if it doesn't find managers_db, remember to execute the cell above which defins it !
assert extract_departments(managers_db) == ["Accounting", "IT", "Human resources","R&D
↪", "Public relations"]
# TEST END
```

managers: avg_age

⊛⊛ RETURN the average age of managers

[51]:

```
def avg_age(db):
    #jupman-raise
    s = 0
    for d in db:
        s += d["age"]

    return s / len(db)
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

# 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 + 45 + 25 + 28 + 30) / 5)
# TEST END
```

managers: buildings

⊛⊛ RETURN the buildings the departments belong to, WITHOUT duplicates !!!

[52]:

```
def buildings(db):
    #jupman-raise
    ret = []
    for d in db:
        building = d["department"]["building"]
        if building not in ret:
            ret.append(building)

    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

assert buildings([]) == []
assert buildings(managers_db) == ["A", "B"]
# TEST END
```

medie

⊛⊛ 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}
    ])
[ 76.0 , 73.5, 80.5 ]
```

which corresponds to

```
[ (70+82)/2 , (73+74)/2, (75+86)/2 ]
```

[53]:
```
def averages(lista):
    ret = [0.0, 0.0, 0.0]

    for i in range(len(lista)):
        ret[i] = (lista[i]['V'] + lista[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 nubers 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

    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
```

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 the 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:

```
ha_pref({
            'aldo':['cinema', 'musica', 'sport'],
            'giovanni':['musica'],
            'giacomo':['cinema', 'videogiochi']

        }, 'aldo', 'musica')
```

deve ritornare `True` perchè ad `aldo` piace la musica, invece

```
has_pref({'aldo':['cinema', 'music', 'sport'],
         'giovanni':['music'],
         'giacomo':['cinema', 'videogames']

        }, 'giacomo', 'sport')
```

Must return `False` because Giacomo does not like sport

[54]:

```
def has_pref(d, name, pref):
    #jupman-raise
    if name in d:
        return pref in d[name]
    else:
        return False
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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'],
```

(continues on next page)

(continued from previous page)

```
                    'giovanni':['music'],
                    'giacomo':['cinema', 'videogames']
                  }, 'giacomo', 'sport') == False
# TEST END
```

[ ]:

```

```

6.8 Control flow solutions

6.8.1 Download exercises zip

Browse files online[210]

6.8.2 Introduction

In this practical we will work with conditionals (branching) and loops.

References:

- Complex statements: Andrea Passerini slides A03[211]

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-exercises
     |- lists
         |- control-flow.ipynb
         |- control-flow-sol.ipynb
```

WARNING 1: 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 `control-flow/control-flow.ipynb`

WARNING 2: DO NOT use the *Upload* button in Jupyter, instead navigate to the unzipped folder while in Jupyter browser!

- Go on reading that notebook, and follow instuctions 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`

[210] https://github.com/DavidLeoni/sciprog-ds/tree/master/control-flow
[211] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf

- to execute Python code inside a Jupyter cell AND a create a new cell aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.8.3 Execution flow

Recall from the lecture that there are at least three types of execution flows. Our statements can be simple and structured **sequentially**, when one instruction is executed right after the previous one, but some more complex flows involve **conditional** branching (when the portion of the code to be executed depends on the value of some condition), or **loops** when a portion of the code is executed multiple times until a certain condition becomes False.

These portions of code are generally called **blocks** and Python, unlike most of the programming languages, **uses indentation (and some keywords like else, ':', 'next', etc.) to define blocks**.

6.8.4 Conditionals

We can use conditionals any time a decision needs to be made depending on the value of some condition. A block of code will be executed if the condition is evaluated to the boolean **True** and another one if the condition is evaluated to **False**.

The basic *if - else* statement

The basic syntax of conditionals is an if statement like:

```
if condition :

    # This is the True branch
    # do something

else:

    # This is the False branch (or else branch)
    # do something else
```

where *condition* is a boolean expression that tells the interpreter which of the two blocks should be executed. **If and only if** the condition is **True** the first branch is executed, otherwise execution goes to the second branch (i.e. the else branch). Note that the **condition is followed by a ":"** character and that **the two branches are indented**. This is the way Python uses to identify the block of instructions that belong to the same branch. The **else keyword is followed by ":"** and **is**

not indented (i.e. it is at the same level of the *if* statement. There is no keyword at the end of the "else branch", but **indentation tells when the block of code is finished**.

Example: Let's get an integer from the user and test if it is even or odd, printing the result to the screen.

```
print("Dear user give me an integer:")
num = int(input())
res = ""
if num % 2 == 0:
    #The number is even
    res = "even"
else:
    #The number is odd
    res = "odd"

print("Number ", num, " is ", res)
```

```
Dear user give me an integer:
34
Number  34  is  even
```

Note that the execution is sequential until the *if* keyword, then it branches until the indentation goes back to the same level of the if (i.e. the two branches rejoin at the *print* statement in the final line). **Remember that the else branch is optional.**

The *if - elif - else* statement

If statements can be chained in such a way that there are more than two possible branches to be followed. Chaining them with the **if - elif - else** statement will make execution follow only one of the possible paths.

The syntax is the following:

```
if condition :

    # This is branch 1
    # do something

elif condition1 :

    # This is branch 2
    # do something

elif condition2 :

    # This is branch 3
    # do something

else:

    # else branch. Executed if all other conditions are false
    # do something else
```

Note that **branch 1** is executed if condition is **True**, **branch 2** if and only if **condition is False and condition1 is True**, **branch 3** if condition is **False, condition 1 is False and condition2 is True**. **If all conditions are False the else branch is executed**.

Example: The tax rate of a salary depends on the income. If the income is < 10000 euros, no tax is due, if the income is between 10000 euros and 20000 the tax rate is 25%, if between 20000 and 45000 it is 35% otherwise it is 40%. What

is the tax due by a person earning 35000 euros per year?

[1]:
```
income = 35000
rate = 0.0

if income < 10000:
    rate = 0
elif income < 20000:
    rate = 0.2
elif income < 45000:
    rate = 0.35
else:
    rate = 0.4

tax = income*rate

print("The tax due is ", tax, " euros (i.e ", rate*100, "%)")
```

```
The tax due is  12250.0  euros (i.e  35.0 %)
```

Note the difference in the two following cases:

[2]:
```
#Example 1

val = 10

if val > 5:
    print("Value >5")
elif val > 5:
    print("I said value is >5!")
else:
    print("Value is <= 5")
```

```
Value >5
```

[3]:
```
#Example 2

val = 10

if(val > 5):
    print("\n\nValue is >5")

if(val > 5):
    print("I said Value is >5!!!")
```

```


Value is >5
I said Value is >5!!!
```

Nested ifs

If statements are *blocks* so they can be nested as any other block.

If you have a point with coordinates `x` and `y` and you want to know into which quadrant it falls

You might write something like this:

[4]:
```
x = 5
y = 9

if x >= 0:
    if y >= 0:
        print('first quadrant')
    else:
        print('fourth quadrant')
else:
    if y >= 0:
        print('second quadrant')
    else:
        print('third quadrant')
```

```
first quadrant
```

an equivalent way could be to use boolean expressions and write:

[5]:
```
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')
```

```
first quadrant
```

Ternary operator

In some cases it is handy to be able to initialize a variable depending on the value of another one.

Example:

The discount rate applied to a purchase depends on the amount of the sale. Create a variable *discount* setting its value to 0 if the variable *amount* is lower than 100 euros, to 10% if it is higher.

[6]:
```
amount = 110
discount = 0

if(amount >100):
    discount = 0.1
else:
    discount = 0 # not necessary

print("Total amount:", amount, "discount:", discount)
```

```
Total amount: 110 discount: 0.1
```

The previous code can be written more coincisely as:

[7]:
```
amount = 110
discount = 0.1 if amount > 100 else 0
print("Total amount:", amount, "discount:", discount)
```

```
Total amount: 110 discount: 0.1
```

The basic syntax of the ternary operator is:

```
variable = value if condition else other_value
```

meaning that the *variable* is initialized to *value* if the *condition* holds, otherwise to *other_value*.

Python also allows in line operations separated by a ";"

[8]:
```
a = 10; b = a + 1; c = b +2
print(a,b,c)
```

```
10 11 13
```

Note: Although the ternary operator and in line operations are sometimes useful and less verbose than the explicit definition, they are considered "non-pythonic" and advised against.

6.8.5 Loops

Looping is the ability of repeating a specific block of code several times (i.e. until a specific condition is True or there are no more elements to process).

For loop

The *for* loop is used to loop over a collection of objects (e.g. a string, list, tuple, …). The basic syntax of the for loop is the following:

```
for elem in collection :
    # OK, do something with elem
    # instruction 1
    # instruction 2
```

the variable `elem` will get the value of each one of the elements present in `collection`one after the other. The end of the block of code to be executed for each element in the collection is again defined by indentation.

Depending on the type of the collection elem will get different values. Recall from the lecture that:

str	**for** iterates over the characters
list	**for** iterates over the elements
tuple	**for** iterates over the elements
dict	**for** iterates over the keys

Let's see this in action:

[9]:
```
S = "Hi there from python"
Slist = S.split(" ")
Stuple = ("Hi","there","from","python")
print("String:", S)
print("List:", Slist)
print("Tuple:", Stuple)
```

```
String: Hi there from python
List: ['Hi', 'there', 'from', 'python']
Tuple: ('Hi', 'there', 'from', 'python')
```

[10]:
```
#for loop on string
print("On strings:")
for c in S:
    print(c)
```

```
On strings:
H
i

t
h
e
r
e

f
r
o
m
```

(continues on next page)

(continued from previous page)

```
p
y
t
h
o
n
```

[11]:
```
print("\nOn lists:")
#for loop on list
for item in Slist:
    print(item)
```

```
On lists:
Hi
there
from
python
```

[12]:
```
print("\nOn tuples:")
#for loop on list
for item in Stuple:
    print(item)
```

```
On tuples:
Hi
there
from
python
```

Looping over a range

It is possible to loop over a range of values with the python built-in function `range`. The `range` function accepts either two or three parameters (all of them are **integers**). Similarly to the slicing operator, it needs the **starting point**, **end point** and **an optional step**.

Three distinct syntaxes are available:

```
range(E)         # ranges from 0 to E-1
range(S,E)       # ranges from S to E-1
range(S,E,step)  # ranges from S to E-1 with +step jumps
```

Remember that *S* is **included** while *E* is **excluded**. Let's see some examples.

Example: Given a list of integers, return a list with all the even numbers.

[13]:
```
myList = [1, 7, 9, 121, 77, 82]
onlyEven = []

for i in range(0, len(myList)):  #this is equivalent to range(len(myList)):
    if( myList[i] % 2 == 0 ):
        onlyEven.append(myList[i])

print("original list:", myList)
```

(continues on next page)

(continued from previous page)

```
print("only even numbers:", onlyEven)
```

```
original list: [1, 7, 9, 121, 77, 82]
only even numbers: [82]
```

Example: Store in a list the multiples of 19 between 1 and 100.

[14]:
```
multiples = []

for i in range(19,101,19):
    multiples.append(i)

print("multiples of 19: ", multiples)

#alternative way:
multiples = []
for i in range(1, (100//19) + 1):
    multiples.append(i*19)
print("multiples of 19:", multiples)
```

```
multiples of 19:  [19, 38, 57, 76, 95]
multiples of 19: [19, 38, 57, 76, 95]
```

Note: range works differently in Python 2.x and 3.x

In Python 3 the *range* function returns an iterator rather storing the entire list.

[15]:
```
#Check out the difference:
print(range(0,10))

print(list(range(0,10)))
```

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Example: Let's consider the two DNA strings s1 = "ATACATATAGGGCCAATTATTATAAGTCAC" and s2 = "CGCCACTTAAGCGCCCTGTATTAAAGTCGC" that have the same length. Let's create a third string out such that $out[i]$ is "|" if $s1[i] == s2[i]$, " " otherwise.

[16]:
```
s1 = "ATACATATAGGGCCAATTATTATAAGTCAC"
s2 = "CGCCACTTAAGCGCCCTGTATTAAAGTCGC"

outSTR = ""
for i in range(len(s1)):
    if(s1[i] == s2[i]):
        outSTR = outSTR + "|"
    else:
        outSTR = outSTR + " "

print(s1)
print(outSTR)
print(s2)
```

```
ATACATATAGGGCCAATTATTATAAGTCAC
   ||  || |  |  |   |  ||||| |
CGCCACTTAAGCGCCCTGTATTAAAGTCGC
```

Nested for loops

In some occasions it is useful to nest one (or more) for loops into another one. The basic syntax is:

```
for i in collection:
    for j in another_collection:
        # do some stuff with i and j
```

Example:

Given the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ stored as a list of lists (i.e. matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]].

Print it out as: $\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$

[17]:
```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

for i in range(len(matrix)):
    line = ""
    for j in range(len(matrix[i])):
        line = line + str(matrix[i][j]) + " " #note int --> str conversion!
    print(line)
```

```
1 2 3
4 5 6
7 8 9
```

While loops

The for loop is great when we have to iterate over a finite sequence of elements. But when one needs to loop until a specific condition holds true, another construct must be used: the *while* statement. The loop will end when the condition becomes false.

The basic syntax is the following:

```
while condition:

    # do something
    # update the value of condition
```

An example follows:

[18]:
```
i = 0
while (i < 5):
    print("i now is:", i)
    i = i + 1 #THIS IS VERY IMPORTANT!
```

```
i now is: 0
i now is: 1
i now is: 2
i now is: 3
i now is: 4
```

Note that if *condition* is false at the beginning the block of code is never executed.

> **Note:** The loop will continue until *condition* holds true and the only code executed is the block defined through the indentation. This block of code must update the value of condition otherwise the interpreter will get stuck in the loop and will never exit.

We can combine *for* loops and *while* loops one into the code block of the other:

Break and continue

Sometimes it is useful to skip an entire iteration of a loop or end the loop before its supposed end. This can be achieved with two different statements: **continue** and **break**.

Continue statement

Within a **for** or **while** loop, **continue** makes the interpreter skip that iteration and move to the next.

Example: Print all the odd numbers from 1 to 20.

[19]:
```
#Two equivalent ways
#1. Testing remainder == 1
for i in range(21):
    if(i % 2 == 1):
        print(i, end = " ")

print("")

#2. Skipping if remainder == 0 in for
for i in range(21):
    if(i % 2 == 0):
        continue
    print(i, end = " ")
```

```
1 3 5 7 9 11 13 15 17 19
1 3 5 7 9 11 13 15 17 19
```

Continue can be used also within while loops but we need to be careful to update the value of the variable before reaching the continue statement or we will get stuck in never-ending loops. **Example:** Print all the odd numbers from 1 to 20.

```
#Wrong code:
i = 0
while (i < 21):
    if(i % 2 == 0):
        continue
    print(i, end = " ")
    i = i + 1 # NEVER EXECUTED IF i % 2 == 0!!!!
```

a possible correct solution using while:

[20]:
```
i = -1
while( i< 20):        #i is incremented in the loop, so 20!!!
    i = i + 1         #the variable is updated no matter what
    if(i % 2 == 0 ):
        continue
    print(i, end = " ")
```

```
1 3 5 7 9 11 13 15 17 19
```

Break statement

Within a **for** or **while** loop, **break** makes the interpreter exit the loop and continue with the sequential execution. Sometimes it is useful to get out of the loop if to complete our task we do not need to get to the end of the loop.

Example: Given the following list of integers [1,5,6,4,7,1,2,3,7] print them until a number already printed is found.

[21]:
```
L = [1,5,6,4,7,1,2,3,7]
found = []
for i in L:
    if(i in found):
        break

    found.append(i)
    print(i, end = " ")
```

```
1 5 6 4 7
```

Example: Pick a random number from 1 and 50 and count how many times it takes to randomly choose number 27. Limit the number of random picks to 40 (i.e. if more than 40 picks have been done and 27 has not been found exit anyway with a message).

[22]:
```
import random

iterations = 1
picks = []
while(iterations <= 40):
    pick = random.randint(1,50)
    picks.append(pick)

    if(pick == 27):
        break
    iterations += 1

if(iterations == 41):
    print("Sorry number 27 was never found!")
else:
    print("27 found in ", iterations, "iterations")

print(picks)
```

```
Sorry number 27 was never found!
[22, 12, 16, 22, 19, 41, 50, 20, 37, 47, 18, 42, 33, 19, 18, 16, 8, 16, 36, 31, 1, 49,
↪ 19, 38, 34, 18, 45, 30, 26, 44, 7, 23, 37, 12, 38, 43, 42, 26, 46, 41]
```

An alternative way without using the break statement makes use of a *flag* variable (that when changes value will make the loop end):

[23]:
```
import random
found = False # This is called flag
iterations = 1
picks = []
while iterations <= 40 and found == False: #the flag is used to exit
    pick = random.randint(1,50)
    picks.append(pick)
    if pick == 27:
        found = True     #update the flag, will exit at next iteration
    iterations += 1

if iterations == 41 and not found:
    print("Sorry number 27 was never found!")
else:
    print("27 found in ", iterations -1, "iterations")

print(picks)
```

```
Sorry number 27 was never found!
[40, 46, 29, 29, 38, 1, 12, 41, 19, 39, 8, 10, 5, 18, 31, 50, 38, 18, 9, 46, 22, 47,␣
↪36, 41, 7, 43, 24, 39, 50, 47, 15, 10, 34, 8, 6, 23, 9, 1, 24, 18]
```

[24]:
```
for i in range(1,10):                          # or without string output
    j = 1                                      # for i in range(1,10):
    output = ""                                #     j = 1
    while(j<= i):                              #     while(j<=i):
        output = str(j) + " " + output         #         print(j, end = " ")
        j = j + 1                              #         j = j + 1
    print(output)                              #     print("")
```

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
6 5 4 3 2 1
7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1
```

6.8.6 Exercises

1. Given the integer 134479170, print if it is divisible for the numbers from 2 to 16. Hint: use for and if.

Show/Hide Solution

[25]:
```
val = 134479170
flag = False

for divisor in range(2,17):
    if val % divisor == 0:
        print(val, " can be divided by ", divisor)
        print("\t", divisor, "*", val//divisor, "=", val )
        flag = True

if flag == False:
    print("Sorry, could not find divisors")
```

```
134479170  can be divided by  2
         2 * 67239585 = 134479170
134479170  can be divided by  3
         3 * 44826390 = 134479170
134479170  can be divided by  5
         5 * 26895834 = 134479170
134479170  can be divided by  6
         6 * 22413195 = 134479170
134479170  can be divided by  7
         7 * 19211310 = 134479170
134479170  can be divided by  9
         9 * 14942130 = 134479170
134479170  can be divided by  10
         10 * 13447917 = 134479170
134479170  can be divided by  14
         14 * 9605655 = 134479170
134479170  can be divided by  15
         15 * 8965278 = 134479170
```

2. Given the DNA string "GATTACATATATCAGTACAGATATATACGCGCGGGCTTACTATTAAAAACCCC", write a Python script that reverse-complements it. To reverse-complement a string of DNA, one needs to replace and A with T, T with A, C with G and G with C, while any other character is complemented in N. Finally, the sequence has to be reversed (e.g. the first base becomes the last). For example, ATCG becomes CGAT.

Show/Hide Solution

[26]:
```
DNA = "GATTACATATATCAGTACAGATATATACGCGCGGGCTTACTATTAAAAACCCC"

revComp = ""
for base in DNA:
    if base == "T":
        revComp = "A"+ revComp
    elif base == "A":
        revComp = "T"+ revComp
    elif base == "C":
        revComp = "G"+ revComp
    elif base == "G":
        revComp = "C" + revComp
    else:
        revComp = "N" + revComp

print("5'-", DNA, "-3'")
print("3'-", revComp, "-5'")
```

```
5'- GATTACATATATCAGTACAGATATATACGCGCGGGCTTACTATTAAAAACCCC -3'
3'- GGGGTTTTTAATAGTAAGCCCGCGCGTATATATCTGTACTGATATATGTAATC -5'
```

[27]:
```
""" Another solution"""

DNA = "GATTACATATATCAGTACAGATATATACGCGCGGGCTTACTATTAAAAACCCC"


dna_list = list(DNA)
print(dna_list)
compl = []
for el in dna_list:
```

(continues on next page)

(continued from previous page)

```
    if el == 'A':
        compl.append('T')
    elif el == 'T':
        compl.append('A')
    elif el == 'C':
        compl.append('G')
    elif el == 'G':
        compl.append('C')
    else:
        compl.append('N')
print(compl)
compl.reverse()
print(compl)
print("".join(compl))
```

```
['G', 'A', 'T', 'T', 'A', 'C', 'A', 'T', 'A', 'T', 'A', 'T', 'C', 'A', 'G', 'T', 'A',
↪'C', 'A', 'G', 'A', 'T', 'A', 'T', 'A', 'T', 'A', 'C', 'G', 'C', 'G', 'C', 'G', 'G',
↪ 'G', 'C', 'T', 'T', 'A', 'C', 'T', 'A', 'T', 'T', 'A', 'A', 'A', 'A', 'A', 'C', 'C
↪', 'C', 'C']
['C', 'T', 'A', 'A', 'T', 'G', 'T', 'A', 'T', 'A', 'T', 'A', 'G', 'T', 'C', 'A', 'T',
↪'G', 'T', 'C', 'T', 'A', 'T', 'A', 'T', 'A', 'T', 'G', 'C', 'G', 'C', 'G', 'C', 'C',
↪ 'C', 'G', 'A', 'A', 'T', 'G', 'A', 'T', 'A', 'A', 'T', 'T', 'T', 'T', 'T', 'G', 'G
↪', 'G', 'G']
['G', 'G', 'G', 'G', 'T', 'T', 'T', 'T', 'T', 'A', 'A', 'T', 'A', 'G', 'T', 'A', 'A',
↪'G', 'C', 'C', 'C', 'G', 'C', 'G', 'C', 'G', 'T', 'A', 'T', 'A', 'T', 'A', 'T', 'C',
↪ 'T', 'G', 'T', 'A', 'C', 'T', 'G', 'A', 'T', 'A', 'T', 'A', 'T', 'G', 'T', 'A', 'A
↪', 'T', 'C']
GGGGTTTTTAATAGTAAGCCCGCGCGTATATATCTGTACTGATATATGTAATC
```

3. Write a python script that creates the following pattern:

```
+
++
+++
++++
+++++
++++++
+++++++ <-- 7
++++++
+++++
++++
+++
++
+
```

Show/Hide Solution

[28]:
```
outStr = ""
for i in range(0,7):
    outStr = ""
    for j in range(0,i+1):
        outStr = outStr + "+"
    if(i == 6):
        outStr = outStr + " <-- 7"

    print(outStr)
```

(continues on next page)

(continued from previous page)

```
for i in range(1,7):
    outStr = ""
    for j in range(0, 7-i):
        outStr = outStr + "+"
    print(outStr)
```

```
+
++
+++
++++
+++++
++++++
+++++++ <-- 7
++++++
+++++
++++
+++
++
+
```

4. Count how many of the first 100 integers are divisible by 2, 3, 5, 7 but not by 10 and print these counts. Be aware that a number can be divisible by more than one of these numbers (e.g. 6) and therefore it must be counted as divisible by all of them (e.g. 6 must be counted as divisible by 2 and 3).

Show/Hide Solution

[29]:
```
cnts = [0,0,0,0] #cnts[0] counts for 2, cnts[1] counts for 3...
vals = [2,3,5,7]
for i in range(1,101):
    if ( i % 2 == 0 and i % 10 != 0 ):
        cnts[0] = cnts[0] + 1

    if ( i % 3 == 0 and i % 10 != 0):
        cnts[1] = cnts[1] + 1

    if ( i % 5 == 0 and i % 10 != 0):
        cnts[2] = cnts[2] + 1

    if ( i % 7 == 0 and i % 10 != 0):
        cnts[3] = cnts[3] + 1

for i in range(0, len(cnts)):
    print(cnts[i], " numbers are divisible by ", vals[i], "(but not 10) in the first␣
↪100")
```

```
40  numbers are divisible by  2 (but not 10) in the first 100
30  numbers are divisible by  3 (but not 10) in the first 100
10  numbers are divisible by  5 (but not 10) in the first 100
13  numbers are divisible by  7 (but not 10) in the first 100
```

5. Given the following fastq entry:

```
@HWI-ST1296:75:C3F7CACXX:1:1101:19142:14904
CCAACAACTTTGACGCTAAGGATAGCTCCATGGCAGCATATCTGGCACAA
+
FHIIJIJJGIJJJJJ1HHHFFFFFEE:;CIDDDDDDDDDDDDEDDD-./0
```

Store the sequence and the quality in two strings. Create a list with all the quality phred scores (given a quality character

"X" the phred score is: ord("X") -33. Finally print all the bases that have quality lower than 25, reporting the base, its position, quality character and phred score. **Output example:** base: C index: 14 qual:1 phred: 16).

Show/Hide Solution

[30]:
```
entry = """@HWI-ST1296:75:C3F7CACXX:1:1101:19142:14904
CCAACAACTTTGACGCTAAGGATAGCTCCATGGCAGCATATCTGGCACAA
+
FHIIJIJJGIJJJJJ1HHHFFFFFEE:;CIDDDDDDDDDDDDEDDD-./0"""

lines = entry.split("\n")
seq = lines[1]
qual = lines[3]

phredScores = []
for i in range(len(qual)):
    phredScores.append(ord(qual[i]) - 33)


for i in range(len(seq)):
    if(phredScores[i] <25):
        print("base:",seq[i],"index:",i,"qual:",qual[i],"phredScore:",phredScores[i])
```

```
base: C index: 15 qual: 1 phredScore: 16
base: A index: 46 qual: - phredScore: 12
base: C index: 47 qual: . phredScore: 13
base: A index: 48 qual: / phredScore: 14
base: A index: 49 qual: 0 phredScore: 15
```

6. Given the following sequence:

```
AUGCUGUCUCCCUCACUGUAUGUAAAUUGCAUCUAGAAUAGCA
UCUGGAGCACUAAUUGACACAUAGUGGGUAUCAAUUAUUA
UUCCAGGUACUAGAGAUACCUGGACCAUUAACGGAUAAAU
AGAAGAUUCAUUUGUUGAGUGACUGAGGAUGGCAGUUCCU
GCUACCUUCAAGGAUCUGGAUGAUGGGGAGAAACAGAGAA
CAUAGUGUGAGAAUACUGUGGUAAGGAAAGUACAGAGGAC
UGGUAGAGUGUCUAACCUAGAUUUGGAGAAGGACCUAGAA
GUCUAUCCCAGGGAAAUAAAAAUCUAAGCUAAGGUUUGAG
GAAUCAGUAGGAAUUGGCAAAGGAAGGACAUGUUCCAGAU
GAUAGGAACAGGUUAUGCAAAGAUCCUGAAAUGGUCAGAG
CUUGGUGCUUUUUGAGAACCAAAAGUAGAUUGUUAUGGAC
CAGUGCUACUCCCUGCCUCUUGCCAAGGGACCCCGCCAAG
CACUGCAUCCCUUCCCUCUGACUCCACCUUUCCACUUGCC
CAGUAUUGUUGGUGU
```

Considering the genetic code and the first forward open reading frame (i.e. the string as it is **remembering to remove newlines**).

1. How many start codons are present in the whole sequence (i.e. AUG)?
2. How many stop codons (i.e. UAA,UAG, UGA)
3. Create another string in which any codon with except the start and stop codons are substituted with "—" and print the resulting string.

Show/Hide Solution

[31]:
```
seq ="""AUGCUGUCUCCCUCACUGUAUGUAAAUUGCAUCUAGAAUAGCA
UCUGGAGCACUAAUUGACACAUAGUGGGUAUCAAUUAUUA
```

(continues on next page)

(continued from previous page)

```
UUCCAGGUACUAGAGAUACCUGGACCAUUAACGGAUAAAU
AGAAGAUUCAUUUGUUGAGUGACUGAGGAUGGCAGUUCCU
GCUACCUUCAAGGAUCUGGAUGAUGGGGAGAAACAGAGAA
CAUAGUGUGAGAAUACUGUGGUAAGGAAAGUACAGAGGAC
UGGUAGAGUGUCUAACCUAGAUUUGGAGAAGGACCUAGAA
GUCUAUCCCAGGGAAAUAAAAAUCUAAGCUAAGGUUUGAG
GAAUCAGUAGGAAUUGGCAAAGGAAGGACAUGUUCCAGAU
GAUAGGAACAGGUUAUGCAAAGAUCCUGAAAUGGUCAGAG
CUUGGUGCUUUUUGAGAACCAAAAGUAGAUUGUUAUGGAC
CAGUGCUACUCCCUGCCUCUUGCCAAGGGACCCCGCCAAG
CACUGCAUCCCUUCCCUCUGACUCCACCUUUCCACUUGCC
CAGUAUUGUUGGUG"""

seq = seq.replace("\n","")


countStart = 0
countEnd = 0

newSeq = ""
i = 0
while i < len(seq):
    codon = seq[i:i+3]
    if(codon == "AUG"):
            countStart = countStart + 1

    elif (codon == "UAA" or codon == "UAG" or codon == "UGA"):
        countEnd = countEnd + 1
    else:
        codon = "---"

    newSeq = newSeq + codon
    i = i + 3


print("\nNumber of start codons:", countStart)
print("Number of end codons:", countEnd)
print("\n")
print(seq)
print("\n")
print(newSeq)
```

```
Number of start codons: 2
Number of end codons: 12


AUGCUGUCUCCCUCACUGUAUGUAAAUUGCAUCUAGAAUAGCAUCUGGAGCACUAAUUGACACAUAGUGGGUAUCAAUUAUUAUUCCAGGUACUAGAGAUA


AUG------------------------------UAG---------------------UGA--------------------------
↪-------UAG---------------UAA------------------------UGA-----------------------------
↪-------------UGA------------------------UGA---------------------------------UAG-----
↪-UAA-------------------------------------------------------UAA----------------UAG-----
↪-------------------------------------------------------AUG--------------------------
↪-----------------------------------------------------------------------------------UGA--
↪-------------------------------
```

7. Playing time! Write a python scripts that:
    1. Picks a random number from 1 to 10, with: import random myInt = random.randint(1,10)
    2. Asks the user to guess a number and checks if the user has guessed the right one
    3. If the guess is right the program will stop with a congratulation message
    4. If the guess is wrong the program will continue asking a number, reporting the numbers already guessed (hint: store them in a list and print it).
    5. Modify the program to notify the user if he/she inputs the same number more than once.

Show/Hide Solution

```
import random
myInt = random.randint(1,10)
guessedNumbers = []

found = False
while (found == False):
    userInt = int(input("Guess a number from 1 to 10: "))
    if(userInt == myInt):
        print("Congratulations. The number I guessed was ", myInt)
        found = True
    else:
        if(userInt in guessedNumbers):
            print("You already guessed ", userInt)
        else:
            guessedNumbers.append(userInt)
        guessedNumbers.sort()
        print("Nope, try again. So far you guessed: ", guessedNumbers)
```

```
Guess a number from 1 to 10: 4
Nope, try again. So far you guessed:  [4]
Guess a number from 1 to 10: 5
Nope, try again. So far you guessed:  [4, 5]
Guess a number from 1 to 10: 1
Congratulations. The number I guessed was  1
```

[32]:
```
# SOLUTION
```

[ ]:
```

```

6.9 Functions - solutions

6.9.1 Download exercises zip

Browse files online[212]

[212] https://github.com/DavidLeoni/sciprog-ds/tree/master/functions

6.9.2 Introduction

References:

A function 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 the scope of variables

References

- Andrea Passerini slides A04[213]
- Thinking in Python, Chapter 3, Functions[214]
- Thinking in Python, Chapter 6, Fruitful functions[215] **NOTE**: in the book they use the weird term 'fruitful functions' for those functions which RETURN a value (mind you, RETURN a value, which is different from PRINTing it), and use also the term 'void functions' for functions which do not return anything but have some effect like PRINTing to screen. Please ignore these terms.

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-exercises
     |- functions
         |- functions.ipynb
         |- functions-sol.ipynb
```

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/functions.ipynb`
- Go on reading that notebook, and follow instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

[213] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf
[214] http://greenteapress.com/thinkpython2/html/thinkpython2004.html
[215] http://greenteapress.com/thinkpython2/html/thinkpython2007.html

What is a function ?

A function is a block of code that has a name and that performs a task. A function can be thought of as a box that gets an input and returns an output.

Why should we use 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 that 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;

both things improve code readability and make your code easier to understand.

The basic definition of a function is:

```
def function_name(input) :
    #code implementing the function
    ...
    ...
    return return_value
```

Functions are defined with the **def** keyword that proceeds the *function_name* and then a list of parameters is passed in the brackets. A colon **:** is used to end the line holding the definition of the function. The code implementing the function is specified by using indentation. A function **might** or **might not** return a value. In the first case a **return** statement is used.

Example:

Define a function that implements the sum of two integer lists (note that there is no check that the two lists actually contain integers and that they have the same size).

[2]:
```
def int_list_sum(la,lb):
    """implements the sum of two lists of integers having the same size
    """
    ret =[]
    for i in range(len(la)):
        ret.append(la[i] + lb[i])
    return ret

La = list(range(1,10))
print("La:", La)
```

```
La: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[3]:
```
Lb = list(range(20,30))
print("Lb:", Lb)
```

```
Lb: [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
```

[4]:
```
res = int_list_sum(La,Lb)
```

[5]:
```
print("La+Lb:", res)
```

```
La+Lb: [21, 23, 25, 27, 29, 31, 33, 35, 37]
```

[6]:
```
res = int_list_sum(La,La)
```

[7]:
```
print("La+La", res)
```

```
La+La [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Note that once the function has been defined, it can be called as many times as wanted with different input parameters. Moreover, **a function does not do anything until it is actually called.** A function can return **0** (in this case the return value would be "None"), **1** or **more** results. Notice also that collecting the results of a function is **not mandatory**.

Example: Let's write a function that, given a list of elements, prints only the even-placed ones without returning anything.

[8]:
```
def get_even_placed(myList):
    """returns the even placed elements of myList"""
    ret = [myList[i] for i in range(len(myList)) if i % 2 == 0]
    print(ret)
```

[9]:
```
L1 = ["hi", "there", "from","python","!"]
```

[10]:
```
L2 = list(range(13))
```

[11]:
```
print("L1:", L1)
```

```
L1: ['hi', 'there', 'from', 'python', '!']
```

[12]:
```
print("L2:", L2)
```

```
L2: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

[13]:
```
print("even L1:")
get_even_placed(L1)
```

```
even L1:
['hi', 'from', '!']
```

[14]:
```
print("even L2:")
get_even_placed(L2)
```

```
even L2:
[0, 2, 4, 6, 8, 10, 12]
```

Note that the function above is polymorphic (i.e. it works on several data types, provided that we can iterate through them).

Example: Let's write a function that, given a list of integers, returns the number of elements, the maximum and minimum.

[15]:
```
def get_info(myList):
    """returns len of myList, min and max value (assumes elements are integers)"""
    tmp = myList[:] #copy the input list
    tmp.sort()
    return len(tmp), tmp[0], tmp[-1] #return type is a tuple

A = [7, 1, 125, 4, -1, 0]

print("Original A:", A, "\n")
```

```
Original A: [7, 1, 125, 4, -1, 0]
```

[16]:
```
result = get_info(A)
```

[17]:
```
print("Len:", result[0], "Min:", result[1], "Max:",result[2], "\n" )
```

```
Len: 6 Min: -1 Max: 125
```

[18]:
```
print("A now:", A)
```

```
A now: [7, 1, 125, 4, -1, 0]
```

[19]:
```
def my_sum(myList):
    ret = 0
    for el in myList:
        ret += el # == ret = ret + el
    return ret

A = [1,2,3,4,5,6]
B = [7, 9, 4]
```

[20]:
```
s = my_sum(A)
```

[21]:
```
print("List A:", A)
print("Sum:", s)
```

```
List A: [1, 2, 3, 4, 5, 6]
Sum: 21
```

[22]:
```
s = my_sum(B)
```

[23]:
```
print("List B:", B)
print("Sum:", s)
```

```
List B: [7, 9, 4]
Sum: 20
```

Please note that the return value above is actually a tuple. Importantly enough, a function needs to be defined (i.e. its code has to be written) before it can actually be used.

[24]:
```
A = [1,2,3]
my_sum(A)

def my_sum(myList):
    ret = 0
    for el in myList:
        ret += el
    return ret
```

6.9.3 Namespace and variable scope

Namespaces are mappings from *names* to objects, or in other words places where names are associated to objects. Namespaces can be considered as the context. According to Python's reference a **scope** is a *textual region of a Python program, where a namespace is directly accessible*, which means that Python will look into that *namespace* to find the object associated to a name. Four **namespaces** are made available by Python:

1. **Local**: the innermost that contains local names (inside a function or a class);
2. **Enclosing**: the scope of the enclosing function, it does not contain local nor global names (nested functions) ;
3. **Global**: contains the global names;
4. **Built-in**: contains all built in names (e.g. print, if, while, for,…)

When one refers to a name, Python tries to find it in the current namespace, if it is not found it continues looking in the namespace that contains it until the built-in namespace is reached. If the name is not found there either, the Python interpreter will throw a **NameError** exception, meaning it cannot find the name. The order in which namespaces are considered is: Local, Enclosing, Global and Built-in (LEGB).

Consider the following example:

[25]:
```
def my_function():
    var = 1  #local variable
    print("Local:", var)
    b = "my string"
    print("Local:", b)
```

```
var = 7 #global variable
my_function()
print("Global:", var)
print(b)
```

```
Local: 1
Local: my string
Global: 7

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-56-7dd8330a24f0> in <module>
      8 my_function()
      9 print("Global:", var)
---> 10 print(b)

NameError: name 'b' is not defined
```

Variables defined within a function can only be seen within the function. That is why variable b is defined only within the function. Variables defined outside all functions are **global** to the whole program. The namespace of the local variable is within the function my_function, while outside it the variable will have its global value.

And the following:

[26]:
```
def outer_function():
    var = 1 #outer

    def inner_function():
        var = 2 #inner
        print("Inner:", var)
```

(continues on next page)

(continued from previous page)

```
        print("Inner:", B)

    inner_function()
    print("Outer:", var)


var = 3 #global
B = "This is B"
outer_function()
print("Global:", var)
print("Global:", B)
```

```
Inner: 2
Inner: This is B
Outer: 1
Global: 3
Global: This is B
```

Note in particular that the variable B is global, therefore it is accessible everywhere and also inside the inner_function. On the contrary, the value of var defined within the inner_function is accessible only in the namespace defined by it, outside it will assume different values as shown in the example.

In a nutshell, remember the three simple rules seen in the lecture. Within a **def**:

```
1. Name assignments create local names by default;
2. Name references search the following four scopes in the order:
local, enclosing functions (if any), then global and finally built-in (LEGB)
3. Names declared in global and nonlocal statements map assigned names to
enclosing module and function scopes.
```

6.9.4 Argument passing

Arguments are the parameters and data we pass to functions. When passing arguments, there are three important things to bear in mind are:

1. Passing an argument is actually assigning an object to a local variable name;
2. Assigning an object to a variable name within a function **does not affect the caller**;
3. Changing a **mutable** object variable name within a function **affects the caller**

Consider the following examples:

[27]:
```
"""Assigning the argument does not affect the caller"""

def my_f(x):
    x = "local value" #local
    print("Local: ", x)

x = "global value" #global
my_f(x)
print("Global:", x)
my_f(x)
```

```
Local:  local value
Global: global value
Local:  local value
```

[28]:
```
"""Changing a mutable affects the caller"""

def my_f(myList):
    myList[1] = "new value1"
    myList[3] = "new value2"
    print("Local: ", myList)

myList = ["old value"]*4
print("Global:", myList)
my_f(myList)
print("Global now: ", myList)
```

```
Global: ['old value', 'old value', 'old value', 'old value']
Local:  ['old value', 'new value1', 'old value', 'new value2']
Global now:  ['old value', 'new value1', 'old value', 'new value2']
```

Recall what seen in the lecture:

The behaviour above is because **immutable objects** are passed **by value** (therefore it is like making a copy), while **mutable objects** are passed **by reference** (therefore changing them effectively changes the original object).

To avoid making changes to a **mutable object** passed as parameter one needs to **explicitely make a copy** of it.

Consider the example seen before. **Example:** Let's write a function that, given a list of integers, returns the number of elements, the maximum and minimum.

[29]:
```
def get_info(myList):
    """returns len of myList, min and max value (assumes elements are integers)"""
    myList.sort()
    return len(myList), myList[0], myList[-1] #return type is a tuple


def get_info_copy(myList):
    """returns len of myList, min and max value (assumes elements are integers)"""
```

(continues on next page)

(continued from previous page)

```
    tmp = myList[:] #copy the input list!!!!
    tmp.sort()
    return len(tmp), tmp[0], tmp[-1] #return type is a tuple

A = [7, 1, 125, 4, -1, 0]
B = [70, 10, 1250, 40, -10, 0, 10]

print("A:", A)
result = get_info(A)
```

```
A: [7, 1, 125, 4, -1, 0]
```

[30]:
```
print("Len:", result[0], "Min:", result[1], "Max:",result[2] )
```

```
Len: 6 Min: -1 Max: 125
```

[31]:
```
print("A now:", A) #whoops A is changed!!!
```

```
A now: [-1, 0, 1, 4, 7, 125]
```

[32]:
```
print("\nB:", B)
```

```

B: [70, 10, 1250, 40, -10, 0, 10]
```

[33]:
```
result = get_info_copy(B)
```

[34]:
```
print("Len:", result[0], "Min:", result[1], "Max:",result[2] )
```

```
Len: 7 Min: -10 Max: 1250
```

[35]:
```
print("B now:", B) #B is not changed!!!
```

```
B now: [70, 10, 1250, 40, -10, 0, 10]
```

Positional arguments

Arguments can be passed to functions following the order in which they appear in the function definition.

Consider the following example:

[36]:
```
def print_parameters(a,b,c,d):
    print("1st param:", a)
    print("2nd param:", b)
    print("3rd param:", c)
    print("4th param:", d)

print_parameters("A", "B", "C", "D")
```

```
1st param: A
2nd param: B
3rd param: C
4th param: D
```

Passing arguments by keyword

Given the name of an argument as specified in the definition of the function, parameters can be passed using the **name = value** syntax.

For example:

[37]:
```
def print_parameters(a,b,c,d):
    print("1st param:", a)
    print("2nd param:", b)
    print("3rd param:", c)
    print("4th param:", d)

print_parameters(a = 1, c=3, d=4, b=2)
```

```
1st param: 1
2nd param: 2
3rd param: 3
4th param: 4
```

[38]:
```
print_parameters("first","second",d="fourth",c="third")
```

```
1st param: first
2nd param: second
3rd param: third
4th param: fourth
```

Arguments passed positionally and by name can be used at the same time, but parameters passed by name must always be to the left of those passed by name. The following code in fact is not accepted by the Python interpreter:

```
def print_parameters(a,b,c,d):
    print("1st param:", a)
    print("2nd param:", b)
    print("3rd param:", c)
    print("4th param:", d)

print_parameters(d="fourth",c="third", "first","second")
```

```
File "<ipython-input-60-4991b2c31842>", line 7
    print_parameters(d="fourth",c="third", "first","second")
                                           ^
SyntaxError: positional argument follows keyword argument
```

Specifying default values

During the definition of a function it is possible to specify default values. The syntax is the following:

```
def my_function(par1 = val1, par2 = val2, par3 = val3):
```

Consider the following example:

[39]:
```
def print_parameters(a="defaultA", b="defaultB",c="defaultC"):
    print("a:",a)
    print("b:",b)
    print("c:",c)

print_parameters("param_A")
```

```
a: param_A
b: defaultB
c: defaultC
```

[40]:
```
print_parameters(b="PARAMETER_B")
```

```
a: defaultA
b: PARAMETER_B
c: defaultC
```

[41]:
```
print_parameters()
```

```
a: defaultA
b: defaultB
c: defaultC
```

[42]:
```
print_parameters(c="PARAMETER_C", b="PAR_B")
```

```
a: defaultA
b: PAR_B
c: PARAMETER_C
```

6.9.5 Simple exercises

sum2

⊛ Write function `sum2` which given two numbers `x` and `y` RETURN their sum

QUESTION: Why do we call it `sum2` instead of just `sum` ??

[43]:
```
sum([2,51])
```

[43]:
```
53
```

ANSWER: `sum` is already defined as standard python function, we do not want to overwrite it. Look at how in the following snippet it displays in green:

```
>>> sum([5,8])
13
```

[44]:
```
# write here

def sum2(x,y):
    return x + y
```

[45]:
```
s = sum2(3,6)
print(s)
```

```
9
```

[46]:
```
s = sum2(-1,3)
print(s)
```

```
2
```

comparep

⊛ Write a function `comparep` which given two numbers `x` and `y`, PRINTS `x is greater than y`, `x is less than y`, `x is equal to y`

NOTE: in print, put real numbers. For example, comparep(10,5) should print:

```
10 is greater than 5
```

HINT: to print numbers and text, use commas in `print`:

```
print(x, " is greater than ")
```

[47]:
```
# write here
def comparep(x,y):
    if x > y:
        print(x, " is greater than ", y)
    elif x < y:
        print(x, " is less than ", y)
    else:
        print(x, " is equal to ", y)
```

[48]:
```
comparep(10,5)
```

```
10  is greater than  5
```

[49]:
```
comparep(3,8)
```

```
3  is less than  8
```

[50]:
```
comparep(3,3)
```

```
3  is equal to  3
```

comparer

⊛ Write function `comparer` which given two numbers `x` and`y` RETURN the STRING `'>'` if `x` is greater than `y`, the STRING `'<'`if `x` is less than `y` or the STRING `'=='` if `x` is equal to `y`

[51]:
```
# write here
def comparer(x,y):
    if x > y:
        return '>'
    elif x < y:
        return '<'
    else:
        return '=='
```

[52]:
```
c = comparer(10,5)
print(c)
```

```
>
```

[53]:
```
c = comparer(3,7)
print(c)
```

```
<
```

[54]:
```
c = comparer(3,3)
print(c)
```

```
==
```

even

⊛ Write a function `even` which given a number `x`, RETURN `True` if `x` is even, otherwise RETURN `False`

HINT: a number is even when the rest of division by two is zero. To obtaing the reminder of division, write `x % 2`

[55]:
```
# Example:
2 % 2
```

[55]:
```
0
```

[56]:
```
3 % 2
```

[56]:
```
1
```

[57]:
```
4 % 2
```

[57]:
```
0
```

[58]:
```
5 % 2
```

[58]:
```
1
```

[59]:
```
# write here
def even(x):
    return x % 2 == 0
```

[60]:
```
p = even(2)
print(p)
```

```
True
```

[61]:
```
p = even(3)
print(p)
```

```
False
```

[62]:
```
p = even(4)
print(p)
```

```
True
```

[63]:
```
p = even(5)
print(p)
```

```
False
```

[64]:
```
p = even(0)
print(p)
```

```
True
```

gre

⊛ Write a function `gre` that given two numbers `x` and `y`, RETURN the greatest number.

If they are equal, RETURN any number.

[65]:
```
# write here

def gre(x,y):
    if x > y:
        return x
    else:
        return y
```

[66]:
```
m = gre(3,5)
print(m)
```

```
5
```

[67]:
```
m = gre(6,2)
print(m)
```

```
6
```

[68]:
```
m = gre(4,4)
print(m)
```

```
4
```

[69]:
```
m = gre(-5,2)
print(m)
```

```
2
```

[70]:
```
m = gre(-5, -3)
print(m)
```

```
-3
```

is_vocal

⊛ Write a function `is_vocal` in which a character `car` is passed as parameter, and PRINTs `'yes'` if the carachter is a vocal, otherwise PRINTs `'no'` (using the `prints`).

```
>>> is_vocal("a")
'yes'

>>> is_vocal("c")
'no'
```

[71]:
```
# write here

def is_vocal(char):
    if char == 'a' or char == 'e' or char == 'i' or char == 'o' or char == 'u':
        print('yes')
    else:
        print('no')
```

sphere_volume

⊛ The volume of a sphere of radius `r` is $4/3r^3$

Write a function `sphere_volume(radius)` which given a `radius` of a sphere, PRINTs the volume.

NOTE: assume pi = 3.14

```
>>> sphere_volume(4)
267.94666666666666
```

[72]:
```
# write here

def sphere_volume(radius):
    print((4/3)*3.14*(radius**3))
```

ciri

⊛ Write a function `ciri(name)` which takes as parameter the string `name` and RETURN `True` if it is equal to the name `'Cirillo'`

```
>>> r = ciri("Cirillo")
>>> r
True

>>> r = ciri("Cirillo")
>>> r
False
```

[73]:
```
# write here

def ciri(name):
    if name == "Cirillo":
        return True
    else:
        return False
```

age

⊛ Write a function `age` which takes as parameter `year` of birth and RETURN the age of the person

**Suppose the current year is known, so to represent it in the function body use a constant like `2019`:

```
>>> a = age(2003)
>>> print(a)
16
```

[74]:
```
# write here

def age(year):
    return 2019 - year
```

6.9.6 Verify comprehension

Following exercises require you to know:

> **ATTENTION**
>
> Following exercises require you to know:
>
> Complex statements: Andrea Passerini slides A03[216]
>
> Tests with asserts[217]: Following exercises contain automated tests to help you spot errors. To understand how to do them, read before Error handling and testing[218]

gre3

⊛⊛ Write a function `gre3(a,b,c)` 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
```

[75]:
```
# write ehere

def gre3(a,b,c):
    if a > b:
        if a>c:
            return a
        else:
```

(continues on next page)

[216] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[217] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-asserts
[218] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html

(continued from previous page)

```
            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
```

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
```

[76]:
```
def final_price(n):
    #jupman-raise
    if n == 0:
        return 0
    else:
        return n* 24.95*0.6 + 3 +(n-1)*0.75
    #/jupman-raise

assert final_price(10) == 159.45
assert final_price(0) == 0
```

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 reminder of integer division, use the module operator `%`

```
>>> arrival_time(2,2)
7:22
```

[77]:
```
def arrival_time(n,m):
    #jupman-raise
    starting_hours = 6
    starting_minutes = 52

    # passed seconds
    seconds = n * 495 + m * 432

    # passed time
    seconds_two = seconds % 60
    minutes = seconds // 60
    hours = minutes // 60


    arrival_hours= hours + starting_hours
    arrival_minutes= minutes + starting_minutes

    final_minutes = arrival_minutes % 60
    final_hours = arrival_minutes // 60 + arrival_hours

    return str(final_hours) + ":" + str(final_minutes)
    #/jupman-raise

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) == '9:34'
```

[ ]:
```

```

6.9.7 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:

[78]:
```
lambda x: x*2
```

[78]:
```
<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`:

[79]:
```
f = lambda x: x*2
```

[80]:
```
f
```

[80]:
```
<function __main__.<lambda>(x)>
```

Great, now we have a function we can call as many times as we want:

[81]:
```
f(5)
```

[81]:
```
10
```

[82]:
```
f(7)
```

[82]:
```
14
```

So writing

[83]:
```
def f(x):
    return x*2
```

or

[84]:
```
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:

[85]:
```
animals = [('dog', 12), ('cat', 14), ('pelican', 30), ('eagle', 25), ('squirrel', 6)]
```

If you want to sort them, you can try the `.sort` method but it will not work:

[86]:
```
animals.sort()
```

[87]:
```
animals
```

[87]:
```
[('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, Pyhton provides us with `key` parameter, which we must pass a function that takes as argument the list element under consideration (in this case a tuple) and will return a trasformation of it (in this case the number at 1-th position):

[88]:
```
animals.sort(key=lambda t: t[1])
```

[89]:
```
animals
```

[89]:
```
[('squirrel', 6), ('dog', 12), ('cat', 14), ('eagle', 25), ('pelican', 30)]
```

Now we got the ordering we wanted. We could have written the thing as

[90]:
```
def myf(t):
    return t[1]

animals.sort(key=myf)
animals
```

[90]:
```
[('squirrel', 6), ('dog', 12), ('cat', 14), ('eagle', 25), ('pelican', 30)]
```

but lambdas clearly save some keyboard typing

Notice lambdas can take multiple parameters:

[91]:
```
mymul = lambda x,y: x * y

mymul(2,5)
```

[91]:
```
10
```

Exercises: lambdas

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 appling `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')
```

[92]:
```
# write here

def apply_borders(f, seq):
    return ( f(seq[0]),   f(seq[-1]) )
```

[93]:
```
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')
```

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

[94]:
```
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
```

[95]:
```
process(f, ['d','b','a','c','e','f'], ['q','s','p','t','r','n'])
```

[95]:
```
['An', 'Bp', 'Cq', 'Dr', 'Es', 'Ft']
```

6.10 Error handling and testing solutions

6.10.1 Download exercises zip

Browse files online[219]

6.10.2 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. In particular, we will describe the exercise format as proposed in Part A and in Part B (they are different!)

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 scrupolous checks? We wouldn't.

[219] https://github.com/DavidLeoni/sciprog-ds/tree/master/errors-and-testing

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-exercises
     |- errors-and-testing
         |- errors-and-testing.ipynb
         |- errors-and-testing-sol.ipynb
```

WARNING 1: 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 `strings/strings.ipynb`

WARNING 2: DO NOT use the *Upload* button in Jupyter, instead navigate to the unzipped folder while in Jupyter browser!

- Go on reading that notebook, and follow instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.10.3 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!")


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 ??

ANSWER: the program above is partying even when we do not have enough ingredients !

6.10.4 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[220]

[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
    #jupman-strip
    if milk > 1.3:
        print("take milk")
        # return True  # NO, it would finish right here
    else:
        print("Don't have enough milk !")
        return False

    if sugar > 0.2:
        print("take sugar")
    else:
        print("Don't have enouch 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
    #/jupman-strip


# 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 :-(

#jupman-strip
made_pie = make_pie(5,1,0.3)

if made_pie == True:
    print("Party")
else:
    print("No party !")
#/jupman-strip
```

```
take milk
take sugar
Don't have enough flour !
No party !
```

[220] http://greenteapress.com/thinkpython2/html/thinkpython2007.html

6.10.5 Exceptions

Real Python - Python Exceptions: an Introduction[221]

Using `return` we improved the previous function, but remains a problem: the responsability 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 tipically end up hidden in system logs. To final users you should only show short messages which are understanble 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()`:

[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
    #jupman-strip

    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!")
    if flour > 1.0:
        print("take flour")
    else:
        raise Exception("Don't have enough flour!")
    print("Mix")
    print("Heat")
    print("I made the pie !")
   #/jupman-strip
```

[221] https://realpython.com/python-exceptions/

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
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 know 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!")
    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
```

(continues on next page)

(continued from previous page)

```
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[222]

6.10.6 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. Plutoniom 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

print("water level: ", water_level)


# a lot of code

# a lot of code

# a lot of code

# a lot of code

water_level = 5  # forgot somewhere this bad row !

print("WARNING: water level low! ", water_level)

# a lot of code

# a lot of code

# a lot of code

# 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")
```

[222] https://realpython.com/python-exceptions/

```
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

# a lot of code

# a lot of code

# a lot of code

water_level = 5  # forgot somewhere this bad row !

print("WARNING: water level low! ", water_level)

# a lot of code

# a lot of code

# a lot of code
```

(continues on next page)

(continued from previous page)

```
# 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 exagerated, 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

# a lot of code

# a lot of code

# a lot of code

water_level = 5  # forgot somewhere this bad row !

print("WARNING: water level low! ", water_level)
```

(continues on next page)

(continued from previous page)

```
# a lot of code

# a lot of code

# a lot of code

# 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")

Exception: Water level too low !
```

Note how the reactor was *not* turned on.

6.10.7 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 euros or even kill people. Suggested reading: Software Horror Stories[223]

[223] https://www.cs.tau.ac.il/~nachumd/horror.html

Where Is Your Software?

As a data scientist, you might likely end up with code which is algorithmically complex, but maybe not too big in size. Either way, when red line is crossed you should start testing properly:

In a typical scenario, you are a junior programmer and your senior colleague ask you to write a function to perform some task, giving only an informal description:

[10]:
```
def my_sum(x,y):
    """ RETURN the sum of x and y
    """
    raise Exception("TODO IMPLEMENT ME!")
```

Even better, your colleague might provide you with some automated tests you might run to check your function meets his/her expectations. If you are smart, you will even write tests for your own functions to make sure every little piece you add to your software is a solid block you can build upon.

According to the part of the course you are following, we will review two kinds of tests:

- Part A: *asserts*
- Part B: *unittests*

6.10.8 Testing with asserts

> **NOTE**: Testing with asserts is only done in PART A of this course

We can use *assert* to quickly test functions, and verify they behave like they should.

For example, from this function:

[11]:
```
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`:

[12]:
```
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 exectued 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`

Part A exercise structure

Exercises in Part A will be 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
```

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("IMPLEMENTAMI")` 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 th efunction 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
----> 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 !

[13]:
```
def my_sum(x,y):
    """ RITORNA 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, congratulatins! The code you have written is correct *with reference to the tests* !
- Try to substitute instead with `return 10` and see what happens.

[14]:
```
def my_mul(x,y):
    """ RETURN the multiplication of numbers x and y
    """
    #jupman-raise
    return x * y
    #/jupman-raise


assert my_mul(2,5) == 10
assert my_mul(0,2) == 0
assert my_mul(3,2) == 6
```

even_numbers example

Let's see a slightly more complex function:

[15]:
```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    raise Exception("TODO IMPLEMENT ME!")
```

In this case, if you run the function as it is, you are reminded to implement it:

```
>>> even_numbers(5)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-2-d2cbc915c576> in <module>()
----> 1 even_numbers(5)

<ipython-input-1-a20a4ea4b42a> in even_numbers(n)
      8     [0,2,4,6,8]
      9     """
---> 10     raise Exception("TODO IMPLEMENT ME!")

Exception: TODO IMPLEMENT ME!
```

Why? The instruction

```
raise Exception("TODO IMPLEMENT ME!")
```

tells Python to immediatly stop execution, and signal an error to the caller of the function `even_number`. If there were commands right after `raise Exception("TODO IMPLEMENT ME")`, they would not be executed. Here, we are

directly calling the function from the prompt, and we didn't tell Python how to handle the `Exception`, so Python just stopped and showed the error message given as parameter to the `Exception`

Spend time reading well the function text!

Always read very well function text and ask yourself questions! What is the supposed input? What should be the output? Is there any output to return at all, or should you instead modify *in-place* a passed parameter (i.e. for example, when you sort a list)? Are there any edge cases, es what happens for `n=0`)? What about `n < 0` ?

Let's code a possible solution. As it often happens, first version may be buggy, in this case for example purposes we intentionally introduce a bug:

[16]:
```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    r = [2 * x for x in range(n)]
    r[n // 2] = 3   # <-- evil bug, puts number '3' in the middle, and 3 is not even .
↪.
    return r
```

Typically the first test we do is printing the output and do some 'visual inspection' of the result, in this case we find many numbers are correct but we might miss errors such as the wrong `3` in the middle:

[17]:
```
print(even_numbers(5))
```

```
[0, 2, 3, 6, 8]
```

Furthermore, if we enter commands a the prompt, each time we fix something in the code, we need to enter commands again to check everything is ok. This is inefficient, boring, and prone to errors.

Let's add assertions

To go beyond the dumb "visual inspection" testing, it's better to write some extra code to allow Python checking for us if the function actually returns what we expect, and throws an error otherwise. We can do so with `assert` command, which verifies if its argument is True. If it is not, it raises an `AssertionError` immediately stopping execution.

Here we check the result of `even_numbers(5)` is actually the list of even numbers `[0,2,4,6,8]` we expect:

```
assert even_numbers(5) == [0,2,4,6,8]
```

Since our code is faulty, `even_numbers` returns the wrong list `[0,2,3,6,8]` which is different from `[0,2,4,6,8]` so assertion fails showing `AssertionError`:

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-21-d4198f229404> in <module>()
----> 1 assert even_numbers(5) != [0,2,4,6,8]

AssertionError:
```

We got some output, but we would like to have it more informative. To do so, we may add a message, separated by a comma:

```
assert even_numbers(5) == [0,2,4,6,8], "even_numbers is not working !!"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-18-8544fcd1b7c8> in <module>()
----> 1 assert even_numbers(5) == [0,2,4,6,8], "even_numbers is not working !!"

AssertionError: even_numbers is not working !!
```

So if we modify code to fix bugs we can just launch the assert commands and have a quick feedback about possible errors.

Error kinds

As a fact of life, errors happen. Sometimes, your program may have inconsistent data, like wrong parameter type passed to a function (i.e. string instead of integer). A good principle to follow in these cases is to try have the program detect weird situations, and stop as early as such a situation is found (i.e. in the Therac 25 case, if you detect excessive radiation, showing a warning sign is not enough, it's better to stop). Note stopping might not always be the desirable solution (if one pidgeon enters one airplane engine, you don't want to stop all the other engines). If you want to check function parameters are correct, you do the so called *precondition checking*.

There are roughly two cases for errors, external user misusing you program, and just plain wrong code. Let's analyize both:

Error kind a) An external user misuses you program.

You can assume whover uses your software, final users or other programmers , they will try their very best to wreck your precious code by passing all sort of non-sense to functions. Everything can come in, strings instead of numbers, empty arrays, `None` objects … In this case you should signal the user he made some mistake. The most crude signal you can have is raising an `Exception` with `raise Exception("Some error occurred")`, which will stop the program and print the stacktrace in the console. Maybe final users won't understand a stacktrace, but at least programmers hopefully will get a clue about what is happening.

In these case you can raise an appropriate Exception, like TypeError[224] for wrong types and ValueError[225] for more generic errors. Other basic exceptions can be found in Python documentation[226]. Notice you can also define your own, if needed (we won't consider custom exceptions in this course).

NOTE: Many times, you can consider yourself the 'careless external user' to guard against.

Let's enrich the function with some appropriate type checking:

Note that for checking input types, you can use the function `type()` :

[18]:
```
type(3)
```

[18]:
```
int
```

[19]:
```
type("ciao")
```

[19]:
```
str
```

[224] https://docs.python.org/3/library/exceptions.html#TypeError
[225] https://docs.python.org/3/library/exceptions.html#ValueError
[226] https://docs.python.org/3/library/exceptions.html#built-in-exceptions

Let's add the code for checking the *even_numbers example*:

[20]:
```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    if type(n) is not int:
        raise TypeError("Passed a non integer number: " + str(n))

    if n < 0:
        raise ValueError("Passed a negative number: " + str(n))

    r = [2 * x for x in range(n)]
    return r
```

Let's pass a wrong type and see what happens:

```
>>> even_numbers("ciao")

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-a908b20f00c4> in <module>()
----> 1 even_numbers("ciao")

<ipython-input-13-b0b3a85f2b2a> in even_numbers(n)
      9     """
     10     if type(n) is not int:
---> 11         raise TypeError("Passed a non integer number: " + str(n))
     12
     13     if n < 0:

TypeError: Passed a non integer number: ciao
```

Now let's try to pass a negative number - it should suddenly stop with a meaningful message:

```
>>> even_numbers(-5)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-3f648fdf6de7> in <module>()
----> 1 even_numbers(-5)

<ipython-input-13-b0b3a85f2b2a> in even_numbers(n)
     12
     13     if n < 0:
---> 14         raise ValueError("Passed a negative number: " + str(n))
     15
     16     r = [2 * x for x in range(n)]

ValueError: Passed a negative number: -5
```

Now, even if you ship your code to careless users, and as soon as they commit a mistrake, they will get properly notified.

Error kind b): Your code is just plain wrong

In this case, it's 100% your fault, and these sort of bugs should never pop up in production. For example your code passes internally wrong stuff, like strings instead of integers, or wrong ranges (typically integer outside array bounds). So if you have an internal function nobody else should directly call, and you suspect it is being passed wrong parameters or at some point it has inconsistent data, to quickly spot the error you could add an assertion:

[21]:
```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    assert type(n) is int, "type of n is not correct: " + str(type(n))
    assert n >= 0, "Found negative n: " + str(n)

    r = [2 * x for x in range(n)]

    return r
```

As before, the function will stop as soon we call it we wrong parameters. The big difference is, this time we are assuming `even_numbers` is just for personal use and nobody else except us should directly call it.

Since assertion consume CPU time, IF we care about performances AND once we are confident our program behaves correctly, we can even remove them from compiled code by using the `-O` compiler flag. For more info, see Python wiki[227]

EXERCISE: try to call latest definition of `even_numbers` with wrong parameters, and see what happens.

NOTE: here we are using the correct definition of `even_numbers`, not the buggy one with the `3` in the middle of returned list !

6.10.9 Testing with Unittest

NOTE: Testing with Unittest is only done in PART B of this course

Is there anything better than `assert`for testing? `assert` can be a quick way to check but doesn't tell us exactly which is the wrong number in the list returned by `even_number(5)`. Luckily, Python offers us a better option, which is a complete testing framework called unittest[228]. We will use unittest because it is the standard one, but if you're doing other projects you might consider using better ones like pytest[229]

So let's give unittest a try. Suppose you have a file called `file_test.py` like this:

[22]:
```
import unittest

def even_numbers(n):
    """
    Return a list of the first n even numbers
```

(continues on next page)

[227] https://wiki.python.org/moin/UsingAssertionsEffectively
[228] https://docs.python.org/3/library/unittest.html
[229] https://docs.pytest.org/

(continued from previous page)

```
    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    r = [2 * x for x in range(n)]
    r[n // 2] = 3   # <-- evil bug, puts number '3' in the middle
    return r

class MyTest(unittest.TestCase):

    def test_long_list(self):
        self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

We won't explain what `class` mean (for classes see the book chpater[230]), the important thing to notice is the method definition:

```
def test_long_list(self):
    self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

In particular:

- method is declared like a function, and begins with `'test_'` word
- method takes `self` as parameter
- `self.assertEqual(even_numbers(5),[0,2,4,6,8])` executes the assertion. Other assertions could be `self.assertTrue(some_condition) or self.assertFalse(some_condition)`

Running tests

To run the tests, enter the following command in the terminal:

```
python -m unittest file_test
```

!!!!! WARNING: In the call above, DON'T append the extension `.py` to `file_test` !!!!!! **!!!!! WARNING**: Still, on the hard-disk the file MUST be named with a `.py` at the end, like `file_test.py`!!!!!!

You should see an output like the following:

[23]:
```
jupman.show_run(MyTest)
```

```
F
======================================================================
FAIL: test_long_list (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-22-397caec8a66f>", line 19, in test_long_list
    self.assertEqual(even_numbers(5),[0,2,4,6,8])
AssertionError: Lists differ: [0, 2, 3, 6, 8] != [0, 2, 4, 6, 8]
```

(continues on next page)

[230] http://interactivepython.org/runestone/static/pythonds/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html

(continued from previous page)

```
First differing element 2:
3
4

- [0, 2, 3, 6, 8]
?        ^

+ [0, 2, 4, 6, 8]
?        ^


----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Now you can see a nice display of where the error is, exactly in the middle of the list!

When tests don't run

When `-m unittest` does not work and you keep seeing absurd errors like Python not finding a module and you are getting desperate (especially because Python has `unittest` included *by default*, there is no need to install it!), try to put the following code at the very end of the file you are editing:

```
unittest.main()
```

Then run your file with just

```
python file_test.py
```

In this case it should REALLY work. If it still doesn't, call the Ghostbusters. Or, better, the IndentationBusters, you're likely having tabs mixed with spaces mixed with bad bad luck.

Adding tests

How can we add (good) tests? Since best ones are usually short, it would be better starting small boundary cases. For example like `n=1` , which according to function documentation should produce a list containing zero:

[24]:
```
class MyTest(unittest.TestCase):

    def test_one_element(self):
        self.assertEqual(even_numbers(1),[0])

    def test_long_list(self):
        self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

Let's call again the command:

```
python -m unittest file_test
```

[25]:
```
jupman.show_run(MyTest)
```

```
FF
======================================================================
FAIL: test_long_list (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-24-306d9f1c7777>", line 7, in test_long_list
    self.assertEqual(even_numbers(5),[0,2,4,6,8])
AssertionError: Lists differ: [0, 2, 3, 6, 8] != [0, 2, 4, 6, 8]

First differing element 2:
3
4

- [0, 2, 3, 6, 8]
?        ^

+ [0, 2, 4, 6, 8]
?        ^


======================================================================
FAIL: test_one_element (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-24-306d9f1c7777>", line 4, in test_one_element
    self.assertEqual(even_numbers(1),[0])
AssertionError: Lists differ: [3] != [0]

First differing element 0:
3
0

- [3]
+ [0]

----------------------------------------------------------------------
Ran 2 tests in 0.002s

FAILED (failures=2)
```

From the tests we can now see there is clearly something wrong with the number 3 that keeps popping up, making both tests fail. You can see immediately which tests have failed by looking at the first two `FF` at the top of the output. Let's fix the code by removing the buggy line:

[26]:
```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    r = [2 * x for x in range(n)]
    # NOW WE COMMENTED THE BUGGY LINE  r[n // 2] = 3   # <-- evil bug, puts number '3
↪' in the middle
    return r
```

(continues on next page)

(continued from previous page)

And call yet again the command:

```
python -m unittest file_test
```

[27]:
```
jupman.show_run(MyTest)
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Wonderful, all the two tests have passed and we got rid of the bug.

WARNING: DON'T DUPLICATE TEST CLASS NAMES AND/OR METHODS!

In the following, you will be asked to add tests. Just add **NEW** methods with **NEW** names to the **EXISTING** class `MyTest` !

Exercise: boundary cases

Think about other boundary cases, and try to add corresponding tests.

- Can we ever have an empty list?
- Can `n` be equal to zero? Add a test **inside MyTest class** for its expected result.
- Can `n` be negative? In this case the function text tells us nothing about the expected behaviour, so we might choose it now: either the function raises an error, or it gives a back something, like i.e. list of even negative numbers. Try to modify `even_numbers` and add a relative test **inside MyTest class** for expecting even negative numbers (starting from zero).

Exercise: expecting assertions

What if user passes us a float like `3.5` instead of an integer? If you try to run `even_numbers(3.5)` you will discover it works anyway, but we might decide to be picky and not accept inputs other than integers. Try to modify `even_numbers` to make so that when input is not of type `int`, raises TypeError[231] (to check for type, you can write `type(n) == int`).

To test for it, add following test **inside MyTest class** :

```
def test_type(self):

    with self.assertRaises(TypeError):
        even_numbers(3.5)
```

The `with` block tells Python to expect the code inside the `with` block to raise the exception TypeError[232]:

- If `even_numbers(3.5)` actually raises `TypeError` exception, nothing happens
- If `even_numbers(3.5)` does not raise `TypeError` exception, with raises `AssertionError`

[231] https://docs.python.org/3/library/exceptions.html#TypeError
[232] https://docs.python.org/3/library/exceptions.html#TypeError

After you completed previous task, consider when the input is the float `4.0`: in this case it might make sense to still accept it, so modify `even_numbers` accordingly and write a test for it.

Exercise: good tests

What difference is there between the following two test classes? Which one is better for testing?

```
class MyTest(unittest.TestCase):

    def test_one_element(self):
        self.assertEqual(even_numbers(1),[0])

    def test_long_list(self):
        self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

and

```
class MyTest(unittest.TestCase):

    def test_stuff(self):
        self.assertEqual(even_numbers(1),[0])
        self.assertEqual(even_numbers(5),[0,2,4,6,8])
```

Running unittests in Visual Studio Code

You can run and debug tests in Visual Studio Code, which is very handy. First, you need to set it up.

1. Hit `Control-Shift-P` (on Mac: `Command-Shift-P`) and type `Python: Configure Tests`

2. Select unittest:

3. Select `. root directory` (we assume tests are in the folder that you've opened):

4. Select `*Python files containing the word 'test'`:

Hopefully, on the currently opened test file new labels should appear above class and test methods, like in the following example. Try to click on them:

stack_test.py stack2_test.py ×

stack2_test.py > WStackTest

```
1   import unittest
2   from stack_solution import *
3
    ✘ Run Test | ✘ Debug Test
4   class WStackTest(unittest.TestCase):
5
        ✔ Run Test | ✔ Debug Test
6       def test_01_init(self):
7           s = WStack()
8
        ✘ Run Test | ✘ Debug Test
9       def test_02_weight(self):
10          s = WStack()
11          self.assertEqual(s.weight(), 1)
12
13
```

In the bottom bar, you should see a recap of run tests (right side of the picture):

master Python 3.7.5 64-bit ⊗ 0 ⚠ 0 ✓ 46 ⚠ 1

TROUBLESHOOTING

If you encounter problems running tests and have Anaconda, sometimes an easy solution can be just closing Visual Studio Code and running it from the Anaconda Navigator. You can also try to update it.

Running tests by console does not work:

- remember to SAVE the files before executing tests: in Windows, a file appears as not saved when its filename in the tab is written in italics; on Linux, you might see a dot to the right of the filename

Run Test label does not show up in code:

- if you see red squiggles in the code, most probably syntax is not correct and thus no test will get discovered ! If this is the case, fix the syntax error, SAVE, and then tell Visual Studio to discover test.
- you might also try *Right click->Run current Test File*.
- try *selecting another testing framework* , try pytest, which is also capable to discover and execute unittests.

- if you are really out of luck with the editor, there is always the option of running tests from the console.

WARNING: spend time also with the console !!!!

During the exam testing in VSCode might not work, so please be prepared to use the console

6.10.10 Functional programming

In functional programming, functions behave as mathematical ones so they always take some parameter and return new data without ever changing the input. They say functional programming is easier to test. Why?

Immutable data structures: all data structures are (or are meant to be) immutable -> no code can ever tweak your data, so other developers just cannot (should not) be able to inadvertently change your data.

Simpler parallel computing: point above is particularly inmportant in parallel computation, wheb the system can schedule thread executions differently *each* time you run the program: this implies that when you have multiple threads it can be very very hard to reproduce a bug where a thread wrongly changes a data which is supposed to be exclusively managed by another one, as it might fail in one run and succeed in another just because the system scheduled differently the code execution ! Functional programming frameworks like Spark[233] solve these problems very nicely.

Easier to reason about code: it is much easier to reason about functions, as we can use standard equational reasoning on input/outputs as traditionally done in algebra. To understand what we're talking about, you can see these slides: Visual functional programming[234] (will talk more about it in class)

[ ]:

6.11 Matrices: list of lists solutions

6.11.1 Download exercises zip

Browse files online[235]

6.11.2 Introduction

Python natively does not provide easy and efficient ways to manipulate matrices. To do so, you would need an external library called numpy which will be seen later in the course. For now we will limit ourselves to using matrices as lists of lists because

1. lists are pervasive in Python, you will probably encounter matrices expressed as lists of lists anyway
2. you get an idea of how to construct a nested data structure
3. we can discuss memory referencies and copies along the way
4. even if numpy internal representation is different, it prints matrices as they were lists of lists

[233] https://spark.apache.org
[234] https://docs.google.com/presentation/d/1hTHty5aML9WDDTvkflvvdGDh0AfZwV_8ZEr10-rUPVA
[235] https://github.com/DavidLeoni/sciprog-ds/tree/master/matrices-lists

6.11.3 What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-exercises
     |- matrix-lists
         |- matrix-list.ipynb
         |- matrix-list-sol.ipynb
```

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-lists.ipynb`
- Go on reading that notebook, and follow instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

Overview

So let's see these lists of lists.For example, we can 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 elemnt `'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:
```

6.11.4 Exercises

Now implement the following functions.

REMEMBER: if the cell is executed and nothing happens, it is 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.

COMMANDMENT 4: You shall never ever reassign function parameters

---

```
def myfun(i, s, L, D):

    # You shall not do any of such evil, no matter what the type of the parameter is:
    i = 666             # basic types (int, float, ...)
    s = "666"           # strings
    L = [666]           # containers
    D = {"evil":666}    # dictionaries
```

(continues on next page)

(continued from previous page)

```
    # For the sole case of composite parameters like lists or dictionaries,
    # you can write stuff like this IF AND ONLY IF the function specification
    # requires you to modify the parameter internal elements (i.e. sorting a list
    # or changing a dictionary field):

    L[4] = 2              # list
    D["my field"] = 5     # dictionary
    C.my_field = 7        # class
```

COMMANDMENT 7: You shall use ``return`` command only if you see written *return* in the 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.

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']
    ]
```

[10]:
```
# write here code for printing rows and columns

# the outer list is a list of rows, so to count htem 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
```

extract_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 you

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 should because you 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 infringment matters aside :-)

extract_row_pointer

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[236]). If 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.

[11]:
```
def extract_row_pointer(mat, i):
    """ RETURN the ith row from mat
        NOTE: the underlying row is returned, so modifications to it will also modify␣
↪original mat
    """
    return mat[i]


m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

row = extract_row_pointer(m, 0)


jupman.pytut()
```

[11]:
```
<IPython.lib.display.IFrame at 0x7fd5acee0be0>
```

extract_row_f

⊛ Now try to implement a version which returns a copy of the row.

You might be tempted to implement something like this:

[12]:
```
# WARNING: WRONG CODE!!!!
# It is adding a LIST as element to another empty list.
# In other words, it is wrapping the row (which is already a list) into another list.
```

(continues on next page)

[236] https://sciprog.davidleoni.it/introduction/introduction-sol.html#Python-Tutor-inside-Jupyter

(continued from previous page)

```
def extract_row(mat, i):
    """ RETURN the ith row from mat. NOTE: the row MUST be a new list ! """

    riga = []
    riga.append(mat[i])
    return riga


# Let's 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.

m = [
      ['a','b'],
      ['c','d'],
      ['a','e'],
]

row = extract_row(m,0)

jupman.pytut()
```

[12]:
```
<IPython.lib.display.IFrame at 0x7fd5acee0898>
```

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 see only *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.

[13]:
```
def extract_row_f(mat, i):
    """ RETURN the ith row from mat.
        NOTE: the row MUST be a new list! To create a new list use a for cycle
              which iterates over the elements, _not_ the indexes (so don't use range!
↪)
    """
    #jupman-raise
    riga = []
    for x in mat[i]:
        riga.append(x)
    return riga
    #/jupman-raise

# 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'],
      ['a','e'],
]

assert extract_row_f(m, 0) == ['a','b']
assert extract_row_f(m, 1) == ['c','d']
assert extract_row_f(m, 2) == ['a','e']

# check it didn't change the original matrix !
```

(continues on next page)

(continued from previous page)

```
r = extract_row_f(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'
# TEST END

# uncomment if you want to visualize execution here (you need to be online for this␣
↪to work)
#jupman.pytut()
```

extract_row_fr

⊛ Now try to iterate over a range of row indexes. Let's have a quick look at `range(n)`. Maybe you think it should return a sequence of integers, from zero to `n - 1`. Does it?

[14]:
```
range(5)
```

[14]:
```
range(0, 5)
```

Maybe you expected to see something like a list `[0,1,2,3,4]`, instead we just discovered Python is pretty lazy here: `range(n)` actually returns an *iterabile* object, not a real sequence materialized in memory.

To get an actual list of integers, we must explicitly ask this iterable object to give us the numbers one by one.

When you write `for i in range(5)` the for cycle is doing exactly this, at each round it is asking the range object to generate a number in the sequence. If we want the whole sequence materialized in memory, we can generate it by converting the range to a list object:

[15]:
```
list(range(5))
```

[15]:
```
[0, 1, 2, 3, 4]
```

Be careful, though. Depending on the size of the sequence, this might be dangerous. A list of billion elements might saturate the RAM of your computer (as of 2018 laptops come with 4 gigabytes of RAM memory, that is 4 billion of bytes).

Now implement the `extract_row_fr` iterating over a range of row indexes:

[16]:
```
def extract_row_fr(mat, i):
    """ RETURN the ith row from mat.
        NOTE: the row MUST be a new list! To create a new list use a for cycle
              which iterates over the indexes, _not_ the elements (so use range!)
    """
    #jupman-raise
    riga = []
    for j in range(len(mat[0])):
        riga.append(mat[i][j])
    return riga
    #/jupman-raise

# 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'],
      ['a','e'],
```

(continues on next page)

(continued from previous page)

```
]

assert extract_row_fr(m, 0) == ['a','b']
assert extract_row_fr(m, 1) == ['c','d']
assert extract_row_fr(m, 2) == ['a','e']

# check it didn't change the original matrix !
r = extract_row_fr(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'
# TEST END

# uncomment if you want to visualize execution here (you need to be online for this␣
↪to work)
#jupman.pytut()
```

extract_row_s

⊛ Remember slices return a *copy* of a list? Now try to use them.

[17]:
```
def extract_row_s(mat, i):
    """ RETURN the ith row from mat.
        NOTE: the row MUST be a new list! To create a new list use slices.
    """
    #jupman-raise
    return mat[i][:]  # if you omit start end end indexes, you get a copy of the␣
↪whole list
    #/jupman-raise

# 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'],
      ['a','e'],
]


assert extract_row_s(m, 0) == ['a','b']
assert extract_row_s(m, 1) == ['c','d']
assert extract_row_s(m, 2) == ['a','e']

# check it didn't change the original matrix !
r = extract_row_s(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'
# TEST END

# uncomment if you want to visualize execution here (you need to be online for this␣
↪to work)
#jupman.pytut()
```

extract_row_c

⊛ Try now to use list comprehensions.

[18]:
```
def extract_row_c(mat, i):
    """ RETURN the ith row from mat.
        NOTE: the row MUST be a new list! To create a new list use list comprehension.
    """
    #jupman-raise
    return [x for x in mat[i]]
    #/jupman-raise

# 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'],
      ['a','e'],
]

assert extract_row_c(m, 0) == ['a','b']
assert extract_row_c(m, 1) == ['c','d']
assert extract_row_c(m, 2) == ['a','e']

# check it didn't change the original matrix !
r = extract_row_c(m, 0)
r[0] = 'z'
assert m[0][0] == 'a'
# TEST END

# uncomment if you want to visualize execution here (you need to be online for this␣
↪to work)
#jupman.pytut()
```

extract_col_f

⊛⊛ Now we can try to extract 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.

[19]:
```
def extract_col_f(mat, j):
    """ RETURN the jth column from mat. To create it, use a for  """

    #jupman-raise
    ret = []
    for row in mat:
        ret.append(row[j])
    return ret
    #/jupman-raise

# 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'],
```

(continues on next page)

(continued from previous page)

```
      ['a','e'],
]

assert extract_col_f(m, 0) == ['a','c','a']
assert extract_col_f(m, 1) == ['b','d','e']

# check returned column does not modify m
c = extract_col_f(m,0)
c[0] = 'z'
assert m[0][0] == 'a'
# TEST END

# uncomment if you want to visualize execution here (you need to be online for this␣
↪to work)
#jupman.pytut()
```

extract_col_c

Difficulty: ⊛⊛

[20]:
```
def extract_col_c(mat, j):
    """ RETURN the jth column from mat. To create it, use a list comprehension  """

    #jupman-raise
    return [row[j] for row in mat]
    #/jupman-raise

# 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'],
      ['a','e'],
]

assert extract_col_c(m, 0) == ['a','c','a']
assert extract_col_c(m, 1) == ['b','d','e']

# check returned column does not modify m
c = extract_col_c(m,0)
c[0] = 'z'
assert m[0][0] == 'a'
# TEST END

# uncomment if you want to visualize execution here (you need to be online for this␣
↪to work)
#jupman.pytut()
```

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.

You might be tempted to write code like this:

[21]:
```
# 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[:] # NOT SUFFICIENT !
                  # This is a SHALLOW clone, it's only copying the _external_ list
                  # and not also the internal ones !

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()
```

[21]:
```
<IPython.lib.display.IFrame at 0x7fd5ac590240>
```

To fix the above code, you will need to iterate through the rows and *for each* row create a copy of that row.

[22]:
```
def deep_clone(mat):
    """ RETURN a NEW list of lists which is a COMPLETE DEEP clone
        of mat (which is a list of lists)
    """
    #jupman-raise

    ret = []
    for row in mat:
        ret.append(row[:])
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

m = [
        ['a','b'],
        ['b','d']
    ]

res = [
        ['a','b'],
        ['b','d']
    ]
```

(continues on next page)

(continued from previous page)

```
# 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'
# TEST END
```

stitch_down

Difficulty: ⊛⊛

[23]:
```
def stitch_down(mat1, mat2):
    """Given matrices mat1 and mat2  as list of lists, with mat1 of size u x n and␣
↪mat2 of size d x n,
       RETURN a NEW matrix of size (u+d) x 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)
    """
    #jupman-raise
    res = []
    for row in mat1:
        res.append(row[:])
    for row in mat2:
        res.append(row[:])
    return res
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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']
     ]
```

(continues on next page)

(continued from previous page)

```
m2 = [
        ['f','b', 'h'],
        ['g','h', 'w']
     ]

res = [
        ['a','b','c'],
        ['d','b','a'],
        ['f','b','h'],
        ['g','h','w']
     ]

assert stitch_down(m1, m2) == res
# TEST END
```

stitch_up

Difficulty: ⊛⊛

[24]:
```
def stitch_up(mat1, mat2):
    """Given matrices mat1 and mat2  as list of lists, with mat1 of size u x n and␣
↪mat2 of size d x n,
       RETURN a NEW matrix of size (u+d) x 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.
    """
    #jupman-raise
    return stitch_down(mat2, mat1)
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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']
```

(continues on next page)

(continued from previous page)

```
     ]
m2 = [
        ['f','b', 'h'],
        ['g','h', 'w']
     ]

res = [
        ['f','b','h'],
        ['g','h','w'],
        ['a','b','c'],
        ['d','b','a']
     ]

assert stitch_up(m1, m2) == res
# TEST END
```

stitch_right

Difficulty: ⊛⊛⊛

[25]:

```
def stitch_right(mata,matb):
    """Given matrices mata and matb  as list of lists, with mata of size n x l and␣
↪matb of size n x r,
       RETURN a NEW matrix of size n x (l + r) as list of lists, by stitching second␣
↪mat to the right end of mat1
    """
    #jupman-raise
    ret = []
    for i in range(len(mata)):
        row_to_add =  mata[i][:]
        row_to_add.extend(matb[i])
        ret.append(row_to_add)
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
ma1 = [
        ['a','b','c'],
        ['d','b','a']
     ]
mb1 = [
        ['f','b'],
        ['g','h']
     ]

r1 = [
        ['a','b','c','f','b'],
        ['d','b','a','g','h']
      ]

assert stitch_right(ma1, mb1) == r1
# TEST END
```

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

[26]:
```
def stitch_left_mod(mat1,mat2):
    """Given matrices mat1 and mat2  as list of lists, with mat1 of size n x l and␣
↪mat2 of size n x r,
       MODIFIES mat1 so that it becomes of size n x (l + r), by stitching second mat␣
↪to the left of mat1

    """
    #jupman-raise
    for i in range(len(mat1)):
        mat1[i][0:0] = mat2[i]
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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
# TEST END
```

Exceptions and parameter checking

Let's look at a parameter validation example (it is not an exercise).

If we wanted to implement a function `mydiv(a,b)` which divides `a` by `b` we could check inside that b is not zero. If it is, we might abruptly stop the function raising a ValueError[237]. In this case the division by zero actually has already a very specific ZeroDivisionError[238], but for the sake of the example we will raise a `ValueError`.

[237] https://docs.python.org/3/library/exceptions.html#ValueError
[238] https://docs.python.org/3/library/exceptions.html#ZeroDivisionError

[27]:
```
def mydiv(a,b):
    """ Divides a by b. If b is zero, raises a ValueError
    """
    if b == 0:
        raise ValueError("Invalid divisor 0")
    return a / b

# to check the function actually raises ValueError when called, we might write a␣
↪quick test like this:

try:
    mydiv(3,0)
    raise Exception("SHOULD HAVE FAILED !")  # if mydiv 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␣
↪catched
    "passed test"  # In an except clause you always need to put some code.
                   # Here we put a placeholder string just to fill in


assert mydiv(6,2) == 3
```

diag

⊛⊛ `diag` extracts the diagonal of a matrix. To do so, `diag` requires an nxn matrix as input. To make sure we actually get an nxn 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 nxn, the function should stop raising an exception. In particular, it shoud raise a ValueError[239], 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 puroposes, 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],
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]
                                   ]
```

[239] https://docs.python.org/3/library/exceptions.html#ValueError

```
                                \ j  0,1,2,3
                                i
                                   [
                                0   [a,b,c,d],
extract from row at i=1  -->    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
                                   [
                                0   [a,b,c,d],
                                1   [e,f,g,h],
extract from row at i=2  -->    2   [p,q,r,s],        'r' is extracted from mat[2][2]
                                3   [t,u,v,z]
                                   ]
```

```
                                 \ j  0,1,2,3
                                 i
                                    [
                                 0   [a,b,c,d],
                                 1   [e,f,g,h],
                                 2   [p,q,r,s],
 extract from row at i=3  -->    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?

About perfomances

For the purposes of the first part of the course, performance considerations won't be part of the evaluation. So if all the tests run in a decent time on your laptop (and the code is actually correct!), then the exercise is considered solved, even if there are better algorithmic ways to solve it. Typically in this first part you won't have many performance problems, except when we will deal with 100 mb files - in that cases you will be forced to use the right method otherwise your laptop will just keep keep heating without spitting out results

In the second part of the course, we will consider performance indeed, so in that part using a double for would be considered an unacceptable waste.

---

[28]:
```
def diag(mat):
    """ Given an nxn 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 nxn raise ValueError
    """
```

(continues on next page)

(continued from previous page)

```
    #jupman-raise
    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
    #/jupman-raise

# 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([              # 1x2 dimension, not square
           ['a','b']
         ])
    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␣
↪catched
    "passed test"  # In an except clause you always need to put some code.
                   # Here we put a placeholder string just to fill in
# TEST END
```

anti_diag

⊛⊛ Before implementing it, be sure to write down understand the required indeces as we did in the example for the *diag* function.

[29]:
```
def anti_diag(mat):
    """ Given an nxn matrix mat as a list of lists, RETURN a list which contains the␣
↪elemets in the antidiagonal
    (top right to bottom left corner). If mat is not nxn raise ValueError
    """
    #jupman-raise
    n = len(mat)
    ret = []
    for i in range(n):
        ret.append(mat[i][n-i-1])
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
```

(continues on next page)

(continued from previous page)

```
m = [
        ['a','b','c'],
        ['d','e','f'],
        ['g','h','i']
     ]

assert anti_diag(m) == ['c','e','g']
# TEST END

# If you have doubts about the indexes remember to try it in python tutor !
# jupman.pytut()
```

is_utriang

⊛⊛⊛ You will now try to iterate only the lower triangular half of a matrix. Let's look at an example:

[30]:
```
m = [
        [3,2,5,8],
        [0,6,2,3],
        [0,0,4,9],
        [0,0,0,5]
    ]
```

Just for illustrative puroposes, 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
```

(continues on next page)

(continued from previous page)

```
                                  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 triangular
```

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 `for`s, 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 is 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:

COMMANDMENT 9: 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 is as confusing as reassigning function parameters.

So avoid this sins:

[31]:
```
i = 7
for i in range(3):  # sin, you lose i variable
    print(i)
```

```
0
1
2
```

[32]:
```
def f(i):
    for i in range(3): # sin again, you lose i parameter
        print(i)
```

[33]:
```
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
```

If you read *all* the above, start implementing the function:

[34]:
```
def is_utriang(mat):
    """ Takes a 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.
    """
    #jupman-raise
    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
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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],
```

(continues on next page)

(continued from previous page)

```
    [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
# TEST END
```

transpose_1

⊛⊛⊛ 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` funciton can also have a start index:

[35]:
```
list(range(3,7))
```

[35]:
```
[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

[36]:
```
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()
```

[36]:
```
<IPython.lib.display.IFrame at 0x7fd5ac587e80>
```

Going back to the transpose, for now we will consider only an nxn matrix. To make sure we actually get an nxn 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 nxn, the function should

stop raising an exception. In particular, it shoud raise a ValueError[240], 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.

COMMANDMENT 4 (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
```

If you read *all* the above, you can now proceed implementing the `transpose_1` function:

[37]:
```
def transpose_1(mat):
    """ MODIFIES given nxn matrix mat by transposing it *in-place*.
        If the matrix is not nxn, raises a ValueError
    """
    #jupman-raise
    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
    #/jupman-raise


# 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:
    transpose_1([
                [3,5]
              ])
    raise Exception("SHOULD HAVE FAILED !")
except ValueError:
    "passed test"
```

(continues on next page)

[240] https://docs.python.org/3/library/exceptions.html#ValueError

(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']
             ]
# TEST END
```

empty matrix

⊛⊛ There are several ways to create a new empty 3x5 matrix as lists of lists which contains zeros. Try to create one with two nested `for` cycle:

[38]:
```
def empty_matrix(n, m):
    """
    RETURN a NEW nxm matrix as list of lists filled with zeros. Implement it with a␣
↪nested for
    """
    #jupman-raise
    ret = []
    for i in range(n):
        row = []
        ret.append(row)
        for j in range(m):
            row.append(0)
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

assert empty_matrix(1,1) == [
    [0]
]

assert empty_matrix(1,2) == [
    [0,0]
]

assert empty_matrix(2,1) == [
    [0],
```

(continues on next page)

(continued from previous page)

```
    [0]
]

assert empty_matrix(2,2) == [
    [0,0],
    [0,0]
]

assert empty_matrix(3,3) == [
    [0,0,0],
    [0,0,0],
    [0,0,0]
]
# TEST END
```

empty_matrix the elegant way

To create a new list of 3 elements filled with zeros, you can write like this:

[39]:
```
[0]*3
```

[39]:
```
[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:

[40]:
```
# WRONG
[[0]*3]*5
```

[40]:
```
[[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:

[41]:
```
bad = [[0]*3]*5
jupman.pytut()
```

[41]:
```
<IPython.lib.display.IFrame at 0x7fd5ac523d30>
```

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:

[42]:
```
# CORRECT
[[0]*3 for i in range(5)]
```

[42]:
```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

transpose_2

⊛⊛ Now let's try to transpose a generic nxm matrix. This time for simplicity we will return a whole new matrix.

[43]:
```
def transpose_2(mat):
    """ RETURN a NEW mxn matrix which is the transpose of the given nxm matrix mat as␣
↪list of lists.
    """
    #jupman-raise
    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]
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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'],
            ]
# TEST END
```

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
```

[44]:
```
def threshold(mat, t):

    #jupman-raise
    ret = []
    for row in mat:
        new_row = []
        ret.append(new_row)
        for el in row:
            new_row.append(el > t)

    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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
# TEST END
```

swap_rows

Difficulty: ⊛⊛

[45]:
```
def swap_rows(mat, i1, i2):
    """Takes a matrix as list of lists, and RETURN a NEW matrix where rows at indexes␣
↪i1 and i2 are swapped
    """
    #jupman-raise

    # deep clones
    ret = []
    for row in mat:
        ret.append(row[:])
    #swaps
    s = ret[i1]
    ret[i1] = ret[i2]
    ret[i2] = s
    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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']
```

(continues on next page)

(continued from previous page)

```
                ]

r2[0][0] = 'z'
assert m2[0][0] == 'a'
# TEST END
```

swap_cols

⊛⊛ RETURN a NEW matrix where the columns `j1` and `j2` are swapped

[46]:
```
def swap_cols(mat, j1, j2):
    #jupman-raise
    ret = []
    for row in mat:
        new_row = row[:]
        new_row[j1] = row[j2]
        new_row[j2] = row[j1]
        ret.append(new_row)
    return ret
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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'
# TEST END
```

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.

Now implement the algorithm.

INPUT:

- students: a list of strings of length <= n*m

- chairs: an nxm 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

Example:

```
ss =  ['b', 'd', 'e', 'g', 'c', 'a', 'h', 'f' ]

mat = [
            [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

[47]:
```
def lab(students, chairs):
    #jupman-raise

    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
    #/jupman-raise


try:
    lab(['a','b'], [[None]])
```

(continues on next page)

(continued from previous page)

```
    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]
              ]


m3 = [
        [None],
        [None],
     ]
lab(['a'], m3) # 1 student 2 chairs in one column
assert m3 == [
                ['a'],
                [None]
              ]

ss4 = ['b', 'a']
m4 = [
        [None, None]
```

(continues on next page)

(continued from previous page)

```
     ]
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],
             ]
```

(continues on next page)

(continued from previous page)

```
assert ss8 == ['a','b','c','d','e','f','g','h']
```

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 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`.

NOTE: the function does **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.

[48]:
```
def dump(mat, q):
    #jupman-raise
    rem = q
    ret = []

    for riga in mat:
        for j in range(len(riga)):
            cellfill = 7 - riga[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:
```

(continues on next page)

(continued from previous page)

```
        raise ValueError("Couldn't fill the dump, %s tons remain!")
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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]


m6 = [          # tons to dump in each cell
    [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
```

(continues on next page)

(continued from previous page)

```
# TEST END
```

matrix multiplication

⊛⊛⊛ Have a look at matrix multiplication definition[241] on Wikipedia and try to implement it in the following function.

Basically, gicen nxm matrix A and mxp matrix B you need to output an nxp 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 nxp cells of C, so sure enough to fill a rectangle you need two `fors`. Do you also need another `for` ? Help yourself with the following visualization.

```
[49]: def mul(mata, matb):
          """ Given matrices n x m mata and m x p matb, RETURN a NEW n x 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.
          """
          #jupman-raise
```

(continues on next page)

[241] https://en.wikipedia.org/w/index.php?title=Matrix_multiplication&section=2#Definition

(continued from previous page)

```
    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
    #/jupman-raise

# 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:
    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)
```

(continues on next page)

(continued from previous page)

```
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]
               ]
# TEST END
```

check_nqueen

⊛⊛⊛⊛ This is a hard problem but don't worry, exam exercises will be simpler!

You have an nxn 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  ........
    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  .../|\..
```

(continues on next page)

(continued from previous page)

```
    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  ..\.|./
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  ....|....
```

(continues on next page)

(continued from previous page)

```
 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.

[50]:

```
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

    """
    #jupman-raise

    # 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):
        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
    #/jupman-raise
```

(continues on next page)

(continued from previous page)

```
# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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
# TEST END
```

6.12 Matrices: Numpy solutions

6.12.1 Download exercises zip

Browse files online[242]

6.12.2 Introduction

References:

- Andrea Passerini slides A08[243]
- Python Data Science Handbook, Numpy part[244]

Previously we've seen Matrices as lists of lists[245], here we focus on matrices using Numpy library

There are substantially two ways to represent matrices in Python: as list of lists, or with the external library numpy[246]. The most used is surely Numpy, let's see the reason the principal differences:

List of lists - see separate notebook[247]

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. give 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. 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)

Here we will see data types and essential commands of Numpy library[248], but we will not get into the details.

The idea is to simply pass using the the data format `ndarray` without caring too much about performances: for example, even if `for` cycles in Python are slow because they operate cell by cell, we will use them anyway. In case you actually need to execute calculations fast, you will want to use operators on vectors but for this we invite you to read links below

ATTENTION: if you want to use Numpy in Python tutor[249], instead of default interpreter `Python 3.6` you will need to select `Python 3.6 with Anaconda` (at May 2019 results marked as experimental)

[242] https://github.com/DavidLeoni/sciprog-ds/tree/master/matrices-numpy
[243] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A08-numpy.pdf
[244] https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html
[245] https://sciprog.davidleoni.it/matrices-lists/matrices-lists-sol.html
[246] https://www.numpy.org
[247] https://sciprog.davidleoni.it/matrices-lists/matrices-lists-sol.html
[248] https://www.numpy.org
[249] http://www.pythontutor.com/visualize.html#mode=edit

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-exercises
     |- matrices-numpy
         |- matrices-numpy.ipynb
         |- matrices-numpy-sol.ipynb
```

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-numpy.ipynb`
- Go on reading that notebook, and follow instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.12.3 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.

To access data or overwrite square bracket notation is used, with the important difference that in Numpy you can write *bot* 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!

Let's put number `0` in cell at row `0` and column `1`

[5]:
```
mat[0,1] = 9
```

[6]:
```
mat
```

[6]:
```
array([[0., 9., 0.],
       [0., 0., 0.]])
```

Let's access cell at row `0` and column `1`

[7]:
```
mat[0,1]
```

[7]:
```
9.0
```

We put number `7` into cell at row `1` and column `2`

[8]:
```
mat[1,2] = 7
```

[9]:
```
mat
```

[9]:
```
array([[0., 9., 0.],
       [0., 0., 7.]])
```

To get the dimension, we write like the following:

> **ATTENTIONE**: after `shape` there are **no** round parenthesis !
>
> `shape` is an attribute, not a function to call

[10]:
```
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
```

⊛ **Exercise**: try to write like the following, what happens?

```
mat[0,0] = "c"
```

[14]:
```
# write here
```

We can also create an `ndarray` starting from a list of lists:

[15]:
```
mat = np.array( [ [5.0,8.0,1.0],
                  [4.0,3.0,2.0]])
```

[16]:
```
mat
```

[16]:
```
array([[5., 8., 1.],
       [4., 3., 2.]])
```

[17]:
```
type(mat)
```

[17]:
```
numpy.ndarray
```

[18]:
```
mat[1,1]
```

[18]:
```
3.0
```

⊛ **Exercise**: Try to write like this and check what happens:

```
mat[1,1.0]
```

[19]:
```
# write here
```

6.12.4 NaNs 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.

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.

NaNs

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:

[20]:
```
import math
math.nan    # notice it prints as 'nan' with lowercase n
```

[20]:
```
nan
```

As we said, a NaN is actually considered a float:

[21]:
```
type(math.nan)
```

[21]:
```
float
```

Still, it behaves very differently from its fellow floats, or any other object in the known universe:

[22]:
```
math.nan == math.nan    # what the F... alse
```

[22]:
```
False
```

Detecting NaN

Given the above, if you want to check if a variable `x` is a NaN, you *cannot* write this:

[23]:
```
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:

[24]:
```
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:

[25]:
```
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:

[26]:
```
[math.nan, math.nan] == [math.nan, math.nan]
```

[26]:
```
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"

[27]:
```
# expected output: same
x = math.nan
y = math.nan
```

(continues on next page)

(continued from previous page)

```
# 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
if math.isnan(x) and math.isnan(y):
    print('same')
elif x == y:
    print('same')
else:
    print('not the same')
```

```
same
```

Operations on NaNs

Any operation on a NaN will generate another NaN:

[28]:
```
5 * math.nan
```

[28]:
```
nan
```

[29]:
```
math.nan + math.nan
```

[29]:
```
nan
```

[30]:
```
math.nan / math.nan
```

[30]:
```
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`:

[31]:
```
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:

[32]:
```
np.nan
```

[32]:
```
nan
```

[33]:
```
math.isnan(np.nan)
```

[33]:
```
True
```

[34]:
```
np.isnan(math.nan)
```

[34]:
```
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:

[35]:
```
import numpy as np
np.array([4.9,None,3.2,5.1])
```

[35]:
```
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:

[36]:
```
np.array([4.9,np.nan,3.2,5.1])    # Notice how the `dtype=object` has disappeared
```

[36]:
```
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:

```
>>> math.log(-1)
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-38-d6e02ba32da6> in <module>
----> 1 math.log(-1)

ValueError: math domain error
```

Let's try the equivalent with Numpy:

[37]:
```
np.log(-1)
```

```
/home/da/Da/bin/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:␣
↪RuntimeWarning: invalid value encountered in log
  """Entry point for launching an IPython kernel.
```

[37]:
```
nan
```

Notice we actually got as a result `np.nan`, even if Jupyter is printing a warning.

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:

[38]:
```
np.log(np.array([3,7,-1,9]))
```

```
/home/da/Da/bin/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:␣
↪RuntimeWarning: invalid value encountered in log
  """Entry point for launching an IPython kernel.
```

[38]:
```
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 misteries 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[250]):

Positive infinity `np.inf`

[39]:
```
np.array( [ 5 ] ) / 0
```

```
/home/da/Da/bin/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[39]:
```
array([inf])
```

[40]:
```
np.array( [ 6,9,5,7 ] ) / np.array( [ 2,0,0,4 ] )
```

```
/home/da/Da/bin/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[40]:
```
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:

[41]:
```
np.inf == np.inf
```

[41]:
```
True
```

so we can safely detect infinity with ==:

[42]:
```
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`:

[43]:
```
np.isinf(np.inf)
```

[43]:
```
True
```

[250] https://numpy.org/devdocs/reference/constants.html

Negative infinity

We can also have negative infinity, which is different from positive infinity:

[44]:
```
-np.inf == np.inf
```

[44]:
```
False
```

Note that `isinf` detects *both* positive and negative:

[45]:
```
np.isinf(-np.inf)
```

[45]:
```
True
```

To actually check for negative infinity you have to use `isneginf`:

[46]:
```
np.isneginf(-np.inf)
```

[46]:
```
True
```

[47]:
```
np.isneginf(np.inf)
```

[47]:
```
False
```

Where do they appear? As an example, let's try `np.log` function:

[48]:
```
np.log(0)
```

```
/home/da/Da/bin/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:␣
↪RuntimeWarning: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

[48]:
```
-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:

[49]:
```
np.inf + np.inf
```

[49]:
```
inf
```

[50]:
```
- np.inf - np.inf
```

[50]:
```
-inf
```

[51]:
```
np.inf * -np.inf
```

[51]:
```
-inf
```

What in classical analysis would be undefined, here becomes NaN:

[52]:
```
np.inf - np.inf
```

[52]:
```
nan
```

[53]:
```
np.inf / np.inf
```

[53]:
```
nan
```

As usual, combining with NaN results in NaN:

[54]:
```
np.inf + np.nan
```

[54]:
```
nan
```

[55]:
```
np.inf / np.nan
```

[55]:
```
nan
```

Negative zero

We can even have a *negative* zero - who would have thought?

[56]:
```
np.NZERO
```

[56]:
```
-0.0
```

Negative zero of course pairs well with the more known and much appreciated *positive* zero:

[57]:
```
np.PZERO
```

[57]:
```
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:

[58]:
```
0.0 == -0.0
```

[58]:
```
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:

[59]:
```
np.array( [ 5.0 ] ) / np.array( [ 0.0 ] )
```

```
/home/da/Da/bin/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[59]:
```
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:

[60]:
```
np.array( [ 5.0 ] ) / np.array( [ -0.0 ] )
```

```
/home/da/Da/bin/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:␣
↪RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[60]:
```
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 costants[251]

[1]:
```
# expected: equal numbers
x = 5
y = 5

# expected: not equal numbers
#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
```

(continues on next page)

[251] https://docs.scipy.org/doc/numpy/reference/constants.html

(continued from previous page)

```
#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')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-32186ec2496f> in <module>()
     35
     36 # SOLUTION 1 - the ugly one
---> 37 if np.isinf(x) or np.isinf(y) or np.isnan(x) or np.isnan(y):
     38     print('not equal numbers')
     39 else:

NameError: name 'np' is not defined
```

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))
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]
```

6.12.5 Verify comprehension

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.]])
```

Hints:

- Since you need to return a matrix, start with creating an empty one
- go through the whole input matrix with indeces `i` and `j`

[62]:
```
import numpy as np

def odd(mat):
    #jupman-raise
    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
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

m1 = np.array([
                [2],
               ])
m2 = np.array([
                [3]
               ])
assert np.allclose(odd(m1),
                   m2)
```

(continues on next page)

(continued from previous page)

```
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)

# TEST END
```

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( [                        #  index
                    [ 2, 5, 6, 3],      #    0      even
                    [ 8, 4, 3, 5],      #    1      odd
                    [ 7, 1, 6, 9],      #    2      even
                    [ 5, 2, 4, 1],      #    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]])
```

[63]:
```
import numpy as np

def doublealt(mat):
    #jupman-raise
    nrows, ncols = mat.shape
    ret = np.zeros( (nrows, ncols) )

    for i in range(nrows):
        for j in range(ncols):
```

(continues on next page)

(continued from previous page)

```
            if i % 2 == 0:
                ret[i,j] = mat[i,j] * 2
            else:
                ret[i,j] = mat[i,j]
    return ret
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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)


# TEST END
```

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]])
```

Ingredients:

- create a matrix filled with zeros. ATTENTION: which dimensions does it have? Do you need `n` or `k` ? Read WELL the text.
- start by filling the cells of first row with `k` values. To iterate along the first row columns, use a `for j in range(n)`
- fill other rows and columns, using appropriate `for`

[64]:
```
def frame(n, k):
    #jupman-raise
    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
    #/jupman-raise


# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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 Ture 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)
# TEST END
```

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]])
```

Ingredients:

- 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, …

[65]:
```
def chessboard(n):
    #jupman-raise
    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
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

expected_mat = 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]])

# all_close return True if all the values in the first matrix are close enough
# (that is, within a certain tolerance) to the corresponding ones in the second matrix
assert np.allclose(chessboard(4), expected_mat)

expected_mat = np.array( [ [1.0]
                          ])
assert np.allclose(chessboard(1), expected_mat)

expected_mat = np.array( [ [1.0, 0.0],
                           [0.0, 1.0]
                        ])
assert np.allclose(chessboard(2), expected_mat)
# TEST END
```

altsum

⊛⊛⊛ MODIFY the input Numpy matrix (n x 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]]
```

Ingredients:

- 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, ..

[66]:
```
def altsum(mat):
    #jupman-raise
    nrows, ncols = mat.shape
    for i in range(1,nrows, 2):
        for j in range(0,ncols):
            mat[i, j] = mat[i,j] + mat[i-1, j]
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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)

m2 = np.array( [ [5.0]  ])
r2 = np.array( [ [5.0] ])
altsum(m1)
assert np.allclose(m2, r2)
```

(continues on next page)

(continued from previous page)

```
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)
# TEST END
```

avg_rows

⊛⊛⊛ Takes a Numpy matrix n x 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
```

Ingredients:

- create a matrix n x 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

[67]:
```
def avg_rows(mat):
    #jupman-raise
    nrows, ncols = mat.shape

    ret = np.zeros( (nrows,1)  )

    for i in range(nrows):

        for j in range(ncols):
            ret[i] += mat[i,j]
```

(continues on next page)

(continued from previous page)

```
        ret[i] = ret[i] / ncols
        # for brevity we could also write
        # ret[i] /= colonne
    #/jupman-raise
    return ret

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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)
# TEST END
```

avg_half

⊛⊛⊛ Takes as input a Numpy matrix withan 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

[68]:
```
def avg_half(mat):
    #jupman-raise
    nrows, ncols = mat.shape
    half_cols = ncols // 2

    avg_sx = 0.0
    avg_dx = 0.0

    # scrivi qui
    for i in range(nrows):
        for j in range(half_cols):
            avg_sx += mat[i,j]
```

(continues on next page)

(continued from previous page)

```
        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])
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

m1 = np.array([[3,2,1,4],
               [6,2,3,5],
               [4,3,6,2],
               [4,6,5,4],
               [7,2,9,3]])

r1 = 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(m1), r1)
# TEST END
```

matxarr

⊛⊛⊛ Takes a Numpy matrix `n` x `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.

[69]:

```
def matxarr(mat, arr):
    #jupman-raise
    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
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`
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]]
```

(continues on next page)

(continued from previous page)

```
assert np.allclose(matxarr(m1,a1), r1)
# TEST END
```

quadrants

⊛⊛⊛ Given a matrix `2n * 2n`, divide the matrix in 4 equal square parts (see example) and RETURN a NEW matrix `2 * 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
-----------------------------             =>         1.0 , 2.0
(2+0+0+2)/4    | (5+1+1+1)/4
```

[70]:

```
import numpy as np

def quadrants(mat):
    #jupman-raise
    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]
```

(continues on next page)

(continued from previous page)

```
    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

    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
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

assert np.allclose(
    quadrants(np.array([
                            [3.0, 5.0],
                            [4.0, 9.0],
                        ])),
                np.array([
                            [3.0, 5.0],
                            [4.0, 9.0],
                        ]))

assert np.allclose(
    quadrants(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]
                        ])),
                np.array([
                          [2.0, 5.0],
                          [1.0, 2.0]
                        ]))

# TEST END
```

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 zertoth 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]
            ])
```

[71]:
```
import numpy as np

def matrot(mat):
    #jupman-raise
    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
    #/jupman-raise

# TEST START - DO NOT TOUCH!
# if you wrote the whole code correct, and execute the cell, Python shouldn't raise␣
↪`AssertionError`

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]
                ])
assert np.allclose(matrot(m4), r4)
```

(continues on next page)

(continued from previous page)

```
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)
# TEST END
```

Other Numpy exercises

- Try to do exercises from liste di liste[252] using Numpy instead.
- try to do the exercises more performant by using Numpy features and functions (i.e. `2*arr` multiplies all numbers in arr without the need of a slow Python `for`)
- (in inglese) machinelearningplus[253] Esercizi su Numpy (Fermarsi a difficoltà L1, L2 e se vuoi prova L3)

[ ]:

```
```

6.13 Data formats solutions

6.13.1 Download exercises zip

Browse files online[254]

6.13.2 Introduction

Here we review how to load and write tabular data such as CSV, tree-like data such as JSON files, and how to fetch it from the web with webapis.

Graph formats[255] are treated in a separate notebook.

In this tutorial we will talk about data formats

- textual files
  - line-based files
  - CSV

[252] https://sciprog.davidleoni.it/matrices-lists/matrices-lists-sol.html
[253] https://www.machinelearningplus.com/python/101-numpy-exercises-python/
[254] https://github.com/DavidLeoni/sciprog-ds/tree/master/formats
[255] https://sciprog.davidleoni.it/graph-formats/graph-formats-sol.html

  - opendata catalogs
  - license mention (creative commons, ..)

In a separate notebook we will discuss graph formats[256]

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-exercises
     |- matrices
         |- formats.ipynb
         |- formats-sol.ipynb
```

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 `matrix-networks/matrix-networks.ipynb`
- Go on reading that notebook, and follow instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.13.3 1. 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
```

[256] https://sciprog.davidleoni.it/graph-formats/graph-formats-sol.html

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 examing 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 dependes 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[257], and Dive into Python - Chapter 11 - File[258], **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.

[257] https://diveintopython3.problemsolving.io/strings.html
[258] https://diveintopython3.problemsolving.io/files.html

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 !

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 from 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.

[2]:
```
# write here

with open('people-simple.txt', encoding='utf-8') as f:
    line=f.readline()
    print(line)
```

```
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))`
- `print(help(f))`
- `print(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

[3]:
```
# write here the code (copy and paste)
with open('people-simple.txt', encoding='utf-8') as f:
    line=f.readline()
    print(type(f))
    print(help(f.readline))
    print(help(f))
    print(line)
```

```
<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.

None
Help on TextIOWrapper object:

class TextIOWrapper(_TextIOBase)
 |  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.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) 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.
 |
 |  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:
 |
 |      * 0 -- start of stream (the default); offset should be zero or positive
 |      * 1 -- current stream position; offset may be negative
```

(continues on next page)

(continued from previous page)

```
 |      * 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).
 |
 |  ----------------------------------------------------------------------
 |  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
 |
 |  newlines
 |      Line endings translated so far.
 |
 |      Only line endings translated during reading are considered.
 |
 |      Subclasses should override.
 |
```

(continues on next page)

(continued from previous page)

```
 |  ----------------------------------------------------------------------
 |  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, /)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from _IOBase:
 |
 |  __dict__

None
Leonardo
```

First we put the content of the first line into the variable `line`, now we might put it in a variable witha 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)`

[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
```

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[259] 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 !

We first read a line, and two cases are possible:

a. we are 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 != "":                    # enter cycle if line contains characters
    name = line.rstrip()             # parses the name
    surname=f.readline().rstrip()    # reads next line and parses surname
    print(name + ' ' + surname)
    line=f.readline()                # read next line
```

[259] https://thispointer.com/5-different-ways-to-read-a-file-line-by-line-in-python/

⊛ **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):

[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
```

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 ue 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':

[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
```

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[260]. 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
```

(continues on next page)

[260] http://softpython.readthedocs.io/it/latest/_static/toc-immersione-in-python-3.txt

(continued from previous page)

```
BookmarkBegin
BookmarkTitle: Ogni cosa &#232; un oggetto
BookmarkLevel: 2
BookmarkPageNumber: 47
```

Write a python program to print the following output:

```
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 qunatity determined by the rows `BookmarkLevel`

QUESTION: what's that weird value `è` 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 `>`, `>`, `&x3e;`, what should we do? As good data cleaners, we recognize these are HTML escape sequences[261], so we could use methods specific to sequences like html.escape[262]. 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:

[261] https://corsidia.com/materia/web-design/caratterispecialihtml
[262] https://docs.python.org/3/library/html.html#html.unescape

[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:

[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
```

6.13.4 2. File CSV

There can be various formats for tabular data, among which you surely know Excel (`.xls` or `.xslx`). 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 witha 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

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 moder 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. Tyipcally 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 ?

ANSWER: no, it is sufficient to keep in memory one row at a time, and hold the count in another variable

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?

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.

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:

[20]:
```
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 sequnce 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 !')
```

(continues on next page)

(continued from previous page)

```
        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 specificed 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 `lettore` 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:

[21]:
```
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 sequnce 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']
```

⊛⊛ **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:

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

HINT: Try creating an empty list and then adding elements with `.append` method

[22]:
```
# 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=',')
```

(continues on next page)

(continued from previous page)

```
    # 'my_reader' is an object considered 'iterable', that is,
    # if used in a 'for' will produce a sequnce 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)
    print(big_list)
```

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

⊛⊛ **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')`

[23]:
```
# 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]]
```

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:

[24]:
```
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 0x7f58767de978>
```

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?

[25]:
```
# 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']]
```

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?

[ ]:
```

```

[26]:
```
# write here the code

#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))
```

⊛ **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 ?

[27]:
```
# 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))     # COMMENTED, AS IT WOULD RAISE ON ERROR OF CLOSED FILE
                            # We can't use commands which read the file outside the␣
↪with !
```

⊛⊛ **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[263] 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)

[28]:
```
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]]
```

⊛ **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:

[29]:
```
# write here
```

ANSWER:

```
animal,lifespan,description
dog,12,dogs walk
cat,14,cats walk
pelican,30,pelicans fly
squirrel,6,squirrels fly
eagle,25,eagles fly
```

⊛ **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:

[263] https://docs.python.org/3/library/itertools.html#itertools.islice

- 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?

ANSWER: yes

QUESTION 2.11.2: if we convert to integer, is the space a problem?

ANSWER: no

QUESTION 2.11.3 Modify only dogs description from `dogs walk` to `dogs walk, but don't fly` and try to riexecute the cell which opens the file. What happens?

ANSWER: Python reads one element more in the list

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 ?

ANSWER: yes

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[264]

NOTE: different Python versions give different dictionaries:

- $<$ 3.6: `dict`
- 3.6, 3.7: `OrderedDict`
- $\geq$ 3.8: `dict`

Python 3.8 returned to old `dict` because in the implementation of its dictionariesthe key order is guaranteed, so it will be consistent with the one of CSV headers

[30]:
```
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)
```

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

[264] https://docs.python.org/3/library/csv.html#csv.DictReader

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 !**

[31]:
```
import csv

# To write, REMEMBER to specify the `w` option.
# WARNING: 'w' *completely* replaces existing files !!
with open('written-file.csv', 'w', 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 !

[32]:
```
import csv

# To write, REMEMBER to specify the `w` option.
# WARNING: 'w' *completely* replaces existing files !!
# WARNING: handle here  is called *csvfile_out*
with open('example-1-enriched.csv', 'w', encoding='utf-8', newline='') as csvfile_out:
    my_writer = csv.writer(csvfile_out, delimiter=',')

    # Notice how this 'with' is *inside* the outer one:
    # 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=',')

        for row in my_reader:
            row.append('something else')
            my_writer.writerow(row)
            my_writer.writerow(row)
            my_writer.writerow(row)
```

Let's see the new file was actually created by reading it:

[33]:
```
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']
['animal', ' lifespan', 'something else']
['animal', ' lifespan', 'something else']
['dog', '12', 'something else']
['dog', '12', 'something else']
['dog', '12', 'something else']
['cat', '14', 'something else']
['cat', '14', 'something else']
['cat', '14', 'something else']
['pelican', '30', 'something else']
['pelican', '30', 'something else']
['pelican', '30', 'something else']
['squirrel', '6', 'something else']
['squirrel', '6', 'something else']
['squirrel', '6', 'something else']
['eagle', '25', 'something else']
['eagle', '25', 'something else']
['eagle', '25', 'something else']
```

CSV Botteghe storiche

Usually in open data catalogs like the popular CKAN platform (for example dati.trentino.it[265], data.gov.uk[266], European data portal[267] 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 organizazion which created the data.

The first dataset we wil 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[268], 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_9_5_2019.1557403385.csv

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):

[265] http://dati.trentino.it/
[266] https://data.gov.uk/
[267] https://www.europeandataportal.eu/
[268] https://creativecommons.org/licenses/by/4.0/deed.en

```
Numero,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 â€œALLA NAVEâ€,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[269]

- **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`

[269] 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 Valsugana	38051		esercizio commerciale
4	3	FOTOGRAFICA TRINTINAGLIA UMBERTO S.N.C.	Largo Dordi	8	Borgo Valsugana	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.'),
            ('Indirizzo', 'Corso Ausugum'),
```

(continues on next page)

(continued from previous page)

```
             ('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)]
```

[34]:
```
def rank_categories(stopwords):
    #jupman-raise
    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)
    #/jupman-raise

stopwords = ['',
             'S.N.C.', 'SNC','S.A.S.', 'S.R.L.', 'S.C.A.R.L.', 'SCARL','S.A.S',
↪'COMMERCIALE','FAMIGLIA','COOPERATIVA',
```

(continues on next page)

(continued from previous page)

```
            '-', '&', 'C.', 'ESERCIZIO',
            'IL', 'DE', 'DI','A', 'DA', 'E', 'LA', 'AL',  'DEL', 'ALLA', ]
categories = rank_categories(stopwords)

categories
```

[34]:
```
[('BAR', 191),
 ('RISTORANTE', 150),
 ('HOTEL', 67),
 ('ALBERGO', 64),
 ('MACELLERIA', 27),
 ('PANIFICIO', 22),
 ('FARMACIA', 21),
 ('CALZATURE', 21),
 ('ALIMENTARI', 20),
 ('PIZZERIA', 16),
 ('SPORT', 16),
 ('FERRAMENTA', 12),
 ('TABACCHI', 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[270]

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'),
```

(continues on next page)

[270] https://docs.python.org/3/library/csv.html#csv.DictWriter

(continued from previous page)

```
                 ('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', '[]')])
```

[35]:
```
def enrich(categories):
    #jupman-raise
    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-enriched-solution.csv', 'w', newline='', encoding='utf-8') as␣
↪csvfile_out:

        writer = csv.DictWriter(csvfile_out, fieldnames=fieldnames)
        writer.writeheader()

        with open('botteghe.csv', newline='',  encoding='utf-8',) as csvfile_in:
            reader = csv.DictReader(csvfile_in,  delimiter=',')
            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)

    #/jupman-raise
```

(continues on next page)

(continued from previous page)

```
enrich(rank_categories(stopwords))
```

[36]:
```
# 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
    pp = pprint.PrettyPrinter(indent=4)
    for i in range(5):
        d = next(reader)
        pp.pprint(d)
```

```
{   'Cap': '38068',
    'Categorie': "['BAR', 'RISTORANTE']",
    'Civico': '30',
    'Comune': 'Sover',
    'Frazione/Località': 'Piscine di Sover',
    'Indirizzo': 'Via del Lagorai',
    'Insegna': 'BAZZANELLA RENATA',
    'Note': 'generi misti, bar - ristorante',
    'Numero': '1'}
{   'Cap': '38051',
    'Categorie': '[]',
    'Civico': '48',
    'Comune': 'Borgo Valsugana',
    'Frazione/Località': '',
    'Indirizzo': 'Corso Ausugum',
    'Insegna': 'CONFEZIONI MONTIBELLER S.R.L.',
    'Note': 'esercizio commerciale',
    'Numero': '2'}
{   'Cap': '38051',
    'Categorie': '[]',
    'Civico': '8',
    'Comune': 'Borgo Valsugana',
    'Frazione/Località': '',
    'Indirizzo': 'Largo Dordi',
    'Insegna': 'FOTOGRAFICA TRINTINAGLIA UMBERTO S.N.C.',
    'Note': 'esercizio commerciale, attività artigianale',
    'Numero': '3'}
{   'Cap': '38055',
    'Categorie': "['BAR']",
    'Civico': '24',
    'Comune': 'Grigno',
    'Frazione/Località': 'Serafini',
    'Indirizzo': '',
    'Insegna': 'BAR SERAFINI DI MINATI RENZO',
    'Note': 'esercizio commerciale',
    'Numero': '4'}
{   'Cap': '38066',
    'Categorie': '[]',
```

(continues on next page)

(continued from previous page)

```
    'Civico': '35',
    'Comune': 'Riva del Garda',
    'Frazione/Località': '',
    'Indirizzo': 'Via S. Francesco',
    'Insegna': 'SEMBENINI GINO & FIGLI S.R.L.',
    'Note': '',
    'Numero': '6'}
```

[ ]:

```

```

6.14 Graph formats solutions

6.14.1 Download exercises zip

Browse files online[271]

6.14.2 Introduction

Usual matrices from linear algebra are of great importance in computer science because they are widely used in many fields, for example in machine learning and network analysis. This tutorial will give you an appreciation of the meaning of matrices when considered as networks or, as we call them in computer science, *graphs*. We will also review other formats for storing graphs, such as *adjacency lists* and a have a quick look at a specialized library called Networkx.

In Part A we will limit ourselves to graph formats in this notebook and see some theory in separate binary relations notebook[272], while in Part B of the course will focus on graph algorithms[273].

What to do

- unzip exercises in a folder, you should get something like this:

```
graph-formats
    graph-formats.ipynb
    graph-formats-sol.ipynb
    jupman.py
    sciprog.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 `graph-formats/graph-formats.ipynb`

WARNING 2: DO NOT use the *Upload* button in Jupyter, instead navigate in Jupyter browser to the unzipped folder !

[271] https://github.com/DavidLeoni/sciprog-ds/tree/master/graph-formats
[272] https://sciprog.davidleoni.it/binary-relations/binary-relations-sol.html
[273] https://sciprog.davidleoni.it/graph-algos/graph-algos.html

- Go on reading that notebook, and follow instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

Required libraries

In order for visualizations to work, you need installed the python library `networkx` and `pydot`. Pydot is an interface to the non-pyhon package GraphViz[274].

Anaconda:

From Anaconda Prompt:

1. Install GraphViz:

```
conda install graphviz
```

2. Install python packages:

```
conda install pydot networkx
```

Ubuntu

From console:

1. Install PyGraphViz (note: you should use apt to install it, pip might give problems):

```
sudo apt install python3-pygraphviz
```

2. Install python packages:

```
python3 -m pip install --user pydot networkx
```

Graph definition

In computer science a *graph* is a set of verteces 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:

[274] http://graphviz.org/

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

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),(d,a)
      (d,c),(d,e),(e,c) }
```

```
V = { a,b,c,d,e,f }
E = { [a,b],[a,d],[b,c],
      [c,d],[d,e],[c,e] }
```

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

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`. Furthemore, 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` verteces, you can represent it as an `n x 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, `mat[i][j]` is the weight of the edge between node `i` to node `j`

Visualization examples

We defined a function `sciprog.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
import sys
```

(continues on next page)

(continued from previous page)

```
sys.path.append('../')
from sciprog 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 a file

If you want (or if you are not using Jupyter), optionally you can save the graph to a .png file by specificing the `save_to` filepath:

[3]:
```
mat = [
        [1,1],
        [0,1]
      ]
draw_mat( mat, save_to='example.png')
```

```
Image saved to file:  example.png
```

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:

[4]:
```
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):

[5]:
```
minimal = [
    [5]
]

draw_mat(minimal)
```

Graph with two nodes example

[6]:
```
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)

[7]:
```
# 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.

[8]:
```
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 some times it is better to use only `True` and `False`

[9]:
```
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.

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. For the sake of the first part of the course the double `for` is acceptable, but in the second part it would be considered a waist of computing cycles.

What would be the graph representation of `diag` ?

[10]:
```
def line(n):
    """ 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 >= 1, otherwise rises ValueError
    """
    #jupman-raise
    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
    #/jupman-raise

assert line(1) == [
                     [0]
                   ]
assert line(2) == [
                     [0,1],
```

(continues on next page)

(continued from previous page)

```
                    [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))
```

cross

⊛⊛ RETURN a nxn matrix filled with zeros except on the crossing lines.

- `n` must be >=1 and odd, otherwise a `ValueError` is thrown

Example for `n=7` :

```
0001000
0001000
0001000
```

(continues on next page)

(continued from previous page)

```
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)

[11]:
```
def cross(n):
    #jupman-raise
    if n < 1 or n % 2 == 0:
        raise ValueError("Invalid n %s" % n)
    ret = [[0]*n for i in range(n)]
    for i in range(n):
        ret[n//2 ][i] = 1
        ret[i][n//2] = 1
    return ret
    #/jupman-raise

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 verteces 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:

[12]:
```
ma =  [
            [True, False, False],
            [False, True, False],
            [True, False, False]
      ]
```

[13]:
```
draw_mat(ma)
```

And another `mb` like this:

[14]:
```
mb =  [
            [True, True, False],
            [False, False, True],
            [False, True, False]

      ]
```

[15]:
```
draw_mat(mb)
```

The result of calling `union(ma, mb)` will be the following:

[16]:
```
res = [[True, True, False], [False, True, True], [True, True, False]]
```

which will be displayed as

[17]:
```
draw_mat(res)
```

So we get same verteces and edges from both `ma` and `mb`

[18]:
```
def union(mata, matb):
    """ Takes two graphs represented as nxn 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
    """
    #jupman-raise

    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
    #/jupman-raise

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"
```

(continues on next page)

(continued from previous page)

```
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]
```

(continues on next page)

(continued from previous page)

```
        ]

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]

                           ]
```

is_subgraph

⊛⊛ 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.

[19]:
```
def is_subgraph(A, B):
    """ RETURN True is 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
    """
    #jupman-raise
    n = len(A)
    m = len(B)
    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 A[i][j] and not B[i][j]:
                return False
    return True
    #/jupman-raise

# 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
```

(continues on next page)

(continued from previous page)

```
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
```

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`

[20]:
```
def remove_node(mat, i):
    """ MODIFIES mat by removing node i.
    """
    #jupman-raise
    del mat[i]
    for row in mat:
        del row[i]
    #/jupman-raise

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
```

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 doesn't provide the best possible representation, try to draw on paper a more intuitive one.

[21]:
```
def utriang(n):
    """ 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
    """
    #jupman-raise
    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
    #/jupman-raise

assert utriang(1) == [
    [1]
]
assert utriang(2) == [
```

(continues on next page)

(continued from previous page)

```
    [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]
]
```

ediff

⊛⊛⊛ 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 verteces but possibly different edges. This time we will try operate on graphs represented as dictionaries of adjacency lists.

For example, if we have

[22]:
```
da =  {
          'a':['a','c'],
          'b':['b', 'c'],
          'c':['b','c']
        }
```

[23]:
```
draw_adj(da)
```

and

[24]:
```
db =  {
          'a':['c'],
          'b':['a','b', 'c'],
```

(continues on next page)

(continued from previous page)

```
            'c':['a']
        }
```

[25]:
```
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)
```

[28]:
```
def ediff(da,db):
    """  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
```

(continues on next page)

(continued from previous page)

```
        no outgoing edge), and that da and db have the same keys

        EXAMPLE:

          da =  {
                    'a':['a','c'],
                    'b':['b', 'c'],
                    'c':['b','c']
                  }

          db =  {
                    'a':['c'],
                    'b':['a','b', 'c'],
                    'c':['a']
                  }

          assert ediff(da, db) == {
                                      'a':['a'],
                                      'b':[],
                                      'c':['b','c']
                                    }

    """
    #jupman-raise

    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
    #/jupman-raise




da1 =  {
          'a': []
       }
db1 =  {
          'a': []
       }


assert ediff(da1, db1) ==   {
                             'a': []
                           }

da2 =  {
          'a': []
       }

db2 =  {
          'a': ['a']
```

(continues on next page)

(continued from previous page)

```
        }

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':[]
         }

assert ediff(da6, db6) == {
                           'a':['b'],
                           'b':[]
                          }

da7 =  {
```

(continues on next page)

(continued from previous page)

```
          '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']
                          }
```

pyramid

⊛⊛⊛ The following function requires to create a matrix filled with non-zero numbers. Even if don't know exactly the network meaning, with ust 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* - where have you already seen it? ;-)

[29]:
```
def pyramid(n):
    """
        Takes an odd number n >= 1 and RETURN a matrix as list of lists containing␣
↪numbers displaced like this
        example for a pyramid of square 7:
        if n is even, raises ValueError

        1111111
        1222221
        1233321
        1234321
        1233321
        1222221
        1111111
    """
    #jupman-raise
    if n % 2 == 0:
        raise ValueError("n should be odd, found instead %s" % n)
    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
    #/jupman-raise

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]
```

(continues on next page)

(continued from previous page)

```
                        ]
```

6.14.3 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
```

(continues on next page)

(continued from previous page)

```
        5: [2]      # node 5 links to node 2
    }
```

In `sciprog.py`, we provide also a function `sciprog.draw_adj` to quickly inspect such data structure:

[33]:
```
from sciprog import draw_adj

draw_adj(d)
```

As expected, the resulting graph is the same as for the equivalent matrix representation.

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()`

[34]:
```
def mat_to_adj(bool_mat):
    #jupman-raise
    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
    #/jupman-raise

m1 = [
        [False]
    ]

d1 =  {
          0:[]
     }
```

(continues on next page)

(continued from previous page)

```
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


m6 = [
        [True,False,False],
        [True, True,False],
```

(continues on next page)

(continued from previous page)

```
        [False,True,False]
    ]

d6 =  {
         0:[0],
         1:[0,1],
         2:[1]
     }


assert mat_to_adj(m6) == d6
```

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`

[35]:
```
def mat_ids_to_adj(bool_mat, ids):
    #jupman-raise

    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
    #/jupman-raise


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':[] }
```

(continues on next page)

(continued from previous page)

```
assert mat_ids_to_adj(m1, ['a']) == d1

m2 = [
        [True]
    ]

d2 =  { 'a':['a'] }
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']
```

(continues on next page)

(continued from previous page)

```
    }


assert mat_ids_to_adj(m6,['a','b','c']) == d6
```

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 the function:

[36]:
```
def adj_to_mat(d):
    """ Take 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)

    """
    #jupman-raise
    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
    #/jupman-raise

from collections import OrderedDict
od1 = OrderedDict([
                    ('a',[])
```

(continues on next page)

(continued from previous page)

```
                  ])
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],
```

(continues on next page)

(continued from previous page)

```
        [True, True, False],
    ]

assert adj_to_mat(od6) == m6
```

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)]
        }
```

[39]:
```
def table_to_adj(table):
    #jupman-raise
    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
    #/jupman-raise

m0 = [
        ['I','P','Q']
    ]
res0 = {}

assert res0 == table_to_adj(m0)

m1 =    [
```

(continues on next page)

(continued from previous page)

```
            ['Identifier','Price','Quantity'],
            ['a',1,1],
            ['b',5,8],
            ['c',2,6],
            ['d',8,5],
            ['e',7,3]
        ]
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)
```

6.14.4 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 alredy 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:

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 0x7fef507c1080>
```

To display it in a way uniform with the rest of the course, we developed a function called `sciprog.draw_nx`:

[42]:
```
from sciprog 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 node, 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 propery `label`. So let's set label equal to the weight:

[48]:
```
G['a']['b']['label'] = 123
```

[49]:
```
draw_nx(G)
```

Converting networkx graphs

If you try to just output the string representation of the graph, networkx will give the empty string:

[50]:
```
print(G)
```

[51]:
```
str(G)
```

[51]:
```
''
```

[52]:
```
repr(G)
```

[52]:
```
'<networkx.classes.digraph.DiGraph object at 0x7fef507c1080>'
```

To convert to the dict of adjacency lists we know, you can use this method:

[53]:
```
nx.to_dict_of_lists(G)
```

[53]:
```
{'a': ['b', 'c'], 'b': ['b', 'c', 'd'], 'c': [], 'd': []}
```

The above works, but it doesn't convert additional edge info. For a complete conversion, use `nx.to_dict_of_dicts`

[54]:
```
nx.to_dict_of_dicts(G)
```

[54]:
```
{'a': {'b': {'weight': 123, 'label': 123}, 'c': {}},
 'b': {'b': {}, 'c': {}, 'd': {}},
 'c': {},
 'd': {}}
```

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[275]. Do you need to explicitly call `add_node` before referring to some node with `add_edge` ?

[55]:
```
def mat_to_nx(mat, ids):
    """ Given a real-valued nxn matrix as list of lists and a list of immutable␣
↪identifiers for the nodes,
        RETURN the corresponding graph in networkx format (as nx.DiGraph).

        If matrix is not nxn or ids length does not match n, raise ValueError

        - DON'T transform into a dict, use add_ methods from networkx object!
        - WARNING: Remember to set the labels to the weights AS STRINGS!
    """

    #jupman-raise

    G = nx.DiGraph()
    n = len(mat)
    m = len(mat[0])
    if n != m:
```

(continues on next page)

[275] https://networkx.github.io/documentation/stable/tutorial.html

(continued from previous page)

```
        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
    #/jupman-raise



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],
```

(continues on next page)

(continued from previous page)

```
        [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'}
             }
     }


assert nx.to_dict_of_dicts(mat_to_nx(m6,['a','b','c'])) == d6
```

6.14.5 Simple statistics

We will now compute simple statistics abour graphs. More advanced stuff will be done in Part B notebook about graph algorithms[276].

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:

[56]:
```
from sciprog 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.

[276] https://sciprog.davidleoni.it/graph-algos/graph-algos.html

outdegree_adj

[57]:
```
def outdegree_adj(d, v):
    """ 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
    """
    #jupman-raise
    if v not in d:
        raise ValueError("Vertex %s is not in %s" % (v, d))

    return len(d[v])
    #/jupman-raise

try:
    outdegree_adj({'a':[]},'b')
    raise Exception("SHOULD HAVE FAILED !")
except ValueError:
    "passed test"

assert outdegree_adj({
        'a':[]
},'a') == 0

assert outdegree_adj({
        'a':['a']
},'a') == 1

assert outdegree_adj({
        'a':['a','b'],
        'b':[]
},'a') == 2

assert outdegree_adj({
        'a':['a','b'],
        'b':['a','b','c'],
        'c':[]
},'b') == 3
```

outdegree_mat

⊛⊛ RETURN the outdegree 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`

[58]:
```
def outdegree_mat(mat, i):
    #jupman-raise
    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):
```

(continues on next page)

(continued from previous page)

```
        if mat[i][j]:
            ret += 1
    return ret
    #/jupman-raise

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"


assert outdegree_mat(
        [
            [False]
        ]
,0) == 0

assert outdegree_mat(
        [
            [True]
        ],0) == 1

assert outdegree_mat(
        [
            [True, True],
            [False, False]
        ],0) == 2

assert outdegree_mat(
        [
            [True, True, False],
            [True, True, True],
            [False, False, False],
        ]
,1) == 3
```

outdegree_avg

⊛⊛ RETURN the average outdegree of nodes in graph `d`, represented as dictionary of adjacency lists.

- Assume all nodes are in the keys.

[59]:
```
def outdegree_avg(d):
    #jupman-raise
    s = 0
    for k in d:
        s += len(d[k])
```

(continues on next page)

(continued from previous page)

```
    return s / len(d)
    #/jupman-raise

assert outdegree_avg({
        'a':[]
}) == 0

assert round(
                outdegree_avg({
                    'a':['a']
                })
            ,2) == 1.00 / 1.00

assert round(
                outdegree_avg({
                    'a':['a','b'],
                    'b':[]
                })
            ,2) == (2 + 0) / 2

assert round(
                outdegree_avg({
                    'a':['a','b'],
                    'b':['a','b','c'],
                    'c':[]
                })
        ,2) == round( (2 + 3) / 3 , 2)
```

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`

[60]:
```
def indegree_adj(d, v):
    #jupman-raise
    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
    #/jupman-raise

try:
    indegree_adj({'a':[]},'b')
    raise Exception("SHOULD HAVE FAILED !")
except ValueError:
    "passed test"


assert indegree_adj({
        'a':[]
```

(continues on next page)

(continued from previous page)

```
},'a') == 0

assert indegree_adj({
        'a':['a']
},'a') == 1

assert indegree_adj({
        'a':['a','b'],
        'b':[]
},'a') == 1

assert indegree_adj({
        'a':['a','b'],
        'b':['a','b','c'],
        'c':[]
},'b') == 2
```

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`

[61]:

```
def indegree_mat(mat, i):
    #jupman-raise
    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
    #/jupman-raise



try:
    indegree_mat([[False]],7)
    raise Exception("SHOULD HAVE FAILED !")
except ValueError:
    "passed test"

assert indegree_mat(
        [
            [False]
        ]
,0) == 0

assert indegree_mat(
        [
            [True]
```

(continues on next page)

(continued from previous page)

```
        ],0) == 1

assert indegree_mat(
        [
            [True, True],
            [False, False]
        ],0) == 1

assert indegree_mat(
        [
            [True, True, False],
            [True, True, True],
            [False, False, False],
        ]
,1) == 2
```

indegree_avg

⊛⊛ RETURN the average indegree of nodes in graph `d`, represented as dictionary of adjacency lists.

- Assume all nodes are in the keys

[62]:
```
def indegree_avg(d):
    #jupman-raise
    s = 0
    for k in d:
        s += len(d[k])
    return s / len(d)
    #/jupman-raise

assert indegree_avg({
        'a':[]
}) == 0

assert round(
                indegree_avg({
                    'a':['a']
                })
            ,2) == 1.00 / 1.00

assert round(
                indegree_avg({
                    'a':['a','b'],
                    'b':[]
                })
            ,2) == (1 + 1) / 2

assert round(
                indegree_avg({
                    'a':['a','b'],
                    'b':['a','b','c'],
                    'c':[]
                })
        ,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` ?

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[277] for more info):

$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |A|$

min_outdeg

Difficulty: ⊛⊛⊛

Before proceeding please make sure you read recursions on lists[278] **chapter**

[63]:
```
def helper(mat, start, end):
    """
        Takes a graph as matrix of list of lists  and RETURN the minimum
        outdegree of nodes with row index between indeces start (included)
        and end included

        This function MUST be recursive, so it must call itself.

        - HINT: REMEMBER to put return instructions in all 'if' branches!
    """
    #jupman-raise
    n = len(mat)
    if start == end:
        return mat[start].count(True)
    else:
        half = (start + end) // 2
        min_left = helper(mat, 0, half)
        min_right = helper(mat, half+1, end)
        return min(min_left, min_right)
    #/jupman-raise

def min_outdeg(mat):
    """
        Takes a graph as matrix of list of lists  and RETURN the minimum
        outdegree of nodes by calling function helper.
        min_outdeg function is *not* recursive, only function helper is.
    """
    #jupman-raise
    n = len(mat)
    return helper(mat, 0, len(mat) - 1)
    #/jupman-raise

assert min_outdeg(
        [
            [False]
        ]) == 0
```

(continues on next page)

[277] https://en.wikipedia.org/wiki/Directed_graph#Indegree_and_outdegree
[278] https://sciprog.davidleoni.it/lists/lists-sol.html

(continued from previous page)

```
assert min_outdeg(
        [
            [True]
        ]) == 1

assert min_outdeg(
        [
            [False, True],
            [True, False]
        ]) == 1

assert min_outdeg(
        [
            [True, True, False],
            [True, True, True],
            [False, True, True],
        ]) == 2


assert min_outdeg(
        [
            [True, True, False],
            [True, True, True],
            [False, True, False],
        ]) == 1


assert min_outdeg(
        [
            [True, True, True],
            [True, True, True],
            [False, True, False],
        ]) == 1
```

networkx Indegrees and outdegrees

With Networkx we can easily calculate indegrees and outdegrees of a node:

[64]:

```
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
})

draw_nx(G)
```

[65]:
```
G.out_degree('a')
```

[65]:
```
2
```

QUESTION: What is the outdegree of `'b'` ? Try to think about it and then confirm your thoughts with networkx:

[66]:
```
# write here
#print("indegree  b:  %s" % G.in_degree('b'))
#print("outdegree b:  %s" % G.out_degree('b'))
```

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.

[67]:
```
# write here
#print("degree  b:  %s" % G.degree('b'))
```

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

[68]:
```
# write here
#G.degree('b')
```

[69]:
```
draw_nx(mat_to_nx([
        [7,0,0],
        [7,9,0],
        [0,7,0]
    ], ['a','b','c']))
```

6.15 Visualization solutions

6.15.1 Download exercises zip

Browse files online[279]

6.15.2 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.

References

- Andrea Passerini slides A08[280]

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-sciprog.py
-exercises
     |- visualization
         |- visualization.ipynb
         |- visualization-sol.ipynb
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

[279] https://github.com/DavidLeoni/sciprog-ds/tree/master/visualization
[280] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A08-numpy.pdf

- 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/visualization.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 instuctions 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.15.3 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 `ro`, 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]

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
```

[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()
```

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')

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[281] 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()
```

6.15.4 Introducting numpy

For functions involving reals, vanilla python starts showing its limits and its 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.

[281] https://matplotlib.org/gallery/lines_bars_and_markers/categorical_variables.html

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

[9]:
```
%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 with one of these methods.

Sine Python range does not allow float increments, we can use `np.arange`:

[10]:
```
import numpy as np

xs = np.arange(0,1.0,0.1)
xs
```

[10]:
```
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`:

[11]:
```
xs = np.linspace(0,0.9,10)

xs
```

[11]:
```
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:

[12]:
```
ys = 2*xs + 1

ys
```

[12]:
```
array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8])
```

Let's put everything together:

[13]:
```
%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()
```

y = sin(x) + 3 exercise

⊛⊛⊛ 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 !!

a) try to solve it without using numpy. For pi, use constant `math.pi` (first you need to import `math` module)

b) 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`

c) For each tick, use the label sequence `"0π/4", "1π/4" , "2π/4", "3π/4" , "4π/4", "5π/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 ?

d) If you are in the mood, try to display them better like 0, π/4 , π/2 π, 3π/4 , π, 5π/4 possibly using Latex (requires some search, this example[282] might be a starting point)

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:

[14]:
```
'\frac{2,3}'
```

[14]:
```
'\x0crac{2,3}'
```

One solution would be to double the slashes, like this:

[15]:
```
'\\frac{2,3}'
```

[15]:
```
'\\frac{2,3}'
```

An even better one is to prepend the string with the `r` character, which allows to write slashes only once:

[16]:
```
r'\frac{2,3}'
```

[16]:
```
'\\frac{2,3}'
```

[17]:
```
# write here solution for a) y = sin(x) + 3 with vanilla python
```

[18]:
```
# 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')
```

(continues on next page)

[282] https://stackoverflow.com/a/40642200

(continued from previous page)

```
plt.show()
```

[19]:
```
# write here solution b.1)      y = sin(x) + 3 with numpy, arange
```

[20]:
```
# 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
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()
```

[21]:
```
# write here solution b.2)      y = sin(x) + 3 with numpy, linspace
```

[22]:
```
# 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()
```

[23]:
```
# write here solution c)        y = sin(x) + 3 with numpy and pi xlabels
```

[24]:
```
# SOLUTION c)    y = sin(x) + 3 with numpy and pi xlabels

%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 xlabels")
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()
```

Showing degrees per node

Going back to the indegrees and outdegrees as seen in Network statistics[283] chapter, we will try to study the distributions visually.

Let's take an example networkx DiGraph:

[59]:
```
import networkx as nx

G1=nx.DiGraph({
    'a':['b','c'],
    'b':['b','c', 'd'],
    'c':['a','b','d'],
    'd':['b', 'd']
})

draw_nx(G1)
```

[283] https://sciprog.davidleoni.it/network-statistics/network-statistics-sol.html#Simple-statistics

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[284] IF you have matplotlib >= 2.1.0

Here we use `xticks` as sometimes you might need to fiddle with them anyway

To get the nodes, you can use the `G1.nodes()` function:

[26]:
```
G1.nodes()
```

[26]:
```
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:

[27]:
```
for n in G1.nodes():
    print(n)
```

```
a
b
c
d
```

Also, you can get the indegree of a node with

[28]:
```
G1.in_degree('b')
```

[28]:
```
4
```

[29]:
```
# write here the solution
```

[284] https://matplotlib.org/gallery/lines_bars_and_markers/categorical_variables.html

[30]:
```
# SOLUTION

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()
```

6.15.5 Bar plots

The previous plot with dots doesn't look so good - we might try to use instead a bar plot. First look at this this example, then proceed with the next exercise

[31]:
```
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
        color='green',    # someone suggested the default blue color is depressing, so␣
↪let's put green
```

(continues on next page)

(continued from previous page)

```
        align='center')  # bars are centered on the xtick

plt.show()
```

indegree per node bar plot

⊛⊛ Display a bar plot[285] for graph `G1` where the xtick labels are the nodes, and the y is the indegree of those nodes.

[32]:
```
# write here
```

[33]:
```
# SOLUTION

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()
```

[285] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.bar.html

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.

[34]:
```
# SOLUTION

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()
```

[35]:
```
# 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. First read the Python documentation[286] and then:

1. create a list of couples (list of tuples) where each tuple is the node identifier and the corresponding indegree
2. sort the list by using the second value of the tuples as a key.

[36]:
```
# write here
```

[37]:
```
# SOLUTION

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')
```

(continues on next page)

[286] https://docs.python.org/3/howto/sorting.html#key-functions

(continued from previous page)

```
plt.ylabel('indegree')

plt.show()
```

out degrees per node sorted

⊛⊛⊛ Do the same graph as before for the outdegrees.

You can get the outdegree of a node with:

[38]:
```
G1.out_degree('b')
```

[38]:
```
3
```

[39]:
```
# SOLUTION
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()
```

[40]:
```
# 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.

[41]:
```
# write here the solution
```

[42]:
```
# SOLUTION

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()
```

⊛⊛⊛⊛ **EXERCISE**: Look at this example[287], 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`.

[43]:
```
# write here
```

[44]:
```
# SOLUTION

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()
```

(continues on next page)

[287] https://matplotlib.org/gallery/lines_bars_and_markers/barchart.html#sphx-glr-gallery-lines-bars-and-markers-barchart-py

(continued from previous page)

```
plt.show()
```

6.15.6 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 matplot histogram, see documentation[288]

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`:

[61]:
```
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)
```

[288] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html

If we take the the degree sequence of `G2` we get this:

[46]:
```
degrees_G2 = [G2.degree(n) for n in G2.nodes()]

degrees_G2
```

[46]:
```
[3, 7, 3, 6, 7, 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:

[47]:
```
min(degrees_G2)
```

[47]:
```
3
```

[48]:
```
max(degrees_G2)
```

[48]:
```
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[289] for more details about it.

[289] https://stackoverflow.com/a/27084005

[49]:
```
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␣
↪*included*
                            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**: 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[290] 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

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

[290] https://stackoverflow.com/a/27084005

[51]:
```
# SOLUTION

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
           xs )                 # labels of ticks
plt.xlim(min_x, max_x + 1)  #  3 *included* to 8 *excluded*
plt.show()
```

6.15.7 Showing plots side by side

You can display plots on a grid. Each cell in the grid is idientified 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
```

[52]:
```
%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 first 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,    # 2 rows
            3,    # 3 columns
            6)    # plotting in sixth cell

ys6 = [-math.cos(x) for x in xs]
plt.plot(xs,ys6)
plt.title('6th cell')

plt.show()
```

Graph models

Let's study frequencies of some known network types.

Erdős–Rényi model

⊛⊛ A simple graph model we can think of is the so-called Erdős–Rényi model[291]: is 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:

[53]:
```
G = nx.erdos_renyi_graph(10, 0.5)
```

In the drawing, by looking the absence of arrows confirms it is undirected:

[62]:
```
draw_nx(G)
```

[291] 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 avoid rewriting the same code again and again, define a `plot_erdos(n,p,j)` function to be called three times.

[55]:
```
# write here the solution
```

[56]:
```
# SOLUTION


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,    # 1 row
                3,    # 3 columns
                j)    # plotting in 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
```

(continues on next page)

(continued from previous page)

```
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()
print("                                        Erdős-Rényi degree distribution␣
↪SOLUTION")
plt.show()
```

```
                                        Erdős-Rényi degree distribution SOLUTION
```

6.15.8 Other plots

Matplotlib allows to display pretty much any you might like, here we collect some we use in the course, for others, see the extensive Matplotlib documentation[292]

Pie chart

[57]:
```
%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='%1.1f%%', shadow=True)
plt.title("Super strict vegan diet (good luck)")
plt.show()
```

[292] https://matplotlib.org/gallery/index.html

6.16 Pandas solutions

6.16.1 Download exercises zip

Browse files online[293]

References:

- Andrea Passerini, Lecture A07[294]
- []

6.16.2 1. Introduction

Today we will try analyzing data with Pandas

- data analysis with Pandas library
- plotting with MatPlotLib
- Examples from AstroPi dataset
- Exercises with meteotrentino dataset

Python gives powerful tools for data analysis:

[293] https://github.com/DavidLeoni/sciprog-ds/tree/master/pandas
[294] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A07-pandas.pdf

One of these is Pandas[295], which gives fast and flexible data structures, especially for interactive data analusis.

What to do

1. 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:

```
-jupman.py
-sciprog.py
-exercises
     |- pandas
         |- pandas.ipynb
         |- pandas-sol.ipynb
```

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 `network-statistics/pandas.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 instuctions inside.

Shortcut keys:

[295] https://pandas.pydata.org/

- 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 aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

6.16.3 2. Data analysis of Astro Pi

Let's try analyzing data recorded on a Raspberry present on the International Space Station, downloaded from here:

raspberrypi.org/learning/astro-pi-flight-data-analysis/worksheet[296]

in which it is possible to find the detailed description of data gathered by sensors, in the month of February 2016 (one record each 10 seconds).

The method `read_csv` imports data from a CSV file and saves them in DataFrame structure.

In this exercise we shall use the file Columbus_Ed_astro_pi_datalog.csv

[296] https://www.raspberrypi.org/learning/astro-pi-flight-data-analysis/worksheet/

[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('Columbus_Ed_astro_pi_datalog.csv', encoding='UTF-8')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 20 columns):
ROW_ID        110869 non-null int64
temp_cpu      110869 non-null float64
temp_h        110869 non-null float64
temp_p        110869 non-null float64
humidity      110869 non-null float64
pressure      110869 non-null float64
pitch         110869 non-null float64
roll          110869 non-null float64
yaw           110869 non-null float64
mag_x         110869 non-null float64
mag_y         110869 non-null float64
mag_z         110869 non-null float64
accel_x       110869 non-null float64
accel_y       110869 non-null float64
accel_z       110869 non-null float64
gyro_x        110869 non-null float64
gyro_y        110869 non-null float64
gyro_z        110869 non-null float64
reset         110869 non-null int64
time_stamp    110869 non-null object
dtypes: float64(17), int64(2), object(1)
memory usage: 16.9+ MB
```

We can quickly see rows and columns of the dataframe with the attribute `shape`:

NOTE: `shape` is not followed by rounded parenthesis !

[3]:
```
df.shape
```

[3]:
```
(110869, 20)
```

The `describe` method gives you on the fly many summary info:

- rows counting
- the average
- standard deviation[297]
- quantiles[298]
- minimum and maximum

[4]:
```
df.describe()
```

[4]:
```
              ROW_ID       temp_cpu         temp_h         temp_p  \
count  110869.000000  110869.000000  110869.000000  110869.000000
mean    55435.000000      32.236259      28.101773      25.543272
```

(continues on next page)

[297] https://it.wikipedia.org/wiki/Scarto_quadratico_medio
[298] https://en.wikipedia.org/wiki/Quantile

(continued from previous page)

```
std     32005.267835       0.360289       0.369256       0.380877
min         1.000000      31.410000      27.200000      24.530000
25%     27718.000000      31.960000      27.840000      25.260000
50%     55435.000000      32.280000      28.110000      25.570000
75%     83152.000000      32.480000      28.360000      25.790000
max    110869.000000      33.700000      29.280000      26.810000

            humidity       pressure          pitch           roll  \
count  110869.000000  110869.000000  110869.000000  110869.000000
mean       46.252005    1008.126788       2.770553      51.807973
std         1.907273       3.093485      21.848940       2.085821
min        42.270000    1001.560000       0.000000      30.890000
25%        45.230000    1006.090000       1.140000      51.180000
50%        46.130000    1007.650000       1.450000      51.950000
75%        46.880000    1010.270000       1.740000      52.450000
max        60.590000    1021.780000     360.000000     359.400000

                yaw          mag_x          mag_y          mag_z  \
count  110869.00000  110869.000000  110869.000000  110869.000000
mean      200.90126     -19.465265      -1.174493      -6.004529
std        84.47763      28.120202      15.655121       8.552481
min         0.01000     -73.046240     -43.810030     -41.163040
25%       162.43000     -41.742792     -12.982321     -11.238430
50%       190.58000     -21.339485      -1.350467      -5.764400
75%       256.34000       7.299000      11.912456      -0.653705
max       359.98000      33.134748      37.552135      31.003047

             accel_x        accel_y        accel_z        gyro_x  \
count  110869.000000  110869.000000  110869.000000  1.108690e+05
mean       -0.000630       0.018504       0.014512 -8.959493e-07
std         0.000224       0.000604       0.000312  2.807614e-03
min        -0.025034      -0.005903      -0.022900 -3.037930e-01
25%        -0.000697       0.018009       0.014349 -2.750000e-04
50%        -0.000631       0.018620       0.014510 -3.000000e-06
75%        -0.000567       0.018940       0.014673  2.710000e-04
max         0.018708       0.041012       0.029938  2.151470e-01

              gyro_y        gyro_z          reset
count  110869.000000  1.108690e+05  110869.000000
mean        0.000007 -9.671594e-07       0.000180
std         0.002456  2.133104e-03       0.060065
min        -0.378412 -2.970800e-01       0.000000
25%        -0.000278 -1.200000e-04       0.000000
50%        -0.000004 -1.000000e-06       0.000000
75%         0.000271  1.190000e-04       0.000000
max         0.389499  2.698760e-01      20.000000
```

QUESTION: is there some missing field from the table produced by describe? Why is it not included?

To limit `describe` to only one column like `humidity`, you can write like this:

[5]:
```
df['humidity'].describe()
```

[5]:
```
count    110869.000000
mean         46.252005
std           1.907273
min          42.270000
```

(continues on next page)

(continued from previous page)

```
25%          45.230000
50%          46.130000
75%          46.880000
max          60.590000
Name: humidity, dtype: float64
```

Notation with the dot is even more handy:

[6]:
```
df.humidity.describe()
```

[6]:
```
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
```

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'].describe()`)

`head` method gives back the first datasets:

[7]:
```
df.head()
```

[7]:
```
   ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
0       1     31.88   27.57   25.01     44.94   1001.68   1.49  52.25  185.21
1       2     31.79   27.53   25.01     45.12   1001.72   1.03  53.73  186.72
2       3     31.66   27.53   25.01     45.12   1001.72   1.24  53.57  186.21
3       4     31.69   27.52   25.01     45.32   1001.69   1.57  53.63  186.03
4       5     31.66   27.54   25.01     45.18   1001.71   0.85  53.66  186.46

       mag_x     mag_y      mag_z   accel_x   accel_y   accel_z    gyro_x  \
0 -46.422753 -8.132907 -12.129346 -0.000468  0.019439  0.014569  0.000942
1 -48.778951 -8.304243 -12.943096 -0.000614  0.019436  0.014577  0.000218
2 -49.161878 -8.470832 -12.642772 -0.000569  0.019359  0.014357  0.000395
3 -49.341941 -8.457380 -12.615509 -0.000575  0.019383  0.014409  0.000308
4 -50.056683 -8.122609 -12.678341 -0.000548  0.019378  0.014380  0.000321

     gyro_y    gyro_z  reset           time_stamp
0  0.000492 -0.000750     20  2016-02-16 10:44:40
1 -0.000005 -0.000235      0  2016-02-16 10:44:50
2  0.000600 -0.000003      0  2016-02-16 10:45:00
3  0.000577 -0.000102      0  2016-02-16 10:45:10
4  0.000691  0.000272      0  2016-02-16 10:45:20
```

`tail` method gives back last dataset:

[8]:
```
df.tail()
```

[8]:
```
        ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
110864  110865     31.56   27.52   24.83     42.94   1005.83   1.58  49.93
```

(continues on next page)

(continued from previous page)

```
110865  110866     31.55   27.50   24.83     42.72   1005.85   1.89  49.92
110866  110867     31.58   27.50   24.83     42.83   1005.85   2.09  50.00
110867  110868     31.62   27.50   24.83     42.81   1005.88   2.88  49.69
110868  110869     31.57   27.51   24.83     42.94   1005.86   2.17  49.77

           yaw      mag_x      mag_y     mag_z   accel_x   accel_y   accel_z  \
110864  129.60 -15.169673 -27.642610  1.563183 -0.000682  0.017743  0.014646
110865  130.51 -15.832622 -27.729389  1.785682 -0.000736  0.017570  0.014855
110866  132.04 -16.646212 -27.719479  1.629533 -0.000647  0.017657  0.014799
110867  133.00 -17.270447 -27.793136  1.703806 -0.000835  0.017635  0.014877
110868  134.18 -17.885872 -27.824149  1.293345 -0.000787  0.017261  0.014380

          gyro_x    gyro_y    gyro_z  reset           time_stamp
110864 -0.000264  0.000206  0.000196      0  2016-02-29 09:24:21
110865  0.000143  0.000199 -0.000024      0  2016-02-29 09:24:30
110866  0.000537  0.000257  0.000057      0  2016-02-29 09:24:41
110867  0.000534  0.000456  0.000195      0  2016-02-29 09:24:50
110868  0.000459  0.000076  0.000030      0  2016-02-29 09:25:00
```

`colums` property gives the column headers:

[9]:
```
df.columns
```

[9]:
```
Index(['ROW_ID', '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', 'time_stamp'],
      dtype='object')
```

Nota: as you see in the above, the type of the found object is not a list, but a special container defined by pandas:

[10]:
```
type(df.columns)
```

[10]:
```
pandas.core.indexes.base.Index
```

Nevertheless, we can access the elements of this container using indeces within the squared parenthesis:

[11]:
```
df.columns[0]
```

[11]:
```
'ROW_ID'
```

[12]:
```
df.columns[1]
```

[12]:
```
'temp_cpu'
```

2.1 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: https://www.meteotrentino.it). **IMPORTANT**: assign the dataframe to a variable called `meteo` (so we avoid confusion whith AstroPi dataframe)

b) Visualize the information about this dataframe.

[13]:
```
# write here - create dataframe

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
print("COLUMNS:")
```

(continues on next page)

(continued from previous page)

```
print()
print(meteo.columns)
print()
print("INFO:")
print(meteo.info())
print()
print("HEAD():")

meteo.head()
```

```
COLUMNS:

Index(['Date', 'Pressure', 'Rain', 'Temp'], dtype='object')

INFO:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2878 entries, 0 to 2877
Data columns (total 4 columns):
Date        2878 non-null object
Pressure    2878 non-null float64
Rain        2878 non-null float64
Temp        2878 non-null float64
dtypes: float64(3), object(1)
memory usage: 90.0+ KB
None

HEAD():
```

[13]:

```
               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
```

6.16.4 3. Indexing, filtering, ordering

To obtain the i-th series you can use the method `iloc[i]` (here we reuse AstroPi dataset) :

[14]:

```
df.iloc[6]
```

[14]:

```
ROW_ID                          7
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                    -50.4473
mag_y                    -7.93731
mag_z                    -12.1886
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
time_stamp    2016-02-16 10:45:41
Name: 6, dtype: object
```

It is possible to select a dataframe by near positions using *slicing*:

Here for example we select the rows from 5th *included* to 7-th *excluded* :

[15]:
```
df.iloc[5:7]
```

[15]:
```
   ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
5       6     31.69   27.55   25.01     45.12   1001.67   0.85  53.53  185.52
6       7     31.68   27.53   25.01     45.31   1001.70   0.63  53.55  186.10

       mag_x     mag_y      mag_z   accel_x   accel_y   accel_z    gyro_x  \
5 -50.246476 -8.343209 -11.938124 -0.000536  0.019453  0.014380  0.000273
6 -50.447346 -7.937309 -12.188574 -0.000510  0.019264  0.014528 -0.000111

     gyro_y    gyro_z  reset           time_stamp
5  0.000494 -0.000059      0  2016-02-16 10:45:30
6  0.000320  0.000222      0  2016-02-16 10:45:41
```

It is possible to filter data according to a condition:

We che discover the data type, for example for `df.ROW_ID >= 6`:

[16]:
```
type(df.ROW_ID >= 6)
```

[16]:
```
pandas.core.series.Series
```

What is contained in this `Series` object ? If we try printing it we will see it is a series of values True or False, according whether the `ROW_ID` is greater or equal than 6:

[17]:
```
df.ROW_ID >= 6
```

[17]:
```
0         False
1         False
2         False
3         False
4         False
5          True
6          True
7          True
8          True
9          True
10         True
11         True
12         True
13         True
14         True
15         True
16         True
17         True
18         True
```

(continues on next page)

(continued from previous page)

```
19         True
20         True
21         True
22         True
23         True
24         True
25         True
26         True
27         True
28         True
29         True
          …
110839     True
110840     True
110841     True
110842     True
110843     True
110844     True
110845     True
110846     True
110847     True
110848     True
110849     True
110850     True
110851     True
110852     True
110853     True
110854     True
110855     True
110856     True
110857     True
110858     True
110859     True
110860     True
110861     True
110862     True
110863     True
110864     True
110865     True
110866     True
110867     True
110868     True
Name: ROW_ID, Length: 110869, dtype: bool
```

In an analogue way `(df.ROW_ID >= 6) & (df.ROW_ID <= 10)` is a series of values `True` or `False`, if `ROW_ID` is at the same time greater or equal than 6 and less or equal of 10

[18]:
```
type((df.ROW_ID >= 6) & (df.ROW_ID <= 10))
```

[18]:
```
pandas.core.series.Series
```

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

[19]:
```
df[  (df.ROW_ID >= 6) & (df.ROW_ID <= 10)  ]
```

[19]:
```
   ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
5       6     31.69   27.55   25.01     45.12   1001.67   0.85  53.53  185.52
6       7     31.68   27.53   25.01     45.31   1001.70   0.63  53.55  186.10
7       8     31.66   27.55   25.01     45.34   1001.70   1.49  53.65  186.08
8       9     31.67   27.54   25.01     45.20   1001.72   1.22  53.77  186.55
9      10     31.67   27.54   25.01     45.41   1001.75   1.63  53.46  185.94

       mag_x     mag_y      mag_z   accel_x   accel_y   accel_z    gyro_x  \
5 -50.246476 -8.343209 -11.938124 -0.000536  0.019453  0.014380  0.000273
6 -50.447346 -7.937309 -12.188574 -0.000510  0.019264  0.014528 -0.000111
7 -50.668232 -7.762600 -12.284196 -0.000523  0.019473  0.014298 -0.000044
8 -50.761529 -7.262934 -11.981090 -0.000522  0.019385  0.014286  0.000358
9 -51.243832 -6.875270 -11.672494 -0.000581  0.019390  0.014441  0.000266

     gyro_y    gyro_z  reset           time_stamp
5  0.000494 -0.000059      0  2016-02-16 10:45:30
6  0.000320  0.000222      0  2016-02-16 10:45:41
7  0.000436  0.000301      0  2016-02-16 10:45:50
8  0.000651  0.000187      0  2016-02-16 10:46:01
9  0.000676  0.000356      0  2016-02-16 10:46:10
```

So if we want to search the record where pressure is maximal, we user `values` property of the series on which we calculate the maximal value:

[20]:
```
df[  (df.pressure == df.pressure.values.max())  ]
```

[20]:
```
       ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
77602   77603     32.44   28.31   25.74     47.57   1021.78    1.1  51.82

          yaw     mag_x      mag_y      mag_z   accel_x  accel_y   accel_z  \
77602  267.39 -0.797428  10.891803 -15.728202 -0.000612  0.01817  0.014295

         gyro_x    gyro_y    gyro_z  reset           time_stamp
77602 -0.000139 -0.000179 -0.000298      0  2016-02-25 12:13:20
```

The method `sort_values` return a dataframe ordered according to one or more columns:

[21]:
```
df.sort_values('pressure',ascending=False).head()
```

[21]:
```
       ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
77602   77603     32.44   28.31   25.74     47.57   1021.78   1.10  51.82
77601   77602     32.45   28.30   25.74     47.26   1021.75   1.53  51.76
77603   77604     32.44   28.30   25.74     47.29   1021.75   1.86  51.83
77604   77605     32.43   28.30   25.74     47.39   1021.75   1.78  51.54
77608   77609     32.42   28.29   25.74     47.36   1021.73   0.86  51.89

          yaw     mag_x      mag_y      mag_z   accel_x   accel_y   accel_z  \
77602  267.39 -0.797428  10.891803 -15.728202 -0.000612  0.018170  0.014295
77601  266.12 -1.266335  10.927442 -15.690558 -0.000661  0.018357  0.014533
77603  268.83 -0.320795  10.651441 -15.565123 -0.000648  0.018290  0.014372
77604  269.41 -0.130574  10.628383 -15.488983 -0.000672  0.018154  0.014602
77608  272.77  0.952025  10.435951 -16.027235 -0.000607  0.018186  0.014232

         gyro_x    gyro_y    gyro_z  reset           time_stamp
77602 -0.000139 -0.000179 -0.000298      0  2016-02-25 12:13:20
77601  0.000152  0.000459 -0.000298      0  2016-02-25 12:13:10
77603  0.000049  0.000473 -0.000029      0  2016-02-25 12:13:30
```

(continues on next page)

(continued from previous page)

```
77604  0.000360  0.000089 -0.000002      0  2016-02-25 12:13:40
77608 -0.000260 -0.000059 -0.000187      0  2016-02-25 12:14:20
```

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 temperature is too much, we write `True` value in the fields of the column with header `'Too hot'`:

[22]:
```
df.loc[(df.temp_cpu > 31.68),'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`, which literally means *not a number* :

[23]:
```
df.head()
```

[23]:
```
   ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
0       1     31.88   27.57   25.01     44.94   1001.68   1.49  52.25  185.21
1       2     31.79   27.53   25.01     45.12   1001.72   1.03  53.73  186.72
2       3     31.66   27.53   25.01     45.12   1001.72   1.24  53.57  186.21
3       4     31.69   27.52   25.01     45.32   1001.69   1.57  53.63  186.03
4       5     31.66   27.54   25.01     45.18   1001.71   0.85  53.66  186.46

       mag_x  …      mag_z   accel_x   accel_y   accel_z    gyro_x  \
0 -46.422753  … -12.129346 -0.000468  0.019439  0.014569  0.000942
1 -48.778951  … -12.943096 -0.000614  0.019436  0.014577  0.000218
2 -49.161878  … -12.642772 -0.000569  0.019359  0.014357  0.000395
3 -49.341941  … -12.615509 -0.000575  0.019383  0.014409  0.000308
4 -50.056683  … -12.678341 -0.000548  0.019378  0.014380  0.000321

     gyro_y    gyro_z  reset           time_stamp Too hot
0  0.000492 -0.000750     20  2016-02-16 10:44:40    True
1 -0.000005 -0.000235      0  2016-02-16 10:44:50    True
2  0.000600 -0.000003      0  2016-02-16 10:45:00     NaN
3  0.000577 -0.000102      0  2016-02-16 10:45:10    True
4  0.000691  0.000272      0  2016-02-16 10:45:20     NaN

[5 rows x 21 columns]
```

Pandas is a very flexible library, and gives several methods to obtain the same results. For example, we can try the same operation as above with the command `np.where` as down below. For example, we add a column telling if pressure is above or below the average:

[24]:
```
avg_pressure = df.pressure.values.mean()
df['check_p'] = np.where(df.pressure <= avg_pressure, 'sotto', 'sopra')
```

3.1 Exercise: Meteo stats

⊛ Analyze data from Dataframe `meteo` and find:

- values of average pression, minimal and maximal
- average temperature
- the dates of rainy days

[25]:
```
# write here
print("Average pressure : %s" % meteo.Pressure.values.mean())
print("Minimal pressure : %s" % meteo.Pressure.values.min())
```

(continues on next page)

(continued from previous page)

```
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
```

[25]:

```
                  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
439   05/11/2017 13:45     978.8   0.4   8.2
440   05/11/2017 14:00     978.7   0.8   8.2
441   05/11/2017 14:15     978.4   0.6   8.3
442   05/11/2017 14:30     978.2   0.6   8.2
443   05/11/2017 14:45     978.1   0.6   8.2
444   05/11/2017 15:00     978.1   0.4   8.1
445   05/11/2017 15:15     977.9   0.4   8.1
446   05/11/2017 15:30     977.9   0.4   8.1
448   05/11/2017 16:00     977.4   0.2   8.1
455   05/11/2017 17:45     977.1   0.2   8.1
456   05/11/2017 18:00     977.1   0.2   8.2
457   05/11/2017 18:15     977.1   0.2   8.2
458   05/11/2017 18:30     976.8   0.2   8.3
459   05/11/2017 18:45     976.7   0.4   8.3
460   05/11/2017 19:00     976.5   0.2   8.4
461   05/11/2017 19:15     976.5   0.2   8.5
462   05/11/2017 19:30     976.3   0.2   8.5
463   05/11/2017 19:45     976.1   0.4   8.6
464   05/11/2017 20:00     976.3   0.2   8.7
465   05/11/2017 20:15     976.1   0.4   8.7
466   05/11/2017 20:30     976.1   0.4   8.7
467   05/11/2017 20:45     976.2   0.2   8.7
468   05/11/2017 21:00     976.4   0.6   8.8
469   05/11/2017 21:15     976.4   0.6   8.7
470   05/11/2017 21:30     976.9   1.2   8.7
…                  …        …     …     …
1150  12/11/2017 23:45     970.1   0.6   5.3
1151  13/11/2017 00:00     969.9   0.4   5.6
1152  13/11/2017 00:15     970.1   0.6   5.5
1153  13/11/2017 00:30     970.4   0.6   5.1
1154  13/11/2017 00:45     970.4   0.6   5.2
1155  13/11/2017 01:00     970.4   0.2   4.7
1159  13/11/2017 02:00     969.5   0.2   5.4
2338  25/11/2017 09:15     985.9   0.2   5.0
2346  25/11/2017 11:15     984.6   0.2   5.0
2347  25/11/2017 11:30     984.2   0.4   5.0
2348  25/11/2017 11:45     984.1   0.2   4.8
2349  25/11/2017 12:00     983.7   0.2   4.9
2350  25/11/2017 12:15     983.6   0.2   4.9
2352  25/11/2017 12:45     983.2   0.2   4.9
2353  25/11/2017 13:00     983.0   0.2   5.0
2354  25/11/2017 13:15     982.6   0.2   5.0
2355  25/11/2017 13:30     982.5   0.2   4.9
```

(continues on next page)

(continued from previous page)

```
2356  25/11/2017 13:45     982.4   0.2   4.9
2358  25/11/2017 14:15     982.0   0.2   4.8
2359  25/11/2017 14:30     982.1   0.2   4.8
2362  25/11/2017 15:15     981.5   0.2   4.9
2363  25/11/2017 15:30     981.2   0.2   5.0
2364  25/11/2017 15:45     981.1   0.2   5.0
2366  25/11/2017 16:15     981.0   0.2   5.0
2736  29/11/2017 12:45     978.0   0.2   0.9
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]
```

6.16.5 4. MatPlotLib review

We've already seen MatplotLib in the part on visualization[299], and today we use Matplotlib[300] to display data.

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:

[26]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
```

[27]:
```
x = [1,2,3,4]
y = [2,4,6,8]
plt.plot(x, y) # we can directly pass x and y lists
plt.title('Some number')
plt.show()
```

[299] https://sciprog.davidleoni.it/visualization/visualization-sol.html
[300] http://matplotlib.org

We can also create the series with numpy. Let's try making a parabola:

[28]:
```
x = np.arange(0.,5.,0.1)
#  '**' is the power operator in  Python, NOT '^'
y = x**2
```

Let's use the `type` function to understand which data types are x and y:

[29]:
```
type(x)
```

[29]:
```
numpy.ndarray
```

[30]:
```
type(y)
```

[30]:
```
numpy.ndarray
```

Hence we have NumPy arrays.

[31]:
```
plt.title('The parabola')
plt.plot(x,y);
```

If we want the x axis units to be same as y axis, we can use function gca[301]

To set x and y limits, we can use `xlim` e `ylim`:

[32]:
```
plt.xlim([0, 5])
plt.ylim([0,10])
plt.title('La parabola')

plt.gca().set_aspect('equal')
plt.plot(x,y);
```

[301] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.gca.html?highlight=matplotlib%20pyplot%20gca#matplotlib.pyplot.gca

Matplotlib plots from pandas datastructures

We can get plots directly from pandas data structures, always using the *matlab style*. Here there is documentation of DataFrame.plot[302]. Let's make an example. In case of big quantity of data, it may be useful to have a qualitative idea of data by putting them in a plot:

[33]:
```
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);
```

We can put pressure values on horizontal axis, and see which humidity values on vertical axis have a certain pressure:

[34]:
```
plt.plot(df['pressure'], df['humidity'])
```

[34]:
```
[<matplotlib.lines.Line2D at 0x7f8e7e6d0978>]
```

[302] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html

Let's select in the new dataframe `df2` the rows between the 12500th (included) and the 15000th (excluded):

[35]:
```
df2=df.iloc[12500:15000]
```

[36]:
```
plt.plot(df2['pressure'], df2['humidity'])
```

[36]:
```
[<matplotlib.lines.Line2D at 0x7f8e7e52f240>]
```

[37]:
```
df2.humidity.plot(label="Humidity", legend=True)
df2.pressure.plot(secondary_y=True, label="Pressure", legend=True)
```

[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e7e4b0710>
```

With `corr` method we can see the correlation between DataFrame columns.

[38]:
```
df2.corr()
```

[38]:
```
            ROW_ID  temp_cpu    temp_h    temp_p  humidity  pressure  \
ROW_ID    1.000000  0.561540  0.636899  0.730764  0.945210  0.760732
temp_cpu  0.561540  1.000000  0.591610  0.670043  0.488038  0.484902
temp_h    0.636899  0.591610  1.000000  0.890775  0.539603  0.614536
temp_p    0.730764  0.670043  0.890775  1.000000  0.620307  0.650015
humidity  0.945210  0.488038  0.539603  0.620307  1.000000  0.750000
pressure  0.760732  0.484902  0.614536  0.650015  0.750000  1.000000
pitch     0.005633  0.025618  0.022718  0.019178  0.012247  0.037081
roll      0.266995  0.165540  0.196767  0.192621  0.231316  0.225112
yaw       0.172192  0.056950 -0.024700  0.007474  0.181905  0.070603
mag_x    -0.108713 -0.019815 -0.151336 -0.060122 -0.108781 -0.246485
mag_y     0.057601 -0.028729  0.031512 -0.039648  0.131218  0.194611
mag_z    -0.270656 -0.193077 -0.260633 -0.285640 -0.191957 -0.173808
accel_x   0.015936 -0.021093 -0.009408 -0.034348  0.040452  0.085183
accel_y   0.121838  0.108878  0.173037  0.187457  0.069717 -0.032049
accel_z   0.075160  0.065628  0.129074  0.144595  0.021627 -0.068296
gyro_x   -0.014346 -0.019478 -0.005255 -0.010679  0.005625 -0.014838
gyro_y   -0.026012 -0.007527 -0.017054 -0.016674 -0.001927 -0.008821
gyro_z    0.011714 -0.006737 -0.016113 -0.017010  0.014431  0.032056
reset           NaN       NaN       NaN       NaN       NaN       NaN

             pitch      roll       yaw     mag_x     mag_y     mag_z  \
ROW_ID    0.005633  0.266995  0.172192 -0.108713  0.057601 -0.270656
temp_cpu  0.025618  0.165540  0.056950 -0.019815 -0.028729 -0.193077
temp_h    0.022718  0.196767 -0.024700 -0.151336  0.031512 -0.260633
temp_p    0.019178  0.192621  0.007474 -0.060122 -0.039648 -0.285640
humidity  0.012247  0.231316  0.181905 -0.108781  0.131218 -0.191957
pressure  0.037081  0.225112  0.070603 -0.246485  0.194611 -0.173808
pitch     1.000000  0.068880  0.030448 -0.008220 -0.002278 -0.019085
roll      0.068880  1.000000 -0.053750 -0.281035 -0.479779 -0.665041
yaw       0.030448 -0.053750  1.000000  0.536693  0.300571  0.394324
mag_x    -0.008220 -0.281035  0.536693  1.000000  0.046591  0.475674
mag_y    -0.002278 -0.479779  0.300571  0.046591  1.000000  0.794756
mag_z    -0.019085 -0.665041  0.394324  0.475674  0.794756  1.000000
accel_x   0.024460  0.057330 -0.028267 -0.097520  0.046693  0.001699
accel_y  -0.053634 -0.049233  0.078585  0.168764 -0.035111 -0.020016
```

(continues on next page)

(continued from previous page)

```
accel_z  -0.029345 -0.153524  0.068321  0.115423 -0.022579 -0.006496
gyro_x    0.040685  0.139427 -0.021071 -0.017739 -0.084045 -0.092749
gyro_y    0.041674  0.134319 -0.009650 -0.006722 -0.061460 -0.060097
gyro_z   -0.024081 -0.078113  0.064290  0.008456  0.115327  0.101276
reset          NaN       NaN       NaN       NaN       NaN       NaN

           accel_x   accel_y   accel_z    gyro_x    gyro_y    gyro_z  reset
ROW_ID    0.015936  0.121838  0.075160 -0.014346 -0.026012  0.011714    NaN
temp_cpu -0.021093  0.108878  0.065628 -0.019478 -0.007527 -0.006737    NaN
temp_h   -0.009408  0.173037  0.129074 -0.005255 -0.017054 -0.016113    NaN
temp_p   -0.034348  0.187457  0.144595 -0.010679 -0.016674 -0.017010    NaN
humidity  0.040452  0.069717  0.021627  0.005625 -0.001927  0.014431    NaN
pressure  0.085183 -0.032049 -0.068296 -0.014838 -0.008821  0.032056    NaN
pitch     0.024460 -0.053634 -0.029345  0.040685  0.041674 -0.024081    NaN
roll      0.057330 -0.049233 -0.153524  0.139427  0.134319 -0.078113    NaN
yaw      -0.028267  0.078585  0.068321 -0.021071 -0.009650  0.064290    NaN
mag_x    -0.097520  0.168764  0.115423 -0.017739 -0.006722  0.008456    NaN
mag_y     0.046693 -0.035111 -0.022579 -0.084045 -0.061460  0.115327    NaN
mag_z     0.001699 -0.020016 -0.006496 -0.092749 -0.060097  0.101276    NaN
accel_x   1.000000 -0.197363 -0.174005 -0.016811 -0.013694 -0.017850    NaN
accel_y  -0.197363  1.000000  0.424272 -0.023942 -0.054733  0.014870    NaN
accel_z  -0.174005  0.424272  1.000000  0.006313 -0.011883 -0.015390    NaN
gyro_x   -0.016811 -0.023942  0.006313  1.000000  0.802471 -0.012705    NaN
gyro_y   -0.013694 -0.054733 -0.011883  0.802471  1.000000 -0.043332    NaN
gyro_z   -0.017850  0.014870 -0.015390 -0.012705 -0.043332  1.000000    NaN
reset          NaN       NaN       NaN       NaN       NaN       NaN    NaN
```

6.16.6 5. Calculating new columns

It is possible to obtain new columns by calculating them from other columns. 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:

[39]:
```
df['mag_tot'] = df['mag_x']**2 + df['mag_y']**2 + df['mag_z']**2
df.mag_tot.plot()
```

[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e7e3e9ba8>
```

Let's find when the magnetic field was maximal:

[40]:
```
df['time_stamp'][(df.mag_tot == df.mag_tot.values.max())]
```

[40]:
```
96156    2016-02-27 16:12:31
Name: time_stamp, dtype: object
```

By filling in the value found on the website isstracker.com/historical[303], we can find the positions where the magnetic field is at the highest.

5.1 Exercise: Meteo Fahrenheit temperature

In `meteo` dataframe, create a column `Temp (Fahrenheit)` with the temperature measured in Fahrenheit degrees.

Formula to calculate conversion from Celsius degrees (C):

$Fahrenheit = \frac{9}{5}C + 32$

[41]:
```
# write here
```

[42]:
```
# SOLUTION
print()
print("       **************  SOLUTION OUTPUT  **************")
meteo['Temp (Fahrenheit)'] = meteo['Temp']* 9/5 + 32
meteo.head()
```

```
       **************  SOLUTION OUTPUT  **************
```

[42]:

	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

5.2 Exercise: Pressure vs Temperature

Pressure should be directly proportional to temperature in a closed environment Gay-Lussac's law[304]:

$\frac{P}{T} = k$

Does this holds true for `meteo` dataset? Try to find out by direct calculation of the formula and compare with `corr()` method results.

[43]:
```
# SOLUTION

# as expected, in an open environment there is not much linear correlation
#meteo.corr()
#meteo['Pressure'] / meteo['Temp']
```

[ ]:
```
```

[303] http://www.isstracker.com/historical
[304] https://en.wikipedia.org/wiki/Gay-Lussac%27s_law

6.16.7 6. Object values

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. (for text in particular there are various ways to manipulate it, for more details (see pandas documentation[305])

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`):

[44]:
```
df[  df['time_stamp'].str.contains('2016-02-2')  ]
```

[44]:
```
        ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
30442    30443     32.30   28.12   25.59     45.05   1008.01   1.47  51.82
30443    30444     32.25   28.13   25.59     44.82   1008.02   0.81  51.53
30444    30445     33.07   28.13   25.59     45.08   1008.09   0.68  51.69
30445    30446     32.63   28.10   25.60     44.87   1008.07   1.42  52.13
30446    30447     32.55   28.11   25.60     44.94   1008.07   1.41  51.86
30447    30448     32.47   28.12   25.61     44.83   1008.08   1.84  51.75
30448    30449     32.41   28.11   25.61     45.00   1008.10   2.35  51.87
30449    30450     32.41   28.12   25.61     45.02   1008.10   1.41  51.92
30450    30451     32.38   28.12   25.61     45.00   1008.12   1.46  52.04
30451    30452     32.36   28.13   25.61     44.97   1008.12   1.18  51.78
30452    30453     32.38   28.12   25.61     45.10   1008.12   1.08  51.81
30453    30454     32.33   28.12   25.61     44.96   1008.14   1.45  51.79
30454    30455     32.32   28.14   25.61     44.86   1008.12   1.89  51.95
30455    30456     32.39   28.13   25.61     45.01   1008.12   1.49  51.60
30456    30457     32.34   28.09   25.61     45.02   1008.14   1.18  51.74
30457    30458     32.34   28.11   25.61     45.02   1008.16   1.92  51.72
30458    30459     32.33   28.10   25.61     44.85   1008.18   1.99  52.06
30459    30460     32.35   28.11   25.61     44.98   1008.15   1.38  51.78
30460    30461     32.34   28.11   25.61     44.93   1008.18   1.41  51.66
30461    30462     32.29   28.11   25.61     44.90   1008.18   1.33  51.99
30462    30463     32.32   28.12   25.61     45.04   1008.17   1.30  51.93
30463    30464     32.30   28.12   25.61     44.86   1008.16   0.98  51.78
30464    30465     32.31   28.10   25.60     44.96   1008.18   1.82  51.95
30465    30466     32.34   28.11   25.60     45.07   1008.19   1.14  51.69
30466    30467     32.37   28.12   25.61     44.92   1008.19   1.73  51.94
30467    30468     32.32   28.11   25.60     44.98   1008.18   1.45  51.67
30468    30469     32.32   28.12   25.60     44.98   1008.20   1.66  51.85
30469    30470     32.30   28.12   25.60     44.93   1008.20   1.45  51.89
30470    30471     32.28   28.11   25.60     44.88   1008.21   1.78  51.88
30471    30472     32.33   28.10   25.60     44.96   1008.21   1.76  51.88
…          …         …       …       …         …         …     …     …
110839  110840     31.60   27.49   24.82     42.74   1005.83   1.12  49.34
110840  110841     31.59   27.48   24.82     42.75   1005.82   2.04  49.53
110841  110842     31.59   27.51   24.82     42.76   1005.82   1.31  49.19
110842  110843     31.60   27.50   24.82     42.74   1005.85   1.19  48.91
110843  110844     31.57   27.49   24.82     42.80   1005.83   1.49  49.17
110844  110845     31.60   27.50   24.82     42.81   1005.84   1.47  49.46
110845  110846     31.61   27.50   24.82     42.81   1005.82   2.28  49.27
110846  110847     31.61   27.50   24.82     42.75   1005.84   2.18  49.64
110847  110848     31.58   27.50   24.82     43.00   1005.85   2.52  49.31
```

(continues on next page)

[305] https://pandas.pydata.org/pandas-docs/stable/text.html

(continued from previous page)

```
110848  110849     31.54   27.51   24.82     42.76   1005.84   2.35  49.55
110849  110850     31.60   27.50   24.82     42.79   1005.82   2.33  48.79
110850  110851     31.61   27.50   24.82     42.79   1005.85   2.11  49.66
110851  110852     31.56   27.50   24.83     42.84   1005.83   1.68  49.91
110852  110853     31.59   27.51   24.83     42.76   1005.82   2.26  49.17
110853  110854     31.58   27.50   24.83     42.98   1005.83   1.96  49.41
110854  110855     31.61   27.51   24.83     42.69   1005.84   2.27  49.39
110855  110856     31.55   27.50   24.83     42.79   1005.83   1.51  48.98
110856  110857     31.55   27.49   24.83     42.81   1005.82   2.12  49.95
110857  110858     31.60   27.51   24.83     42.92   1005.82   1.53  49.33
110858  110859     31.58   27.50   24.83     42.81   1005.83   1.60  49.65
110859  110860     31.61   27.50   24.83     42.82   1005.84   2.65  49.47
110860  110861     31.57   27.50   24.83     42.80   1005.84   2.63  50.08
110861  110862     31.58   27.51   24.83     42.90   1005.85   1.70  49.81
110862  110863     31.60   27.51   24.83     42.80   1005.85   1.66  49.13
110863  110864     31.64   27.51   24.83     42.80   1005.85   1.91  49.31
110864  110865     31.56   27.52   24.83     42.94   1005.83   1.58  49.93
110865  110866     31.55   27.50   24.83     42.72   1005.85   1.89  49.92
110866  110867     31.58   27.50   24.83     42.83   1005.85   2.09  50.00
110867  110868     31.62   27.50   24.83     42.81   1005.88   2.88  49.69
110868  110869     31.57   27.51   24.83     42.94   1005.86   2.17  49.77

           yaw      mag_x  …   accel_y   accel_z    gyro_x    gyro_y  \
30442    51.18   9.215883  …  0.018792  0.014558 -0.000042  0.000275
30443    52.21   8.710130  …  0.019290  0.014667  0.000260  0.001011
30444    57.36   7.383435  …  0.018714  0.014598  0.000299  0.000343
30445    59.95   7.292313  …  0.018857  0.014565  0.000160  0.000349
30446    61.83   6.699141  …  0.018871  0.014564 -0.000608 -0.000381
30447    64.10   6.339477  …  0.018833  0.014691 -0.000233 -0.000403
30448    66.59   5.861904  …  0.018828  0.014534 -0.000225 -0.000292
30449    68.70   5.235877  …  0.018724  0.014255  0.000134 -0.000310
30450    70.98   4.775404  …  0.018730  0.014372  0.000319  0.000079
30451    73.10   4.300375  …  0.018814  0.014518 -0.000023  0.000186
30452    74.90   3.763551  …  0.018526  0.014454 -0.000184 -0.000075
30453    77.31   3.228626  …  0.018607  0.014330 -0.000269 -0.000547
30454    78.88   2.888813  …  0.018698  0.014548 -0.000081 -0.000079
30455    80.46   2.447253  …  0.018427  0.014576 -0.000349 -0.000269
30456    82.41   1.983143  …  0.018866  0.014438  0.000248  0.000172
30457    84.46   1.623884  …  0.018729  0.014770  0.000417  0.000231
30458    86.72   1.050999  …  0.018867  0.014592  0.000377  0.000270
30459    89.42   0.297179  …  0.018609  0.014593  0.000622  0.000364
30460    91.11  -0.136305  …  0.018504  0.014502 -0.000049 -0.000104
30461    93.09  -0.659496  …  0.018584  0.014593  0.000132 -0.000542
30462    94.25  -1.002867  …  0.018703  0.014584  0.000245  0.000074
30463    96.42  -1.634671  …  0.018833  0.014771  0.000343 -0.000154
30464    98.65  -2.204607  …  0.018867  0.014664 -0.000058 -0.000366
30465   101.53  -3.065968  …  0.018461  0.014735  0.000263 -0.000071
30466   103.40  -3.533967  …  0.018810  0.014541  0.000442  0.000022
30467   104.59  -4.009444  …  0.018657  0.014586 -0.000125  0.000013
30468   105.99  -4.438902  …  0.019021  0.014753 -0.000055  0.000126
30469   107.38  -4.940700  …  0.018959  0.014662  0.000046 -0.000504
30470   108.78  -5.444541  …  0.019012  0.014606 -0.000177 -0.000407
30471   110.70  -6.101692  …  0.018822  0.014834  0.000044  0.000042
…        …        …  …       …        …        …        …
110839   90.42   0.319629  …  0.017461  0.014988 -0.000209 -0.000005
110840   92.11   0.015879  …  0.017413  0.014565 -0.000472 -0.000478
110841   93.94  -0.658624  …  0.017516  0.015014 -0.000590 -0.000372
```

(continues on next page)

(continued from previous page)

```
110842   95.57  -1.117541  …  0.017400  0.014982 -0.000039  0.000059
110843   98.11  -1.860475  …  0.017580  0.014704  0.000223  0.000278
110844   99.67  -2.286044  …  0.017428  0.014325 -0.000283 -0.000187
110845  103.17  -3.182359  …  0.017537  0.014575 -0.000451 -0.000100
110846  105.05  -3.769940  …  0.017739  0.014926  0.000476  0.000452
110847  107.23  -4.431722  …  0.017588  0.015077  0.000822  0.000739
110848  108.68  -4.944477  …  0.017487  0.014864  0.000613  0.000763
110849  109.52  -5.481255  …  0.017455  0.014638  0.000196  0.000519
110850  111.90  -6.263577  …  0.017489  0.014960  0.000029 -0.000098
110851  113.38  -6.844946  …  0.017778  0.014703 -0.000177 -0.000452
110852  114.42  -7.437300  …  0.017733  0.014838  0.000396  0.000400
110853  116.50  -8.271114  …  0.017490  0.014582  0.000285  0.000312
110854  117.61  -8.690470  …  0.017465  0.014720 -0.000001  0.000371
110855  119.13  -9.585351  …  0.017554  0.014910 -0.000115  0.000029
110856  120.81 -10.120745  …  0.017494  0.014718 -0.000150  0.000147
110857  121.74 -10.657858  …  0.017544  0.014762  0.000161  0.000029
110858  123.50 -11.584851  …  0.017608  0.015093 -0.000073  0.000158
110859  124.51 -12.089743  …  0.017433  0.014930  0.000428  0.000137
110860  125.85 -12.701497  …  0.017805  0.014939  0.000263  0.000163
110861  126.86 -13.393369  …  0.017577  0.015026 -0.000077  0.000179
110862  127.35 -13.990712  …  0.017508  0.014478  0.000119 -0.000204
110863  128.62 -14.691672  …  0.017789  0.014891  0.000286  0.000103
110864  129.60 -15.169673  …  0.017743  0.014646 -0.000264  0.000206
110865  130.51 -15.832622  …  0.017570  0.014855  0.000143  0.000199
110866  132.04 -16.646212  …  0.017657  0.014799  0.000537  0.000257
110867  133.00 -17.270447  …  0.017635  0.014877  0.000534  0.000456
110868  134.18 -17.885872  …  0.017261  0.014380  0.000459  0.000076

          gyro_z  reset           time_stamp  Too hot  check_p      mag_tot
30442   0.000157      0  2016-02-20 00:00:00     True    sotto   269.091903
30443   0.000149      0  2016-02-20 00:00:10     True    sotto   260.866157
30444  -0.000025      0  2016-02-20 00:00:41     True    sotto   265.421154
30445  -0.000190      0  2016-02-20 00:00:50     True    sotto   269.572476
30446  -0.000243      0  2016-02-20 00:01:01     True    sotto   262.510966
30447  -0.000337      0  2016-02-20 00:01:10     True    sotto   273.997653
30448  -0.000004      0  2016-02-20 00:01:20     True    sotto   272.043915
30449  -0.000101      0  2016-02-20 00:01:30     True    sotto   268.608057
30450  -0.000215      0  2016-02-20 00:01:40     True    sotto   271.750032
30451  -0.000118      0  2016-02-20 00:01:51     True    sotto   277.538126
30452  -0.000077      0  2016-02-20 00:02:00     True    sotto   268.391448
30453  -0.000262      0  2016-02-20 00:02:11     True    sopra   271.942019
30454  -0.000240      0  2016-02-20 00:02:20     True    sotto   264.664070
30455  -0.000198      0  2016-02-20 00:02:31     True    sotto   267.262186
30456  -0.000474      0  2016-02-20 00:02:40     True    sopra   270.414588
30457  -0.000171      0  2016-02-20 00:02:50     True    sopra   278.210856
30458  -0.000074      0  2016-02-20 00:03:00     True    sopra   288.728974
30459  -0.000134      0  2016-02-20 00:03:10     True    sopra   303.816530
30460  -0.000286      0  2016-02-20 00:03:21     True    sopra   305.475482
30461  -0.000221      0  2016-02-20 00:03:30     True    sopra   306.437506
30462  -0.000308      0  2016-02-20 00:03:41     True    sopra   318.703894
30463  -0.000286      0  2016-02-20 00:03:50     True    sopra   324.412585
30464  -0.000091      0  2016-02-20 00:04:01     True    sopra   331.006515
30465  -0.000370      0  2016-02-20 00:04:10     True    sopra   332.503688
30466  -0.000193      0  2016-02-20 00:04:20     True    sopra   330.051496
30467   0.000209      0  2016-02-20 00:04:31     True    sopra   340.085476
30468   0.000070      0  2016-02-20 00:04:40     True    sopra   354.350961
30469   0.000041      0  2016-02-20 00:04:51     True    sopra   364.753950
```

(continues on next page)

(continued from previous page)

```
30470  -0.000427      0  2016-02-20 00:05:00     True    sopra   379.362654
30471  -0.000327      0  2016-02-20 00:05:11     True    sopra   388.749366
…           …    …                    …       …       …          …
110839  0.000138      0  2016-02-29 09:20:10      NaN    sotto   574.877314
110840  0.000126      0  2016-02-29 09:20:20      NaN    sotto   593.855683
110841  0.000207      0  2016-02-29 09:20:31      NaN    sotto   604.215692
110842  0.000149      0  2016-02-29 09:20:40      NaN    sotto   606.406098
110843  0.000038      0  2016-02-29 09:20:51      NaN    sotto   622.733559
110844  0.000077      0  2016-02-29 09:21:00      NaN    sotto   641.480748
110845 -0.000351      0  2016-02-29 09:21:10      NaN    sotto   633.949204
110846 -0.000249      0  2016-02-29 09:21:20      NaN    sotto   643.508698
110847 -0.000012      0  2016-02-29 09:21:30      NaN    sotto   658.512439
110848 -0.000227      0  2016-02-29 09:21:41      NaN    sotto   667.095455
110849 -0.000234      0  2016-02-29 09:21:50      NaN    sotto   689.714415
110850 -0.000073      0  2016-02-29 09:22:01      NaN    sotto   707.304506
110851 -0.000232      0  2016-02-29 09:22:10      NaN    sotto   726.361255
110852 -0.000188      0  2016-02-29 09:22:21      NaN    sotto   743.185242
110853 -0.000058      0  2016-02-29 09:22:30      NaN    sotto   767.328522
110854 -0.000274      0  2016-02-29 09:22:40      NaN    sotto   791.907055
110855 -0.000223      0  2016-02-29 09:22:50      NaN    sotto   802.932850
110856 -0.000320      0  2016-02-29 09:23:00      NaN    sotto   820.194642
110857 -0.000210      0  2016-02-29 09:23:11      NaN    sotto   815.462202
110858 -0.000006      0  2016-02-29 09:23:20      NaN    sotto   851.154631
110859  0.000201      0  2016-02-29 09:23:31      NaN    sotto   879.563826
110860  0.000031      0  2016-02-29 09:23:40      NaN    sotto   895.543882
110861  0.000148      0  2016-02-29 09:23:50      NaN    sotto   928.948693
110862  0.000041      0  2016-02-29 09:24:01      NaN    sotto   957.695014
110863  0.000221      0  2016-02-29 09:24:10      NaN    sotto   971.126355
110864  0.000196      0  2016-02-29 09:24:21      NaN    sotto   996.676408
110865 -0.000024      0  2016-02-29 09:24:30      NaN    sotto  1022.779594
110866  0.000057      0  2016-02-29 09:24:41      NaN    sotto  1048.121268
110867  0.000195      0  2016-02-29 09:24:50      NaN    sotto  1073.629703
110868  0.000030      0  2016-02-29 09:25:00      NaN    sotto  1095.760426

[80427 rows x 23 columns]
```

Extracting strings

To extract only the day from `timestamp` column, we can use `str` and use slice operator with square brackets:

[45]:
```
df['time_stamp'].str[8:10]
```

[45]:
```
0         16
1         16
2         16
3         16
4         16
5         16
6         16
7         16
8         16
9         16
10        16
11        16
12        16
```

(continues on next page)

(continued from previous page)

```
13        16
14        16
15        16
16        16
17        16
18        16
19        16
20        16
21        16
22        16
23        16
24        16
25        16
26        16
27        16
28        16
29        16
          ..
110839    29
110840    29
110841    29
110842    29
110843    29
110844    29
110845    29
110846    29
110847    29
110848    29
110849    29
110850    29
110851    29
110852    29
110853    29
110854    29
110855    29
110856    29
110857    29
110858    29
110859    29
110860    29
110861    29
110862    29
110863    29
110864    29
110865    29
110866    29
110867    29
110868    29
Name: time_stamp, Length: 110869, dtype: object
```

[46]:
```
count, division = np.histogram(df['temp_h'])
print(count)
print(division)
```

```
[ 2242  8186 15692 22738 20114 24683  9371  5856  1131   856]
[27.2   27.408 27.616 27.824 28.032 28.24  28.448 28.656 28.864 29.072
 29.28 ]
```

6.16.8 7. Transforming

Suppose we want to convert all values of column temperature which are floats to integers.

We know that to convert a float to an integer there the predefined python function `int`

[47]:
```
int(23.7)
```

[47]:
```
23
```

We would like 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 `int` *as a parameter*

NOTE: there are no round parenthesis after `int` !!!

[48]:
```
df['humidity'].transform(int)
```

[48]:
```
0          44
1          45
2          45
3          45
4          45
5          45
6          45
7          45
8          45
9          45
10         45
11         45
12         45
13         45
14         45
15         45
16         45
17         45
18         45
19         45
20         45
21         45
22         45
23         45
24         45
25         45
26         45
27         45
28         45
29         45
           ..
110839     42
110840     42
110841     42
110842     42
110843     42
110844     42
110845     42
110846     42
110847     43
110848     42
110849     42
```

(continues on next page)

(continued from previous page)

```
110850    42
110851    42
110852    42
110853    42
110854    42
110855    42
110856    42
110857    42
110858    42
110859    42
110860    42
110861    42
110862    42
110863    42
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 !)

[49]:
```
def myf(x):
    return int(x)

df['humidity'].transform(myf)
```

[49]:
```
0          44
1          45
2          45
3          45
4          45
5          45
6          45
7          45
8          45
9          45
10         45
11         45
12         45
13         45
14         45
15         45
16         45
17         45
18         45
19         45
20         45
21         45
22         45
23         45
24         45
```

(continues on next page)

(continued from previous page)

```
25        45
26        45
27        45
28        45
29        45
          ..
110839    42
110840    42
110841    42
110842    42
110843    42
110844    42
110845    42
110846    42
110847    43
110848    42
110849    42
110850    42
110851    42
110852    42
110853    42
110854    42
110855    42
110856    42
110857    42
110858    42
110859    42
110860    42
110861    42
110862    42
110863    42
110864    42
110865    42
110866    42
110867    42
110868    42
Name: humidity, Length: 110869, dtype: int64
```

lamda function: We could have used as well a lambda function, that is, a function without a name which is defined on one line:

[50]:
```
df['humidity'].transform( lambda x: int(x) )
```

[50]:
```
0         44
1         45
2         45
3         45
4         45
5         45
6         45
7         45
8         45
9         45
10        45
11        45
12        45
13        45
```

(continues on next page)

(continued from previous page)

```
14        45
15        45
16        45
17        45
18        45
19        45
20        45
21        45
22        45
23        45
24        45
25        45
26        45
27        45
28        45
29        45
          ..
110839    42
110840    42
110841    42
110842    42
110843    42
110844    42
110845    42
110846    42
110847    43
110848    42
110849    42
110850    42
110851    42
110852    42
110853    42
110854    42
110855    42
110856    42
110857    42
110858    42
110859    42
110860    42
110861    42
110862    42
110863    42
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:

[51]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 23 columns):
ROW_ID        110869 non-null int64
```

(continues on next page)

(continued from previous page)

```
temp_cpu      110869 non-null float64
temp_h        110869 non-null float64
temp_p        110869 non-null float64
humidity      110869 non-null float64
pressure      110869 non-null float64
pitch         110869 non-null float64
roll          110869 non-null float64
yaw           110869 non-null float64
mag_x         110869 non-null float64
mag_y         110869 non-null float64
mag_z         110869 non-null float64
accel_x       110869 non-null float64
accel_y       110869 non-null float64
accel_z       110869 non-null float64
gyro_x        110869 non-null float64
gyro_y        110869 non-null float64
gyro_z        110869 non-null float64
reset         110869 non-null int64
time_stamp    110869 non-null object
Too hot       105315 non-null object
check_p       110869 non-null object
mag_tot       110869 non-null float64
dtypes: float64(18), int64(2), object(3)
memory usage: 19.5+ MB
```

If we want to add a new column, say `huimdity_int`, we have to explicitly assigne the result of transform to a new series:

[52]:
```
df['humidity_int'] = df['humidity'].transform( lambda x: int(x) )
```

Notice how pandas automatically infers type `int64` for the newly created column:

[53]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 24 columns):
ROW_ID          110869 non-null int64
temp_cpu        110869 non-null float64
temp_h          110869 non-null float64
temp_p          110869 non-null float64
humidity        110869 non-null float64
pressure        110869 non-null float64
pitch           110869 non-null float64
roll            110869 non-null float64
yaw             110869 non-null float64
mag_x           110869 non-null float64
mag_y           110869 non-null float64
mag_z           110869 non-null float64
accel_x         110869 non-null float64
accel_y         110869 non-null float64
accel_z         110869 non-null float64
gyro_x          110869 non-null float64
gyro_y          110869 non-null float64
gyro_z          110869 non-null float64
reset           110869 non-null int64
time_stamp      110869 non-null object
```

(continues on next page)

(continued from previous page)

```
Too hot         105315 non-null object
check_p         110869 non-null object
mag_tot         110869 non-null float64
humidity_int    110869 non-null int64
dtypes: float64(18), int64(3), object(3)
memory usage: 20.3+ MB
```

6.16.9 8. Grouping

Reference:

- PythonDataScienceHandbook: Aggregation and Grouping[306]

It is pretty easy to group items and perform aggregated calculations by using `groupby` method. Let's say we want to count how many huidity readings were taken for each integer humidity (here we use pandas groupby, but for histograms you could also use numpy[307])

After `groupby` we can use `count()` aggregation function (other common ones are `sum()`, `mean()`, `min()`, `max()`):

[54]:
```
df.groupby(['humidity_int'])['humidity'].count()
```

[54]:
```
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
58      237
59      271
60      300
Name: humidity, dtype: int64
```

Notice we got only 19 rows. To have a series that fills the whole table, assigning to each row the count of its own group, we can use `transform` like this:

[55]:
```
df.groupby(['humidity_int'])['humidity'].transform('count')
```

[55]:
```
0         13029
1         32730
2         32730
3         32730
4         32730
```

(continues on next page)

[306] https://jakevdp.github.io/PythonDataScienceHandbook/03.08-aggregation-and-grouping.html
[307] https://stackoverflow.com/a/13130357

(continued from previous page)

```
5          32730
6          32730
7          32730
8          32730
9          32730
10         32730
11         32730
12         32730
13         32730
14         32730
15         32730
16         32730
17         32730
18         32730
19         32730
20         32730
21         32730
22         32730
23         32730
24         32730
25         32730
26         32730
27         32730
28         32730
29         32730
           …
110839      2776
110840      2776
110841      2776
110842      2776
110843      2776
110844      2776
110845      2776
110846      2776
110847      2479
110848      2776
110849      2776
110850      2776
110851      2776
110852      2776
110853      2776
110854      2776
110855      2776
110856      2776
110857      2776
110858      2776
110859      2776
110860      2776
110861      2776
110862      2776
110863      2776
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:

[56]:
```
df['Humidity counts'] = df.groupby(['humidity_int'])['humidity'].transform('count')
```

[57]:
```
df
```

[57]:

	ROW_ID	temp_cpu	temp_h	temp_p	humidity	pressure	pitch	roll	\
0	1	31.88	27.57	25.01	44.94	1001.68	1.49	52.25	
1	2	31.79	27.53	25.01	45.12	1001.72	1.03	53.73	
2	3	31.66	27.53	25.01	45.12	1001.72	1.24	53.57	
3	4	31.69	27.52	25.01	45.32	1001.69	1.57	53.63	
4	5	31.66	27.54	25.01	45.18	1001.71	0.85	53.66	
5	6	31.69	27.55	25.01	45.12	1001.67	0.85	53.53	
6	7	31.68	27.53	25.01	45.31	1001.70	0.63	53.55	
7	8	31.66	27.55	25.01	45.34	1001.70	1.49	53.65	
8	9	31.67	27.54	25.01	45.20	1001.72	1.22	53.77	
9	10	31.67	27.54	25.01	45.41	1001.75	1.63	53.46	
10	11	31.68	27.53	25.00	45.16	1001.72	1.32	53.52	
11	12	31.67	27.52	25.00	45.48	1001.72	1.51	53.47	
12	13	31.63	27.53	25.00	45.20	1001.72	1.55	53.75	
13	14	31.69	27.53	25.00	45.28	1001.71	1.07	53.63	
14	15	31.70	27.52	25.00	45.14	1001.72	0.81	53.40	
15	16	31.72	27.53	25.00	45.31	1001.75	1.51	53.34	
16	17	31.71	27.52	25.00	45.14	1001.72	1.82	53.49	
17	18	31.67	27.53	25.00	45.23	1001.71	0.46	53.69	
18	19	31.67	27.53	25.00	45.28	1001.71	0.67	53.55	
19	20	31.69	27.53	25.00	45.21	1001.71	1.23	53.43	
20	21	31.69	27.51	25.00	45.18	1001.71	1.44	53.58	
21	22	31.66	27.52	25.00	45.18	1001.73	1.25	53.34	
22	23	31.68	27.54	25.00	45.25	1001.72	1.18	53.49	
23	24	31.67	27.53	24.99	45.30	1001.72	1.34	53.32	
24	25	31.65	27.53	25.00	45.40	1001.71	1.36	53.56	
25	26	31.67	27.52	25.00	45.33	1001.72	1.17	53.44	
26	27	31.74	27.54	25.00	45.27	1001.71	0.88	53.41	
27	28	31.63	27.52	25.00	45.33	1001.75	0.78	53.84	
28	29	31.68	27.52	25.00	45.33	1001.73	0.88	53.41	
29	30	31.67	27.51	25.00	45.21	1001.74	0.86	53.29	
…	…	…	…	…	…	…	…	…	
110839	110840	31.60	27.49	24.82	42.74	1005.83	1.12	49.34	
110840	110841	31.59	27.48	24.82	42.75	1005.82	2.04	49.53	
110841	110842	31.59	27.51	24.82	42.76	1005.82	1.31	49.19	
110842	110843	31.60	27.50	24.82	42.74	1005.85	1.19	48.91	
110843	110844	31.57	27.49	24.82	42.80	1005.83	1.49	49.17	
110844	110845	31.60	27.50	24.82	42.81	1005.84	1.47	49.46	
110845	110846	31.61	27.50	24.82	42.81	1005.82	2.28	49.27	
110846	110847	31.61	27.50	24.82	42.75	1005.84	2.18	49.64	
110847	110848	31.58	27.50	24.82	43.00	1005.85	2.52	49.31	
110848	110849	31.54	27.51	24.82	42.76	1005.84	2.35	49.55	
110849	110850	31.60	27.50	24.82	42.79	1005.82	2.33	48.79	
110850	110851	31.61	27.50	24.82	42.79	1005.85	2.11	49.66	
110851	110852	31.56	27.50	24.83	42.84	1005.83	1.68	49.91	
110852	110853	31.59	27.51	24.83	42.76	1005.82	2.26	49.17	
110853	110854	31.58	27.50	24.83	42.98	1005.83	1.96	49.41	
110854	110855	31.61	27.51	24.83	42.69	1005.84	2.27	49.39	
110855	110856	31.55	27.50	24.83	42.79	1005.83	1.51	48.98	
110856	110857	31.55	27.49	24.83	42.81	1005.82	2.12	49.95	
110857	110858	31.60	27.51	24.83	42.92	1005.82	1.53	49.33	

(continues on next page)

(continued from previous page)

```
110858  110859     31.58   27.50   24.83     42.81   1005.83   1.60  49.65
110859  110860     31.61   27.50   24.83     42.82   1005.84   2.65  49.47
110860  110861     31.57   27.50   24.83     42.80   1005.84   2.63  50.08
110861  110862     31.58   27.51   24.83     42.90   1005.85   1.70  49.81
110862  110863     31.60   27.51   24.83     42.80   1005.85   1.66  49.13
110863  110864     31.64   27.51   24.83     42.80   1005.85   1.91  49.31
110864  110865     31.56   27.52   24.83     42.94   1005.83   1.58  49.93
110865  110866     31.55   27.50   24.83     42.72   1005.85   1.89  49.92
110866  110867     31.58   27.50   24.83     42.83   1005.85   2.09  50.00
110867  110868     31.62   27.50   24.83     42.81   1005.88   2.88  49.69
110868  110869     31.57   27.51   24.83     42.94   1005.86   2.17  49.77

           yaw      mag_x  …    gyro_x    gyro_y    gyro_z  reset  \
0       185.21 -46.422753  …  0.000942  0.000492 -0.000750     20
1       186.72 -48.778951  …  0.000218 -0.000005 -0.000235      0
2       186.21 -49.161878  …  0.000395  0.000600 -0.000003      0
3       186.03 -49.341941  …  0.000308  0.000577 -0.000102      0
4       186.46 -50.056683  …  0.000321  0.000691  0.000272      0
5       185.52 -50.246476  …  0.000273  0.000494 -0.000059      0
6       186.10 -50.447346  … -0.000111  0.000320  0.000222      0
7       186.08 -50.668232  … -0.000044  0.000436  0.000301      0
8       186.55 -50.761529  …  0.000358  0.000651  0.000187      0
9       185.94 -51.243832  …  0.000266  0.000676  0.000356      0
10      186.24 -51.616473  …  0.000268  0.001194  0.000106      0
11      186.17 -51.781714  …  0.000859  0.001221  0.000264      0
12      186.38 -51.992696  …  0.000589  0.001151  0.000002      0
13      186.60 -52.409175  …  0.000497  0.000610 -0.000060      0
14      186.32 -52.648488  … -0.000053  0.000593 -0.000141      0
15      186.42 -52.850708  … -0.000238  0.000495  0.000156      0
16      186.39 -53.449140  …  0.000571  0.000770  0.000331      0
17      186.72 -53.679986  … -0.000187  0.000159  0.000386      0
18      186.61 -54.159015  … -0.000495  0.000094  0.000084      0
19      186.21 -54.400646  … -0.000338  0.000013  0.000041      0
20      186.40 -54.609398  … -0.000266  0.000279 -0.000009      0
21      186.50 -54.746114  …  0.000139  0.000312  0.000050      0
22      186.69 -55.091416  … -0.000489  0.000086  0.000065      0
23      186.84 -55.516313  …  0.000312  0.000175  0.000308      0
24      187.02 -55.560991  … -0.000101 -0.000023  0.000377      0
25      186.95 -56.016359  …  0.000147  0.000054  0.000147      0
26      186.57 -56.393694  … -0.000125 -0.000193  0.000269      0
27      186.85 -56.524545  … -0.000175 -0.000312  0.000361      0
28      186.62 -56.791585  … -0.000382 -0.000253  0.000132      0
29      186.71 -56.915466  …  0.000031 -0.000260  0.000069      0
…         …         …  …       …        …        …    …
110839   90.42   0.319629  … -0.000209 -0.000005  0.000138      0
110840   92.11   0.015879  … -0.000472 -0.000478  0.000126      0
110841   93.94  -0.658624  … -0.000590 -0.000372  0.000207      0
110842   95.57  -1.117541  … -0.000039  0.000059  0.000149      0
110843   98.11  -1.860475  …  0.000223  0.000278  0.000038      0
110844   99.67  -2.286044  … -0.000283 -0.000187  0.000077      0
110845  103.17  -3.182359  … -0.000451 -0.000100 -0.000351      0
110846  105.05  -3.769940  …  0.000476  0.000452 -0.000249      0
110847  107.23  -4.431722  …  0.000822  0.000739 -0.000012      0
110848  108.68  -4.944477  …  0.000613  0.000763 -0.000227      0
110849  109.52  -5.481255  …  0.000196  0.000519 -0.000234      0
110850  111.90  -6.263577  …  0.000029 -0.000098 -0.000073      0
110851  113.38  -6.844946  … -0.000177 -0.000452 -0.000232      0
```

(continues on next page)

(continued from previous page)

```
110852  114.42  -7.437300  …  0.000396  0.000400 -0.000188      0
110853  116.50  -8.271114  …  0.000285  0.000312 -0.000058      0
110854  117.61  -8.690470  … -0.000001  0.000371 -0.000274      0
110855  119.13  -9.585351  … -0.000115  0.000029 -0.000223      0
110856  120.81 -10.120745  … -0.000150  0.000147 -0.000320      0
110857  121.74 -10.657858  …  0.000161  0.000029 -0.000210      0
110858  123.50 -11.584851  … -0.000073  0.000158 -0.000006      0
110859  124.51 -12.089743  …  0.000428  0.000137  0.000201      0
110860  125.85 -12.701497  …  0.000263  0.000163  0.000031      0
110861  126.86 -13.393369  … -0.000077  0.000179  0.000148      0
110862  127.35 -13.990712  …  0.000119 -0.000204  0.000041      0
110863  128.62 -14.691672  …  0.000286  0.000103  0.000221      0
110864  129.60 -15.169673  … -0.000264  0.000206  0.000196      0
110865  130.51 -15.832622  …  0.000143  0.000199 -0.000024      0
110866  132.04 -16.646212  …  0.000537  0.000257  0.000057      0
110867  133.00 -17.270447  …  0.000534  0.000456  0.000195      0
110868  134.18 -17.885872  …  0.000459  0.000076  0.000030      0

                 time_stamp  Too hot  check_p      mag_tot  humidity_int  \
0       2016-02-16 10:44:40     True    sotto  2368.337207            44
1       2016-02-16 10:44:50     True    sotto  2615.870247            45
2       2016-02-16 10:45:00      NaN    sotto  2648.484927            45
3       2016-02-16 10:45:10     True    sotto  2665.305485            45
4       2016-02-16 10:45:20      NaN    sotto  2732.388620            45
5       2016-02-16 10:45:30     True    sotto  2736.836291            45
6       2016-02-16 10:45:41      NaN    sotto  2756.496929            45
7       2016-02-16 10:45:50      NaN    sotto  2778.429164            45
8       2016-02-16 10:46:01      NaN    sotto  2773.029554            45
9       2016-02-16 10:46:10      NaN    sotto  2809.446772            45
10      2016-02-16 10:46:20      NaN    sotto  2851.426683            45
11      2016-02-16 10:46:30      NaN    sotto  2864.856376            45
12      2016-02-16 10:46:40      NaN    sotto  2880.392591            45
13      2016-02-16 10:46:50     True    sotto  2921.288936            45
14      2016-02-16 10:47:00     True    sotto  2946.615432            45
15      2016-02-16 10:47:11     True    sotto  2967.640766            45
16      2016-02-16 10:47:20     True    sotto  3029.683044            45
17      2016-02-16 10:47:31      NaN    sotto  3052.251538            45
18      2016-02-16 10:47:40      NaN    sotto  3095.501435            45
19      2016-02-16 10:47:51     True    sotto  3110.640598            45
20      2016-02-16 10:48:00     True    sotto  3140.151110            45
21      2016-02-16 10:48:10      NaN    sotto  3156.665111            45
22      2016-02-16 10:48:21      NaN    sotto  3188.235806            45
23      2016-02-16 10:48:30      NaN    sotto  3238.850567            45
24      2016-02-16 10:48:41      NaN    sotto  3242.425155            45
25      2016-02-16 10:48:50      NaN    sotto  3288.794716            45
26      2016-02-16 10:49:01     True    sotto  3320.328854            45
27      2016-02-16 10:49:10      NaN    sotto  3339.433796            45
28      2016-02-16 10:49:20      NaN    sotto  3364.310107            45
29      2016-02-16 10:49:30      NaN    sotto  3377.217368            45
…                     …       …       …       …           …
110839  2016-02-29 09:20:10      NaN    sotto   574.877314            42
110840  2016-02-29 09:20:20      NaN    sotto   593.855683            42
110841  2016-02-29 09:20:31      NaN    sotto   604.215692            42
110842  2016-02-29 09:20:40      NaN    sotto   606.406098            42
110843  2016-02-29 09:20:51      NaN    sotto   622.733559            42
110844  2016-02-29 09:21:00      NaN    sotto   641.480748            42
110845  2016-02-29 09:21:10      NaN    sotto   633.949204            42
```

(continues on next page)

(continued from previous page)

```
110846  2016-02-29 09:21:20      NaN    sotto   643.508698            42
110847  2016-02-29 09:21:30      NaN    sotto   658.512439            43
110848  2016-02-29 09:21:41      NaN    sotto   667.095455            42
110849  2016-02-29 09:21:50      NaN    sotto   689.714415            42
110850  2016-02-29 09:22:01      NaN    sotto   707.304506            42
110851  2016-02-29 09:22:10      NaN    sotto   726.361255            42
110852  2016-02-29 09:22:21      NaN    sotto   743.185242            42
110853  2016-02-29 09:22:30      NaN    sotto   767.328522            42
110854  2016-02-29 09:22:40      NaN    sotto   791.907055            42
110855  2016-02-29 09:22:50      NaN    sotto   802.932850            42
110856  2016-02-29 09:23:00      NaN    sotto   820.194642            42
110857  2016-02-29 09:23:11      NaN    sotto   815.462202            42
110858  2016-02-29 09:23:20      NaN    sotto   851.154631            42
110859  2016-02-29 09:23:31      NaN    sotto   879.563826            42
110860  2016-02-29 09:23:40      NaN    sotto   895.543882            42
110861  2016-02-29 09:23:50      NaN    sotto   928.948693            42
110862  2016-02-29 09:24:01      NaN    sotto   957.695014            42
110863  2016-02-29 09:24:10      NaN    sotto   971.126355            42
110864  2016-02-29 09:24:21      NaN    sotto   996.676408            42
110865  2016-02-29 09:24:30      NaN    sotto  1022.779594            42
110866  2016-02-29 09:24:41      NaN    sotto  1048.121268            42
110867  2016-02-29 09:24:50      NaN    sotto  1073.629703            42
110868  2016-02-29 09:25:00      NaN    sotto  1095.760426            42

       Humidity counts
0                13029
1                32730
2                32730
3                32730
4                32730
5                32730
6                32730
7                32730
8                32730
9                32730
10               32730
11               32730
12               32730
13               32730
14               32730
15               32730
16               32730
17               32730
18               32730
19               32730
20               32730
21               32730
22               32730
23               32730
24               32730
25               32730
26               32730
27               32730
28               32730
29               32730
…                …
110839            2776
```

(continues on next page)

(continued from previous page)

```
110840            2776
110841            2776
110842            2776
110843            2776
110844            2776
110845            2776
110846            2776
110847            2479
110848            2776
110849            2776
110850            2776
110851            2776
110852            2776
110853            2776
110854            2776
110855            2776
110856            2776
110857            2776
110858            2776
110859            2776
110860            2776
110861            2776
110862            2776
110863            2776
110864            2776
110865            2776
110866            2776
110867            2776
110868            2776

[110869 rows x 25 columns]
```

6.16.10 9. Exercise: meteo average temperatures

9.1 meteo plot

⊛ Put in a plot the temperature from dataframe *meteo*:

[58]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

# write here
```

[59]:
```
# SOLUTION
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

meteo.Temp.plot()
```

[59]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e74689828>
```

9.2 meteo pressure and raining

⊛ In the same plot as above show the pressure and amount of raining.

[60]:
```
# write here
```

[61]:
```
# 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);
```

9.3 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 perfomant and elegant is with `groupby` operator, see Pandas trasform - more than meets the eye[308]
- 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.

[62]:
```
# write here
```

[63]:
```
print()
print('    ****************    SOLUTION 1 - recalculate average for every row - slow !
↪')
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()
print("WITH AVERAGE TEMPERATURE")
print(meteo.head())
meteo.Temp.plot(label="Temperatura", legend=True)
meteo.Avg_day_temp.plot(label="Average temperature", legend=True)
```

```
    ****************    SOLUTION 1 - recalculate average for every row - slow !
WITH DAY
               Date  Pressure  Rain  Temp         Day
0  01/11/2017 00:00     995.4   0.0   5.4  01/11/2017
1  01/11/2017 00:15     995.5   0.0   6.0  01/11/2017
2  01/11/2017 00:30     995.5   0.0   5.9  01/11/2017
3  01/11/2017 00:45     995.7   0.0   5.4  01/11/2017
4  01/11/2017 01:00     995.7   0.0   5.3  01/11/2017
```

(continues on next page)

[308] https://towardsdatascience.com/pandas-transform-more-than-meets-the-eye-928542b40b56

(continued from previous page)

```
WITH AVERAGE TEMPERATURE
                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
```

[63]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e74499c50>
```

[64]:
```
print()
print('********    SOLUTION 2 - recalculate average only 30 times by using a␣
↪dictionary d_avg,')
print('                          faster but not yet optimal')

meteo = pd.read_csv('meteo.csv', encoding='UTF-8')
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()
print("WITH AVERAGE TEMPERATURE")
print(meteo.head())
meteo.Temp.plot(label="Temperature", legend=True)
meteo.Avg_day_temp.plot(label="Average temperature", legend=True)
```

```
********    SOLUTION 2 - recalculate average only 30 times by using a dictionary d_
↪avg,
```

(continues on next page)

(continued from previous page)

```
                        faster but not yet optimal

WITH DAY
                Date  Pressure  Rain  Temp         Day
0  01/11/2017 00:00     995.4   0.0   5.4  01/11/2017
1  01/11/2017 00:15     995.5   0.0   6.0  01/11/2017
2  01/11/2017 00:30     995.5   0.0   5.9  01/11/2017
3  01/11/2017 00:45     995.7   0.0   5.4  01/11/2017
4  01/11/2017 01:00     995.7   0.0   5.3  01/11/2017

WITH AVERAGE TEMPERATURE
                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
```

[64]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e74661668>
```

[65]:
```
print()
print('****************  SOLUTION 3  -  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')
meteo
print()
print("WITH AVERAGE TEMPERATURE")
print(meteo.head())
meteo.Temp.plot(label="Temperatura", legend=True)
meteo.Avg_day_temp.plot(label="Average temperature", legend=True)
```

```
****************  SOLUTION 3  -  best solution with groupby and transform

WITH AVERAGE TEMPERATURE
                Date  Pressure  Rain  Temp         Day  Avg_day_temp
```

(continues on next page)

(continued from previous page)

```
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
```

[65]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8e7e704780>
```

6.16.11 10. 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

[66]:
```
iss_coords = pd.read_csv('iss-coords.csv', encoding='UTF-8')
```

[67]:
```
iss_coords
```

[67]:
```
                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
5    2016-01-01 18:24:00 -15.990725 -106.400927
6    2016-01-01 22:45:51  31.602388   85.647998
7    2016-01-02 07:48:31 -51.578009  -26.736801
8    2016-01-02 10:50:19 -36.512021   14.452174
9    2016-01-02 14:01:27 -27.459029   10.991151
10   2016-01-02 14:01:27 -27.458783   10.991398
11   2016-01-02 20:30:13  29.861877  156.955941
12   2016-01-03 11:43:18   9.065825 -172.436293
13   2016-01-03 14:39:47  15.529901   35.812502
14   2016-01-03 14:39:47  15.530149   35.812698
15   2016-01-03 21:12:17 -44.793666  -28.679197
16   2016-01-03 22:39:52  28.061007  178.935724
17   2016-01-04 13:40:02 -14.153170 -139.759391
18   2016-01-04 13:51:36   9.461309   30.520802
19   2016-01-04 13:51:36   9.461560   30.520986
```

(continues on next page)

(continued from previous page)

```
20   2016-01-04 18:42:18  44.974327   84.801522
21   2016-01-04 21:46:03 -51.551958  -75.103323
22   2016-01-04 21:46:03 -51.551933  -75.102954
23   2016-01-05 12:57:50 -41.439217    3.847215
24   2016-01-05 14:36:00 -13.581246   39.166522
25   2016-01-05 14:36:00 -13.581024   39.166692
26   2016-01-05 17:51:36  26.103252 -151.570312
27   2016-01-05 22:28:56 -26.458448 -108.642807
28   2016-01-06 12:07:09 -51.204236  -19.679525
29   2016-01-06 13:41:23 -51.166546  -19.318519
..                   …         …          …
308  2016-02-25 21:19:08  14.195431 -133.777268
309  2016-02-25 21:38:48 -14.698631  -85.875320
310  2016-02-26 00:51:29  -4.376121  -94.773870
311  2016-02-26 00:51:29 -51.097174  -21.117794
312  2016-02-26 13:09:56  -1.811782  -99.010499
313  2016-02-26 14:28:13 -15.363988  -87.986579
314  2016-02-26 14:28:13 -15.364276  -87.986354
315  2016-02-26 17:49:36 -32.517607   47.514800
316  2016-02-26 22:37:28 -41.292043   29.733597
317  2016-02-27 01:43:10 -41.049112   30.193004
318  2016-02-27 01:43:10  -8.402991 -100.981726
319  2016-02-27 13:34:30  18.406130 -126.884570
320  2016-02-27 13:52:46 -22.783724  -90.869452
321  2016-02-27 13:52:46 -22.784018  -90.869189
322  2016-02-27 21:47:45  -7.038283 -106.607037
323  2016-02-28 00:51:03 -31.699384  -84.328371
324  2016-02-28 08:13:04  40.239764 -155.465692
325  2016-02-28 09:48:40  50.047523  175.566751
326  2016-02-28 14:29:36  37.854997  106.124377
327  2016-02-28 14:29:36  37.855237  106.124735
328  2016-02-28 20:56:33  51.729529  163.754128
329  2016-02-29 04:39:20 -10.946978 -100.874429
330  2016-02-29 08:56:28  46.885514 -167.143393
331  2016-02-29 10:32:56  46.773608 -166.800893
332  2016-02-29 11:53:49  46.678097 -166.512208
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 that `time_stamp` column (notice the underscore _) in original astropi dataset:

[68]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110869 entries, 0 to 110868
Data columns (total 25 columns):
ROW_ID             110869 non-null int64
temp_cpu           110869 non-null float64
temp_h             110869 non-null float64
temp_p             110869 non-null float64
humidity           110869 non-null float64
```

(continues on next page)

(continued from previous page)

```
pressure            110869 non-null float64
pitch               110869 non-null float64
roll                110869 non-null float64
yaw                 110869 non-null float64
mag_x               110869 non-null float64
mag_y               110869 non-null float64
mag_z               110869 non-null float64
accel_x             110869 non-null float64
accel_y             110869 non-null float64
accel_z             110869 non-null float64
gyro_x              110869 non-null float64
gyro_y              110869 non-null float64
gyro_z              110869 non-null float64
reset               110869 non-null int64
time_stamp          110869 non-null object
Too hot             105315 non-null object
check_p             110869 non-null object
mag_tot             110869 non-null float64
humidity_int        110869 non-null int64
Humidity counts     110869 non-null int64
dtypes: float64(18), int64(4), object(3)
memory usage: 21.1+ MB
```

To merge datasets according to the columns, we can use the command `merge` like this:

[69]:
```
# 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)
```

[70]:
```
geo_astropi
```

[70]:
```
   ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll     yaw  \
0   23231     32.53   28.37   25.89     45.31   1006.04   1.31  51.63   34.91
1   27052     32.30   28.12   25.62     45.57   1007.42   1.49  52.29  333.49
2   27052     32.30   28.12   25.62     45.57   1007.42   1.49  52.29  333.49
3   46933     32.21   28.05   25.50     47.36   1012.41   0.67  52.40   27.57
4   64572     32.32   28.18   25.61     47.45   1010.62   1.14  51.41   33.68
5   68293     32.39   28.26   25.70     46.83   1010.51   0.61  51.91  287.86
6   73374     32.38   28.18   25.62     46.52   1008.28   0.90  51.77   30.80
7   90986     32.42   28.34   25.76     45.72   1006.79   0.57  49.85   10.57
8   90986     32.42   28.34   25.76     45.72   1006.79   0.57  49.85   10.57
9  102440     32.62   28.62   26.02     45.15   1006.06   1.12  50.44  301.74

       mag_x  …    gyro_z  reset           time_stamp  Too hot  check_p  \
0  21.125001  …  0.000046      0  2016-02-19 03:49:00     True    sotto
1  16.083471  …  0.000034      0  2016-02-19 14:30:40     True    sotto
2  16.083471  …  0.000034      0  2016-02-19 14:30:40     True    sotto
3  15.441683  …  0.000221      0  2016-02-21 22:14:11     True    sopra
4  11.994554  …  0.000030      0  2016-02-23 23:40:50     True    sopra
5   6.554283  …  0.000171      0  2016-02-24 10:05:51     True    sopra
6   9.947132  … -0.000375      0  2016-02-25 00:23:01     True    sopra
7   7.805606  … -0.000047      0  2016-02-27 01:43:10     True    sotto
8   7.805606  … -0.000047      0  2016-02-27 01:43:10     True    sotto
```

(continues on next page)

(continued from previous page)

```
9  10.348327  … -0.000061      0  2016-02-28 09:48:40     True    sotto

       mag_tot  humidity_int  Humidity counts        lat         lon
0  2345.207992            45            32730  31.434741   52.917464
1   323.634786            45            32730 -46.620658  -57.311657
2   323.634786            45            32730 -46.620477  -57.311138
3   342.159257            47            14176  19.138359 -140.211489
4   264.655601            47            14176   4.713819   80.261665
5   436.876111            46            35775 -46.061583   22.246025
6   226.089258            46            35775  47.047346  137.958918
7   149.700293            45            32730 -41.049112   30.193004
8   149.700293            45            32730  -8.402991 -100.981726
9   381.014223            45            32730  50.047523  175.566751

[10 rows x 27 columns]
```

Exercise 10.1 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 stategies, read about attribute `how` in Why And How To Use Merge With Pandas in Python[309]
- To fill missing values don't use fancy interpolation techniques, just put the station position in that given day or hour

[71]:
```
geo_astropi = df.merge(iss_coords, left_on='time_stamp', right_on='timestamp', how=
↪'left')
```

[72]:
```
pd.merge_ordered(df, iss_coords, fill_method='ffill', how='left', left_on='time_stamp
↪', right_on='timestamp')
geo_astropi
```

[72]:
```
        ROW_ID  temp_cpu  temp_h  temp_p  humidity  pressure  pitch   roll  \
0            1     31.88   27.57   25.01     44.94   1001.68   1.49  52.25
1            2     31.79   27.53   25.01     45.12   1001.72   1.03  53.73
2            3     31.66   27.53   25.01     45.12   1001.72   1.24  53.57
3            4     31.69   27.52   25.01     45.32   1001.69   1.57  53.63
4            5     31.66   27.54   25.01     45.18   1001.71   0.85  53.66
5            6     31.69   27.55   25.01     45.12   1001.67   0.85  53.53
6            7     31.68   27.53   25.01     45.31   1001.70   0.63  53.55
7            8     31.66   27.55   25.01     45.34   1001.70   1.49  53.65
8            9     31.67   27.54   25.01     45.20   1001.72   1.22  53.77
9           10     31.67   27.54   25.01     45.41   1001.75   1.63  53.46
10          11     31.68   27.53   25.00     45.16   1001.72   1.32  53.52
11          12     31.67   27.52   25.00     45.48   1001.72   1.51  53.47
12          13     31.63   27.53   25.00     45.20   1001.72   1.55  53.75
13          14     31.69   27.53   25.00     45.28   1001.71   1.07  53.63
14          15     31.70   27.52   25.00     45.14   1001.72   0.81  53.40
15          16     31.72   27.53   25.00     45.31   1001.75   1.51  53.34
16          17     31.71   27.52   25.00     45.14   1001.72   1.82  53.49
17          18     31.67   27.53   25.00     45.23   1001.71   0.46  53.69
18          19     31.67   27.53   25.00     45.28   1001.71   0.67  53.55
19          20     31.69   27.53   25.00     45.21   1001.71   1.23  53.43
```

(continues on next page)

[309] https://towardsdatascience.com/why-and-how-to-use-merge-with-pandas-in-python-548600f7e738

(continued from previous page)

```
20          21     31.69   27.51   25.00     45.18   1001.71   1.44  53.58
21          22     31.66   27.52   25.00     45.18   1001.73   1.25  53.34
22          23     31.68   27.54   25.00     45.25   1001.72   1.18  53.49
23          24     31.67   27.53   24.99     45.30   1001.72   1.34  53.32
24          25     31.65   27.53   25.00     45.40   1001.71   1.36  53.56
25          26     31.67   27.52   25.00     45.33   1001.72   1.17  53.44
26          27     31.74   27.54   25.00     45.27   1001.71   0.88  53.41
27          28     31.63   27.52   25.00     45.33   1001.75   0.78  53.84
28          29     31.68   27.52   25.00     45.33   1001.73   0.88  53.41
29          30     31.67   27.51   25.00     45.21   1001.74   0.86  53.29
…        …       …      …      …       …       …    …     …
110841  110840     31.60   27.49   24.82     42.74   1005.83   1.12  49.34
110842  110841     31.59   27.48   24.82     42.75   1005.82   2.04  49.53
110843  110842     31.59   27.51   24.82     42.76   1005.82   1.31  49.19
110844  110843     31.60   27.50   24.82     42.74   1005.85   1.19  48.91
110845  110844     31.57   27.49   24.82     42.80   1005.83   1.49  49.17
110846  110845     31.60   27.50   24.82     42.81   1005.84   1.47  49.46
110847  110846     31.61   27.50   24.82     42.81   1005.82   2.28  49.27
110848  110847     31.61   27.50   24.82     42.75   1005.84   2.18  49.64
110849  110848     31.58   27.50   24.82     43.00   1005.85   2.52  49.31
110850  110849     31.54   27.51   24.82     42.76   1005.84   2.35  49.55
110851  110850     31.60   27.50   24.82     42.79   1005.82   2.33  48.79
110852  110851     31.61   27.50   24.82     42.79   1005.85   2.11  49.66
110853  110852     31.56   27.50   24.83     42.84   1005.83   1.68  49.91
110854  110853     31.59   27.51   24.83     42.76   1005.82   2.26  49.17
110855  110854     31.58   27.50   24.83     42.98   1005.83   1.96  49.41
110856  110855     31.61   27.51   24.83     42.69   1005.84   2.27  49.39
110857  110856     31.55   27.50   24.83     42.79   1005.83   1.51  48.98
110858  110857     31.55   27.49   24.83     42.81   1005.82   2.12  49.95
110859  110858     31.60   27.51   24.83     42.92   1005.82   1.53  49.33
110860  110859     31.58   27.50   24.83     42.81   1005.83   1.60  49.65
110861  110860     31.61   27.50   24.83     42.82   1005.84   2.65  49.47
110862  110861     31.57   27.50   24.83     42.80   1005.84   2.63  50.08
110863  110862     31.58   27.51   24.83     42.90   1005.85   1.70  49.81
110864  110863     31.60   27.51   24.83     42.80   1005.85   1.66  49.13
110865  110864     31.64   27.51   24.83     42.80   1005.85   1.91  49.31
110866  110865     31.56   27.52   24.83     42.94   1005.83   1.58  49.93
110867  110866     31.55   27.50   24.83     42.72   1005.85   1.89  49.92
110868  110867     31.58   27.50   24.83     42.83   1005.85   2.09  50.00
110869  110868     31.62   27.50   24.83     42.81   1005.88   2.88  49.69
110870  110869     31.57   27.51   24.83     42.94   1005.86   2.17  49.77

           yaw      mag_x  …  reset           time_stamp  Too hot  check_p  \
0       185.21 -46.422753  …     20  2016-02-16 10:44:40     True    sotto
1       186.72 -48.778951  …      0  2016-02-16 10:44:50     True    sotto
2       186.21 -49.161878  …      0  2016-02-16 10:45:00      NaN    sotto
3       186.03 -49.341941  …      0  2016-02-16 10:45:10     True    sotto
4       186.46 -50.056683  …      0  2016-02-16 10:45:20      NaN    sotto
5       185.52 -50.246476  …      0  2016-02-16 10:45:30     True    sotto
6       186.10 -50.447346  …      0  2016-02-16 10:45:41      NaN    sotto
7       186.08 -50.668232  …      0  2016-02-16 10:45:50      NaN    sotto
8       186.55 -50.761529  …      0  2016-02-16 10:46:01      NaN    sotto
9       185.94 -51.243832  …      0  2016-02-16 10:46:10      NaN    sotto
10      186.24 -51.616473  …      0  2016-02-16 10:46:20      NaN    sotto
11      186.17 -51.781714  …      0  2016-02-16 10:46:30      NaN    sotto
12      186.38 -51.992696  …      0  2016-02-16 10:46:40      NaN    sotto
13      186.60 -52.409175  …      0  2016-02-16 10:46:50     True    sotto
```

(continues on next page)

(continued from previous page)

```
14      186.32 -52.648488  …      0  2016-02-16 10:47:00     True    sotto
15      186.42 -52.850708  …      0  2016-02-16 10:47:11     True    sotto
16      186.39 -53.449140  …      0  2016-02-16 10:47:20     True    sotto
17      186.72 -53.679986  …      0  2016-02-16 10:47:31      NaN    sotto
18      186.61 -54.159015  …      0  2016-02-16 10:47:40      NaN    sotto
19      186.21 -54.400646  …      0  2016-02-16 10:47:51     True    sotto
20      186.40 -54.609398  …      0  2016-02-16 10:48:00     True    sotto
21      186.50 -54.746114  …      0  2016-02-16 10:48:10      NaN    sotto
22      186.69 -55.091416  …      0  2016-02-16 10:48:21      NaN    sotto
23      186.84 -55.516313  …      0  2016-02-16 10:48:30      NaN    sotto
24      187.02 -55.560991  …      0  2016-02-16 10:48:41      NaN    sotto
25      186.95 -56.016359  …      0  2016-02-16 10:48:50      NaN    sotto
26      186.57 -56.393694  …      0  2016-02-16 10:49:01     True    sotto
27      186.85 -56.524545  …      0  2016-02-16 10:49:10      NaN    sotto
28      186.62 -56.791585  …      0  2016-02-16 10:49:20      NaN    sotto
29      186.71 -56.915466  …      0  2016-02-16 10:49:30      NaN    sotto
…         …         …  …    …                  …       …       …
110841   90.42   0.319629  …      0  2016-02-29 09:20:10      NaN    sotto
110842   92.11   0.015879  …      0  2016-02-29 09:20:20      NaN    sotto
110843   93.94  -0.658624  …      0  2016-02-29 09:20:31      NaN    sotto
110844   95.57  -1.117541  …      0  2016-02-29 09:20:40      NaN    sotto
110845   98.11  -1.860475  …      0  2016-02-29 09:20:51      NaN    sotto
110846   99.67  -2.286044  …      0  2016-02-29 09:21:00      NaN    sotto
110847  103.17  -3.182359  …      0  2016-02-29 09:21:10      NaN    sotto
110848  105.05  -3.769940  …      0  2016-02-29 09:21:20      NaN    sotto
110849  107.23  -4.431722  …      0  2016-02-29 09:21:30      NaN    sotto
110850  108.68  -4.944477  …      0  2016-02-29 09:21:41      NaN    sotto
110851  109.52  -5.481255  …      0  2016-02-29 09:21:50      NaN    sotto
110852  111.90  -6.263577  …      0  2016-02-29 09:22:01      NaN    sotto
110853  113.38  -6.844946  …      0  2016-02-29 09:22:10      NaN    sotto
110854  114.42  -7.437300  …      0  2016-02-29 09:22:21      NaN    sotto
110855  116.50  -8.271114  …      0  2016-02-29 09:22:30      NaN    sotto
110856  117.61  -8.690470  …      0  2016-02-29 09:22:40      NaN    sotto
110857  119.13  -9.585351  …      0  2016-02-29 09:22:50      NaN    sotto
110858  120.81 -10.120745  …      0  2016-02-29 09:23:00      NaN    sotto
110859  121.74 -10.657858  …      0  2016-02-29 09:23:11      NaN    sotto
110860  123.50 -11.584851  …      0  2016-02-29 09:23:20      NaN    sotto
110861  124.51 -12.089743  …      0  2016-02-29 09:23:31      NaN    sotto
110862  125.85 -12.701497  …      0  2016-02-29 09:23:40      NaN    sotto
110863  126.86 -13.393369  …      0  2016-02-29 09:23:50      NaN    sotto
110864  127.35 -13.990712  …      0  2016-02-29 09:24:01      NaN    sotto
110865  128.62 -14.691672  …      0  2016-02-29 09:24:10      NaN    sotto
110866  129.60 -15.169673  …      0  2016-02-29 09:24:21      NaN    sotto
110867  130.51 -15.832622  …      0  2016-02-29 09:24:30      NaN    sotto
110868  132.04 -16.646212  …      0  2016-02-29 09:24:41      NaN    sotto
110869  133.00 -17.270447  …      0  2016-02-29 09:24:50      NaN    sotto
110870  134.18 -17.885872  …      0  2016-02-29 09:25:00      NaN    sotto

            mag_tot  humidity_int  Humidity counts  timestamp  lat lon
0       2368.337207            44            13029        NaN  NaN NaN
1       2615.870247            45            32730        NaN  NaN NaN
2       2648.484927            45            32730        NaN  NaN NaN
3       2665.305485            45            32730        NaN  NaN NaN
4       2732.388620            45            32730        NaN  NaN NaN
5       2736.836291            45            32730        NaN  NaN NaN
6       2756.496929            45            32730        NaN  NaN NaN
7       2778.429164            45            32730        NaN  NaN NaN
```

(continues on next page)

(continued from previous page)

```
8       2773.029554            45             32730        NaN  NaN NaN
9       2809.446772            45             32730        NaN  NaN NaN
10      2851.426683            45             32730        NaN  NaN NaN
11      2864.856376            45             32730        NaN  NaN NaN
12      2880.392591            45             32730        NaN  NaN NaN
13      2921.288936            45             32730        NaN  NaN NaN
14      2946.615432            45             32730        NaN  NaN NaN
15      2967.640766            45             32730        NaN  NaN NaN
16      3029.683044            45             32730        NaN  NaN NaN
17      3052.251538            45             32730        NaN  NaN NaN
18      3095.501435            45             32730        NaN  NaN NaN
19      3110.640598            45             32730        NaN  NaN NaN
20      3140.151110            45             32730        NaN  NaN NaN
21      3156.665111            45             32730        NaN  NaN NaN
22      3188.235806            45             32730        NaN  NaN NaN
23      3238.850567            45             32730        NaN  NaN NaN
24      3242.425155            45             32730        NaN  NaN NaN
25      3288.794716            45             32730        NaN  NaN NaN
26      3320.328854            45             32730        NaN  NaN NaN
27      3339.433796            45             32730        NaN  NaN NaN
28      3364.310107            45             32730        NaN  NaN NaN
29      3377.217368            45             32730        NaN  NaN NaN
…             …            …                 …          …   …   ..
110841   574.877314            42              2776        NaN  NaN NaN
110842   593.855683            42              2776        NaN  NaN NaN
110843   604.215692            42              2776        NaN  NaN NaN
110844   606.406098            42              2776        NaN  NaN NaN
110845   622.733559            42              2776        NaN  NaN NaN
110846   641.480748            42              2776        NaN  NaN NaN
110847   633.949204            42              2776        NaN  NaN NaN
110848   643.508698            42              2776        NaN  NaN NaN
110849   658.512439            43              2479        NaN  NaN NaN
110850   667.095455            42              2776        NaN  NaN NaN
110851   689.714415            42              2776        NaN  NaN NaN
110852   707.304506            42              2776        NaN  NaN NaN
110853   726.361255            42              2776        NaN  NaN NaN
110854   743.185242            42              2776        NaN  NaN NaN
110855   767.328522            42              2776        NaN  NaN NaN
110856   791.907055            42              2776        NaN  NaN NaN
110857   802.932850            42              2776        NaN  NaN NaN
110858   820.194642            42              2776        NaN  NaN NaN
110859   815.462202            42              2776        NaN  NaN NaN
110860   851.154631            42              2776        NaN  NaN NaN
110861   879.563826            42              2776        NaN  NaN NaN
110862   895.543882            42              2776        NaN  NaN NaN
110863   928.948693            42              2776        NaN  NaN NaN
110864   957.695014            42              2776        NaN  NaN NaN
110865   971.126355            42              2776        NaN  NaN NaN
110866   996.676408            42              2776        NaN  NaN NaN
110867  1022.779594            42              2776        NaN  NaN NaN
110868  1048.121268            42              2776        NaN  NaN NaN
110869  1073.629703            42              2776        NaN  NaN NaN
110870  1095.760426            42              2776        NaN  NaN NaN

[110871 rows x 28 columns]
```

6.16.12 11. Other exercises

See 31 October 2019 Midterm simulation[310]

```
[ ]:
```

6.17 Binary relations solutions

6.17.1 Download exercises zip

Browse files online[311]

6.17.2 Introduction

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 chapter Graph formats[312]

What to do

- unzip exercises in a folder, you should get something like this:

```
binary-relations
    binary-relations.ipynb
    binary-relations-sol.ipynb
    jupman.py
    sciprog.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 `binary-relations/binary-relations.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 instuctions 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 aftwerwards, press `Alt + Enter`

[310] https://sciprog.davidleoni.it/exams/2019-10-31/exam-2019-10-31-sol.html
[311] https://github.com/DavidLeoni/sciprog-ds/tree/master/binary-relations
[312] https://sciprog.davidleoni.it/graph-formats/graph-formats-sol.html

- 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 sciprog 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 sciprog import draw_mat

draw_mat(
    [
        [True, False, False, False],
        [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 homerworks at all. Alarmingly, we notice student 2 did the homeworks for student 1. Resulting conspiration shall be severely punished

with a one year ban from having spritz at Emma's bar.

6.17.3 Exercises

is_reflexive_mat

⊛⊛ Implement now this function for matrices.

[4]:
```
def is_reflexive_mat(mat):

    """ RETURN True if nxn 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.
    """
    #jupman-raise
    n = len(mat)
    for i in range(n):
        if not mat[i][i]:
            return False
    return True
    #/jupman-raise



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],
```

(continues on next page)

(continued from previous page)

```
                    ]) == 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

assert is_reflexive_mat([
                        [True, True, True],
                        [True, True, True],
                        [True, True, True],

                    ]) == True    # m12
```

is_reflexive_adj

⊛⊛ Implement now the same function for dictionaries of adjacency lists.

[5]:
```
def is_reflexive_adj(d):

    """ 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.
    """
    #jupman-raise
```

(continues on next page)

(continued from previous page)

```
    for v in d:
        if not v in d[v]:
            return False
    return True
    #/jupman-raise



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']
```

(continues on next page)

(continued from previous page)

```
                        }) == 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
```

Symmetric relations

A graph is symmetric when for all nodes, if a node A links to another node B, there is a also a link from node B to A.

In real life, the typical symmetric relation is "is friend of". If you are friend to somene, 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 sciprog 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 a 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?

ANSWER: No, in a symmetric relation some nodes can be linked to themseves, 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.

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?

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 your friends list the information that you are a friend of yourself.

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 ?

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.

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 lovel 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 sciprog 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[313] (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):

[8]:
```
from sciprog import draw_adj

draw_adj({
    'Donald Duck' : ['Daisy Duck'],
    'Daisy Duck' : ['Donald Duck'],
    'Mickey Mouse' : ['Minnie'],
    'Minnie' : ['Mickey Mouse'],
    'Gladstone Gander' : ['Daisy Duck']

})
```

[313] https://en.wikipedia.org/wiki/Gladstone_Gander

is_symmetric_mat

⊛⊛ Implement an automated procedure to check whether or not a graph is symmetrical. Implement this function for matrices:

[9]:
```
def is_symmetric_mat(mat):
    """ 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 a also a link from node B to A.

        NOTE: if
    """
    #jupman-raise
    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
    #/jupman-raise

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
```

(continues on next page)

(continued from previous page)

```
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],
                        [True, False, False],
                        [True, True, True],

                    ]) == False    # m11

assert is_symmetric_mat([
                        [False, True, True],
                        [True, False, True],
                        [True, True, True],

                    ]) == True    # m12
```

is_symmetric_adj

⊛⊛ Now implement the same as before but for a dictionary of adjacency lists:

[10]:
```
def is_symmetric_adj(d):
    """ 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 a also a link from node B to A.

    """
    #jupman-raise
    for k in d:
        for v in d[k]:
            if not k in d[v]:
                return False
    return True
    #/jupman-raise

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
```

(continues on next page)

(continued from previous page)

```
assert is_symmetric_adj({
                        'a' : ['b'],
                        'b' : ['a','b']
                    }) == 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
```

surjective

⊛⊛ 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)
```

[15]:
```
def surjective(mat):
    """ RETURN True if provided graph mat as list of boolean lists is an
        nxn surjective binary relation, otherwise return False
    """
    #jupman-raise
    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
    #/jupman-raise



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],
      ]
```

(continues on next page)

(continued from previous page)

```
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],
         [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
```

6.17.4 Further resources

- Rule based design[314] 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 theorical 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

[314] https://www.researchgate.net/profile/Stef_Joosten/publication/327022933_Rule_Based_Design/links/5b7321be45851546c903234a/Rule-Based-Design.pdf

[ ]:

```

```

CHAPTER

SEVEN

PART B

7.1 OOP

7.1.1 Download exercises zip

Browse files online[315]

7.1.2 What to do

- unzip exercises in a folder, you should get something like this:

```
oop
    oop.ipynb
    ComplexNumber_sol.py
    ComplexNumber.py
    jupman.py
    sciprog.py
```

This time you will not write in the notebook, instead you will edit .py files in Visual Studio Code.

Now proceed reading.

7.1.3 1. Abstract Data Types (ADT) Theory

1.1. Intro

- Theory from the slides:
  - Programming paradagims, Object-Oriented Python[316]
  - Data structures slides[317] (First slides until slide 13 'Comments')
- Object Oriented programming on the the book[318] (In particular, Fraction class[319], in this course we won't focus on inheritance)

[315] https://github.com/DavidLeoni/sciprog-ds/tree/master/oop-formats
[316] http://disi.unitn.it/~montreso/sp/slides/A09-oop.pdf#Navigation5
[317] http://disi.unitn.it/~montreso/sp/slides/B02-strutture.pdf
[318] http://interactivepython.org/runestone/static/pythonds/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html
[319] http://interactivepython.org/runestone/static/pythonds/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html#a-fraction-class

1.2. Complex number theory

A **complex number** is a **number** that can be expressed in the form a + bi, where a and b are real **numbers** and i is the imaginary unit which satisfies the equation $i^2 = -1$. In this expression, a is the real part and b is the imaginary part of the **complex number**.

Complex number - Wikipedia
https://en.wikipedia.org/wiki/**Complex_number**

1.3. Datatypes the old way

From the definition we see that to identify a complex number **we need two float values** . One number is for the ***real* part**, and another number is for the ***imaginary* part**.

How can we represent this in Python? So far, you saw there are many ways to put two numbers together. One way could be to put the numbers in a list of two elements, and implicitly assume the first one is the *real* and the second the *imaginary* part:

[2]:
```
c = [3.0, 5.0]
```

Or we could use a tuple:

[3]:
```
c = (3.0, 5.0)
```

A problem with the previous representations is that a casual observer might not know exactly the meaning of the two numbers. We could be more explicit and store the values into a dictionary, using keys to identify the two parts:

[4]:
```
c = {'real': 3.0, 'imaginary': 5.0}
```

[5]:
```
print(c)
```

```
{'real': 3.0, 'imaginary': 5.0}
```

[6]:
```
print(c['real'])
```

```
3.0
```

[7]:
```
print(c['imaginary'])
```

```
5.0
```

Now, writing the whole record `{'real': 3.0, 'imaginary': 5.0}` each time we want to create a complex number might be annoying and error prone. To help us, we can create a little shortcut function named `complex_number` that creates and returns the dictionary:

[8]:
```
def complex_number(real, imaginary):
    d = {}
    d['real'] = real
```

(continues on next page)

(continued from previous page)

```
    d['imaginary'] = imaginary
    return d
```

[9]:
```
c = complex_number(3.0, 5.0)
```

[10]:
```
print(c)
```

```
{'real': 3.0, 'imaginary': 5.0}
```

To do something with our dictionary, we would then define functions, like for example `complex_str` to show them nicely:

[11]:
```
def complex_str(cn):
    return str(cn['real']) + " + " + str(cn['imaginary']) + "i"
```

[12]:
```
c = complex_number(3.0, 5.0)
print(complex_str(c))
```

```
3.0 + 5.0i
```

We could do something more complex, like defining the `phase` of the complex number which returns a `float`:

IMPORTANT: In these exercises, we care about programming, not complex numbers theory. There's no need to break your head over formulas!

[13]:
```
import math
def phase(cn):
        """ Returns a float which is the phase (that is, the vector angle) of the␣
↪complex number

            See definition: https://en.wikipedia.org/wiki/Complex_number#Absolute_
↪value_and_argument
        """
        return math.atan2(cn['imaginary'], cn['real'])
```

[14]:
```
c = complex_number(3.0, 5.0)
print(phase(c))
```

```
1.0303768265243125
```

We could even define functions that that take the complex number and some other parameter, for example we could define the `log` of complex numbers, which return another complex number (mathematically it would be infinitely many, but we just pick the first one in the series):

[15]:
```
import math
def log(cn, base):
        """ Returns another complex number which is the logarithm of this complex␣
↪number

            See definition (accomodated for generic base b):
            https://en.wikipedia.org/wiki/Complex_number#Natural_logarithm
        """
        return {'real':math.log(cn['real']) / math.log(base),
                'imaginary' : phase(cn) / math.log(base)}
```

[16]:
```
print(log(c,2))
```

```
{'real': 1.5849625007211563, 'imaginary': 1.4865195378735334}
```

You see we got our dictionary representing a complex number. If we want a nicer display we can call on it the `complex_str` we defined:

[17]:
```
print(complex_str(log(c,2)))
```

```
1.5849625007211563 + 1.4865195378735334i
```

1.4. Finding the pattern

So, what have we done so far?

1) Decided a data format for the complex number, saw that the dictionary is quite convenient
2) Defined a function to quickly create the dictionary:

```
def complex_number(real, imaginary):
```

3) Defined some function like `phase` and `log` to do stuff on the complex number

```
def phase(cn):
def log(cn, base):
```

4) Defined a function `complex_str` to express the complex number as a readable string:

```
def complex_str(cn):
```

Notice that: * all functions above take a `cn` complex number dictionary as first parameter * the functions `phase` and `log` are quite peculiar to complex number, and to know what they do you need to have deep knowledge of what a complex number is. * the function `complex_str` is more intuitive, because it covers the common need of giving a nice string representation to the data format we just defined. Also, we used the word `str` as part of the name to give a hint to the reader that probably the function behaves in a way similar to the Python function `str()`.

When we encounter a new datatype in our programs, we often follow the procedure of thinking listed above. Such procedure is so common that software engineering people though convenient to provide a specific programming paradigm to represent it, called *Object Oriented* programming. We are now going to rewrite the complex number example using such paradigm.

1.5. Object Oriented Programming

In Object Oriented Programming, we usually

1. Introduce new datatypes by declaring a *class*, named for example `ComplexNumber`
2. Are given a dictionary and define how data is stored in the dictionary (i.e. in fields `real` and `imaginary`)
3. Define a way to *construct* specific *instances* , like `3 + 2i`, `5 + 6i` (instances are also called *objects*)
4. Define some *methods* to operate on the *instances* (like `phase`)
5. Define some special *methods* to customize how Python treats *instances* (for example for displaying them as strings when printing)

Let's now create our first *class*.

7.1.4 2. ComplexNumber class

2.1. Class declaration

A minimal class declaration will at least declare the class name and the `__init__` method:

[18]:
```
class ComplexNumber:

    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary
```

Here we declare to Python that we are starting defining a template for a new *class* called `ComplexNumber`. This template will hold a collection of functions (called methods) that manipulate *instances* of complex numbers (instances are `1.0 + 2.0i`, `3.0 + 4.0i`, ...).

IMPORTANT: Although classes can have any name (i.e. complex_number, complexNumber, ...), by convention you SHOULD use a camel cased name like ComplexNumber, with capital letters as initials and no underscores.

2.2. Constructor `__init__`

With the dictonary model, to create complex numbers remember we defined that small utility function `complex_number`, where inside we were creating the dictionary:

```
def complex_number(real, imaginary):
    d = {}
    d['real'] = real
    d['imaginary'] = imaginary
    return d
```

With classes, to create objects we have instead to define a so-called *constructor method* called `__init__`:

[19]:
```
class ComplexNumber:

    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary
```

`__init__` is a very special method, that has the job to initialize an *instance* of a complex number. It has three important features:

a) it is defined like a function, inside the `ComplexNumber` declaration (as usual, indentation matters!)

b) it always takes as first parameter `self`, which is an instance of a special kind of dictionary that will hold the fields of the complex number. Inside the previous `complex_number` function, we were creating a dictionary `d`. In `__init__` method, the dictionary instead is automatically created by Python and given to us in the form of parameter `self`

c) `__init__` does not return anything: this is different from the previous `complex_number` function where instead we were returning the dictionary `d`.

Later we will explain better these properties. For now, let's just concentrate on the names of things we see in the declaration.

WARNING: There can be only one constructor method per class, and MUST be named `__init__`

WARNING: **init** MUST take at least one parameter, by convention it is usually named `self`

IMPORTANT: `self` is just a name we give to the first parameter. It could be any name our fantasy suggest and the program would behave exactly the same!

If the editor you are using will evidence it in some special color, it is because it is aware of the convention but *not* because `self` is some special Python keyword.

IMPORTANT: In general, any of the `__init__` parameters can have completely arbitrary names, so for example the following code snippet would work exactly the same as the initial definition:

[20]:
```
class ComplexNumber:

    def __init__(donald_duck, mickey_mouse, goofy):
        donald_duck.real = mickey_mouse
        donald_duck.imaginary = goofy
```

Once the `__init__` method is defined, we can create a specific `ComplexNumber` *instance* with a call like this:

[21]:
```
c = ComplexNumber(3.0,5.0)
print(c)
```

```
<__main__.ComplexNumber object at 0x7f0c4c380f60>
```

What happend here?

init 2.2.1) We told Python we want to create a new particular *instance* of the template defined by *class* `ComplexNumber`. As parameters for the instance we indicated `3.0` and `5.0`.

WARNING: to create the instance, we used the name of the class `ComplexNumber` following it by an open round parenthesis and parameters like a function call: `c=ComplexNumber(3.0,5.0)` Writing just: `c = ComplexNumber` would NOT instantiate anything and we would end up messing with the *template* ``*ComplexNumber*``, which is a collection of functions for complex numbers.

init 2.2.2) Python created a new special dictionary for the instance

init 2.2.3) Python passed the special dictionary as first parameter of the method `__init__`, so it will be bound to parameter `self`. As second and third arguments passed *3.0* and *5.0*, which will be bound respectively to parameters `real` and `imaginary`

WARNING: When instantiating an object with a call like c=ComplexNumber(3.0,5.0) you don't need to pass a dictionary as first parameter! Python will implicitly create it and pass it as first parameter to `__init__`

init 2.2.4) In the `__init__` method, the instructions

```
self.real = real
self.imaginary = imaginary
```

first create a key in the dictionary called `real` associating to the key the value of the parameter `real` (in the call is *3.0*). Then the value *5.0* is bound to the key `imaginary`.

IMPORTANT: we said Python provides **init** with a special kind of dictionary as first parameter. One of the reason it is special is that you can access keys using the dot like self.my_key. With ordinary dictionaries you would have to write the brackets like self["my_key"]

IMPORTANT: like with dictionaries, we can arbitrarily choose the name of the keys, and which values to associate to them.

IMPORTANT: In the following, we will often refer to keys of the self dictionary with the terms *field*, and/or *attribute*.

Now one important word of wisdom:

!!!!!! COMMANDMENT 5: YOU SHALL NEVER EVER REASSIGN ``self`` !!!!!!!

Since self is a kind of dictionary, you might be tempted to do like this:

[22]:
```
class EvilComplexNumber:
    def __init__(self, real, imaginary):
        self = {'real':real, 'imaginary':imaginary}
```

but to the outside world this will bring no effect. For example, let's say somebody from outside makes a call like this:

[23]:
```
ce = EvilComplexNumber(3.0, 5.0)
```

At the first attempt of accessing any field, you would get an error because after the initalization `c` will point to the yet untouched `self` created by Python, and not to your dictionary (which at this point will be simply lost):

```
print(ce.real)
```

AttributeError: EvilComplexNumber instance has no attribute 'real'

In general, you *DO NOT* reassign `self` to anything. Here are other example *DON'Ts*:

```
self = ['666']  # self is only supposed to be a sort of dictionary which is passed by
↪Python
self = 6        # self is only supposed to be a sort of dictionary which is passed by
↪Python</p>
```

init 2.2.5) Python automatically returns from `__init__` the special dictionary `self`

> **WARNING**: `__init__` must **NOT** have a `return` statement ! Python will implicitly return `self` !

init 2.2.6) The result of the call (so the special dictionary) is bound to external variable 'c`:

```
c = ComplexNumber(3.0, 5.0)
```

init 2.2.7) You can then start using `c` as any variable

[24]:
```
print(c)
```

```
<__main__.ComplexNumber object at 0x7f0c4c380f60>
```

From the output, you see we have indeed an *instance* of the *class* `ComplexNumber`. To see the difference between *instance* and *class*, you can try printing the *class* instead:

[25]:
```
print(ComplexNumber)
```

```
<class '__main__.ComplexNumber'>
```

IMPORTANT: You can create an infinite number of different *instances* (i.e. `ComplexNumber(1.0, 1.0)`, `ComplexNumber(2.0, 2.0)`, `ComplexNumber(3.0, 3.0)`, ...), but you will have only one *class* definition for them (`ComplexNumber`).

We can now access the fields of the special dictionary by using the dot notation as we were doing with the 'self`:

[26]:
```
print(c.real)
```

```
3.0
```

[27]:
```
print(c.imaginary)
```

```
5.0
```

If we want, we can also change them:

[28]:
```
c.real = 6.0
print(c.real)
```

```
6.0
```

2.3. Defining methods

2.3.1 phase

Let's make our class more interesting by adding the method `phase(self)` to operate on the complex number:

[29]:
```
import unittest
import math

class ComplexNumber:

    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary

    def phase(self):
        """ Returns a float which is the phase (that is, the vector angle) of the␣
↪complex number
```

(continues on next page)

(continued from previous page)

```
            This method is something we introduce by ourselves, according to the␣
↪definition:
            https://en.wikipedia.org/wiki/Complex_number#Absolute_value_and_argument
        """
        return math.atan2(self.imaginary, self.real)
```

The method takes as first parameter `self` which again is a special dictionary. We expect the dictionary to have already been initialized with some values for `real` and `imaginary` fields. We can access them with the dot notation as we did before:

```
return math.atan2(self.imaginary, self.real)
```

How can we call the method on instances of complex numbers? We can access the method name from an instance using the dot notation as we did with other keys:

[30]:
```
c = ComplexNumber(3.0,5.0)
print(c.phase())
```

```
1.0303768265243125
```

What happens here?

By writing `c.phase()` , we call the method `phase(self)` which we just defined. The method expects as first parameter `self` a class instance, but in the call `c.phase()` apparently we don't provide any parameter. Here some magic is going on, and Python implicitly is passing as first parameter the special dictionary bound to `c`. Then it executes the method and returns the desired float.

WARNING: Put round parenthesis in method calls!

When *calling* a method, you MUST put the round parenthesis after the method name like in `c.phase()`! If you just write `c.phase` without parenthesis you will get back an address to the physical location of the method code:

```
>>> c.phase
<bound method ComplexNumber.phase of <__main__.ComplexNumber instance at 0xb465a4cc>
↪>
```

2.3.2 log

We can also define methods that take more than one parameter, and also that create and return `ComplexNumber` instances, like for example the method `log(self, base)`:

[31]:
```
import math

class ComplexNumber:

    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary

    def phase(self):
        """ Returns a float which is the phase (that is, the vector angle) of the␣
↪complex number

            This method is something we introduce by ourselves, according to the␣
↪definition:
```

(continues on next page)

(continued from previous page)

```
            https://en.wikipedia.org/wiki/Complex_number#Absolute_value_and_argument
        """
        return math.atan2(self.imaginary, self.real)

    def log(self, base):
        """ Returns another ComplexNumber which is the logarithm of this complex␣
↪number

            This method is something we introduce by ourselves, according to the␣
↪definition:
            (accomodated for generic base b)
            https://en.wikipedia.org/wiki/Complex_number#Natural_logarithm
        """
        return ComplexNumber(math.log(self.real) / math.log(base), self.phase() /␣
↪math.log(base))
```

WARNING: *ALL* METHODS MUST HAVE AT LEAST ONE PARAMETER, WHICH BY CONVENTION IS NAMED self !

To call `log`, you can do as with `phase` but this time you will need also to pass one parameter for the `base` parameter, in this case we use the exponential `math.e`:

[32]:
```
c = ComplexNumber(3.0, 5.0)
logarithm = c.log(math.e)
```

WARNING: As before for `phase`, notice we didn't pass any dictionary as first parameter! Python will implicitly pass as first argument the instance `c` as `self`, and `math.e` as `base`

[33]:
```
print(logarithm)
```

```
<__main__.ComplexNumber object at 0x7f0c4c39e470>
```

To see if the method worked and we got back we got back a different complex number, we can print the single fields:

[34]:
```
print(logarithm.real)
```

```
1.0986122886681098
```

[35]:
```
print(logarithm.imaginary)
```

```
1.0303768265243125
```

2.3.3 __str__ for printing

As we said, printing is not so informative:

[36]:
```
print(ComplexNumber(3.0, 5.0))
```

```
<__main__.ComplexNumber object at 0x7f0c4c3f53c8>
```

It would be nice to instruct Python to express the number like *"3.0 + 5.0i"* whenever we want to see the `ComplexNumber` represented as a string. How can we do it? Luckily for us, defining the `__str__(self) method` (see bottom of class definition)

> **WARNING**: There are **two** underscores _ before and **two** underscores _ after in `__str__` !

[37]:
```
import math

class ComplexNumber:

    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary

    def phase(self):
        """ Returns a float which is the phase (that is, the vector angle) of the␣
↪complex number

            This method is something we introduce by ourselves, according to the␣
↪definition:
            https://en.wikipedia.org/wiki/Complex_number#Absolute_value_and_argument
        """
        return math.atan2(self.imaginary, self.real)

    def log(self, base):
        """ Returns another ComplexNumber which is the logarithm of this complex␣
↪number

            This method is something we introduce by ourselves, according to the␣
↪definition:
            (accomodated for generic base b)
            https://en.wikipedia.org/wiki/Complex_number#Natural_logarithm
        """
        return ComplexNumber(math.log(self.real) / math.log(base), self.phase() /␣
↪math.log(base))

    def __str__(self):
        return str(self.real) + " + " + str(self.imaginary) + "i"
```

IMPORTANT: all methods starting and ending with a double underscore __ have a special meaning in Python: depending on their name, they override some default behaviour. In this case, with `__str__` we are overriding how Python represents a `ComplexNumber` instance into a string.

WARNING:

Since we are overriding Python default behaviour, it is very important that we follow the specs of the method we are overriding *to the letter*. In our case, the specs for __str__[320] obviously state you MUST return a string. **Do read them!**

[38]:
```
c = ComplexNumber(3.0, 5.0)
```

We can also pretty print the whole complex number. Internally, `print` function will look if the class `ComplexNumber` has defined a method named `__str__`. If so, it will pass to the method the instance `c` as the first argument, which in our methods will end up in the `self` parameter:

[39]:
```
print(c)
```

```
3.0 + 5.0i
```

[40]:
```
print(c.log(2))
```

```
1.5849625007211563 + 1.4865195378735334i
```

Special Python methods are like any other method, so if we wish, we can also call them directly:

[41]:
```
c.__str__()
```

[41]:
```
'3.0 + 5.0i'
```

EXERCISE: There is another method for getting a string representation of a Python object, called `__repr__`. Read carefully __repr__ documentation[321] and implement the method. To try it and see if any difference appear with respect to str, call the standard Python functions `repr` and `str` like this:

```
c = ComplexNumber(3,5)
print(repr(c))
print(str(c))
```

QUESTION: Would `3.0 + 5.0i` be a valid Python expression ? Should we return it with `__repr__`? Read again also __str__ documentation[322]

2.4. ComplexNumber code skeleton

We are now ready to write methods on our own. Open Visual Studio Code (no jupyter in part B !) and proceed editing file `ComplexNumber.py`

To see how to test, try running this in the console, tests should pass (if system doesn't find `python3` write `python`):

```
python3 -m unittest ComplexNumber_test.ComplexNumberTest
```

[320] https://docs.python.org/3/reference/datamodel.html#object.__str__
[321] https://docs.python.org/3/reference/datamodel.html#object.__repr__
[322] https://docs.python.org/3/reference/datamodel.html#object.__str__

2.5. Complex numbers magnitude

The *absolute value* (or *modulus* or *magnitude*) of a complex number $z = x + yi$ is

$$r = |z| = \sqrt{x^2 + y^2}.$$

Implement the `magnitude` method, using this signature:

```
def magnitude(self):
    """ Returns a float which is the magnitude (that is, the absolute value) of the␣
↪complex number

        This method is something we introduce by ourselves, according to the␣
↪definition:
        https://en.wikipedia.org/wiki/Complex_number#Absolute_value_and_argument
    """
    raise Exception("TODO implement me!")
```

To test it, check this test in `MagnitudeTest` class passes (notice the `almost` in `assertAlmostEquals` !!!):

```
def test_01_magnitude(self):
    self.assertAlmostEqual(ComplexNumber(3.0,4.0).magnitude(),5, delta=0.001)
```

To run the test, in the console type:

```
python3 -m unittest ComplexNumber_test.MagnitudeTest
```

2.6. Complex numbers equality

Here we will try to give you a glimpse of some aspects related to Python equality, and trying to respect interfaces when overriding methods. Equality can be a nasty subject, here we will treat it in a simplified form.

First of all, try to execute this command, you should get back `False`

```
[42]: ComplexNumber(1,2) == ComplexNumber(1,2)
```

```
[42]: False
```

How comes we get `False`? The reason is whenever we write `ComplexNumber(1,2)` we are creating a new object in memory. Such object will get assigned a unique address number in memory, and by default equality between class instances is calculated considering only equality among memory addresses. In this case we create one object to the left of the expression and another one to the right. So far we didn't tell Python how to deal with equality for `ComplexNumber` classes, so default equality testing is used by checking memory addresses, which are different - so we get `False`.

To get `True` as we expect, we need to implement `__eq__` special method. This method should tell Python to compare the fields within the objects, and not just the memory address.

REMEMBER: as all methods starting and ending with a double underscore `__`, `__eq__` has a special meaning in Python: depending on their name, they override some default behaviour. In this case, with `__eq__` we are overriding how Python checks equality. Please review `__eq__` documentation[323] before continuing.

QUESTION: What is the return type of `__eq__` ?

Equality [edit]

Two complex numbers are equal if and only if both their real and imaginary parts are equal. In symbols:

$$z_1 = z_2 \;\leftrightarrow\; (\mathrm{Re}(z_1) = \mathrm{Re}(z_2) \wedge \mathrm{Im}(z_1) = \mathrm{Im}(z_2)).$$

- Implement equality for `ComplexNumber` more or less as it was done for `Fraction`

  Use this method signature:

  ```
  def __eq__(self, other):
  ```

  Since `__eq__` is a binary operation, here `self` will represent the object to the left of the ==, and `other` the object to the right.

Use this simple test case to check for equality in class `EqTest`:

```
def test_01_integer_equality(self):
    """
        Note all other tests depend on this test !

        We want also to test the constructor, so in c we set stuff by hand
    """
    c = ComplexNumber(0,0)
    c.real = 1
    c.imaginary = 2
    self.assertEquals(c, ComplexNumber(1,2))
```

To run the test, in the console type:

[323] https://docs.python.org/3/reference/datamodel.html#object.__eq__

```
python3 -m unittest ComplexNumber_test.EqTest
```

- Beware 'equality' is tricky in Python for float numbers! Rule of thumb: when overriding `__eq__`, use 'dumb' equality, two things are the same only if their parts are literally equal
- If instead you need to determine if two objects are similar, define other 'closeness' functions.
- Once done, check again `ComplexNumber(1,2) == ComplexNumber(1,2)` command and see what happens, this time it should give back `True`.

QUESTION: What about `ComplexNumber(1,2) != ComplexNumber(1,2)`? Does it behaves as expected?

- (Non mandatory read) if you are interested in the gory details of equality, see
  - How to Override comparison operators in Python[324]
  - Messing with hashing[325]

2.7. Complex numbers isclose

Complex numbers can be represented as vectors, so intuitively we can determine if a complex number is close to another by checking that the distance between its vector tip and the the other tip is less than a given delta. There are more precise ways to calculate it, but here we prefer keeping the example simple.

Given two complex numbers

$$z_1 = a + bi$$

and

$$z_2 = c + di$$

We can consider them as close if they satisfy this condition:

$$\sqrt{(a-c)^2 + (b-d)^2} < delta$$

- Implement the method in `ComplexNumber` class:

```
def isclose(self, c, delta):
    """ Returns True if the complex number is within a delta distance from complex␣
↪number c.
    """
    raise Exception("TODO Implement me!")
```

Check this test case `IsCloseTest` class pass:

```
def test_01_isclose(self):
    """  Notice we use `assertTrue` because we expect `isclose` to return a `bool`␣
↪value, and
         we also test a case where we expect `False`
    """
    self.assertTrue(ComplexNumber(1.0,1.0).isclose(ComplexNumber(1.0,1.1), 0.2))
    self.assertFalse(ComplexNumber(1.0,1.0).isclose(ComplexNumber(10.0,10.0), 0.2))
```

To run the test, in the console type:

[324] http://jcalderone.livejournal.com/32837.html
[325] http://www.asmeurer.com/blog/posts/what-happens-when-you-mess-with-hashing-in-python/

```
python3 -m unittest ComplexNumber_test.IscloseTest
```

REMEMBER: Equality with `__eq__` and closeness functions like `isclose` are very different things. Equality should check if two objects have the same memory address or, alternatively, if they contain the same things, while closeness functions should check if two objects are similar. You should never use functions like `isclose` inside `__eq__` methods, unless you really know what you're doing.

2.8. Complex numbers addition

Complex numbers are added by separately adding the real and imaginary parts of the summands. That is to say:

$$(a + bi) + (c + di) = (a + c) + (b + d)i.$$

Similarly, subtraction is defined by

$$(a + bi) - (c + di) = (a - c) + (b - d)i.$$

- `a` and `c` correspond to `real`, `b` and `d` correspond to `imaginary`
- implement addition for `ComplexNumber` more or less as it was done for `Fraction` in theory slides
- write some tests as well!

Use this definition:

```
def __add__(self, other):
    raise Exception("TODO implement me!")
```

Check these two tests pass in `AddTest` class:

```
def test_01_add_zero(self):
    self.assertEquals(ComplexNumber(1,2) + ComplexNumber(0,0), ComplexNumber(1,2));

def test_02_add_numbers(self):
    self.assertEquals(ComplexNumber(1,2) + ComplexNumber(3,4), ComplexNumber(4,6));
```

To run the tests, in the console type:

```
python3 -m unittest ComplexNumber_test.AddTest
```

2.9. Adding a scalar

We defined addition among ComplexNumbers, but what about addition among a ComplexNumber and an `int` or a `float`?

Will this work?

```
ComplexNumber(3,4) + 5
```

What about this?

```
ComplexNumber(3,4) + 5.0
```

Try to add the following method to your class, and check if it does work with the scalar:

[43]:
```
    def __add__(self, other):
         # checks other object is instance of the class ComplexNumber
        if isinstance(other, ComplexNumber):
            return ComplexNumber(self.real + other.real,self.imaginary + other.
↪imaginary)

        # else checks the basic type of other is int or float
        elif type(other) is int or type(other) is float:
            return ComplexNumber(self.real + other, self.imaginary)

        # other is of some type we don't know how to process.
        # In this case the Python specs say we MUST return 'NotImplemented'
        else:
            return NotImplemented
```

Hopefully now you have a better add. But what about this? Will this work?

```
5 + ComplexNumber(3,4)
```

Answer: it won't, Python needs further instructions. Usually Python tries to see if the class of the object on left of the expression defines addition for operands *to the right* of it. In this case on the left we have a `float` number, and float numbers don't define any way to deal to the right with your very own `ComplexNumber` class. So as a last resort Python tries to see if your `ComplexNumber` class has defined also a way to deal with operands *to the left* of the `ComplexNumber`, by looking for the method `__radd__` , which means *reverse addition* . Here we implement it :

```
def __radd__(self, other):
    """ Returns the result of expressions like    other + self      """
    if (type(other) is int or type(other) is float):
        return ComplexNumber(self.real + other, self.imaginary)
    else:
        return NotImplemented
```

To check it is working and everything is in order for addition, check these tests in `RaddTest` class pass:

```
def test_01_add_scalar_right(self):
    self.assertEquals(ComplexNumber(1,2) + 3, ComplexNumber(4,2));

def test_02_add_scalar_left(self):
    self.assertEquals(3 + ComplexNumber(1,2), ComplexNumber(4,2));

def test_03_add_negative(self):
    self.assertEquals(ComplexNumber(-1,0) + ComplexNumber(0,-1), ComplexNumber(-1,-
↪1));
```

2.10. Complex numbers multiplication

Multiplication and division [edit]

The multiplication of two complex numbers is defined by the following formula:

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i.$$

In particular, the square of the imaginary unit is −1:

$$i^2 = i \times i = -1.$$

- Implement multiplication for `ComplexNumber`, taking inspiration from previous `__add__` implementation
- Can you extend multiplication to work with scalars (both left and right) as well?

To implement `__mul__`, implement definition into `ComplexNumber` class:

```
def __mul__(self, other):
    raise Exception("TODO Implement me!")
```

and make sure these tests cases pass in `MulTest` class:

```
def test_01_mul_by_zero(self):
    self.assertEquals(ComplexNumber(0,0) * ComplexNumber(1,2), ComplexNumber(0,0));

def test_02_mul_just_real(self):
    self.assertEquals(ComplexNumber(1,0) * ComplexNumber(2,0), ComplexNumber(2,0));

def test_03_mul_just_imaginary(self):
    self.assertEquals(ComplexNumber(0,1) * ComplexNumber(0,2), ComplexNumber(-2,0));

def test_04_mul_scalar_right(self):
    self.assertEquals(ComplexNumber(1,2) * 3, ComplexNumber(3,6));

def test_05_mul_scalar_left(self):
    self.assertEquals(3 * ComplexNumber(1,2), ComplexNumber(3,6));
```

7.1.5 3. MultiSet

You are going to implement a class called `MultiSet`, where you are only given the class skeleton, and you will need to determine which Python basic datastructures like `list`, `set`, `dict` (or combinations thereof) is best suited to actually hold the data.

In math a multiset (or bag) generalizes a set by allowing multiple instances of the multiset's elements.

The multiplicity of an element is the number of instances of the element in a specific multiset.

For example:

- The multiset `a, b` contains only elements `a` and `b`, each having multiplicity 1
- In multiset `a, a, b`, `a` has multiplicity 2 and `b` has multiplicity 1
- In multiset `a, a, a, b, b, b`, `a` and `b` both have multiplicity 3

NOTE: order of insertion does not matter, so `a, a, b` and `a, b, a` are the same multiset, where `a` has multiplicity 2 and `b` has multiplicity 1.

[44]:
```
from multiset_sol import *
```

7.1.6 3.1 `__init__` add and get

Now implement *all* of the following methods: `__init__`, `add` and `get`:

```
def __init__(self):
    """ Initializes the MultiSet as empty."""
    raise Exception("TODO IMPLEMENT ME !!!")

def add(self, el):
    """ Adds one instance of element el to the multiset

        NOTE: MUST work in O(1)
    """
    raise Exception("TODO IMPLEMENT ME !!!")

def get(self, el):
    """ Returns the multiplicity of element el in the multiset.

        If no instance of el is present, return 0.

        NOTE: MUST work in O(1)
    """
    raise Exception("TODO IMPLEMENT ME !!!")
```

Testing

Once done, running this will run only the tests in `AddGetTest` class and hopefully they will pass.

Notice that `multiset_test` is followed by a dot and test class name `.AddGetTest`:

```
python3 -m unittest multiset_test.AddGetTest
```

7.1.7 3.2 `removen`

Implement the following `removen` method:

```
def removen(self, el, n):
    """ Removes n instances of element el from the multiset (that is, reduces el␣
↪multiplicity by n)

        If n is negative, raises ValueError.
        If n represents a multiplicity bigger than the current multiplicity, raises␣
↪LookupError

        NOTE: multiset multiplicities are never negative
        NOTE: MUST work in O(1)
    """
```

Testing: `python3 -m unittest multiset_test.RemovenTest`

7.2 Sorting

7.2.1 Download exercises zip

Browse files online[326]

7.2.2 Introduction

References

- Alberto Montresor Algorithm analysis slides[327]
- Python DS Chapter 2.6: Algorithm analysis[328]

What to do

- unzip exercises in a folder, you should get something like this:

```
sorting
    sorting.ipynb
    selection_sort.py
    selection_sort_test.py
    selection_sort_sol.py
    jupman.py
    sciprog.py
```

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), you will edit the files ending in `.py` files
- Go on reading this notebook, and follow instuctions inside.

7.2.3 List performance

Python lists are generic containers, they are useful in a variety of scenarios but sometimes their perfomance can be disappointing, so it's best to know and avoid potentially expensive operations. Table from the book Chapter 2.6: Lists[329]

[326] https://github.com/DavidLeoni/sciprog-ds/tree/master/sorting
[327] http://disi.unitn.it/~montreso/sp/handouts/B01-analisi.pdf
[328] https://runestone.academy/runestone/books/published/pythonds/AlgorithmAnalysis/toctree.html
[329] http://interactivepython.org/runestone/static/pythonds/AlgorithmAnalysis/Lists.html

Operation	Big-O Efficiency
index []	O(1)
index assignment	O(1)
append	O(1)
pop()	O(1)
pop(i)	O(n)
insert(i,item)	O(n)
del operator	O(n)
iteration	O(n)

contains (in)	O(n)
get slice [x:y]	O(k)
del slice	O(n)
set slice	O(n+k)
reverse	O(n)
concatenate	O(k)
sort	O(n log n)
multiply	O(nk)

Fast or not?

```
x = ["a", "b", "c"]

x[2]
x[2] = "d"
x.append("d")
x.insert(0, "d")
x[3:5]
x.sort()
```

What about `len(x)` ? If you don't know the answer, try googling it!

Sublist iteration performance

`get slice` time complexity is `O(k)`, but what about memory? It's the same!

So if you want to iterate a part of a list, beware of slicing! For example, slicing a list like this can occupy much more memory than necessary:

[2]:
```
x = range(1000)

print([2*y for y in x[100:200]])
```

```
[200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232,␣
↪234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254, 256, 258, 260, 262, 264, 266,
↪ 268, 270, 272, 274, 276, 278, 280, 282, 284, 286, 288, 290, 292, 294, 296, 298,␣
↪300, 302, 304, 306, 308, 310, 312, 314, 316, 318, 320, 322, 324, 326, 328, 330, 332,
↪ 334, 336, 338, 340, 342, 344, 346, 348, 350, 352, 354, 356, 358, 360, 362, 364,␣
↪366, 368, 370, 372, 374, 376, 378, 380, 382, 384, 386, 388, 390, 392, 394, 396, 398]
```

The reason is that, depending on the Python interpreter you have, slicing like `x[100:200]` at loop start can create a *new* list. If we want to explicitly tell Python we just want to iterate through the list, we can use the so called itertools. In particular, the islice method is handy, with it we can rewrite the list comprehension above like this:

[3]:
```
import itertools

print([2*y for y in itertools.islice(x, 100, 200)])
```

```
[200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232,␣
↪234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254, 256, 258, 260, 262, 264, 266,
↪ 268, 270, 272, 274, 276, 278, 280, 282, 284, 286, 288, 290, 292, 294, 296, 298,␣
↪300, 302, 304, 306, 308, 310, 312, 314, 316, 318, 320, 322, 324, 326, 328, 330, 332,
↪ 334, 336, 338, 340, 342, 344, 346, 348, 350, 352, 354, 356, 358, 360, 362, 364,␣
↪366, 368, 370, 372, 374, 376, 378, 380, 382, 384, 386, 388, 390, 392, 394, 396, 398]
```

(continues on next page)

(continued from previous page)

7.2.4 Exercises

7.2.5 1 Selection Sort

We will try to implement Selection Sort on our own. Montresor slides already contain the Python solution, but don't look at them (we will implement a slightly different solution anyway). In this exercises, you will only be allowed to look at this picture:

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$i=0$	7	4	2	1	8	3	5
$i=1$	1	4	2	7	8	3	5
$i=2$	1	2	4	7	8	3	5
$i=3$	1	2	3	7	8	4	5
$i=4$	1	2	3	4	8	7	5
$i=5$	1	2	3	4	5	7	8
$i=6$	1	2	3	4	5	7	8

To start with, open `selection_sort.py` in an editor of your choice.

Now proceed reading.

1.1 Implement `swap`

[4]:
```
def swap(A, i, j):
    """ MODIFIES the array A by swapping the elements at position i and j
    """
    raise Exception("TODO implement me!")
```

In order to succeed in this part of the course, you are strongly invited to *first* think hard about a function, and *then* code! So to start with, pay particular attention at the required inputs and expected outputs of functions. *Before* start coding, answer these questions:

QUESTION 1.1.1: What are the input types of `swap`? In particular

- What is the type of the elements in `A`?
  - Can we have both strings and floats inside `A` ?
- What is the type of `i` and `j` ?

COMMANDMENT 2: You shall also write on paper!

Help yourself by drawing a representation of input array. Staring at the monitor doesn't always work, so help yourself and draw a representation of the state sof the program. Tables, nodes, arrows, all can help figuring out a solution for the problem.

QUESTION 1.1.2: What should be the result of the three prints here? Should the function `swap` *return* something at all ? Try to answer this question before proceeding.

```
A = ['a','b','c']
print(A)
print(swap(A, 0, 2))
print(A)
```

HINT: Remember this:

COMMANDMENT 7: You shall use `return` command only if you see written *return* in the function description!

If there is no `return` in function description, the function is intended to return `None`.

QUESTION 1.1.3: Try to answer this question before proceeding:

- What is the result of the first and second print down here?
- What is the result of the final print if we have arbitrary indeces i and j with $0 \leq i, j \leq 2$?

```
A = ['a','b','c']
swap(A, 0, 2)
print(A)
swap(A, 0, 2)
print(A)
```

QUESTION 1.1.3: Try to answer this question before proceeding:

- What is the result of the first and second print down here?
- What is the result of the final print if we have arbitrary indeces i and j with $0 \leq i, j \leq 2$?

```
A = ['a','b','c']
swap(A, 0, 2)
print(A)
swap(A, 2, 0)
print(A)
```

QUESTION 1.1.4: What is the result of the final print here? Try to answer this question before proceeding:

```
A = ['a','b','c']
swap(A, 1, 1)
print(A)
```

QUESTION 1.1.5:

- In the same file `selection_sort.py` copy at the end the test code at the end of this question.
- Read carefully all the test cases, in particular `test_swap_property` and `test_double_swap`. They show two important properties of the swap function that you should have discovered while ansering the questions before.
  - Why should these tests succeed with implemented code? Make sure to answer.

EXERCISE: implement swap

Proceed implementing the `swap` function

To test the function, run:

```
python3  -m unittest selection_sort_test.SwapTest
```

Notice that:

- In the command above there is no `.py` at the end of `selection_sort_test`
- We are executing the command in the operating system shell, not Python (there must *not* be >>> at the beginning)
- At the end of the filename, there is a dot followed by a test class name `SwapTest`, which means Python will only execute tests contained in `SwapTest`. Of course, in this case those are all the tests we have, but if we add many test classes to our file, it will be useful to able to filter executed tests.
- According to your distribution (i.e. Anaconda), you might need to write `python` instead of `python3`

QUESTION 1.1.6: Read Error kinds[330] section in Testing. Suppose you will be the only one calling `swap`, and you suspect your program somewhere is calling swap with wrong parameters. Which kind of error would that be? Add to `swap` some appropriate *precondition checking*.

1.2 Implement `argmin`

Try to code and test the partial `argmin` pos function:

[5]:
```
def argmin(A, i):
    """ RETURN the *index* of the element in list A which is lesser than or equal
        to all other elements in A that start from index i included

        - MUST execute in O(n) where n is the length of A
    """
    raise Exception("TODO implement me!")
```

QUESTION 1.2.1: What are the input types of `argmin`? In particular

- What could be the type of the elements in `A`?
  - Can we have both strings and floats inside `A` ?
- What is the type of `i` ?
  - What is the range of `i` ?

QUESTION 1.2.2: Should the function `argmin` *return* something ? What would be the result type? Try to answer this question before proceeding.

QUESTION 1.2.3: Look again at the `selection_sort` matrix, and compare it to the `argmin` function definition:

[330] https://sciprog.davidleoni.it/testing/testing.html#Error-kinds

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$i=0$	7	4	2	1	8	3	5
$i=1$	1	4	2	7	8	3	5
$i=2$	1	2	4	7	8	3	5
$i=3$	1	2	3	7	8	4	5
$i=4$	1	2	3	4	8	7	5
$i=5$	1	2	3	4	5	7	8
$i=6$	1	2	3	4	5	7	8

- Can you understand the meaning of orange and white boxes?
- What does the yellow box represent?

QUESTION 1.2.4:

- Draw a matrix like the above for the array `A = ['b','a','c']`, adding the corresponding row and column numbers for `i` and `j`
- What should be the result of the three prints here?

```
A = ['a','b','c']
print(argmin(A,0))
print(argmin(A,1))
print(argmin(A,2))
print(A)
```

EXERCISE 1.2.5: Copy the following test code at the end of the file `selection_sort.py`, and start coding a solution.

To test the function, run:

```
python3  -m unittest selection_sort_test.ArgminTest
```

Notice how now we are appending `.ArgminTest` at the end of the command.

> **Warning:** *Don't* use slices ! Remember their computational complexity, and that in these labs we do care about performances!

1.3: Full `selection_sort`

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$i=0$	7	4	2	1	8	3	5
$i=1$	1	4	2	7	8	3	5
$i=2$	1	2	4	7	8	3	5
$i=3$	1	2	3	7	8	4	5
$i=4$	1	2	3	4	8	7	5
$i=5$	1	2	3	4	5	7	8
$i=6$	1	2	3	4	5	7	8

Let's talks about implementing `selection_sort` function in `selection_sort.py`

[6]:
```
def selection_sort(A):
    """ Sorts the list A in-place in O(n^2) time this ways:
        1. Looks at minimal element in the array [i:n],
           and swaps it with first element.
        2. Repeats step 1, but considering the subarray [i+1:n]

        Remember selection sort has complexity O(n^2) where n is the
        size of the list.
    """

    raise Exception("TODO implement me!")
```

Note: on the book website there is an implementation of the selection sort[331] with a nice animated histogram showing a sorting process. Differently from the slides, instead of selecting the minimal element the algorithm on the book selects the *maximal* element and puts it to the right of the array.

QUESTION 1.3.1:

- What is the expected *return* type? Does it return anything at all?
- What is the meaning of 'Sorts the list A in-place' ?

QUESTION 1.3.2:

- At the beginning, which array indeces are considered?
- At the end, which array indeces are considered ? Is `A[len(A) - 1:len(A)]` ever considered ?

EXERCISE 1.3.3:

Try now to implement `selection_sort` in `selection_sort.py`, using the two previously defined functions `swap` and `argmin`.

[331] http://interactivepython.org/runestone/static/pythonds/SortSearch/TheSelectionSort.html

HINT: If you are having troubles because your selection sort passes wrong arguments to either swap or argmin, feel free to add further assertions to both. They are much more effective than prints !

To test the function, run:

```
python3  -m unittest selection_sort_test.SelectionSortTest
```

7.2.6 2 Insertion sort

Insertion sort is a basic sorting algorithm. This animation[332] gives you an idea of how it works

From the animation, you can see these things are going on:

1. The red square selects one number starting from the leftomost (question: does it actually need to be the leftmost ? Can we save one iteration?). Let's say it starts at position `i`.
2. *While* the number in the red square is lesser then the previous one, it is shifted back one position at a time
3. The red square now selects the number immediately following the previous starting point of the red square, that is, selects position `i + 1`

From the analysis above:

- how many cycles do we need ? One, Two, Three?
- Are they nested?
- Is there one cycle with a fixed number of iterations ? Is there one with an unknown number of iterations?
- What is the worst-case complexity of the algorithm?

As always, if you have troubles finding a generic solution, take a fixed list and manually write down all the steps to do the algorithm. Here we give a sketch:

```
   i=0,1,2,3,4,5
A = [3,8,9,7,6,2]
```

Let's say we have red square at `i=4`

```
i = 4
red =  A[4]    # in red we put the value in A[4] which is 6

               #  0,1,2,3,4,5
               # [3,7,8,9,6,2]  start
A[4] = A[3]    # [3,7,8,9,9,2]
A[3] = A[2]    # [3,7,8,8,9,2]
A[2] = A[1]    # [3,7,7,8,9,2]
A[1] = red     # [3,6,7,8,9,2]  A[1] < red, stop
```

We can generalize A index with a `j`:

```
i = 4
red = A[4]
j = 4
while ...
    A[j] = A[j-1]
    j -= 1

A[j] = red
```

[332] https://github.com/DavidLeoni/sciprog-ds/blob/master/sorting/img/insertion-sort-example.gif

Start editing the file `insertion_sort.py` and implement `insertion_sort` without looking at theory slides.

```
def insertion_sort(A):
    """ Sorts in-place list A with insertion sort
    """
```

7.2.7 3 Merge sort

With merge sort we model lists to ordered as stacks, so it is important to understand how to take elements from the end of a list and how to reverse a list to change its order.

Taking last element

To take last element from a list you may use `[-1]`:

[7]:
```
[9,7,8][-1]
```

[7]:
```
8
```

Reversing a list

REMEMBER: `.reverse()` method MODIFIES the list it is called on and returns `None` !

[8]:
```
lst = [9,7,8]
lst.reverse()
```

Notice how above Jupyter did not show anything, because implicitly the result of the call was `None`. Still, we have an effect, `lst` is now reversed:

[9]:
```
lst
```

[9]:
```
[8, 7, 9]
```

If you want to reversed version of a list without actually changing it, you can use `reversed` function:

[10]:
```
lst = [9,7,8]
reversed(lst)
```

[10]:
```
<list_reverseiterator at 0x7f08d439a860>
```

The returned value is an iterator, so something which is able to produce a reversed version of the list but it is still not a list. If you actually want to get back a list, you need to explicitly cast it to `list`:

[11]:
```
lst = [9,7,8]
list(reversed(lst))
```

[11]:
```
[8, 7, 9]
```

Notice `lst` itself was not changed:

[12]:
```
lst
```

[12]:
```
[9, 7, 8]
```

Removing last element with .pop()

To remove an element, you can use `.pop()` method, which does two things:

1. if not given any argument, removes the **last** element **in** $O(1)$ **time**
2. returns it to the caller of the method, so for example we can conveniently store it in a variable

[13]:
```
A = [9,7,8]
x = A.pop()
```

[14]:
```
print(A)
```

```
[9, 7]
```

[15]:
```
print(x)
```

```
8
```

> **WARNING**: internal deletion is expensive !
>
> If you pay attention to performance (and in this course part you are), whenever you have to remove elements from a Python list be very careful about the complexity! Removal at the end is a very fast O(1), but internal removal is O(n) !

Costly internal del

You can remove an internal element with `del`

NOTE: `del` returns `None`

[16]:
```
lst = [9,5,6,7]
del lst[2]      # internal delete is O(n)
```

[17]:
```
lst
```

[17]:
```
[9, 5, 7]
```

Costly internal pop

You can remove an internal element with `pop(i)`

[18]:
```
lst = [9,5,6,7]

lst.pop(2)  # internal pop is O(n)
```

[18]:
```
6
```

[19]:
```
lst
```

[19]:
```
[9, 5, 7]
```

3.1 merge 1

Start editing `merge_sort.py`

`merge1` takes two *already ordered* lists of size `n` and `m` and return a new one made with the elements of both in $O(n+m)$ time. For example:

[20]:
```
from merge_sort_sol import *

merge1([3,6,9,13,15], [2,4,8,9])
```

[20]:
```
[2, 3, 4, 6, 8, 9, 9, 13, 15]
```

To implement it, keep comparing the last elements of the two lists, and at each round append the greatest in a temporary list, which you shall return at the end of the function (remember to reverse it!).

Example:

If we imagine the numbers as ordered card decks, we can picture them like this:

```
                                2                 15
                                4                 13
                                4                 10
                                6                 9
          15                    8                 8
          13        10          9                 6
          9         8           10                4
          6         4           13                4
          4         2           15                2

          A         B           TMP             RESULT

As Python lists, they would look like:

A=[4,6,9,13,15]
B=[2,4,8,10]
TMP=[15,13,10,9,8,6,4,4,2]
RESULT=[2,4,4,6,8,9,10,13,15]
```

The algorithm would:

1. compare 15 and 10, pop 15 and put it in TMP
2. compare 13 and 10, pop 13 and put it in TMP
3. compare 9 and 10, pop 10 and put it in TMP
4. compare 9 and 8, pop 9 and put it in TMP
5. etc …
6. finally return a reversed TMP

It remains to decide what to do when one of the two lists remains empty, but this is up to you.

To test:

```
python3 -m unittest merge_sort_test.Merge1Test
```

3.2 merge2

`merge2` takes A and B as two ordered lists (from smallest to greatest) of (possibly negative) integers. Lists are of size n and m respectively, and RETURN a NEW list composed of the items in A and B ordered from smallest to greatest

- MUST RUN IN O(m+n)
- in this version, do **NOT** use `.pop()` on input lists to reduce their size. Instead, **use indeces to track at which point you are**, starting at zero and putting **minimal** elements in result list, so this time you don't even need a temporary list.

```
  8                           15
  7                           13
  6                           10
  5                           9
  4        15                 8
  3        13       10        6
  2        9        8         4
  1        6        4         4
  0        4        2         2

index      A        B         RESULT
```

Sketch:

0. set i=0 (left index) and j=0 (right index)
1. compare 4 and 2, put 2 in RESULT, set i=0, j=1
2. compare 4 and 4, put 4 in RESULT, set i=1, j=1
3. compare 6 and 4, put 4 in RESULT, set i=1, j=2
4. compare 6 and 8, put 6 in RESULT, set i=2, j=2
5. etc …
6. finally return RESULT

To test:

```
python3 -m unittest merge_sort_test.Merge2Test
```

7.2.8 4 quick sort

Quick sort is a widely used sorting algorithm and in this exercise you will implement it following the pseudo code.

IMPORTANT: Array `A` in the pseudo code has indexes starting from zero *included*

IMPORTANT: The functions pivot and quicksort operate an a subarray that goes from indeces first included and last **included** !!!

Start editing the file `quick_sort.py`:

4.1 pivot

Try look at this pseudocode and implement `pivot` method.

IMPORTANT: If something goes wrong (it will), find the problem using the debugger !

int pivot (**int**[] A, **int** $first$, **int** $last$)

int $p \leftarrow A[first]$
int $j \leftarrow first$
for $i \leftarrow first + 1$ **to** $last$ **do**
  if $A[i] < p$ **then**
    $j \leftarrow j + 1$
    swap(A, i, j)
$A[first] \leftarrow A[j]$
$A[j] \leftarrow p$
return j

```
def pivot(A, first, last):
    """ MODIFIES in-place the slice of the array A with indeces between first included
        and last **included**. RETURN the new pivot index.

    """
    raise Exception("TODO IMPLEMENT ME!")
```

You can run tests only for `pivot` with this command:

```
python3 -m unittest quick_sort_test.PivotTest
```

4.2 quicksort and qs

Implement `quicksort` and `qs` method:

QuickSort(**int**[] A, **int** $first$, **int** $last$)

if $first < last$ **then**
  int $j \leftarrow$ pivot$(A, first, last)$
  QuickSort$(A, first, j - 1)$
  QuickSort$(A, j + 1, last)$

```
def quicksort(A, first, last):
    """
        Sorts in-place the slice of the array A with indeces between
        first included and last included.
    """
    raise Exception("TODO IMPLEMENT ME !")

def qs(A):
    """
        Sorts in-place the array A by calling quicksort function on the
        full array.
    """
    raise Exception("TODO IMPLEMENT ME !")
```

You can run tests only for both `quicksort` and `qs` with this command:

```
python3 -m unittest quick_sort_test.QuicksortTest
```

7.2.9 5. chaining

You will be doing exercises about chainable lists, using plain old Python lists. This time we don't actually care about sorting, we just want to detect duplicates and chain sequences fast.

Start editing the file `exerciseB2.py` and read the following.

5.1 has_duplicates

Implement the function `has_duplicates`

```
def has_duplicates(external_list):
    """
        Returns True if internal lists inside external_list contain duplicates,
        False otherwise. For more info see exam and tests.

        INPUT: a list of list of strings, possibily containing repetitions, like:

            [
                ['ab', 'c', 'de'],
                ['v', 'a'],
                ['c', 'de', 'b']
            ]

        OUTPUT: Boolean  (in the example above it would be True)

    """
```

- **MUST RUN IN** $O(m * n)$, where m is the number of internal lists and n is the length of the longest internal list (just to calculate complexity think about the scenario where all lists have equal size)
- **HINT**: Given the above constraint, whenever you find an item, you cannot start another for loop to check if the item exists elsewhere - that would cost around $O(m^2 * n)$. Instead, you need to keep track of found items with some other data structure of your choice, which must allow fast read and writes.

Testing: `python3 -m unittest chains_test.TestHasDuplicates`

B.2.2 chain

Implement the function `chain`:

```
def chain(external_list):
    """
        Takes a list of list of strings and return a list containing all the strings
        from external_list in sequence, joined by the ending and starting strings
        of the internal lists. For more info see exam and tests.

        INPUT: a list of list of strings , like:

                [
                    ['ab', 'c', 'de'],
                    ['gh', 'i'],
                    ['de', 'f', 'gh']
                ]

        OUTPUT: a list of strings, like   ['ab', 'c', 'de', 'f', 'gh', 'i']
```

It is assumed that

- `external_list` always contains at least one internal list
- internal lists always contain at least two strings
- no string is duplicated among all internal lists

Output sequence is constructed as follows:

- it starts will all the items from the first internal list
- successive items are taken from an internal list which starts with a string equal to the previous taken internal list last string
- sequence must not contain repetitions (so joint strings are taken only once).
- *all* internal lists must be used. If this is not possible (because there are no joint strings), raise `ValueError`

Be careful that:

- **MUST BE WRITTEN WITH STANDARD PYTHON FUNCTIONS**
- **MUST RUN IN** $O(m * n)$, where m is the number of internal lists and n is the length of the longest internal list (just to calculate complexity think about the scenario where all lists have equal size)
- **HINT**: Given the above constraint, whenever you find a string, you cannot start another for loop to check if the string exists elsewhere (that would likely introduce a quadratic m^2 factor) Instead, you need to first keep track of both starting strings and the list they are contained within using some other data structure of your choice, which must allow fast read and writes.
- if possible avoid slicing (which doubles memory usage) and use `itertools.islice` instead

Testing: `python3 -m unittest chains_test.TestChain`

7.2.10 6 SwapArray

> **NOTE: This exercise was given at an exam. Solving it could have been quite easy, if students had just read** the book[333] (which is available when doing the exam)!
>
> Interpret it as a warning that reading these worksheets alone is *not* enough to pass the exam.

You are given a class `SwapArray` that models an array where the only modification you can do is to swap an element with the successive one.

[21]:
```
from swap_array_sol import *
```

To create a `SwapArray`, just call it passing a python list:

[22]:
```
sarr = SwapArray([7,8,6])
print(sarr)
```

```
SwapArray: [7, 8, 6]
```

Then you can query in $O(1)$ it by calling `get()` and `get_last()`

[23]:
```
sarr.get(0)
```

[23]:
```
7
```

[24]:
```
sarr.get(1)
```

[24]:
```
8
```

[25]:
```
sarr.get_last()
```

[25]:
```
6
```

You can know the size in $O(1)$ with `size()` method:

[26]:
```
sarr.size()
```

[26]:
```
3
```

As we said, the only modification you can do to the internal array is to call `swap_next` method:

```
def swap_next(self, i):
""" Swaps the elements at indeces i and i + 1

            If index is negative or greater or equal of the last index, raises
            an IndexError

        """
```

For example:

[27]:
```
sarr = SwapArray([7,8,6,3])
print(sarr)
```

```
SwapArray: [7, 8, 6, 3]
```

[333] https://runestone.academy/runestone/static/pythonds/SortSearch/sorting.html

[28]:
```
sarr.swap_next(2)
print(sarr)
```

```
SwapArray: [7, 8, 3, 6]
```

[29]:
```
sarr.swap_next(0)
print(sarr)
```

```
SwapArray: [8, 7, 3, 6]
```

Now start editing the file `swap_array.py`:

6.1 is_sorted

Implement the `is_sorted` function, which is a function *external* to the class `SwapArray`:

```
def is_sorted(sarr):
    """ Returns True if the provided SwapArray sarr is sorted, False otherwise

        NOTE: Here you are a user of SwapArray, so you *MUST NOT* access
              directly the field _arr.
        NOTE: MUST run in O(n) where n is the length of the array
    """
    raise Exception("TODO IMPLEMENT ME !")
```

Once done, running this will run only the tests in `IsSortedTest` class and hopefully they will pass.

```
python3 -m unittest swap_array_test.IsSortedTest
```

Example usage:

[30]:
```
is_sorted(SwapArray([8,5,6]))
```

[30]:
```
False
```

[31]:
```
is_sorted(SwapArray([5,6,6,8]))
```

[31]:
```
True
```

6.2 max_to_right

Implement `max_to_right` function, which is a function *external* to the class `SwapArray`. There are two ways to implement it, try to minimize the reads from the SwapArray.

```
def max_to_right(sarr,i):
    """ Modifies the provided SwapArray sarr so that its biggest element
        in the subarray from 0 to i is moved at index i.
        Elements *after* i are *not* considered.

        The order in which the other elements will be after a call
        to this function is left unspecified (so it could be any).

        NOTE: Here you are a user of SwapArray, so you *MUST NOT* access
              directly the field _arr. To do changes, you can only use
              the method swap_next(self, i).
```

(continues on next page)

(continued from previous page)

```
        NOTE: does *not* return anything!
        NOTE: MUST run in O(n) where n is the length of the array

    """
```

** Testing **: `python3 -m unittest swap_array_test.MaxToRightTest`

Example usage:

[32]:
```
sarr = SwapArray([7, 9, 6, 5, 8])
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[33]:
```
max_to_right(sarr,4)  # 4 is an *index*
print(sarr)
```

```
SwapArray: [7, 6, 5, 8, 9]
```

[34]:
```
sarr = SwapArray([7, 9, 6, 5, 8])
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[35]:
```
max_to_right(sarr,3)
print(sarr)
```

```
SwapArray: [7, 6, 5, 9, 8]
```

[36]:
```
sarr = SwapArray([7, 9, 6, 5, 8])
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[37]:
```
max_to_right(sarr,1)
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[38]:
```
sarr = SwapArray([7, 9, 6, 5, 8])
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[39]:
```
max_to_right(sarr,0)   # changes nothing
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

6.6 swapsort

When you know how to push a maximum element to the rightmost position of an array, you almost have a sorting algorithm. So now you can try to implement `swapsort` function, taking inspiration from `max_to_right`. Note `swapsort` is a function *external* to the class `SwapArray`:

```
def swapsort(sarr):
    """ Sorts in-place provided SwapArray.

        NOTE: Here you are a user of SwapArray, so you *MUST NOT* access
              directly the field _arr. To do changes, you can only use
              the method swap_next(self, i).
        NOTE: does *not* return anything!
        NOTE: MUST execute in O(n^2), where n is the length of the array
    """

    raise Exception("TODO IMPLEMENT ME !")
```

You can run tests only for `swapsort` with this command:

```
python3 -m unittest swap_array_test.SwapSortTest
```

Example usage:

[40]:
```
sar = SwapArray([8,4,2,4,2,7,3])
```

[41]:
```
swapsort(sar)
```

[42]:
```
print(sar)
```

```
SwapArray: [2, 2, 3, 4, 4, 7, 8]
```

[ ]:
```

```

7.3 Linked lists

7.3.1 Download exercises zip

Browse files online[334]

7.3.2 0 Introduction

In these exercises, you will be implementing several versions of a `LinkedList`, improving its performances with each new version.

[334] https://github.com/DavidLeoni/sciprog-ds/tree/master/linked-lists

References

- theory slides[335](Monodirectional list)
- LinkedList Abstract Data Type[336] on the book
- Implementing LinkedListLinkedLists[337] on the book

NOTE: What the book calls `UnorderedList`, in this lab is just called `LinkedList`. May look confusing, but in the wild you will never find code called `UnorderedList` so let's get rid of the weird name right now!

What to do

- unzip exercises in a folder, you should get something like this:

```
linked-lists
    linked-lists.ipynb
    linked_list_test.py
    linked_list.py
    linked_list_sol.py
    linked_list_v2_sol.py
    linked_list_v2_test_sol.py
    linked_list_v3_sol.py
    linked_list_v3_test_sol.py
    jupman.py
    sciprog.py
```

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), you will edit the files ending in `.py` files
- Go on reading this notebook, and follow instuctions inside.

0.1 Initialization

A `LinkedList` for us is a linked list starting with a pointer called *head* that points to the first `Node` (if the list is empty the pointer points to `None`). Think of the list as a chain where each `Node` can contain some data retriavable with `Node.get_data()` method and you can access one `Node` at a time by calling the method `Node.get_next()` on each node.

Let's see how a LinkedList should behave:

[2]:
```
from linked_list_sol import *
```

[3]:
```
ll = LinkedList()
```

At the beginning the `LinkedList` is empty:

[4]:
```
print(ll)
```

```
LinkedList:
```

NOTE: `print` calls `__str__` method, which in our implementation was overridden to produce a nice string you've just seen. Still, we did not override `__repr__` method which is the default one used by Jupyter when displaying on object without using `print`, so if you omit it you won't get nice display:

[335] https://sciproalgo2019.readthedocs.io/slides/Lecture4.pdf
[336] http://interactivepython.org/runestone/static/pythonds/BasicDS/TheUnorderedListAbstractDataType.html
[337] http://interactivepython.org/runestone/static/pythonds/BasicDS/ImplementinganUnorderedListLinkedLists.html

[5]:
```
ll
```

[5]:
```
<linked_list_sol.LinkedList at 0x7f19247363c8>
```

0.2 Growing

Main way to grow a `LinkedList` is by using the `.add` method, which executes in costant time $O(1)$:

[6]:
```
ll.add('a')
```

Internally, each time you call `.add` a new `Node` object is created which will hold the actual data that you are passing. In this implementation, users of the class are supposed to never get instances of `Node`, they will just be able to see the actual data contained in the `Node`s:

[7]:
```
print(ll)
```

```
LinkedList: a
```

Notice that `.add` actually inserts nodes *at the beginning* :

[8]:
```
ll.add('b')
```

[9]:
```
print(ll)
```

```
LinkedList: b,a
```

[10]:
```
ll.add('c')
```

[11]:
```
print(ll)
```

```
LinkedList: c,b,a
```

Our basic `LinkedList` instance will only hold a pointer to the *first* `Node` of the chain (such pointer is called `_next`). When you add an element:

1. a new Node is created
2. provided data is stored inside new node
3. the new node `_next` field is set to point to current first Node
4. the new node becomes the first node of the `LinkedList`, by setting `LinkedList._next` to new node

0.3 Visiting

Any method that needs to visit the `LinkedList` will have to start from the first Node pointed by `LinkedList._next` and then follow the chain of `_next` links from one Node to the next one. This is why the data structure is called 'linked'. While insertion at the beginning is very fast, retrieving an element at arbitrary position requires a linear scan which in worst case costs $O(n)$.

7.3.3 1 v1: a slow LinkedList

Implement the missing methods in `linked_list.py`, in the order they are presented in the skeleton. **Before implementing, read carefully all this point 1) and all its subsections (1.a,b and c)**

1.a) Testing

You will have two files to look at, the code in `linked_list.py` and the test code in a separate `linked_list_test.py` file:

- `linked_list.py`
- `linked_list_test.py`

You can run tests with this shell command:

```
python3 -m unittest linked_list_test
```

Let's look inside the first lines of `linked_list_test.py` code, you will see a structure like this:

```
from linked_list import *
import unittest

class LinkedListTest(unittest.TestCase):

    def myAssert(self, linked_list, python_list):
        #####  etc #####


class AddTest(LinkedListTest):

    def test_01_init(self):
        #####  etc #####

    def test_04_add(self):
        #####  etc #####

class SizeTest(LinkedListTest):
    #####  etc  #####
```

Note:

- the test automatically imports everything from first module `linked_list`, so when you run the test, it automatically loads the file you will be working on.) :

```
from linked_list import *
```

- there is a base class for testing called `LinkedListTest`
- there are many classes for testing individual methods, each class inherits from `LinkedListTest`
- You will be writing several versions of the linked list. For the first one, you won't need `myAssert`
- This time there is not much Python code to find around, you should rely solely on theory from the slides and book, method definitions and your intuition

1.b) Differences with the book

- We don't assume the list has all different values
- We used more pythonic names[338] for properties and methods, so for example private attribute `Node.data` was renamed to `Node._data` and accessor method `Node.getData()` was renamed to `Node.get_data()`. There are nicer ways to handle these kind of getters/setters pairs called 'properties' but we won't address them here.
- In boundary cases like removing a non-existing element we prefer to raise an `LookupError` with the command

```
raise LookupError("Some error occurred!")
```

In general, this is the behaviour you also find in regular Python lists.

1.c) Please remember...

WARNING: Methods of the class `LinkedList` are supposed to *never* return instances of `Node`. If you see them returned in the tests, then you are making some mistake. Users of `LinkedList` are should only be able to get access to items inside the Node `data` fields.

WARNING: Do *not* use a Python list to hold data inside the data structure. Differently from the `CappedStack` exercise, here you can only use `Node` class. Each `Node` in the `_data` field can hold only one element which is provided by the user of the class, and we don't care about the type of the value the user gives us (so it can be an `int`, a `float`, a `string`, or even a Python list !)

COMMANDMENT 2: You shall also draw lists on paper, helps a lot avoiding mistakes

COMMANDMENT 5: You shall never ever reassign ``self``:

Never ever write horrors such as:

```
class MyClass
    def my_method(self, x, y):
        self = {a:666}  # since self is a kind of dictionary, you might be tempted to␣
↪do like this
                        # but to the outside world this will bring no effect.
                        # For example, let's say somebody from outside makes a call␣
↪like this:
                        #    mc = MyClass()
                        #    mc.my_method()
                        # after the call mc will not point to {a:666}
        self = ['666']  # self is only supposed to be a sort of dictionary and passed␣
↪from outside
        self = 6        # self is only supposed to be a sort of dictionary and passed␣
↪from outside
```

[338] https://www.python.org/dev/peps/pep-0008/#id45

COMMANDMENT 7: You shall use `return` command only if you see written *return* in the 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.

7.3.4 2 v2 faster size

2.1 Save a copy of your work

You already wrote a lot of code, and you don't want to lose it, right? Since we are going to make many modifications, when you reach a point when the code does something useful, it is good practice to save a copy of what you have done somewhere, so if you later screw up something, you can always restore the copy.

- Copy the whole folder `linked-lists` in a new folder `linked-lists-v1`
- Add also in the copied folder a separate `README.txt` file, writing inside the version (like `1.0`), the date, and a description of the main features you implemented (for example "Simple linked list, not particularly performant").
- Backing up the work is a form of the so-called *versioning* : there are much better ways to do it (like using git[339]) but we don't address them here.

WARNING: DO NOT SKIP THIS STEP!

No matter how smart you are, you *will* fail, and a backup may be the only way out.

WARNING: HAVE YOU READ WHAT I JUST WROTE ????

Just. Copy. The. Folder.

2.2. Improve size

Once you saved your precious work in the copy folder `linked-lists-v1`, you can now more freely improve the current folder `linked-lists`, being sure your previous efforts are not going to get lost!

As a first step, in `linked-lists/linked_list.py` implement a `size()` method that works in `O(1)`. To make this work without going through the whole list each time, we will need a new `_size` field that keeps track of the size. When the list is mutated with methods like `add`, `append`, etc you will also need to update the `_size` field accordingly. Proceed like this:

2.2.1) add a new field `_size` in the class constructor and initialize it to zero

2.2.2) modify the `size()` method to just return the `_size` field.

2.2.3) The data structure starts to be complex, and we need better testing. If you look at the tests, very often there are lines of code like `self.assertEquals(to_py(ul), ['a', 'b'])` in the `test_add` method:

```
def test_add(self):
    ul = LinkedList()
    self.myAssert(ul, [])
    ul.add('b')
```

(continues on next page)

[339] https://git-scm.com

(continued from previous page)

```
    self.assertEquals(to_py(ul), ['b'])
    ul.add('a')
    self.assertEquals(to_py(ul), ['a', 'b'])
```

Last line checks our linked list `ul` contains a sequence of linked nodes that once transformed to a python list actually equals `['a', 'b']`. Since in the new implementation we are going to mutate `_size` field a lot, it could be smart to also check that `ul.size()` equals `len(["a", "b"])`. Repeating this check in every test method could be quite verbose. Instead, we can do a smarter thing, and develop in the `LinkedListTest` class a new assertion method on our own:

If you noticed, there is a method `myAssert` in `LinkedListTest` class (in the current `linked-lists/linked_list_test.py` file) which we never used so far, which performs a more thourough check:

```
class LinkedListTest(unittest.TestCase):

    def myAssert(self, linked_list, python_list):
        """ Checks provided linked_list can be represented as the given python_list.␣
↪Since v2.
        """
        self.assertEquals(to_py(linked_list), python_list)
        # check this new invariant about the size
        self.assertEquals(linked_list.size(), len(python_list))
```

WARNING: method `myAssert` must *not* start with `test`, otherwise `unittest` will run it as a test!

2.3.4) Now, how to use this powerful new `myAssert` method? In the test class, just replace every occurence of

```
self.assertEquals(to_py(ul), ['a', 'b'])
```

into calls like this:

```
self.myAssert(ul, ['a', 'b'])
```

WARNING: Notice the `to_py( )` enclosing `ul` is gone.

2.3.5) Actually update `_size` in the various methods where data is mutated, like `add`, `insert`, etc.

2.3.6) Run the tests and hope for the best ;-)

```
python3 -m unittest linked_list_test
```

7.3.5 3 v3 Faster append

We are now better equipped to make further improvements. Once you're done implementing the above and made sure everything works, you can implement an `append` method that works in $O(1)$ by adding an additional pointer in the data structure that always point at the last node. To further exploit the pointer, you can also add a fast `last(self)` method that returns the last value in the list. Proceed like this:

3.1 Save a copy of your work

- Copy the whole folder `linked-lists` in a new folder `linked-lists-v2`
- Add also in the copied folder a separate `README.txt` file, writing inside the version (like `2.0`), the date, and a description of the main features you implemented (for example "Simple linked list, not particularly performant").

WARNING: DO NOT SKIP THIS STEP!

3.2 add `_last` field

Work on `linked_list.py` and simply add an additional pointer called `_last` in the constructor.

3.3 add method skeleton

Copy this method `last` into the class. Just copy it, don't implement it for now.

```
def last(self):
    """ Returns the last element in the list, in O(1).

        - If list is empty, raises a ValueError. Since v3.
    """
    raise ValueError("TODO implement me!")
```

3.4 test driven development

Let's do some so-called *test driven development*, that is, first we write the tests, then we write the implementation.

WARNING: During the exam you *may* be asked to write tests, so don't skip writing them now !!

3.4.1 LastTest

Create a class `LastTest` which inherits from `LinkedListTest`, and add this method Implement a test for `last()` method, by adding this to `LinkedListTest` class:

```
def test_01_last(self):
    raise Exception("TODO IMPLEMENT ME !")
```

In the method, create a list and add elements using only calls to `add` method and checks using the `myAssert` method. When done, ask your instructor if the test is correct (or look at the proposed solution), it is important you get it right otherwise you won't be able to properly test your code.

3.4.2 improve myAssert

You already have a test for the `append()` method, but, how can you be sure the `_last` pointer is updated correctly throughout the code? When you implemented the fast `size()` method you wrote some invariant in the `myAssert` method. We can do the same this time, too. Find the invariant and add the corresponding check to the `myAssert` method. When done, ask your instructor if the invariant is correct (or look at the proposed solution): it is important you get it right otherwise you won't be able to properly test your code.

3.5 update methods that mutate the LinkedList

Update the methods that mutate the data structure (`add`, `insert`, `remove` ...) so they keep `_last` pointed to last element. If the list is empty, `_last` will point to `None`. Take particular care of corner cases such as empty list and one element list.

3.6 Run tests

Cross your fingers and run the tests!

```
python3 -m unittest linked_list_test
```

7.3.6 4 v4 Go bidirectional

Our list so far has links that allow us to traverse it fast in one direction. But what if we want fast traversal in the reverse direction, from last to first element? What if we want a `pop()` that works in $O(1)$? To speed up these operations we could add backward links to each `Node`. Note no solution is provided for this part (yet).

Proceed in the following way:

4.1 Save your work

Once you're done with previous points, save the version you have in a folder `linked-list-v3` somewhere adding in the `README.txt` comments about the improvements done so far, the version number (like 3.0) and the date. Then start working on a new copy.

4.2 Node backlinks

In `Node` class, add backlinks by adding the attribute `_prev` and methods `get_prev(self)` and `set_prev(self, pointer)`.

4.3 Better str

Improve `__str__` method so it shows presence or absence of links, along with the size of the list (note you might need to adapt the test for str method):

- `next` pointers presence must be represented with > character , absence with `*` character. They must be put after the item representation.
- `prev` pointers presence must be represented with < character , absence with `*` character. They must be put befor the item representation.

For example, for the list `['a','b','c']`, you would have the following representation:

```
LinkedList(size=3):*a><b><c*
```

As a special case for empty list you should print the following:

```
LinkedList(size=0):**
```

Other examples of proper lists, with 3, 2, and 1 element can be:

```
LinkedList(size=3):*a><b><c*
LinkedList(size=2):*a><b*
LinkedList(size=1):*a*
```

This new `__str__` method should help you to spot broken lists like the following, were some pointers are not correct:

```
Broken list, all prev pointers are missing:
LinkedList(size=3):*a>*b>*c*

Broken list, size = 3 but shows only one element with next pointer set to None:
LinkedList(size=3):*a*

Broken list, first backward pointer points to something other than None
LinkedList(size=3):<a>*b><c*
```

4.4 Modify add

Update the `LinkedList add` method to take into account you now have backlinks. Take particular care for the boundary cases when the list is empty, has one element, or for nodes at the head and at the tail of the list.

4.5 Add to_python_reversed

Implement `to_python_reversed` method with a linear scan by using the newly added backlinks:

```
def to_python_reversed(self):
    """ Returns a regular Python list with the elements in reverse order,
        from last to first. Since v3. """
    raise Exception("TODO implement me")
```

Add also this test, and make sure it pass:

```
def test_to_python_reversed(self):
    ul = LinkedList()
    ul.add('c')
    ul.add('b')
    ul.add('a')
    pr = to_py(ul)
    pr.reverse()  # we are reversing pr with Python's 'reverse()' method
    self.assertEquals(pr, ul.to_python_reversed())
```

4.6 Add invariant

By using the method `to_python_reversed()`, add a new invariant to the `myAssert` method. If implemented correctly, this will surely spot a lot of possible errors in the code.

4.7 Modify other methods

Modify all other methods that mutate the data structure (`insert`, `remove`, etc) so that they update the backward links properly.

4.8 Run the tests

If you wrote meaningful tests and all pass, congrats!

7.3.7 5 EqList

Open file `eqlist.py` , which is a simple linked list, and start editing the following methods.

5.1 eq

Implement the method `__eq__` (with TWO underscores before and TWO underscores after 'eq') !:

```
def __eq__(self, other):
    """ Returns True if self is equal to other, that is, if all the data elements in␣
↪the respective
        nodes are the same. Otherwise, return False.

        NOTE: compares the *data* in the nodes, NOT the nodes themselves !
    """
```

Testing: `python -m unittest eqlist_test.EqTest`

5.2 remsub

Implement the method `remsub`:

```
def remsub(self, rem):
    """ Removes the first elements found in this LinkedList that match subsequence rem
        Parameter rem is the subsequence to eliminate, which is also a LinkedList.

        Examples:
            aabca  remsub ac  =  aba
            aabca  remsub cxa =  aaba  # when we find a never matching character in␣
↪rem like 'x' here,
                                         the rest of rem after 'x' is not considered.
            aabca  remsub ba  =  aac
            aabca  remsub a   =  abca
            abcbab remsub bb  =  acab
    """
```

Testing: `python3 -m unittest eqlist_test.RemsubTest`

7.3.8 6 Cloning

Start editing the file `cloning.py`, which contains a simplified LinkedList.

6.1 rev

Implement the method `rev(self)` that you find in the skeleton and check provided tests pass.

Testing: `python3 -m unittest cloning_test.RevTest`

6.2 clone

Implement the method `clone(self)` that you find in the skeleton and check provided tests pass.

Testing: `python3 -m unittest cloning_test.CloneTest`

7.3.9 7 More exercises

Start editing the file `more.py`, which contains a simplified LinkedList.

7.1 occurrences

Implement this method:

```
def occurrences(self, item):
    """
        Returns the number of occurrences of item in the list.

        - MUST execute in O(n) where 'n' is the length of the list.
    """
```

Testing: `python3 -m unittest more_test.CloneTest`

**Examples: **

[17]:
```
from more_sol import *

ul = LinkedList()
ul.add('a')
ul.add('c')
ul.add('b')
ul.add('a')
print(ul)
```

```
LinkedList: a,b,c,a
```

[18]:
```
print(ul.occurrences('a'))
```

```
2
```

[19]:
```
print(ul.occurrences('c'))
```

```
1
```

[20]:
```
print(ul.occurrences('z'))
```

```
0
```

7.2 shrink

Implement this method in `LinkedList` class:

```
def shrink(self):
    """
        Removes from this LinkedList all nodes at odd indeces (1, 3, 5, ...),
        supposing that the first node has index zero, the second node
        has index one, and so on.

        So if the LinkedList is
            'a','b','c','d','e'
        a call to shrink will transform the UnorderedList into
            'a','c','e'

        - MUST execute in O(n) where 'n' is the length of the list.
        - Does *not* return anything.
    """
    raise Exception("TODO IMPLEMENT ME!")
```

Testing: `python3 -m unittest more_test.ShrinkTest`

[21]:
```
ul = LinkedList()
ul.add('e')
ul.add('d')
ul.add('c')
ul.add('b')
ul.add('a')
print(ul)
```

```
LinkedList: a,b,c,d,e
```

[22]:
```
ul.shrink()
print(ul)
```

```
LinkedList: a,c,e
```

7.3 dup_first

Implement the method `dup_first`:

```
def dup_first(self):
    """ MODIFIES this list by adding a duplicate of first node right after it.

        For example, the list 'a','b','c' should become 'a','a','b','c'.
        An empty list remains unmodified.

        - DOES NOT RETURN ANYTHING !!!

    """

    raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python3 -m unittest more_test.DupFirstTest`

7.4 dup_all

Implement the method `dup_all`:

```
def dup_all(self):
    """ Modifies this list by adding a duplicate of each node right after it.

        For example, the list 'a','b','c' should become 'a','a','b','b','c','c'.
        An empty list remains unmodified.

        - MUST PERFORM IN O(n) WHERE n is the length of the list.

        - DOES NOT RETURN ANYTHING !!!
    """

    raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python3 -m unittest more_test.DupAllTest`

7.5 mirror

Implement following `mirror` function. **NOTE**: the function is external to class `LinkedList`.

```
def mirror(lst):
    """ Returns a new LinkedList having double the nodes of provided lst
        First nodes will have same elements of lst, following nodes will
        have the same elements but in reversed order.

        For example:

            >>> mirror(['a'])
            LinkedList: a,a

            >>> mirror(['a','b'])
            LinkedList: a,b,b,a

            >>> mirror(['a','c','b'])
            LinkedList: a,c,b,b,c,a

    """
    raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python -m unittest more_test.MirrorTest`

7.6 norep

Implement the method `norep`:

```
def norep(self):
    """ MODIFIES this list by removing all the consecutive
        repetitions from it.

        - MUST perform in O(n), where n is the list size.

        For example, after calling norep:

        'a','a','b','c','c','c'   will become  'a','b','c'

        'a','a','b','a'   will become   'a','b','a'

    """

    raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python -m unittest more_test.NorepTest`

7.8 find_couple

Implement following `find_couple` method.

```
def find_couple(self,a,b):
    """ Search the list for the first two consecutive elements having data equal to
        provided a and b, respectively. If such elements are found, the position
        of the first one is returned, otherwise raises LookupError.

        - MUST run in O(n), where n is the size of the list.
        - Returned index start from 0 included

    """
```

Testing: `python3 -m unittest more_test.FindCoupleTest`

7.9 swap

Implement the method `swap`:

```
def swap (self, i, j):
    """
        Swap the data of nodes at index i and j. Indeces start from 0 included.
        If any of the indeces is out of bounds, rises IndexError.

        NOTE: You MUST implement this function with a single scan of the list.

    """
```

Testing: `python3 -m unittest more_test.SwapTest`

7.10 gaps

Given a linked list of size n which only contains integers, a gap is an **index** `i`, `0<i<n`, such that `L[i-1]<L[i]`. For the purpose of this exercise, we assume an empy list or a list with one element have zero gaps

Example:

```
 data:  9 7 6 8 9 2 2 5
index:  0 1 2 3 4 5 6 7
```

contains three gaps [3,4,7] because:

- number 8 at index 3 is greater than previous number 6 at index 2
- number 9 at index 4 is greater than previous number 8 at index 3
- number 5 at index 7 is greater than previous number 2 at index 6

Implement this method:

```
def gaps(self):
    """ Assuming all the data in the linked list is made by numbers,
        finds the gaps in the LinkedList and return them as a Python list.

        - we assume empty list and list of one element have zero gaps
        - MUST perform in O(n) where n is the length of the list

        NOTE: gaps to return are *indeces* , *not* data!!!!
    """
```

Testing: `python3 -m unittest more_test.GapsTest`

7.11 flatv

Suppose a `LinkedList` only contains integer numbers, say 3,8,8,7,5,8,6,3,9. Implement method `flatv` which scans the list: when it finds the *first* occurence of a node which contains a number which is less then the previous one, and the less than successive one, it inserts after the current one another node with the same data as the current one, and exits.

Example:

for Linked list 3,8,8,7,5,8,6,3,9

calling `flatv` should modify the linked list so that it becomes

Linked list 3,8,8,7,5,5,8,6,3,9

Note that it only modifies the first occurrence found 7,5,8 to 7,5,5,8 and the successive sequence 6,3,9 is not altered

Implement this method:

```
def flatv(self):
```

Testing: `python3 -m unittest more_test.FlatvTest`

7.12 bubble_sort

You will implement bubble sort on a `LinkedList`.

```
def bubble_sort(self):
    """ Sorts in-place this linked list using the method of bubble sort

        - MUST execute in O(n^2) where n is the length of the linked list
    """
```

As a reference, you can look at this `example_bubble` implementation below that operates on regular python lists. Basically, you will have to translate the `for` cycles into two suitable `while` and use node pointers.

NOTE: this version of the algorithm is inefficient as we do not use `j` in the inner loop: your linked list implementation can have this inefficiency as well.

Testing: `python3 -m unittest more_test.BubbleSortTest`

[23]:
```
def example_bubble(plist):
    for j in range(len(plist)):
        for i in range(len(plist)):
            if i + 1 < len(plist) and plist[i]>plist[i+1]:
                temp = plist[i]
                plist[i] = plist[i+1]
                plist[i+1] = temp

my_list = [23, 34, 55, 32, 7777, 98, 3, 2, 1]
example_bubble(my_list)
print(my_list)
```

```
[1, 2, 3, 23, 32, 34, 55, 98, 7777]
```

7.13 merge

Implement this method:

```
def merge(self,l2):
    """ Assumes this linkedlist and l2 linkedlist contain integer numbers
        sorted in ASCENDING order, and  RETURN a NEW LinkedList with
        all the numbers from this and l2 sorted in DESCENDING order

        IMPORTANT 1: *MUST* EXECUTE IN O(n1+n2) TIME where n1 and n2 are
                     the sizes of this and l2 linked_list, respectively

        IMPORTANT 2: *DO NOT* attempt to convert linked lists to
                     python lists!
    """
```

Testing: `python3 -m unittest more_test.MergeTest`

[ ]:
```

```

7.4 Stacks

7.4.1 Download exercises zip

Browse files online[340]

7.4.2 0. Introduction

References

- theory: https://sciproalgo2019.readthedocs.io/slides/Lecture5.pdf
- stack definition on the book[341]

and following sections :

- Stack Abstract Data Type[342]
- Implementing a Stack in Python[343]
- Simple Balanced Parenthesis[344]
- Balanced Symbols - a General Case

What to do

- unzip exercises in a folder, you should get something like this:

```
stacks
    stacks.ipynb
    capped_stack.py
    capped_stack_sol.py
    capped_stack_test.py
    ...
    jupman.py
    sciprog.py
```

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), you will edit the files ending in `.py` files
- Go on reading this notebook, and follow instuctions inside

[340] https://github.com/DavidLeoni/sciprog-ds/tree/master/stacks
[341] http://interactivepython.org/runestone/static/pythonds/BasicDS/WhatisaStack.html
[342] http://interactivepython.org/runestone/static/pythonds/BasicDS/TheStackAbstractDataType.html
[343] http://interactivepython.org/runestone/static/pythonds/BasicDS/ImplementingaStackinPython.html
[344] http://interactivepython.org/runestone/static/pythonds/BasicDS/SimpleBalancedParentheses.html

7.4.3 1. CappedStack

You will try to implement a so called *capped* stack, which has a limit called *cap* over which elements are discarded.

- Your internal implementation will use python lists
- Please name internal variables that you don't want to expose to class users by prepending them with one underscore `'_'`, like `_elements` or `_cap`
  - The underscore is just a convention, class users will still be able to get internal variables by accessing them with field accessors like `mystack._elements`
  - If users manipulate private fields and complain something is not working, you can tell them it's their fault!
- try to write robust code. In general, when implementing code in the real world you might need to think more about boundary cases. In this case, we add the additional constraint that if you pass to the stack a negative or zero cap, your class initalization is expected to fail and raise a `ValueError`.
- For easier inspection of the stack, implement also an `__str__` method so that calls to `print` show text like `CappedStack: cap=4 elements=['a', 'b']`

IMPORTANT: you can exploit any Python feature you deem correct to implement the data structure. For example, internally you could represent the elements as a list , and use its own methods to grow it.

QUESTION: If we already have Python lists that can more or less do the job of the stack, why do we need to wrap them inside a Stack? Can't we just give our users a Python list?

QUESTION: When would you *not* use a Python list to hold the data in the stack?

Notice that:

- We tried to use pythonic names for methods, so for example `isEmpty` was renamed to `is_empty`
- In this case, when this stack is required to `pop` or `peek` but it is found to be empty, an `IndexError` is raised

CappedStack Examples

To get an idea of the class to be made, in the terminal you may run the python interpreter and load the solution module like we are doing here:

[2]:
```
from capped_stack_sol import *
```

[3]:
```
s = CappedStack(2)
```

[4]:
```
print(s)
```

```
CappedStack: cap=2 elements=[]
```

[5]:
```
s.push('a')
```

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

```
CappedStack: cap=2 elements=['a']
```

[7]:
```
s.peek()
```

[7]:
```
'a'
```

[8]:
```
s.push('b')
```

[9]:
```
s.peek()
```

[9]:
```
'b'
```

[10]:
```
print(s)
```

```
CappedStack: cap=2 elements=['a', 'b']
```

[11]:
```
s.peek()
```

[11]:
```
'b'
```

[12]:
```
s.push('c')  # exceeds cap, gets silently discarded
```

[13]:
```
print(s)   # no c here ...
```

```
CappedStack: cap=2 elements=['a', 'b']
```

[14]:
```
s.pop()
```

[14]:
```
'b'
```

[15]:
```
print(s)
```

```
CappedStack: cap=2 elements=['a']
```

[16]:
```
s.pop()
```

[16]:
```
'a'
```

```
s.pop()   # can't pop empty stack

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-41-c88c8c48122b> in <module>()
----> 1 s.pop()

~/Da/prj/sciprog-ds/prj/stacks/capped_stack_sol.py in pop(self)
     63         #jupman-raise
     64         if len(self._elements) == 0:
---> 65             raise IndexError("Empty stack !")
     66         else:
     67             return self._elements.pop()

IndexError: Empty stack !
```

```
s.peek()     # can't peek empty stack


---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-18-f056e7e54f5d> in <module>()
----> 1 s.peek()

~/Da/prj/sciprog-ds/prj/stacks/capped_stack_sol.py in peek(self)
     77         #jupman-raise
     78         if len(self._elements) == 0:
---> 79             raise IndexError("Empty stack !")
     80
     81         return self._elements[-1]

IndexError: Empty stack !
```

Capped Stack basic methods

Now open `capped_stack.py` and start implementing the methods in the order you find them.

All basic methods are grouped within the CappedStackTest class: to execute single tests you can put the test method name after the test class name, see examples below.

1.1 __init__

Test: `python3 -m unittest capped_stack_test.CappedStackTest.test_01_init`

1.2 cap

Test: `python3 -m unittest capped_stack_test.CappedStackTest.test_02_cap`

1.3 size

Test: `python3 -m unittest capped_stack_test.CappedStackTest.test_03_size`

1.4 __str__

Test: `python3 -m unittest capped_stack_test.CappedStackTest.test_04_str`

1.5 is_empty

Test: `python3 -m unittest capped_stack_test.CappedStackTest.test_05_is_empty`

1.6 push

Test: `python3 -m unittest capped_stack_test.CappedStackTest.test_06_push`

1.7 peek

Test: `python3 -m unittest capped_stack_test.CappedStackTest.test_07_peek`

1.8 pop

Test: `python3 -m unittest capped_stack_test.CappedStackTest.test_08_pop`

1.9 peekn

Implement the `peekn` method:

```
def peekn(self, n):
    """
        RETURN a list with the n top elements, in the order in which they
        were pushed. For example, if the stack is the following:

            e
            d
            c
            b
            a

        peekn(3) will return the list ['c','d','e']
```

(continues on next page)

(continued from previous page)

```
        - If there aren't enough element to peek, raises IndexError
        - If n is negative, raises an IndexError

    """
    raise Exception("TODO IMPLEMENT ME!")
```

Test: `python3 -m unittest capped_stack_test.PeeknTest`

1.10 popn

Implement the `popn` method:

```
def popn(self, n):
    """ Pops the top n elements, and RETURN them as a list, in the order in
        which they where pushed. For example, with the following stack:

            e
            d
            c
            b
            a

        popn(3)

        will give back ['c','d','e'], and stack will become:

            b
            a

        - If there aren't enough element to pop, raises an IndexError
        - If n is negative, raises an IndexError
    """
```

Test: `python3 -m unittest capped_stack_test.PopnTest`

1.11 set_cap

Implement the `set_cap` method:

```
def set_cap(self, cap):
    """ MODIFIES the cap, setting its value to the provided cap.

        If the cap is less then the stack size, all the elements above
        the cap are removed from the stack.

        If cap < 1, raises an IndexError
        Does *not* return anything!

        For example, with the following stack, and cap at position 7:

        cap ->  7
                6
                5  e
```

(continues on next page)

(continued from previous page)

```
                4   d
                3   c
                2   b
                1   a


        calling method set_cap(3) will change the stack to this:

        cap ->  3   c
                2   b
                1   a

    """
```

Test: `python3 -m unittest capped_stack_test.SetCapTest`

7.4.4 2. SortedStack

You are given a class `SortedStack` that models a simple stack. This stack is similar to the `CappedStack` you already saw, the differences being:

- **it can only contain integers**, trying to put other type of values will raise a `ValueError`
- integers **must** be inserted sorted in the stack, either ascending or descending
- there is no cap

```
Example:

     Ascending:       Descending

        8                 3
        5                 5
        3                 8
```

[17]:
```
from sorted_stack_sol import *
```

To create a `SortedStack` sorted in ascending order, just call it passing `True`:

[18]:
```
s = SortedStack(True)
print(s)
```

```
SortedStack (ascending):   elements=[]
```

[19]:
```
s.push(5)
print(s)
```

```
SortedStack (ascending):   elements=[5]
```

[20]:
```
s.push(7)
print(s)
```

```
SortedStack (ascending):   elements=[5, 7]
```

[21]:
```
print(s.pop())
```

```
7
```

[22]:
```
print(s)
```

```
SortedStack (ascending):   elements=[5]
```

[23]:
```
print(s.pop())
```

```
5
```

[24]:
```
print(s)
```

```
SortedStack (ascending):   elements=[]
```

For descending order, pass `False` when you create it:

[25]:
```
sd = SortedStack(False)
sd.push(7)
sd.push(5)
sd.push(4)
print(sd)
```

```
SortedStack (descending):   elements=[7, 5, 4]
```

2.1 transfer

Now implement the `transfer` function.

NOTE: function is external to class `SortedStack`, so you must **NOT** access fields which begin with underscore (like `_elements`), which are meant to be private !!

```
def transfer(s):
    """ Takes as input a SortedStack s (either ascending or descending) and
        returns a new SortedStack with the same elements of s, but in reverse order.
        At the end of the call s will be empty.

        Example:

            s       result

            2         5
            3         3
            5         2
    """
    raise Exception("TODO IMPLEMENT ME !!")
```

Testing

Once done, running this will run only the tests in `TransferTest` class and hopefully they will pass.

**Notice that `exercise1` is followed by a dot and test class name `.TransferTest` : **

```
python -m unittest sorted_stack_test.TransferTest
```

2.2 merge

Implement following `merge` function. **NOTE**: function is external to class `SortedStack`.

```
def merge(s1,s2):
    """ Takes as input two SortedStacks having both ascending order,
        and returns a new SortedStack sorted in descending order, which will be the␣
↪sorted merge
        of the two input stacks. MUST run in O(n1 + n2) time, where n1 and n2 are s1␣
↪and s2 sizes.

        If input stacks are not both ascending, raises ValueError.
        At the end of the call the input stacks will be empty.


        Example:

        s1 (asc)    s2 (asc)       result (desc)

           5           7              2
           4           3              3
           2                          4
                                      5
                                      7

    """

    raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python -m unittest sorted_stack_test.MergeTest`

7.4.5 3. WStack

Using a text editor, open file `wstack.py`. You will find a `WStack` class skeleton which represents a simple stack that can only contain integers.

3.1 implement class WStack

Fill in missing methods in class `WStack` in the order they are presented so to have a `.weight()` method that returns the total sum of integers in the stack in `O(1)` time.

Example:

[26]:
```
from wstack_sol import *
```

[27]:
```
s = WStack()
```

[28]:
```
print(s)
```

```
WStack: weight=0 elements=[]
```

[29]:
```
s.push(7)
```

[30]:
```
print(s)
```

```
WStack: weight=7 elements=[7]
```

[31]:
```
s.push(4)
```

[32]:
```
print(s)
```

```
WStack: weight=11 elements=[7, 4]
```

[33]:
```
s.push(2)
```

[34]:
```
s.pop()
```

[34]:
```
2
```

[35]:
```
print(s)
```

```
WStack: weight=11 elements=[7, 4]
```

3.2 accumulate

Implement function `accumulate`:

```
def accumulate(stack1, stack2, min_amount):
    """ Pushes on stack2 elements taken from stack1 until the weight of
        stack2 is equal or exceeds the given min_amount

        - if the given min_amount cannot possibly be reached because
          stack1 has not enough weight, raises early ValueError without
          changing stack1.
        - DO NOT access internal fields of stacks, only use class methods.
        - MUST perform in O(n) where n is the size of stack1
        - NOTE: this function is defined *outside* the class !
    """
```

Testing: `python -m unittest wstack_test.AccumulateTest`

Example:

[36]:
```


s1 = WStack()


print(s1)
```

```
WStack: weight=0 elements=[]
```

[37]:
```
s1.push(2)
s1.push(9)
s1.push(5)
s1.push(3)
```

[38]:
```
print(s1)
```

```
WStack: weight=19 elements=[2, 9, 5, 3]
```

[39]:
```
s2 = WStack()
print(s2)
```

```
WStack: weight=0 elements=[]
```

[40]:
```
s2.push(1)
s2.push(7)
s2.push(4)
```

[41]:
```
print(s2)
```

```
WStack: weight=12 elements=[1, 7, 4]
```

[42]:
```
# attempts to reach in s2 a weight of at least 17
```

[43]:
```
accumulate(s1,s2,17)
```

[44]:
```
print(s1)
```

```
WStack: weight=11 elements=[2, 9]
```

Two top elements were taken from s1 and now s2 has a weight of 20, which is >= 17

7.4.6 4. Backpack

Open a text editor and edit file `backpack_sol.py`

We can model a backpack as stack of elements, each being a tuple with a name and a weight.

A sensible strategy to fill a backpack is to place heaviest elements to the bottom, so our backback will allow pushing an element only if that element weight is equal or lesser than current topmost element weight.

The backpack has also a maximum weight: you can put any number of items you want, as long as its maximum weight is not exceeded.

Example

[45]:
```
from backpack_sol import *

bp = Backpack(30)  # max_weight = 30

bp.push('a',10)   # item 'a' with weight 10
```

```
DEBUG:  Pushing (a,10)
```

[46]:
```
print(bp)
```

```
Backpack: weight=10 max_weight=30
          elements=[('a', 10)]
```

[47]:
```
bp.push('b',8)
```

```
DEBUG:  Pushing (b,8)
```

[48]:
```
print(bp)
```

```
Backpack: weight=18 max_weight=30
          elements=[('a', 10), ('b', 8)]
```

```
>>> bp.push('c', 11)

DEBUG:  Pushing (c,11)

ValueError: ('Pushing weight greater than top element weight! %s > %s', (11, 8))
```

[49]:
```
bp.push('c', 7)
```

```
DEBUG:  Pushing (c,7)
```

[50]:
```
print(bp)
```

```
Backpack: weight=25 max_weight=30
          elements=[('a', 10), ('b', 8), ('c', 7)]
```

```
>>> bp.push('d', 6)

DEBUG:  Pushing (d,6)

ValueError: Can't exceed max_weight ! (31 > 30)
```

4.1 class

⊛⊛ Implement methods in the class `Backpack`, in the order they are shown. If you want, you can add debug prints by calling the `debug` function

IMPORTANT: the data structure should provide the total current weight in O(1), so make sure to add and update an appropriate field to meet this constraint.

Testing: `python3 -m unittest backpack_test.BackpackTest`

4.2 remove

⊛⊛ Implement function `remove`:

```
# NOTE: this function is implemented *outside* the class !

def remove(backpack, el):
    """
        Remove topmost occurrence of el found in the backpack,
        and RETURN it (as a tuple name, weight)

        - if el is not found, raises ValueError

        - DO *NOT* ACCESS DIRECTLY FIELDS OF BACKPACK !!!
          Instead, just call methods of the class!
```

(continues on next page)

(continued from previous page)

```
        - MUST perform in O(n), where n is the backpack size

        - HINT: To remove el, you need to call Backpack.pop() until
                the top element is what you are looking for. You need
                to save somewhere the popped items except the one to
                remove, and  then push them back again.

    """
```

Testing: `python3 -m unittest backpack_test.RemoveTest`

Example:

[51]:
```
bp = Backpack(50)

bp.push('a',9)
bp.push('b',8)
bp.push('c',8)
bp.push('b',8)
bp.push('d',7)
bp.push('e',5)
bp.push('f',2)
```

```
DEBUG:  Pushing (a,9)
DEBUG:  Pushing (b,8)
DEBUG:  Pushing (c,8)
DEBUG:  Pushing (b,8)
DEBUG:  Pushing (d,7)
DEBUG:  Pushing (e,5)
DEBUG:  Pushing (f,2)
```

[52]:
```
print(bp)
```

```
Backpack: weight=47 max_weight=50
          elements=[('a', 9), ('b', 8), ('c', 8), ('b', 8), ('d', 7), ('e', 5), ('f',␣
↪2)]
```

[53]:
```
remove(bp, 'b')
```

```
DEBUG:  Popping ('f', 2)
DEBUG:  Popping ('e', 5)
DEBUG:  Popping ('d', 7)
DEBUG:  Popping ('b', 8)
DEBUG:  Pushing (d,7)
DEBUG:  Pushing (e,5)
DEBUG:  Pushing (f,2)
```

[53]:
```
('b', 8)
```

[54]:
```
print(bp)
```

```
Backpack: weight=39 max_weight=50
          elements=[('a', 9), ('b', 8), ('c', 8), ('d', 7), ('e', 5), ('f', 2)]
```

[55]:
```
print(s2)
```

```
WStack: weight=20 elements=[1, 7, 4, 3, 5]
```

7.4.7 5. Tasks

Very often, you begin to do a task just to discover it requires doing 3 other tasks, so you start carrying them out one at a time and discover one of them actually requires to do yet another two other subtasks….

To represent the fact a task may have subtasks, we will use a dictionary mapping a task label to a list of subtasks, each represented as a label. For example:

[56]:
```
subtasks = {
        'a':['b','g'],
        'b':['c','d','e'],
        'c':['f'],
        'd':['g'],
        'e':[],
        'f':[],
        'g':[]
    }
```

Task `a` requires subtasks `b` and`g` to be carried out (in this order), but task `b` requires subtasks `c`, `d` and `e` to be done. `c` requires `f` to be done, and `d` requires `g`.

You will have to implement a function called `do` and use a Stack data structure, which is already provided and you don't need to implement. Let's see an example of execution.

IMPORTANT: In the execution example, there are many prints just to help you understand what's going on, but the only thing we actually care about is the final list returned by the function!

IMPORTANT: notice subtasks are scheduled in reversed order, so the item on top of the stack will be the first to get executed !

[57]:
```
from tasks_sol import *

do('a', subtasks)
```

```
DEBUG:  Stack:   elements=['a']
DEBUG:  Doing task a, scheduling subtasks ['b', 'g']
DEBUG:          Stack:   elements=['g', 'b']
DEBUG:  Doing task b, scheduling subtasks ['c', 'd', 'e']
DEBUG:          Stack:   elements=['g', 'e', 'd', 'c']
DEBUG:  Doing task c, scheduling subtasks ['f']
DEBUG:          Stack:   elements=['g', 'e', 'd', 'f']
DEBUG:  Doing task f, scheduling subtasks []
DEBUG:          Nothing else to do!
DEBUG:          Stack:   elements=['g', 'e', 'd']
DEBUG:  Doing task d, scheduling subtasks ['g']
DEBUG:          Stack:   elements=['g', 'e', 'g']
DEBUG:  Doing task g, scheduling subtasks []
DEBUG:          Nothing else to do!
DEBUG:          Stack:   elements=['g', 'e']
DEBUG:  Doing task e, scheduling subtasks []
DEBUG:          Nothing else to do!
DEBUG:          Stack:   elements=['g']
DEBUG:  Doing task g, scheduling subtasks []
DEBUG:          Nothing else to do!
DEBUG:          Stack:   elements=[]
```

[57]:
```
['a', 'b', 'c', 'f', 'd', 'g', 'e', 'g']
```

The `Stack` you must use is simple and supports `push`, `pop`, and `is_empty` operations:

[58]:
```
s = Stack()
```

[59]:
```
print(s)
```

```
Stack:   elements=[]
```

[60]:
```
s.is_empty()
```

[60]:
```
True
```

[61]:
```
s.push('a')
```

[62]:
```
print(s)
```

```
Stack:   elements=['a']
```

[63]:
```
s.push('b')
```

[64]:
```
print(s)
```

```
Stack:   elements=['a', 'b']
```

[65]:
```
s.pop()
```

[65]:
```
'b'
```

[66]:
```
print(s)
```

```
Stack:   elements=['a']
```

5.1 do

Now open `tasks.py` and implement function `do`:

```
def do(task, subtasks):
    """ Takes a task to perform and a dictionary of subtasks,
        and RETURN a list of performed tasks

        - To implement it, inside create a Stack instance and a while cycle.
        - DO *NOT* use a recursive function
        - Inside the function, you can use a print like "I'm doing task a',
          but that is only to help yourself in debugging, only the
          list returned by the function will be considered in the evaluation!
    """
```

Testing: `python3 -m unittest tasks_test.DoTest`

5.2 do_level

In this exercise, you are asked to implement a slightly more complex version of the previous function where on the `Stack` you push two-valued tuples, containing the task label and the associated level. The first task has level 0, the immediate subtask has level 1, the subtask of the subtask has level 2 and so on and so forth. In the list returned by the function, you will put such tuples.

One possibile use is to display the executed tasks as an indented tree, where the indentation is determined by the level. Here we see an example:

IMPORTANT: Again, the prints are only to let you understand what's going on, and you are *not* required to code them. The only thing that really matters is the list the function must return !

[67]:
```
subtasks = {
        'a':['b','g'],
        'b':['c','d','e'],
        'c':['f'],
        'd':['g'],
        'e':[],
        'f':[],
        'g':[]
    }

do_level('a', subtasks)
```

```
DEBUG:                                                  Stack:   elements=[('a', 0)]
DEBUG:  I'm doing       a               level=0 Stack:   elements=[('g', 1), ('b', 1)]
DEBUG:  I'm doing         b             level=1 Stack:   elements=[('g', 1), ('e', 2),
↪ ('d', 2), ('c', 2)]
DEBUG:  I'm doing           c           level=2 Stack:   elements=[('g', 1), ('e', 2),
↪ ('d', 2), ('f', 3)]
DEBUG:  I'm doing             f         level=3 Stack:   elements=[('g', 1), ('e', 2),
↪ ('d', 2)]
DEBUG:  I'm doing           d           level=2 Stack:   elements=[('g', 1), ('e', 2),
↪ ('g', 3)]
DEBUG:  I'm doing             g         level=3 Stack:   elements=[('g', 1), ('e', 2)]
DEBUG:  I'm doing           e           level=2 Stack:   elements=[('g', 1)]
DEBUG:  I'm doing         g             level=1 Stack:   elements=[]
```

[67]:
```
[('a', 0),
 ('b', 1),
 ('c', 2),
 ('f', 3),
 ('d', 2),
 ('g', 3),
 ('e', 2),
 ('g', 1)]
```

Now implement the function:

```
def do_level(task, subtasks):
    """ Takes a task to perform and a dictionary of subtasks,
        and RETURN a list of performed tasks, as tuples (task label, level)

        - To implement it, use a Stack and a while cycle
        - DO *NOT* use a recursive function
        - Inside the function, you can use a print like "I'm doing task a',
          but that is only to help yourself in debugging, only the
```

(continues on next page)

(continued from previous page)

```
        list returned by the function will be considered in the evaluation
    """
```

Testing: `python3 -m unittest tasks_test.DoLevelTest`

7.4.8 6. Stacktris

Open a text editor and edit file `stacktris.py`

A `Stacktris` is a data structure that operates like the famous game Tetris, with some restrictions:

- Falling pieces can be either of length 1 or 2. We call them `1-block` and `2-block` respectively
- The pit has a fixed width of 3 columns
- `2-block`s can only be in horizontal

We print a `Stacktris` like this:

```
\ j 012
i
4  | 11|    # two 1-block
3  | 22|    # one 2-block
2  | 1 |    # one 1-block
1  |22 |    # one 2-block
0  |1 1|    # on the ground there are two 1-block
```

In Python, we model the `Stacktris` as a class holding in the variable `_stack` a list of lists of integers, which models the pit:

```
class Stacktris:

    def __init__(self):
        """ Creates a Stacktris
        """
        self._stack = []
```

So in the situation above the `_stack` variable would look like this (notice row order is inverted with respect to the print)

```
[
    [1,0,1],
    [2,2,0],
    [0,1,0],
    [0,2,2],
    [0,1,1],
]
```

The class has three methods of interest which you will implement, `drop1(j)` , `drop2h(j)` and `_shorten`

Example

Let's see an example:

```
[68]: from stacktris_sol import *

st = Stacktris()
```

At the beginning the pit is empty:

[69]:
```
st
```

[69]:
```
Stacktris:
EMPTY
```

We can start by dropping from the ceiling a block of dimension 1 into the last column at index `j=2`. By doing so, a new row will be created, and will be a list containing the numbers `[0,0,1]`

IMPORTANT: zeroes are not displayed

[70]:
```
st.drop1(2)
```

```
DEBUG:  Stacktris:
        |  1|
```

[70]:
```
[]
```

Now we drop an horizontal block of dimension 2 (a `2-block`) having the leftmost block at column `j=1`. Since below in the pit there is already the `1` block we previosly put, the new block will fall and stay upon it. Internally, we will add a new row as a python list containing the numbers `[0,2,2]`

[71]:
```
st.drop2h(1)
```

```
DEBUG:  Stacktris:
        | 22|
        |  1|
```

[71]:
```
[]
```

We see the zeroth column is empty, so if we drop there a `1-block` it will fall to the ground. Internally, the zeroth list will become `[1,0,1]`:

[72]:
```
st.drop1(0)
```

```
DEBUG:  Stacktris:
        | 22|
        |1 1|
```

[72]:
```
[]
```

Now we drop again a `2-block` at column `j=2`, on top of the previously laid one. This will add a new row as list `[0,2,2]`.

[73]:
```
st.drop2h(1)
```

```
DEBUG:  Stacktris:
        | 22|
        | 22|
        |1 1|
```

[73]:
```
[]
```

In the game Tetris, when a row becomes completely filled it disappears. So if we drop a `1-block` to the leftmost column, the mid line should be removed.

NOTE: The messages on the console are just debug print, the function `drop1` only returns the extracted line `[1,2,2]`:

[74]:
```
st.drop1(0)
```

```
DEBUG:  Stacktris:
        | 22|
        |122|
        |1 1|

DEBUG:  POPPING [1, 2, 2]
DEBUG:  Stacktris:
        | 22|
        |1 1|
```

[74]:
```
[1, 2, 2]
```

Now we insert another `2-block` starting at `j=0`. It will fall upon the previously laid one:

[75]:
```
st.drop2h(0)
```

```
DEBUG:  Stacktris:
        |22 |
        | 22|
        |1 1|
```

[75]:
```
[]
```

We can complete teh topmost row by dropping a `1-block` to the rightmost column. As a result, the row will be removed from the stack and the row will be returned by the call to `drop1`:

[76]:
```
st.drop1(2)
```

```
DEBUG:  Stacktris:
        |221|
        | 22|
        |1 1|

DEBUG:  POPPING [2, 2, 1]
DEBUG:  Stacktris:
        | 22|
        |1 1|
```

[76]:
```
[2, 2, 1]
```

Another line completion with a `drop1` at column `j=0`:

[77]:
```
st.drop1(0)
```

```
DEBUG:  Stacktris:
        |122|
        |1 1|

DEBUG:  POPPING [1, 2, 2]
DEBUG:  Stacktris:
        |1 1|
```

[77]:
```
[1, 2, 2]
```

We can finally empty the Stacktris by dropping a `1-block` in the mod column:

[78]:
```
st.drop1(1)
```

```
DEBUG:  Stacktris:
        |111|

DEBUG:  POPPING [1, 1, 1]
DEBUG:  Stacktris:
        EMPTY
```

[78]:
```
[1, 1, 1]
```

6.1 _shorten

Start by implementing this private method:

```
def _shorten(self):
    """ Scans the Stacktris from top to bottom searching for a completely filled line:
        - if found, remove it from the Stacktris and return it as a list.
        - if not found, return an empty list.
    """
```

If you wish, you can add debug prints but they are not mandatory

Testing: `python3 -m unittest stacktris_test.ShortenTest`

6.2 drop1

Once you are done with the previous function, implement `drop1` method:

NOTE: In the implementation, feel free to call the previously implemented `_shorten` method.

```
def drop1(self, j):
    """ Drops a 1-block on column j.

         - If another block is found,  place the 1-block on top of that block,
           otherwise place it on the ground.

        - If, after the 1-block is placed, a row results completely filled, removes
          the row and RETURN it. Otherwise, RETURN an empty list.

        - if index `j` is outside bounds, raises ValueError
    """
```

Testing: `python3 -m unittest stacktris_test.Drop1Test`

6.3 drop2h

Once you are done with the previous function, implement `drop2` method:

```
def drop2h(self, j):
    """ Drops a 2-block horizontally with left block on column j,

         - If another block is found,  place the 2-block on top of that block,
           otherwise place it on the ground.
```

(continues on next page)

(continued from previous page)

```
        - If, after the 2-block is placed, a row results completely filled,
          removes the row and RETURN it. Otherwise, RETURN an empty list.

        - if index `j` is outside bounds, raises ValueError
    """
```

Testing: `python3 -m unittest stacktris_test.Drop2hTest`

[ ]:

```

```

7.5 Queues

7.5.1 Download exercises zip

Browse files online[345]

7.5.2 Introduction

In these exercises, you will be implementing several queues.

- See theory slides[346]
- See Queue Abstract Data Type[347] on the book
- See Implementing a Queue in Python[348] on the book

What to do

- unzip exercises in a folder, you should get something like this:

```
queues
    queues.ipynb
    circular_queue.py
    circular_queue_test.py
    circular_queue_sol.py
    jupman.py
    sciprog.py
```

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), you will edit the files ending in `.py` files
- Go on reading this notebook, and follow instuctions inside.

[345] https://github.com/DavidLeoni/sciprog-ds/tree/master/queues
[346] https://sciproalgo2019.readthedocs.io/slides/Lecture5.pdf
[347] http://interactivepython.org/runestone/static/pythonds/BasicDS/WhatIsaQueue.html
[348] http://interactivepython.org/runestone/static/pythonds/BasicDS/ImplementingaQueueinPython.html

7.5.3 1. LinkedQueue

Open `linked_queue.py`.

You are given a queue implemented as a LinkedList, with usual `_head` pointer plus additional `_tail` pointer and `_size` counter

- Data in enqueued at the right, in the tail
- Data is dequeued at the left, removing it from the head

Example, where the arrows represent _next pointers:

```
_head                        _tail
    a -> b -> c -> d -> e -> f
```

In this exercise you will implement the methods `enqn(lst)` and `deqn(n)` which respectively enqueue a python list of n elements and dequeue n elements, returning python a list of them.

Here we show an example usage, see to next points for detailed instructions.

Example:

[2]:
```
from linked_queue_sol import *
```

[3]:
```
q = LinkedQueue()
```

[4]:
```
print(q)
```

```
LinkedQueue:
```

[5]:
```
q.enqn(['a','b','c'])
```

Return nothing, queue becomes:

```
_head         _tail
     a -> b -> c
```

[6]:
```
q.enqn(['d'])
```

Return nothing, queue becomes:

```
_head              _tail
    a -> b -> c -> d
```

[7]:
```
q.enqn(['e','f'])
```

Return nothing, queue becomes:

```
_head                        _tail
    a -> b -> c -> d -> e -> f
```

[8]:
```
q.deqn(3)
```

[8]:
```
['a', 'b', 'c']
```

Returns ['d', 'e', 'f'] and queue becomes:

```
_head         _tail
      a -> b -> c
```

[9]:
```
q.deqn(1)
```

[9]:
```
['d']
```

Returns ['c'] and queue becomes:

```
_head    _tail
    a -> b
```

```
q.deqn(5)

---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
<ipython-input-55-e68c2e9949d0> in <module>()
      1
----> 2 q.deqn(5)

~/Da/prj/sciprog-ds/prj/queues/linked_queue_sol.py in deqn(self, n)
    202         #jupman-raise
    203         if n > self._size:
--> 204             raise LookupError('Asked to dequeue %s elements, but only %s are␣
↪available!' % (n, self._size))
    205
    206         ret = []

LookupError: Asked to dequeue 5 elements, but only 2 are available!
```

Raises `LookupError` as there aren't enough elements to remove

1.1 enqn

Implement the method `enqn`:

```
def enqn(self, lst):
    """ Enqueues provided list of elements at the tail of the queue

        - Required complexity: O(len(lst))
        - NOTE: remember to update the _size and _tail

        Example: supposing arrows represent _next pointers:

      _head         _tail
          a -> b -> c

        Calling

        q.enqn(['d', 'e', 'f', 'g'])
```

(continues on next page)

(continued from previous page)

```
        will produce the queue:

      _head                             _tail
          a -> b -> c -> d -> e -> f -> g
```

Testing: `python3 -m unittest linked_queue_test.EnqnTest`

1.2 deqn

Implement the method `deqn`:

```
def deqn(self, n):
    """ Removes n elements from the head, and return them as a Python list,
        where the first element that was enqueued will appear at the
        *beginning* of the returned Python list.

        - if n is greater than the size of the queue, raises a LookupError.
        - required complexity: O(n)

        NOTE 1: return a list of the *DATA* in the nodes, *NOT* the nodes
                themselves
        NOTE 2: DO NOT try to convert the whole queue to a Python
                list for playing with splices.
        NOTE 3: remember to update _size, _head and _tail when needed.


        For example, supposing arrows represent _next pointers:


      _head                             _tail
          a -> b -> c -> d -> e -> f -> g

        q.deqn(3) will return the Python list ['a', 'b', 'c']

        After the call, the queue will be like this:

      _head              _tail
          d -> e -> f -> g

    """
```

Testing: `python3 -m unittest linked_queue_test.DeqnTest`

7.5.4 2. CircularQueue

A circular queue is a data structure which when initialized occupies a **fixed** amount of memory called *capacity*. Typically, fixed size data structures are found in systems programming (i.e. programming drivers), when space is constrained and you want predictable results as much as possible. For us, it will be an example of modular arithmetic usage. In our implementation, to store data we will use a Python list, which we initialize with a number of empty cells equal to *capacity*. During initialization, it does't matter what we actually put inside cells, in this case we will use `None`. Note that **capacity never changes**, and **cells are never added nor remove** from the list. What varies during execution is the actual content of the cells, the index pointing to the head of the queue (from which elements are dequeued) and another number we call *size* which is a number telling us how many elements are present in the queue. Summing *head* and *size* numbers will allow

us to determine where to enqueue elements at the tail of the queue - to avoid overflow, we will have to take modulus of the sum. Keep reading for details.

To implement the circular queue you can use this pseudo code:

QUESTION 2.1: Pseudo code is meant to give a general overview of the algorithms, and can often leave out implementation details, such as defining what to do when things don't work as expected. If you were to implement this in a real life scenario, do you see any particular problem?

In our implementation, we will:

- use more pythonic names, with underscores instead of camelcase.
- explicitly handle exceptions and corner cases
- be able to insert any kind of object in the queue
- Initial queue will be populated with `None` objects, and will have length set to provided capacity
- `_size` is the current dimension of the queue, which is different from the initial provided `capacity`.
- we consider `capacity` as fixed: it will never change during execution. For this reason, since we use a Python list to represent the data, we don't need an extra variable to hold it, just getting the list length will suffice.
- `_head` is an *index* pointing to the *next* element to be dequeued
- elements are inserted at the position pointed to by `(_head + _size) % capacity()`, and dequeued from position pointed by `_head`. The module `%` operator allows using a list as it were circular, that is, if an index apparently falls outside the list, with the modulus it gets transformed to a small index. Since `_size` can never exceed `capacity()`, the formula `(_head + _size) % capacity()` never points to a place which could

overwrite elements not yet dequeued, except cases when the queue has `_size==0` or `_size==capacity()` which are to be treated as special.

- enqueuing and dequeing operations **don't** modify list length !

QUESTION 2.2: If we can insert any kind of object in the queue including `None`, are we going to have troubles with definitions like `top()` above?

2.1 Implementation

Implement methods in file `circular_queue.py` in the order they are presented, and test them with `circular_queue_test.py`

```
python3 -m unittest circular_queue_test
```

7.5.5 3. ItalianQueue

You will implement an `ItalianQueue`, modelled as a LinkedList with two pointers, a `_head` and a `_tail`.

- an element is enqueued scanning from `_head` until a matching group is found, in which case are inserted after (that is, at the right) of the matching group, otherwise the element is appended at the `_tail`
- an element is dequeued from the `_head`

3.1 Slow v1

To gain some understanding about the data structure, look at the following excerpts.

Excerpt from `Node`:

```
class Node:
    """ A Node of an ItalianQueue.
        Holds both data and group provided by the user.
    """

    def __init__(self, initdata, initgroup):
    def get_data(self):
    def get_group(self):
    def get_next(self):

    # etc ..
```

Excerpt from `ItalianQueue` class:

```
class ItalianQueue:
    """ An Italian queue, v1.

        - Implemented as a LinkedList
        - Worst case enqueue is O(n)
        - has extra methods, for accessing groups and tail:
            - top_group()
            - tail()
            - tail_group()

        Each element is assigned a group; during enqueing, queue is scanned
```

(continues on next page)

(continued from previous page)

```
        from head to tail to find if there is another element with a
        matching group.
            - If there is, element to be enqueued is inserted after the last
              element in the same group sequence (that is, to the right of
              the group)
            - otherwise the element is inserted at the end of the queue
    """

    def __init__(self):
        """ Initializes the queue. Note there is no capacity as parameter

            - MUST run in O(1)
        """
```

Example:

[10]:
```
from italian_queue_sol import *

q = ItalianQueue()
print(q)
```

```
ItalianQueue:

       _head: None
       _tail: None
```

[11]:
```
q.enqueue('a','x')    # 'a' is the element,'x' is the group
```

[12]:
```
print(q)
```

```
ItalianQueue: a
              x
       _head: Node(a,x)
       _tail: Node(a,x)
```

[13]:
```
q.enqueue('c','y')     # 'c' belongs to new group 'y', goes to the end of the queue
```

[14]:
```
print(q)
```

```
ItalianQueue: a->c
              x  y
       _head: Node(a,x)
       _tail: Node(c,y)
```

[15]:
```
q.enqueue('d','y')     # 'd' belongs to existing group 'y', goes to the end of the␣
↪group
```

[16]:
```
print(q)
```

```
ItalianQueue: a->c->d
              x  y  y
       _head: Node(a,x)
       _tail: Node(d,y)
```

[17]:
```
q.enqueue('b','x')     # 'b' belongs to existing group 'x', goes to the end of the␣
↪group
```

[18]:
```
print(q)
```

```
ItalianQueue: a->b->c->d
              x  x  y  y
       _head: Node(a,x)
       _tail: Node(d,y)
```

[19]:
```
q.enqueue('f','z')    # 'f' belongs to new group, goes to the end of the queue
```

[20]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->f
              x  x  y  y  z
       _head: Node(a,x)
       _tail: Node(f,z)
```

[21]:
```
q.enqueue('e','y')   # 'e' belongs to an existing group 'y', goes to the end of the␣
↪group
```

[22]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->e->f
              x  x  y  y  y  z
       _head: Node(a,x)
       _tail: Node(f,z)
```

[23]:
```
q.enqueue('g','z')   # 'g' belongs to an existing group 'z', goes to the end of the␣
↪group
```

[24]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->e->f->g
              x  x  y  y  y  z  z
       _head: Node(a,x)
       _tail: Node(g,z)
```

[25]:
```
q.enqueue('h','z')  # 'h' belongs to an existing group 'z', goes to the end of the␣
↪group
```

[26]:
```
print(q)
```

```
ItalianQueue: a->b->c->d->e->f->g->h
              x  x  y  y  y  z  z  z
       _head: Node(a,x)
       _tail: Node(h,z)
```

Dequeue is always from the head, without taking in consideration the group:

[27]:
```
q.dequeue()
```

[27]:
```
'a'
```

[28]:
```
print(q)
```

```
ItalianQueue: b->c->d->e->f->g->h
              x  y  y  y  z  z  z
       _head: Node(b,x)
       _tail: Node(h,z)
```

[29]:
```
q.dequeue()
```

[29]:
```
'b'
```

[30]:
```
print(q)
```

```
ItalianQueue: c->d->e->f->g->h
              y  y  y  z  z  z
       _head: Node(c,y)
       _tail: Node(h,z)
```

[31]:
```
q.dequeue()
```

[31]:
```
'c'
```

[32]:
```
print(q)
```

```
ItalianQueue: d->e->f->g->h
              y  y  z  z  z
       _head: Node(d,y)
       _tail: Node(h,z)
```

3.1.1 init

Implement methods in file `italian_queue.py` in the order they are presented **up until** `enqueue` **excluded**

Testing: `python3 -m unittest italian_queue_test.InitEmptyTest`

3.1.2 Slow enqueue

Implement version 1 of `enqueue` running in $O(n)$ where n is the queue size.

```
def enqueue(self, v, g):
    """ Enqueues provided element v having group g, with the following
        criteria:

        Queue is scanned from head to find if there is another element
        with a matching group:
            - if there is, v is inserted after the last element in the
              same group sequence (so to the right of the group)
            - otherwise v is inserted at the end of the queue

        - MUST run in O(n)
    """
```

Testing: `python3 -m unittest italian_queue_test.EnqueueTest`

QUESTION: The ItalianQueue was implemented as a LinkedList. Even if this time we don't care much about perfomance, if we wanted an efficient `enqueue` operation, could we start with a circular data structure ? Or would you prefer improving a LinkedList ?

3.1.2 dequeue

Implement version 1 of `dequeue` running in $O(1)$

```
def dequeue(self):
    """ Removes head element and returns it.

        - If the queue is empty, raises a LookupError.
        - MUST run in O(1)
    """
```

Testing: `python3 -m unittest italian_queue_test.DequeueTest`

3.2 Fast v2

3.2.1 Save a copy

You already wrote a lot of code, and you don't want to lose it, right? Since we are going to make many modifications, when you reach a point when the code does something useful, it is good practice to save a copy of what you have done somewhere, so if you later screw up something, you can always restore the copy.

- Copy the whole folder `queues` in a new folder `queues_v1`
- Add also in the copied folder a separate `README.txt` file, writing inside the version (like `1.0`), the date, and a description of the main features you implemented (for example "Simple Italian Queue, not particularly performant").
- Backing up the work is a form of the so-called *versioning* : there are much better ways to do it (like using git[349]) but we don't address them here.

WARNING: DO NOT SKIP THIS STEP!

No matter how smart you are, you *will* fail, and a backup may be the only way out.

WARNING: NOT CONVINCED YET?

If you still don't understand why you should spend time with this copy bureaucracy, to help you enter the right mood imagine tomorrow is demo day with your best client and you screw up the only working version: your boss will *skin you alive*.

3.2.2 Improve enqueue

Improve `enqueue` so it works in $O(1)$

HINT:

- You will need an extra data structure that keeps track of the starting points of each group and how they are ordered
- You will also need to update this data structure as `enqueue` and `dequeue` calls are made

[349] https://git-scm.com

7.5.6 4. Supermarket queues

In this exercises, you will try to model a supermarket containing several cash queues.

CashQueue

WARNING: DO *NOT* MODIFY CashQueue CLASS

For us, a `CashQueue` is a simple queue of clients represented as strings. A `CashQueue` supports the `enqueue`, `dequeue`, `size` and `is_empty` operations:

- Clients are enqueued at the right, in the tail
- Clients are dequeued from the left, removing them from the head

For example:

```
q = CashQueue()

q.is_empty()       # True

q.enqueue('a')     #  a
q.enqueue('b')     #  a,b
q.enqueue('c')     #  a,b,c

q.size()           # 3

q.dequeue()    #       returns:  a
               # queue becomes:  [b,c]

q.dequeue()    #       returns:  b
               # queue becomes:  [c]

q.dequeue()    #       returns:  c
               # queue becomes:  []

q.dequeue()    # raises LookupError as there aren't enough elements to remove
```

Supermarket

A `Supermarket` contains several cash queues. It is possible to initialize a `Supermarket` by providing queues as simple python lists, where the first clients arrived are on the left, and the last clients are on the right.

For example, by calling:

```
s = Supermarket([
    ['a','b','c'],      # <------ clients arrive from right
    ['d'],
    ['f','g']
])
```

internally three `CashQueue` objects are created. Looking at the first queue with clients `['a','b','c']`, `a` at the head arrived first and `c` at the tail arrived last

```
>>> print(s)

Supermarket
0 CashQueue: ['a', 'b', 'c']
1 CashQueue: ['d']
2 CashQueue: ['f', 'g']
```

Note a supermarket must have at least one queue, which may be empty:

```
s = Supermarket( [[]] )

>>> print(s)

Supermarket
0 CashQueue: []
```

Supermarket as a queue

Our `Supermarket` should maximize the number of served clients (we assume each clients is served in an equal amount of time). To do so, the whole supermarket itself can be seen as a particular kind of queue, which allows the `enqueue` and `dequeue` operations described as follows:

- by calling `supermarket.enqueue(client)` a client gets enqueued in the shortest `CashQueue`.
- by calling `supermarket.dequeue()`, all clients which are at the heads of non-empty `CashQueues` are dequeued all at once, and their list is returned (this simulates parallelism).

Implementation

Now start editing `supermarket.py` implementing methods in the following points.

4.1 Supermarket size

Implement `Supermarket.size`:

```
def size(self):
    """ Return the total number of clients present in all cash queues.
    """
```

Testing: `python3 -m unittest supermarket_test.SizeTest`

4.2 Supermarket dequeue

Implement `Supermarket.dequeue`:

```
def dequeue(self):
    """ Dequeue all the clients which are at the heads of non-empty cash queues,
        and return a list of such clients.

        - clients are returned in the same order as found in the queues
        - if supermarket is empty, an empty list is returned
```

(continues on next page)

(continued from previous page)

```
        For example, suppose we have following supermarket:

        0  ['a','b','c']
        1  []
        2  ['d','e']
        3  ['f']


        A call to deque() will return ['a','d','f']
        and the supermarket will now look like this:

        0  ['b','c']
        1  []
        2  ['e']
        3  []
    """
```

Testing: `python3 -m unittest supermarket_test.DequeueTest`

4.3 Supermarket enqueue

Implement `Supermarket.enqueue`:

```
def enqueue(self, client):
    """ Enqueue provided client in the cash queue with minimal length.

        If more than one minimal length cash queue is available, the one
        with smallest index is chosen.

        For example:

        If we have supermarket

        0  ['a','b','c']
        1  ['d','e','f','g']
        2  ['h','i']
        3  ['m','n']

        since queues 2 and 3 have both minimal length 2,
        supermarket.enqueue('z') will enqueue the client on queue 2:

        0  ['a','b','c']
        1  ['d','e','f','g']
        2  ['h','i','z']
        3  ['m','n']
    """
```

Testing: `python3 -m unittest supermarket_test.EnqueueTest`

7.5.7 5. Shopping mall queues

In this exercises, you will try to model a shopping mall containing several shops and clients.

Client

WARNING: DO *NOT* MODIFY Client CLASS

For us, a `Client` is composed by a name (in the exercise we will use `a`, `b`, `c` …) and a list of shops he wants to visit as a list. We will identify the shops with letters such as `x`, `y`, `z` …

Note: shops to visit are a Python list intended as a stack, so **the first shop to visit is at end (top) of the list**

Example:

```
c = Client('f', ['y','x','z'])
```

creates a `Client` named `f` who wants to visit first the shop `z`, then `x` and finally `y`

Methods:

```
>>> print(c.name())
a
>>> print(c.to_visit())
['z','x','y']
```

Shop

WARNING: DO *NOT* MODIFY Shop CLASS

For us, a `Shop` is a class with a name and a queue of clients. A `Shop` supports the `name`, `enqueue`, `dequeue`, `size` and `is_empty` operations:

- Clients are enqueued at the right, in the tail
- Clients are dequeued from the left, removing them from the head

For example:

```
s = Shop('x')  # creates a shop named 'x'

print(s.name())   # prints  x

s.is_empty()      # True

s.enqueue('a')    #  a        enqueues client 'a'
s.enqueue('b')    #  a,b
s.enqueue('c')    #  a,b,c

s.size()          # 3

s.dequeue()   #       returns:  a
              # queue becomes:  [b,c]
```

(continues on next page)

(continued from previous page)

```
s.dequeue()    #       returns:  b
               # queue becomes:  [c]

s.dequeue()    #       returns:  c
               # queue becomes:  []

s.dequeue()    # raises LookupError as there aren't enough elements to remove
```

Mall

A shopping `Mall` contains several shops and clients. It is possible to initialize a `Mall` by providing

1. shops as a list of values `shop name , client list`, where the first clients arrived are on the left, and the last clients are on the right.
2. clients as a list of values `client name , shop to visit list`

For example, by calling:

```
m = Mall(
[
    'x', ['a','b','c'],      # <------ clients arrive from right
    'y', ['d'],
    'z', ['f','g']
],
[
    'a',['y','x'],
    'b',['x'],
    'c',['x'],
    'd',['z','y'],         # IMPORTANT: shops to visit stack grows from right, so
    'f',['y','x','z'],     # client 'f' wants to visit first shop 'z', then 'x', and␣
↪finally 'y'
    'g',['x','z']
])
```

Internally:

- three `Shop` objects are created in an `OrderedDict`. Looking at the first queue with clients `['a','b','c']`, `a` at the head arrived first and `c` at the tail arrived last.
- 6 `Client` objects are created in an `OrderedDict`. Note if a client is in a particular shop queue, that shop must be his top desired shop to visit in its stack.

```
>>> print(s)

Mall
  Shop x: ['a', 'b', 'c']
  Shop y: ['d']
  Shop z: ['f', 'g']

  Client a: ['y','x']
  Client b: ['x']
  Client c: ['x']
  Client d: ['z','y']
  Client f: ['x','y','z']
  Client g: ['x','z']
```

Methods:

```
>>> m.shops()

OrderedDict([
              ('x', Shop x: ['a', 'b', 'c'])
              ('y', Shop y: ['d'])
              ('z', Shop z: ['f', 'g'])
            ])

>>> m.clients()

OrderedDict([
  ('a', Client a: ['y','x']),
  ('b', Client b: ['x']),
  ('c', Client c: ['x']),
  ('d', Client d: ['z','y']),
  ('f', Client f: ['x','y','z']),
  ('g', Client g: ['x','z'])
])
```

Note a mall must have at least one shop and may have zero clients:

```
m = Mall( {'x':[]}, {} )

>>> print(m)

Mall
   Shop x: []
```

Mall as a queue

Our `Mall` should maximize the number of served clients (we assume each clients is served in an equal amount of time). To do so, the whole mall itself can be seen as a particular kind of queue, which allows the `enqueue` and `dequeue` operations described as follows:

- by calling `mall.enqueue(client)` a client gets enqueued in the top `Shop` he wants to visit (its desired shop to visit list doesn't change)
- by calling `mall.dequeue()`
  - all clients which are at the heads of non-empty `Shops` are dequeued all at once
  - their top desired shop to visit is removed
  - if a client has any shop to visit left, he is automatically enqueued in that `Shop`
  - the list of clients with no shops to visit is returned (this simulates parallelism)

Implementation

Now start editing `mall.py` implementing methods in the following points.

6.1 Mall enqueue

Implement `Mall.enqueue` method:

```
def enqueue(self, client):
    """ Enqueue provided client in the top shop he wants to visit

        - If client is already in the mall, raise ValueError
        - if client has no shop to visit, raise ValueError
        - If any of the shops to visit are not in the mall, raise ValueError

        For example:

        If we have this mall:

        Mall
            Shop x: ['a','b']
            Shop y: ['c']

            Client a: ['y','x']
            Client b: ['x']
            Client c: ['x','y']

        mall.enqueue(Client('d',['x','y'])) will enqueue the client in Shop y :

        Mall
            Shop x: ['a','b']
            Shop y: ['c','d']

            Client a: ['y','x']
            Client b: ['x']
            Client c: ['x','y']
            Client d: ['x','y']

    """
```

Testing: `python3 -m unittest mall_test.EnqueueTest`

6.2 Mall dequeue

Implement `Mall.dequeue` method:

```
def dequeue(self):
    """ Dequeue all the clients which are at the heads of non-empty
        shop queues,enqueues clients in their next shop to visit and return
        a list of names of clients that exit the mall.

        In detail:
        - shop list is scanned, and all clients which are at the heads
          of non-empty Shops are dequeued
```

(continues on next page)

(continued from previous page)

```
            VERY IMPORTANT HINT: FIRST put all this clients in a list,
                                 THEN using that list do all of the following

        - for each dequeued client, his top desired shop is removed from
          his visit list
        - if a client has a shop to visit left, he is automatically
          enqueued in that Shop
            - clients are enqueued in the same order they were dequeued
              from shops
        - the list of clients with no shops to visit anymore
          is returned (this  simulates parallelism)
            - clients are returned in the same order they were dequeued
              from shops
        - if mall has no clients, an empty list is returned

    """
```

Testing: `python3 -m unittest mall_test.DequeueTest`

For example, suppose we have following mall:

[33]:
```
from mall_sol import *
```

[34]:
```
m = Mall([
            'x', ['a', 'b', 'c'],
            'y', ['d'],
            'z', ['f', 'g']
        ],
        [
            'a', ['y', 'x'],
            'b', ['x'],
            'c', ['x'],
            'd', ['z','y'],
            'f', ['y','x','z'],
            'g', ['x','z']
        ])
```

[35]:
```
print(m)
```

```
Mall
  Shop x : ['a', 'b', 'c']
  Shop y : ['d']
  Shop z : ['f', 'g']

  Client a : ['y', 'x']
  Client b : ['x']
  Client c : ['x']
  Client d : ['z', 'y']
  Client f : ['y', 'x', 'z']
  Client g : ['x', 'z']
```

[36]:
```
m.dequeue()  # first call
```

[36]:
```
[]
```

Clients 'a', 'd' and 'f' change shop, the others stay in their current shop. The mall will now look like this:

[37]:
```
print(m)
```

```
Mall
  Shop x : ['b', 'c', 'f']
  Shop y : ['a']
  Shop z : ['g', 'd']

  Client a : ['y']
  Client b : ['x']
  Client c : ['x']
  Client d : ['z']
  Client f : ['y', 'x']
  Client g : ['x', 'z']
```

[38]:
```
m.dequeue()  # second call
```

[38]:
```
['b', 'a']
```

because client 'b' was top shop in the list, 'a' in the second, and both clients had nothing else to visit. Client 'g' changes shop, the others remain in their current shop.

The mall will now look like this:

[39]:
```
print(m)   # Clients a and b are gone
```

```
Mall
  Shop x : ['c', 'f', 'g']
  Shop y : []
  Shop z : ['d']

  Client c : ['x']
  Client d : ['z']
  Client f : ['y', 'x']
  Client g : ['x']
```

[40]:
```
m.dequeue() # third call
```

[40]:
```
['c', 'd']
```

[41]:
```
print(m)
```

```
Mall
  Shop x : ['f', 'g']
  Shop y : []
  Shop z : []

  Client f : ['y', 'x']
  Client g : ['x']
```

[42]:
```
m.dequeue()  # fourth call
```

[42]:
```
[]
```

[43]:
```
print(m)
```

```
Mall
  Shop x : ['g']
  Shop y : ['f']
  Shop z : []

  Client f : ['y']
  Client g : ['x']
```

[44]:
```
m.dequeue()   # fifth call
```

[44]:
```
['g', 'f']
```

[45]:
```
print(m)
```

```
Mall
  Shop x : []
  Shop y : []
  Shop z : []
```

7.5.8 6. Company queues

We can model a company as a list of many employees ordered by their rank, the highest ranking being the first in the list. We assume all employees have different rank. Each employee has a name, a rank, and a queue of tasks to perform (as a Python deque).

When a new employee arrives, it is inserted in the list in the right position according to his rank:

[46]:
```
from company_sol import *

c = Company()
print(c)
```

```

Company:
  name   rank   tasks
```

[47]:
```
c.add_employee('x',9)
```

[48]:
```
print(c)
```

```

Company:
  name   rank   tasks
  x      9      deque([])
```

[49]:
```
c.add_employee('z',2)
```

[50]:
```
print(c)
```

```
Company:
  name   rank   tasks
  x      9      deque([])
  z      2      deque([])
```

[51]:
```
c.add_employee('y',6)
```

[52]:
```
print(c)
```

```
Company:
  name   rank   tasks
  x      9      deque([])
  y      6      deque([])
  z      2      deque([])
```

7.1 add_employee

Implement this method:

```
def add_employee(self, name, rank):
    """
        Adds employee with name and rank to the company, maintaining
        the _employees list sorted by rank (higher rank comes first)

        Represent the employee as a dictionary with keys 'name', 'rank'
        and 'tasks' (a Python deque)

        - here we don't mind about complexity, feel free to use a
          linear scan and .insert
        - If an employee of the same rank already exists, raise ValueError
        - if an employee of the same name already exists, raise ValueError
    """
```

Testing: `python3 -m unittest company_test.AddEmployeeTest`

7.2 add_task

Each employee has a queue of tasks to perform. Tasks enter from the right and leave from the left. Each task has associated a required rank to perform it, but when it is assigned to an employee the required rank may exceed the employee rank or be far below the employee rank. Still, when the company receives the task, it is scheduled in the given employee queue, ignoring the task rank.

[53]:
```
c.add_task('a',3,'x')
```

[54]:
```
c
```

[54]:
```
Company:
  name   rank   tasks
  x      9      deque([('a', 3)])
```

(continues on next page)

(continued from previous page)

```
  y     6     deque([])
  z     2     deque([])
```

[55]:
```
c.add_task('b',5,'x')
```

[56]:
```
c
```

[56]:
```
Company:
  name  rank  tasks
  x     9     deque([('a', 3), ('b', 5)])
  y     6     deque([])
  z     2     deque([])
```

[57]:
```
c.add_task('c',12,'x')
c.add_task('d',1,'x')
c.add_task('e',8,'y')
c.add_task('f',2,'y')
c.add_task('g',8,'y')
c.add_task('h',10,'z')
```

[58]:
```
c
```

[58]:
```
Company:
  name  rank  tasks
  x     9     deque([('a', 3), ('b', 5), ('c', 12), ('d', 1)])
  y     6     deque([('e', 8), ('f', 2), ('g', 8)])
  z     2     deque([('h', 10)])
```

Implement this function:

```
def add_task(self, task_name, task_rank, employee_name):
    """ Append the task as a (name, rank) tuple to the tasks of
        given employee

        - If employee does not exist, raise ValueError
    """
```

Testing: `python3 -m unittest company_test.AddTaskTest`

7.3 work

Work in the company is produced in work steps. Each work step produces a list of all task names executed by the company in that work step.

A work step is done this way:

For each employee, starting from the highest ranking one, dequeue its current task (from the left), and than compare the task required rank with the employee rank according to these rules:

- When an employee discovers a task requires a rank strictly greater than his rank, he will append the task to his supervisor tasks. Note the highest ranking employee may be forced to do tasks that are greater than his rank.

- When an employee discovers he should do a task requiring a rank strictly less than his, he will try to see if the next lower ranking employee can do the task, and if so append the task to that employee tasks.
- When an employee cannot pass the task to the supervisor nor the next lower ranking employee, he will actually execute the task, adding it to the work step list

Example:

[59]:
```
c
```

[59]:
```
Company:
  name  rank  tasks
  x     9     deque([('a', 3), ('b', 5), ('c', 12), ('d', 1)])
  y     6     deque([('e', 8), ('f', 2), ('g', 8)])
  z     2     deque([('h', 10)])
```

[60]:
```
c.work()
```

```
DEBUG: Employee x gives task ('a', 3) to employee y
DEBUG: Employee y gives task ('e', 8) to employee x
DEBUG: Employee z gives task ('h', 10) to employee y
DEBUG: Total performed work this step: []
```

[60]:
```
[]
```

[61]:
```
c
```

[61]:
```
Company:
  name  rank  tasks
  x     9     deque([('b', 5), ('c', 12), ('d', 1), ('e', 8)])
  y     6     deque([('f', 2), ('g', 8), ('a', 3), ('h', 10)])
  z     2     deque([])
```

[62]:
```
c.work()
```

```
DEBUG: Employee x gives task ('b', 5) to employee y
DEBUG: Employee y gives task ('f', 2) to employee z
DEBUG: Employee z executes task ('f', 2)
DEBUG: Total performed work this step: ['f']
```

[62]:
```
['f']
```

[63]:
```
c
```

[63]:
```
Company:
  name  rank  tasks
  x     9     deque([('c', 12), ('d', 1), ('e', 8)])
  y     6     deque([('g', 8), ('a', 3), ('h', 10), ('b', 5)])
  z     2     deque([])
```

[64]:
```
c.work()
```

```
DEBUG: Employee x executes task ('c', 12)
DEBUG: Employee y gives task ('g', 8) to employee x
DEBUG: Total performed work this step: ['c']
```

[64]:
```
['c']
```

[65]:
```
c
```

[65]:
```
Company:
  name  rank  tasks
  x     9     deque([('d', 1), ('e', 8), ('g', 8)])
  y     6     deque([('a', 3), ('h', 10), ('b', 5)])
  z     2     deque([])
```

[66]:
```
c.work()
```

```
DEBUG: Employee x gives task ('d', 1) to employee y
DEBUG: Employee y executes task ('a', 3)
DEBUG: Total performed work this step: ['a']
```

[66]:
```
['a']
```

[67]:
```
c
```

[67]:
```
Company:
  name  rank  tasks
  x     9     deque([('e', 8), ('g', 8)])
  y     6     deque([('h', 10), ('b', 5), ('d', 1)])
  z     2     deque([])
```

[68]:
```
c.work()
```

```
DEBUG: Employee x executes task ('e', 8)
DEBUG: Employee y gives task ('h', 10) to employee x
DEBUG: Total performed work this step: ['e']
```

[68]:
```
['e']
```

[69]:
```
c
```

[69]:
```
Company:
  name  rank  tasks
  x     9     deque([('g', 8), ('h', 10)])
  y     6     deque([('b', 5), ('d', 1)])
  z     2     deque([])
```

[70]:
```
c.work()
```

```
DEBUG: Employee x executes task ('g', 8)
DEBUG: Employee y executes task ('b', 5)
DEBUG: Total performed work this step: ['g', 'b']
```

[70]:
```
['g', 'b']
```

[71]:
```
c
```

[71]:
```
Company:
  name  rank  tasks
  x     9     deque([('h', 10)])
  y     6     deque([('d', 1)])
  z     2     deque([])
```

[72]:
```
c.work()
```

```
DEBUG: Employee x executes task ('h', 10)
DEBUG: Employee y gives task ('d', 1) to employee z
DEBUG: Employee z executes task ('d', 1)
DEBUG: Total performed work this step: ['h', 'd']
```

[72]:
```
['h', 'd']
```

[73]:
```
c
```

[73]:
```
Company:
  name  rank  tasks
  x     9     deque([])
  y     6     deque([])
  z     2     deque([])
```

Now implement this method:

```
def work(self):
    """ Performs a work step and RETURN a list of performed task names.

        For each employee, dequeue its current task from the left and:
        - if the task rank is greater than the rank of the
          current employee, append the task to his supervisor queue
          (the highest ranking employee must execute the task)
        - if the task rank is lower or equal to the rank of the
          next lower ranking employee, append the task to that employee
          queue
        - otherwise, add the task name to the list of
          performed tasks to return
    """
```

Testing: `python3 -m unittest company_test.WorkTest`

7.5.9 7. Concert

Start editing file `concert.py`.

When there are events with lots of potential visitors such as concerts, to speed up check-in there are at least two queues: one for cash where tickets are sold, and one for the actual entrance at the event.

Each visitor may or may not have a ticket. Also, since people usually attend in groups (coupls, families, and so on), in the queue lines each group tends to move as a whole.

In Python, we will model a `Person` as a class you can create like this:

[74]:
```
from concert_sol import *
```

[75]:
```
Person('a', 'x', False)
```

[75]:
```
Person(a,x,False)
```

`a` is the name, `'x'` is the group, and `False` indicates the person doesn't have ticket

To model the two queues, in `Concert` class we have these fields and methods:

```
class Concert:

    def __init__(self):
        self._cash = deque()
        self._entrance = deque()


    def enqc(self, person):
        """ Enqueues at the cash from the right """

        self._cash.append(person)

    def enqe(self, person):
        """ Enqueues at the entrance from the right """

        self._entrance.append(person)
```

7.1 dequeue

⊛⊛⊛ Implement `dequeue`. If you want, you can add debug prints by calling the `debug` function.

```
def dequeue(self):
    """ RETURN the names of people admitted to concert

        Dequeuing for the whole queue system is done in groups, that is,
        with a _single_ call to dequeue, these steps happen, in order:

        1. entrance queue: all people belonging to the same group at
           the front of entrance queue who have the ticket exit the queue
           and are admitted to concert. People in the group without the
           ticket are sent to cash.
        2. cash queue: all people belonging to the same group at the front
           of cash queue are given a ticket, and are queued at the entrance queue
    """
```

Testing: `python3 -m unittest concert_test.DequeueTest`

Example:

[76]:
```
con = Concert()

con.enqc(Person('a','x',False))  # a,b,c belong to same group x
con.enqc(Person('b','x',False))
con.enqc(Person('c','x',False))
con.enqc(Person('d','y',False))  # d belongs to group y
con.enqc(Person('e','z',False))  # e,f belongs to group z
con.enqc(Person('f','z',False))
con.enqc(Person('g','w',False))  # g belongs to group w
```

[77]:
```
con
```

[77]:
```
Concert:
      cash: deque([Person(a,x,False),
                   Person(b,x,False),
                   Person(c,x,False),
```

(continues on next page)

(continued from previous page)

```
                    Person(d,y,False),
                    Person(e,z,False),
                    Person(f,z,False),
                    Person(g,w,False)])
  entrance: deque([])
```

First time we dequeue, entrance queue is empty so no one enters concert, while at the cash queue people in group `x` are given a ticket and enqueued at the entrance queue

NOTE: The messages on the console are just debug print, the function `dequeue` only return name sof people admitted to concert

[78]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  giving ticket to a (group x)
DEBUG:  giving ticket to b (group x)
DEBUG:  giving ticket to c (group x)
DEBUG:  Concert:
              cash: deque([Person(d,y,False),
                           Person(e,z,False),
                           Person(f,z,False),
                           Person(g,w,False)])
          entrance: deque([Person(a,x,True),
                           Person(b,x,True),
                           Person(c,x,True)])
```

[78]:
```
[]
```

[79]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  a (group x) admitted to concert
DEBUG:  b (group x) admitted to concert
DEBUG:  c (group x) admitted to concert
DEBUG:  giving ticket to d (group y)
DEBUG:  Concert:
              cash: deque([Person(e,z,False),
                           Person(f,z,False),
                           Person(g,w,False)])
          entrance: deque([Person(d,y,True)])
```

[79]:
```
['a', 'b', 'c']
```

[80]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  d (group y) admitted to concert
DEBUG:  giving ticket to e (group z)
DEBUG:  giving ticket to f (group z)
DEBUG:  Concert:
              cash: deque([Person(g,w,False)])
          entrance: deque([Person(e,z,True),
                           Person(f,z,True)])
```

[80]:
```
['d']
```

[81]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  e (group z) admitted to concert
DEBUG:  f (group z) admitted to concert
DEBUG:  giving ticket to g (group w)
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([Person(g,w,True)])
```

[81]:
```
['e', 'f']
```

[82]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  g (group w) admitted to concert
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([])
```

[82]:
```
['g']
```

[83]:
```
# calling dequeue on empty lines gives empty list:
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([])
```

[83]:
```
[]
```

Special dequeue case: broken group

In the special case when there is a group at the entrance with one or more members without a ticket, it is assumed that the group gets broken, so whoever has the ticket enters and the others get enqueued at the cash.

[84]:
```
con = Concert()

con.enqe(Person('a','x',True))
con.enqe(Person('b','x',False))
con.enqe(Person('c','x',True))
con.enqc(Person('f','y',False))

con
```

[84]:
```
Concert:
      cash: deque([Person(f,y,False)])
  entrance: deque([Person(a,x,True),
                   Person(b,x,False),
                   Person(c,x,True)])
```

[85]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  a (group x) admitted to concert
DEBUG:  b (group x) has no ticket! Sending to cash
DEBUG:  c (group x) admitted to concert
DEBUG:  giving ticket to f (group y)
```

(continues on next page)

(continued from previous page)

```
DEBUG:  Concert:
              cash: deque([Person(b,x,False)])
          entrance: deque([Person(f,y,True)])
```

[85]:
```
['a', 'c']
```

[86]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  f (group y) admitted to concert
DEBUG:  giving ticket to b (group x)
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([Person(b,x,True)])
```

[86]:
```
['f']
```

[87]:
```
con.dequeue()
```

```
DEBUG:  DEQUEUING ..
DEBUG:  b (group x) admitted to concert
DEBUG:  Concert:
              cash: deque([])
          entrance: deque([])
```

[87]:
```
['b']
```

[88]:
```
con
```

[88]:
```
Concert:
      cash: deque([])
  entrance: deque([])
```

[89]:
```
 m.dequeue()  # no clients left
```

[89]:
```
[]
```

[ ]:
```

```

7.6 Trees

7.6.1 Download exercises zip

(before editing read whole introduction sections 0.x)

Browse files online[350]

[350] https://github.com/DavidLeoni/sciprog-ds/tree/master/trees

7.6.2 0. Introduction

We will deal with both binary and generic trees.

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-sciprog.py
-exercises
     |-trees
         |- trees.ipynb
         |- bin_tree_test.py
         |- bin_tree.py
         |- bin_tree_sol.py
         |- gen_tree_test.py
         |- gen_tree.py
         |- gen_tree_sol.py
```

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), you will edit the files ending in `.py` files
- Go on reading this notebook, and follow instuctions inside.

7.6.3 BT 0. Binary Tree Introduction

BT 0.1 References

See

- Luca Bianco theory here[351]
- Trees on the book[352]
  - In particular, Vocabulary and definitions[353]

[351] https://sciproalgo2019.readthedocs.io/slides/Lecture6.pdf
[352] https://interactivepython.org/runestone/static/pythonds/Trees/toctree.html
[353] https://interactivepython.org/runestone/static/pythonds/Trees/VocabularyandDefinitions.html

BT 0.2 Terminology - relations

BT 0.3 Terminology - levels

Depth of a node

The length of the simple path from the root to the node (measured in number of edges)

Level

The set of nodes having the same depth

Height of the tree

The maximum depth of all its leaves

BT 0.4 Terminology - shapes

Binary tree

A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

Note: *Two trees T and U having the same nodes, the same children for each node and the same root, are said to be different if a node u is a left child of a node v in T and a right child of the same node in U.*

In this worksheet we are first going to provide an implementation of a `BinaryTree` class:

- Differently from the `LinkedList`, which actually had two classes `Node` and `LinkedList` that was pointing to the first node, in this case we just have one `BinaryTree` class.
- Each `BinaryTree` instance *may* have a left `BinaryTree` instance and *may* have a right `BinaryTree` instance, while absence of a branch is marked with `None`. This reflects the recursive nature of trees.
- To grow a tree, first you need to create an instance of `BinaryTree`, and then you call `.insert_left` or `.insert_right` methods on it and pass *data*. Keep reading to see how to do it.

BT 0.2 Code skeleton

Look at the files:

- `trees/bin_tree.py` : the exercise to edit
- `trees/bin_tree_test.py`: the tests to run. Do not modify this file.

Before starting to implement methods in `BinaryTree` class, read all the following sub sections (starting with '0.x')

BT 0.3 Building trees

Let's learn how to build `BinaryTree`. For these trials, feel free to launch a Python 3 interpreter and load this module:

[2]:
```
from bin_tree_sol import *
```

BT 0.3.1 Pointers

A `BinaryTree` class holds 2 pointers that link it to other nodes: `_left`, and `_right`

It also holds a value `data` which is provided by the user to store arbitrary data (could be ints, strings, lists, even other trees, we don't care):

```
class BinaryTree:

    def __init__(self, data):
        self._data = data
        self._left = None
        self._right = None
```

NOTE: `BinaryTree` as defined here is unidirectional, that is, has no backlinks (so no `_parent` field).

Formally, a tree as described in discrete mathematics books is always unidirectional (can't have any cycle) and every node can have at most one incoming link. When we program, though, for convenience we may decide to have or not have backlinks (later with `GenericTree` we will see an example)

To create a `BinaryTree` of one node, just call the constructor passing whatever you want like this:

[3]:
```
tblah = BinaryTree("blah")
tn = BinaryTree(5)
```

Note that with the provided constructor you can't pass children.

BT 0.3.2 Building with `insert_left`

To grow a `BinaryTree`, as basic building block you will have to implement `insert_left`:

```
def insert_left(self, data):
    """ Takes as input DATA (*NOT* a node !!) and MODIFIES current
        node this way:

        - First creates a new BinaryTree (let's call it B) into which
          provided data is wrapped.
        - Then:
            - if there is no left node in self, new node B is attached to
              the left of self
            - if there already is a left node L, it is substituted by
              new node B, and L becomes the left node of B
    """
```

You can call it like this:

[4]:
```
t = BinaryTree('a')

t.insert_left('c')
```

[5]:
```
print(t)
```

```
a
├c
└
```

[6]:
```
t.insert_left('b')
```

[7]:
```
print(t)
```

```
a
├b
│├c
│└
└
```

[8]:
```
t.left().data()
```

[8]:
```
'b'
```

[9]:
```
t.left().left().data()
```

[9]:
```
'c'
```

BT 0.3.3 Building with `bt`

If you need to test your data structure, we provide you with this handy function `bt` in `bin_tree_test` module that allows to easily construct trees from other trees.

> **WARNING:** DO NOT USE `bt` inside your implementation code !!!! `bt` is just meant for testing.

```
def bt(*args):
    """ Shorthand function that returns a GenericTree containing the provided
        data and children. First parameter is the data, the following ones are the␣
↪children.
```

[10]:
```
from bin_tree_test import bt

bt('a')
print(bt('a'))
```

```
a
```

[11]:
```
print(bt('a', None, bt('b')))
```

```
a
├
└b
```

[12]:
```
print(bt('a', bt('b'), bt('c')))
```

(continues on next page)

(continued from previous page)

```
a
├b
└c
```

[13]:
```
print(bt('a', bt('b'), bt('c', bt('d'), None)) )
```

```
a
├b
└c
 ├d
 └
```

7.6.4 BT 1. Insertions

BT 1.1 insert_left

Implement `insert_left`

```
def insert_left(self, data):
        """ Takes as input DATA (*NOT* a node !!) and MODIFIES current node
            this way:

            - First creates a new BinaryTree (let's call it B) into which
              provided data is wrapped.
            - Then:
                - if there is no left node in self, new node B is attached to
                  the left of self
                - if there already is a left node L, it is substituted by
                  new node B, and L becomes the left node of B
```

Testing: `python3 -m unittest bin_tree_test.InsertLeftTest`

BT 1.2 insert_right

```
def insert_right(self, data):
        """ Takes as input DATA (*NOT* a node !!) and MODIFIES current node
            this way:

            - First creates a new BinaryTree (let's call it B) into which
              provided data is wrapped.
            - Then:
                - if there is no right node in self, new node B is attached
                  to the right of self
                - if there already is a right node L, it is substituted by
                  new node B, and L becomes the right node of B
        """
```

Testing: `python3 -m unittest bin_tree_test.InsertRightTest`

7.6.5 BT 2. Recursive visit

In these exercises, we are going to implement methods which do *recursive* calls. Before doing it, we should ask oursevles why. Tyipically, recursive calls are present in funcitonal languages. Is Python one of them? Python is a general purpose language, that allows writing imperative, object-oriented code and also sports *some*, but not *all* functional programming features. Unfortunately, one notably missing feature is the capability to efficiently perform recursive calls. If too many recursive calls happen, you will probably get a 'Recursion limit exceed' error. So why should we bother?

It turns out that recursive code is much shorter and elegant than corrisponding imperative one (which would often use stacks). So to gain a first understanding of problems, it might be beneficial to think about a recursive solution. After that, we may increase efficiency by explicitly using a stack instead of recursive calls.

BT 2.1 sum_rec

Supposing all nodes hold a number, let's see how to write a method that returns the sum of all numbers in the tree. We can define sum recursively:

- if a node has no children: the sum is equal to the node data.
- if a node has only left child: the sum is equal to the node data plus the (recursive) sum of left child
- if a node has only right child: the sum is equal to the node data plus the (recursive) sum of right child
- if a node has both left and right child: the sum is equal to the node data plus the (recursive) sum of left child and the (recursive) sum of the right child

Example: black numbers are node data, purple numbers are the respective sums.

Let's look at node with black number `10`: its sum is `23`, which is given by its data `10`, plus `1` (the recursive sum of the left child `1`), plus `12` (recursive sum of the right child `7`)

```
def sum_rec(self):
    """ Supposing the tree holds integer numbers in all nodes,
        RETURN the sum of the numbers.

        - implement it as a recursive Depth First Search (DFS) traversal
          NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.ContainsRecTest`

Code example:

[14]:
```
t = bt(3,
            bt(10,
                 bt(1),
                 bt(7,
                        bt(5))),
            bt(9,
                 bt(6,
                         bt(2,
                                None,
                                bt(4)),
                         bt(8))))
print(t)
```

```
3
├10
│├1
│└7
│ ├5
│ └
└9
 ├6
 │├2
 ||├
 ||└4
 │└8
 └
```

[15]:
```
t.sum_rec()
```

[15]:
```
55
```

BT 2.2 height_rec

Let's say we want to know the height a tree, which is defined as 'the maximum depth of all the leaves'. We can think recursively as:

- the height of a node without children is 0
- the height of a node with only a left child is the height of the left node plus one
- the height of a node with only a right child is the height of the right node plus one
- the height of a node with both left and right children is the maximum of the height of the left node and height of the right node, plus one

Look at the example and try to convince yourself this makes sense:

- in purple you see nodes corresponding heights
- notice how leaves have all height 0

```
def height_rec(self):
    """ RETURN an integer which is the height of the tree

        - implement it as recursive call which does NOT modify the tree
          NOTE: with big trees a recursive solution would surely exceed
                the call stack, but here we don't mind
        - A tree with only one node has height zero.
```

Testing: `python3 -m unittest bin_tree_test.HeightRecTest`

BT 2.3 depth_rec

```
def depth_rec(self, level):
    """
        - MODIFIES the tree by putting in the data field the provided
          value level (which is an integer),
          and recursively calls itself on left and right nodes
          (if present) passing level + 1
        - implement it as a recursive Depth First Search (DFS) traversal
          NOTE: with big trees a recursive solution would surely exceed
                the  call stack, but here we don't mind
        - The root of a tree has depth zero.
        - does not return anything
```

Testing: `python3 -m unittest bin_tree_test.DepthDfsTest`

Example: For example, if we take this tree:

[16]:
```
t = bt('a', bt('b', bt('c'), None), bt('d', None, bt('e', bt('f'))))

print(t)
```

```
a
├b
│├c
│└
└d
 ├
 └e
  ├f
  └
```

After a call do `depth_rec` on `t` passing 0 as starting level, all letters will be substituted by the tree depth at that point:

[17]:
```
t.depth_rec(0)
```

[18]:
```
print(t)
```

```
0
├1
│├2
│└
└1
 ├
 └2
  ├3
  └
```

BT 2.4 contains_rec

```
def contains_rec(self, item):
    """ RETURN True if at least one node in the tree has data equal
        to item,  otherwise RETURN False.

        - implement it as a recursive Depth First Search (DFS) traversal
          NOTE: with big trees a recursive solution would surely exceed
                the  call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.ContainsRecTest`

Example:

[19]:
```
t = bt('a',
            bt('b',
                    bt('c'),
                    bt('d',
                            None,
                            bt('e'))),
            bt('f',
                    bt('g',
                            bt('h')),
                    bt('i')))
```

[20]:
```
print(t)
```

```
a
├b
│├c
│└d
│ ├
│ └e
└f
 ├g
 │├h
 │└
 └i
```

[21]:
```
t.contains_rec('g')
```

[21]:
```
True
```

[22]:
```
t.contains_rec('z')
```

[22]:
```
False
```

BT 2.5 join_rec

```
def join_rec(self):
    """ Supposing the tree nodes hold a character each, RETURN a STRING
        holding all characters IN-ORDER

        - implement it as a recursive Depth First Search (DFS) traversal
          NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.JoinRecTest`

[23]:
```
t = bt('e',
            bt('b',
                    bt('a'),
                    bt('c',
                            None,
                            bt('d'))),
            bt('h',
                    bt('g',
                            bt('f')),
                    bt('i')))
```

[24]:
```
print(t)
```

```
e
├b
│├a
│└c
│ ├
│ └d
└h
 ├g
```

(continues on next page)

(continued from previous page)

```
 │├f
 │└
 └i
```

[25]:
```
t.join_rec()
```

[25]:
```
'abcdefghi'
```

BT 2.6 fun_rec

```
def fun_rec(self):
    """ Supposing the tree nodes hold expressions which can either be
        functions or single variables, RETURN a string holding
        the complete formula with needed parenthesis.

        - implement it as a recursive Depth First Search (DFS)
          PRE-ORDER visit
        - NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.FunRecTest`

Example:

[26]:
```
t = bt('f',
            bt('g',
                    bt('x'),
                    bt('y')),
            bt('f',
                    bt('h',
                            bt('z')),
                    bt('w')))
```

[27]:
```
print(t)
```

```
f
├g
│├x
│└y
└f
 ├h
 │├z
 │└
 └w
```

[28]:
```
t.fun_rec()
```

[28]:
```
'f(g(x,y),f(h(z),w))'
```

BT 2.7 bin_search_rec

You are given a so-called binary search tree, which holds numbers as data, and all nodes respect this constraint:

- if a node A holds a number strictly less than the number held by its parent node B, then node A must be a left child of B
- if a node C holds a number greater or equal than its parent node B, then node C must be a right child of B

[29]:
```
t = bt(7,
              bt(3,
                     bt(2),
                     bt(6)),
              bt(12,
                     bt(8,
                            None,
                            bt(11,
                                   bt(9))),
                     bt(14,
                            bt(13))))
print(t)
```

```
7
├3
│├2
│└6
└12
 ├8
 │├
```

(continues on next page)

(continued from previous page)

```
 |└11
 | ├9
 | └
 └14
  ├13
  └
```

Implement following method:

```
def bin_search_rec(self, m):
    """ Assuming the tree is a binary search tree of integer numbers,
        RETURN True if m is present in the tree, False otherwise

        - MUST EXECUTE IN O(height(t))
        - NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
    raise Exception("TODO IMPLEMENT ME !")
```

- **QUESTION**: what is the complexity in worst case scenario?
- **QUESTION**: what is the complexity when tree is balanced?

Testing: `python3 -m unittest bin_tree_test.BinSearchRecTest`

BT 2.8 bin_insert_rec

```
def bin_insert_rec(self, m):
    """ Assuming the tree is a binary search tree of integer numbers,
        MODIFIES the tree by inserting a new node with the value m
        in the appropriate position. Node is always added as a leaf.

        - MUST EXECUTE IN O(height(t))
        - NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.BinInsertRecTest`

Example:

[30]:
```
t = bt(7)
print(t)
```

```
7
```

[31]:
```
t.bin_insert_rec(3)
print(t)
```

```
7
├3
└
```

[32]:
```
t.bin_insert_rec(6)
print(t)
```

```
7
├3
│├
│└6
└
```

[33]:
```
t.bin_insert_rec(2)
print(t)
```

```
7
├3
│├2
│└6
└
```

[34]:
```
t.bin_insert_rec(12)
print(t)
```

```
7
├3
│├2
│└6
└12
```

[35]:
```
t.bin_insert_rec(14)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├
 └14
```

[36]:
```
t.bin_insert_rec(13)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├
 └14
  ├13
  └
```

[37]:
```
t.bin_insert_rec(8)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├8
 └14
```

(continues on next page)

(continued from previous page)

```
  ├13
  └
```

[38]:
```
t.bin_insert_rec(11)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├8
 │├
 │└11
 └14
  ├13
  └
```

[39]:
```
t.bin_insert_rec(9)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├8
 │├
 │└11
 │ ├9
 │ └
 └14
  ├13
  └
```

BT 2.9 univalued_rec

```
def univalued_rec(self):
    """ RETURN True if the tree is univalued, otherwise RETURN False.

        - a tree is univalued when all nodes have the same value as data
        - MUST execute in O(n) where n is the number of nodes of the tree
        - NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.UnivaluedRecTest`

Example:

[40]:
```
t = bt(3, bt(3), bt(3, bt(3, bt(3, None, bt(3)))))
print(t)
```

```
3
├3
```

(continues on next page)

(continued from previous page)

```
└3
 ├3
 │├3
 │└
 │└3
 │└
 └
```

[41]:
```
t.univalued_rec()
```

[41]:
```
True
```

[42]:
```
t = bt(2, bt(3), bt(6, bt(3, bt(3, None, bt(3)))))
print(t)
```

```
2
├3
└6
 ├3
 │├3
 │└
 │└3
 │└
 └
```

[43]:
```
t.univalued_rec()
```

[43]:
```
False
```

BT 2.10 same_rec

```
def same_rec(self, other):
    """ RETURN True if this binary tree is equal to other binary tree,
        otherwise return False.

        - MUST execute in O(n) where n is the number of nodes of the tree
        - NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
        - HINT: defining a helper function

                def helper(t1, t2):

                which recursively calls itself and assumes both of the
                inputs can be None may reduce the number of ifs to write.
    """
```

Testing: `python3 -m unittest bin_tree_test.SameRecTest`

7.6.6 BT 3. Stack visit

To avoid getting 'Recursion limit exceeded' errors which can happen with Python, instead of using recursion we can implement tree operations with a while cycle and a stack (or a queue, depending on the case).

Typically, in these algorithms you follow this recipe:

- at the beginning you put inside the stack the current node on which the method is called
- you keep executing the while until the stack is empty
- inside the while, you pop the stack and do some processing on the popped node data
- if the node has children, you put them on the stack

We will try to reimplement this way methods we've already seen.

BT 3.1 sum_stack

Implement `sum_stack`

```
def sum_stack(self):
    """ Supposing the tree holds integer numbers in all nodes,
        RETURN the sum of the numbers.

        - DO *NOT* use recursion
        - implement it with a while and a stack (as a python list)
        - In the stack place nodes to process
    """
```

Testing: `python3 -m unittest bin_tree_test.SumStackTest`

BT 3.3 height_stack

The idea of this function is not that different from the Tasks do_level exercise[354] we've seen in the lab about stacks

```
def height_stack(self):
    """ RETURN an integer which is the height of the tree

        - A tree with only one node has height zero.
        - DO *NOT* use recursion
        - implement it with a while and a stack (as a python list).
        - In the stack place *tuples* holding a node *and* its level

    """
```

Testing: `python3 -m unittest bin_tree_test.HeightStackTest`

BT 3.3 others

Hopefully you got an idea of how stack recursion works, now you could try to implement by yourself previously defined recursive functions, this time using a while and a stack (or a queue, depending on what you are trying to achieve).

[354] https://sciprog.davidleoni.it/stacks/stacks.html#5.2-do_level

7.6.7 BT Further resources

See Trees exercises[355] on LeetCode (sort by easy difficulty), for example:

- univalued[356]
- same_tree[357] (give recursive solution)
- Sum range of BST[358]

7.6.8 GT 0. Generic Tree Introduction

See Luca Bianco Generic Tree theory[359]

In this worksheet we are going to provide an implementation of a `GenericTree` class:

- Why `GenericTree` ? Because many object hierarchies in real life tend to have many interlinked pointers this, in one form or another
- Differently from the `LinkedList`, which actually had two classes `Node` and `LinkedList` that was pointing to the first node, in this case we just have one `GenericTree` class. So to grow a tree like the above one in the picture, for each of the boxes that you see we will need to create one instance of `GenericTree` and link it to the other instances.
- Ordinary simple trees just hold pointers to the children. In this case, we have an enriched tree which holds ponters also up to the *parent* and on the right to the *siblings*. Whenever we are going to manipulate the tree, we need to take good care of updating these pointers.

[355] https://leetcode.com/tag/tree/
[356] https://leetcode.com/problems/univalued-binary-tree/
[357] https://leetcode.com/problems/same-tree/
[358] https://leetcode.com/problems/range-sum-of-bst/
[359] https://sciproalgo2019.readthedocs.io/slides/Lecture6.pdf

Do we need sidelinks and backlinks ?:

Here we use sidelinks and backlinks like `_sibling` and `_parent` for exercise purposes, but keep in mind such extra links need to be properly managed when you write algorithms and thus increase the likelihood of introducing bugs.

As a general rule of thumb, if you are to design a data structure, always first try to start making it unidirectional (like for example the BinaryTree we've seen before). Then, if you notice you really need extra links (for example to quickly traverse a tree from a node up to the root), you can always add them in a later development iteration.

ROOT NODE: In this context, we call a node *root* if has no incoming edges *and* it has no parent nor sibling

DETACHING A NODE: In this context, when we *detach* a node from a tree, the node becomes the *root* of a new tree, which means it will have no link anymore with the tree it was in.

GT 0.2 Code skeleton

Look at the files:

- `trees/gen_tree.py` : the exercise to edit
- `trees/gen_tree_test.py`: the tests to run. Do not modify this file.

Before starting to implement methods in `GenericTree` class, read all the following sub sections (starting with '0.x')

GT 0.3 Building trees

Let's learn how to build `GenericTree`. For these trials, feel free to launch a Python 3 interpreter and load this module:

[44]:
```
from gen_tree_sol import *
```

GT 0.3.1 Pointers

A `GenericTree` class holds 3 pointers that link it to the other nodes: `_child`, `_sibling` and `_parent`. So this time we have to manage more pointers, in particular beware of the `_parent` one which as a matter of fact creates cycles in the structure.

It also holds a value `data` which is provided by the user to store arbitrary data (could be ints, strings, lists, even other trees, we don't care):

```
class GenericTree:

    def __init__(self, data):
        self._data = data
        self._child = None
        self._sibling = None
        self._parent = None
```

To create a tree of one node, just call the constructor passing whatever you want like this:

[45]:
```
tblah = GenericTree("blah")
tn = GenericTree(5)
```

Note that with the provided constructor you can't pass children.

GT 0.3.2 Building with `insert_child`

To grow a `GenericTree`, as basic building block you will have to implement `insert_child`:

```
def insert_child(self, new_child):
    """ Inserts new_child at the beginning of the children sequence. """
```

> **WARNING**: here we insert a **node** !!
>
> Differently from the `BinaryTree`, this time instead of passing *data* we pass a *node*. This can cause more troubles than before, as when we add a `new_child` we must be careful it doesn't have wrong pointers. For example, think the case when you insert node B as child of node A, but by mistake you previously set B `_child` field to point to A. Such a cycle would not be a tree anymore and would basically disrupt any algorithm you would try to run.

You can call it like this:

[46]:
```
ta = GenericTree('a')
print(ta)    # 'a' is the root
```

```
a
```

[47]:
```
tb = GenericTree('b')
ta.insert_child(tb)
print(ta)
```

```
a
└b
```

```
a     'a' is the root
└b    'b' is the child . The '└' means just that it is also the last child of the␣
↪siblings sequence
```

[48]:
```
tc = GenericTree('c')
ta.insert_child(tc)
print(ta)
```

```
a
├c
└b
```

```
a           # 'a' is the root
├c          # 'c' is inserted as the first child (would be shown on the left in the␣
↪graph image)
└b          # 'b' is now the next sibling of c  The '\' means just that it
            #  is also the last child of the siblings sequence
```

[49]:
```
td = GenericTree('d')
tc.insert_child(td)
print(ta)
```

```
a
├c
│└d
└b
```

```
a          # 'a' is the root
├c         # 'c' is the first child of 'a'
│└d        # 'd' is the first child of 'c'
└b         # 'b' is the next sibling of c
```

GT 0.3.3 Building with `gt`

If you need to test your data structure, we provide you with this handy function `gt` in `gen_tree_test` module that allows to easily construct trees from other trees.

> **WARNING:** DO NOT USE `gt` inside your implementation code !!!! `gt` is just meant for testing.

```
def gt(*args):
    """ Shorthand function that returns a GenericTree containing the provided
        data and children. First parameter is the data, the following ones are the␣
↪children.
```

[50]:
```
# first remember to import it from gen_tree_test:

from gen_tree_test import gt

# NOTE: this function is _not_ a class method, you can directly invoke it like this:
print(gt('a'))
```

```
a
```

[51]:
```
# NOTE: the external call gt('a', ......... )  INCLUDES gt('b') and gt('c') in the␣
↪parameters !

print(gt('a', gt('b'), gt('c')))
```

```
a
├b
└c
```

GT 0.4 Displaying trees side by side with `str_trees`

If you have a couple of trees, like the actual one you get from your method calls and the one you expect, it might be useful to display them side by side with the `str_trees` method in `gen_tree_test` module:

[52]:
```
# first remember to import it:

from gen_tree_test import str_trees

# NOTE: this function is _not_ a class method, you can directly invoke it like this:
print(str_trees(gt('a', gt('b')), gt('x', gt('y'), gt('z'))))
```

```
ACTUAL     EXPECTED
a          x
└b         ├y
           └z
```

GT 0.5 Look at the tests

Have a look at the `gen_tree_test.py` file header, notice it imports `GenericTree` class from exercises file `gen_tree`:

```
from gen_tree import *
import unittest
```

GT 0.6 Look at gen_tree_test.GenericTreeTest

Have a quick look at `GenericTreeTest` definitions inside `gen_tree_test` :

```
class GenericTreeTest(unittest.TestCase):

    def assertReturnNone(self, ret, function_name):
        """ Asserts method result ret equals None """

    def assertRoot(self, t):
        """ Checks provided node t is a root, if not raises Exception """

    def assertTreeEqual(self, t1, t2):
        """ Asserts the trees t1 and t2 are equal """
```

We see we added extra asserts you will later find used around in test methods. Of these ones, the most important is `assertTreeEqual`: when you have complex data structures like trees, it is helpful being able to compare the tree you obtain from your method calls to the tree you expect. This assertion we created provides a way to quickly display such differences.

7.6.9 GT 1 Implement basic methods

Start editing `gen_tree.py`, implementing methods in `GenericTree` in the order you find them in the next points.

IMPORTANT: All methods and functions without written inside `raise Exception("TODO IMPLEMENT ME!")` are already provided and you don't need to edit them !

GT 1.1 insert_child

Implement method `insert_child`, which is the basic building block for our `GenericTree`:

> **WARNING**: here we insert a **node** !!
>
> Differently from the `BinaryTree`, this time instead of passing *data* we pass a *node*. This implies that inside the `insert_child` method you will have to take care of pointers of `new_child`: for example, you will need to set the `_parent` pointer of `new_child` to point to the current node you are attaching to (that is, `self`)

```
def insert_child(self, new_child):
    """ Inserts new_child at the beginning of the children sequence. """
```

IMPORTANT: before proceding, make sure the tests for it pass by running:

`python3 -m unittest gen_tree_test.InsertChildTest`

QUESTION: Look at the tests, they are quite thourough and verbose. Why ?

GT 1.2 insert_children

Implement `insert_children`:

```
def insert_children(self, new_children):
    """ Takes a list of children and inserts them at the beginning of the
        current children sequence,

        NOTE: in the new sequence new_children appear in the order they
              are passed to the function!


        For example:
            >>> t = gt('a', gt('b'), gt('c'))
            >>> print t

            a
            ├b
            └c

            >>>  t.insert_children([gt('d'), gt('e')])
            >>> print t

            a
            ├d
            ├e
            ├b
            └c
    """
```

HINT 1: try to reuse `insert_child`, but note it inserts only to the left. Calling it on the input sequence you would get wrong ordering in the tree.

> **WARNING**: Function description does not say anything about changing the input `new_children`, so users calling your method don't expect you to modify it ! However, you can internally produce a new Python list out of the input one, if you wish to.

Testing: `python3 -m unittest gen_tree_test.InsertChildrenTest`

GT 1.3 insert_sibling

Implement `insert_sibling`:

```
def insert_sibling(self, new_sibling):
    """ Inserts new_sibling as the *immediate* next sibling.

        If self is a root, raises an Exception
    """
```

Testing: `python3 -m unittest tree_test.InsertSiblingTest`

Examples:

[53]:
```
tb = gt('b')
ta = gt('a', tb, gt('c'))
print(ta)
```

```
a
├b
└c
```

[54]:
```
tx = gt('x', gt('y'))
print(tx)
```

```
x
└y
```

[55]:
```
tb.insert_sibling(tx)
print(ta)
```

```
a
├b
├x
│└y
└c
```

QUESTION: if you call `insert_sibling` an a root node such as `ta`, you should get an Exception. Why? Does it make sense to have parentless brothers ?

```
ta.insert_sibling(g('z'))
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-35-a1e4ba8b1ee5> in <module>()
----> 1 ta.insert_sibling(gt('z'))

~/Da/prj/sciprolab2/prj/trees/tree_sol.py in insert_sibling(self, new_sibling)
    128         """
    129         if (self.is_root()):
--> 130             raise Exception("Can't add siblings to a root node !!")
    131
    132         new_sibling._parent = self._parent

Exception: Can't add siblings to a root node !!
```

GT 1.4 insert_siblings

Testing: `python3 -m unittest tree_test.InsertSiblingsTest`

GT 1.5 detach_child

QUESTION: does a detached child have still any parent or sibling ?

Testing: `python3 -m unittest tree_test.DetachChildTest`

GT 1.6 detach_sibling

Testing: `python3 -m unittest tree_test.DetachSiblingTest`

GT 1.7 detach

Testing: `python3 -m unittest tree_test.DetachTest`

GT 1.8 ancestors

Implement `ancestors`:

```
def ancestors(self):
        """ Return the ancestors up until the root as a Python list.
            First item in the list will be the parent of this node.

            NOTE: this function return the *nodes*, not the data.
        """
```

(continues on next page)

(continued from previous page)

```
        raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python3 -m unittest gen_tree_test.AncestorsTest`

Examples:

- ancestors of p: f, b, a
- ancestors of h: c, a
- ancestors of a: empty list

7.6.10 GT 2 Implement more complex functions

After you understood well and implemented the previous methods, you can continue with the following ones:

GT 2.1 `grandchildren`

Implement the `grandchildren` method. *NOTE*: it returns the data inside the nodes, NOT the nodes !!!!!

```
def grandchildren(self):
    """ Returns a python list containing the data of all the
        grandchildren of this node.

        - Data must be from left to right order in the tree horizontal
          representation (or up to down in the vertical representation).
        - If there are no grandchildren, returns an empty array.

        For example, for this tree:

        a
        ├b
        │├c
        │├d
        │ └g
        ├e
        ├f
         └h

        Returns ['c','d','h']
    """
```

Testing: `python3 -m unittest gen_tree_test.ZagTest`

Examples:

[56]:
```
ta = gt('a', gt('b', gt('c')))
print(ta)
```

```
a
└b
 └c
```

[57]:
```
print(ta.grandchildren())
```

```
['c']
```

[58]:
```
ta = gt('a', gt('b'))
print(ta)
```

```
a
└b
```

[59]:
```
print(ta.grandchildren())
```

```
[]
```

[60]:
```
ta = gt('a', gt('b', gt('c'), gt('d')), gt('e', gt('f')) )
print(ta)
```

```
a
├b
│ ├c
│ └d
└e
  └f
```

[61]:
```
print(ta.grandchildren())
```

```
['c', 'd', 'f']
```

GT 2.2 Zig Zag

Here you will be visiting a generic tree in various ways.

GT 2.2.1 zig

The method zig must return as output a list of data of the root and all the nodes in the chain of child attributes. Basically, you just have to follow the red lines and gather data in a list, until there are no more red lines to follow.

Testing: `python3 -m unittest tree_test.ZigTest`

Examples: in the labeled tree in the image, these would be the results of calling zig on various nodes:

```
From a: ['a','b', 'e']
From b: ['b', 'e']
From c: ['c', 'g']
From h: ['h']
From q: ['h']
```

GT 2.2.2 zag

This function is quite similar to zig, but this time it gathers data going right, along the sibling arrows.

Testing: `python3 -m unittest gen_tree_test.ZagTest`

Examples: in the labeled tree in the image, these would be the results of calling zag on various nodes:

```
From a : ['a']
From b : ['b', 'c', 'd']
From o : ['o', 'p']
```

GT 2.2.3 zigzag

As you are surely thinking, zig and zag alone are boring. So let's mix the concepts, and go zigzaging. This time you will write a function zigzag, that first zigs collecting data along the child vertical red chain as much as it can. Then, if the last node links to at least a sibling, the method continues to collect data along the siblings horizontal chain as much as it can. At this point, if it finds a child, it goes zigging again along the child vertical red chain as much as it can, and then horizontal zaging, and so on. It continues zig-zaging like this until it reaches a node that has no child nor sibling: when this happens returns the list of data found so far.

Testing: `python3 -m unittest tree_test.ZigZagTest`

Examples: in the labeled tree in the image, these would be the results of calling zigzag on various nodes:

```
From a: ['a', 'b', 'e', 'f', 'o']
From c: ['c', 'g', 'h', 'i', 'q'] NOTE: if node h had a child z, the process would␣
↪still proceed to i
From d: ['d', 'm', 'n']
From o: ['o', 'p']
From n: ['n']
```

GT 2.3 uncles

Implement the `uncles` method:

```
def uncles(self):
    """ RETURN a python list containing the data of all the uncles
        of this node (that is, *all* the siblings of its parent).

        NOTE: returns also the father siblings which are *BEFORE*
              the father !!

        - Data must be from left to right order in the tree horizontal
          representation (or up to down in the vertical representation)
        - If there are no uncles, returns an empty array.

        For example, for this tree:

        a
        ├b
        │├c
        │├d
        │ ├g
        ├e
        │├h
        ├f

        calling this method on 'h' returns ['b','f']
    """
```

Testing: `python3 -m unittest gen_tree_test.UnclesTest`

Example usages:

[62]:
```
td = gt('d')
tb = gt('b')
ta = gt('a', tb,  gt('c', td), gt('e'))
print(ta)
```

```
a
├b
├c
│└d
└e
```

[63]:
```
print(td.uncles())
```

```
['b', 'e']
```

[64]:
```
print(tb.uncles())
```

```
[]
```

GT 2.4 common_ancestor

Implement the method `common_ancestor`:

```
def common_ancestor(self, gt2):
    """ RETURN the first common ancestor of current node and the provided
        gt2 node

        - If gt2 is not a node of the same tree, raises LookupError

        NOTE: this function returns a *node*, not the data.

        Ideally, this method should perform in O(h) where h is the height
        of the tree.

        HINT: you should use a Python Set). If you can't figure out how
              to make it that fast, try to make it at worst O(h^2)

    """

    raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python3 -m unittest gen_tree_test.CommonAncestorTest`

Examples:

- common ancestor of g and i: tree rooted at c
- common_ancestor of g and q: tree rooted at c
- common_ancestor of e and d: tree rooted at a

GT 2.5 mirror

```
def mirror(self):
    """ Modifies this tree by mirroring it, that is, reverses the order
        of all children of this node and of all its descendants

        - MUST work in O(n) where n is the number of nodes
        - MUST change the order of nodes, NOT the data (so don't touch the
              data !)
        - DON'T create new nodes
        - It is acceptable to use a recursive method.


        Example:

        a      <-    Becomes:    a
        ⍰b                       ⍰i
        ⍰⍰c                      ⍰e
        ⍰⍰d                      ⍰⍰h
        ⍰e                       ⍰⍰g
        ⍰⍰f                      ⍰⍰f
        ⍰⍰g                      ⍰b
        ⍰⍰h                        ⍰d
        ⍰i                         ⍰c

    """
```

Testing: `python3 -m unittest gen_tree_test.MirrorTest`

GT 2.6 clone

Implement the method clone:

```
def clone(self):
    """ Clones this tree, by returning an *entirely* new tree which is an
        exact copy of this tree (so returned node and *all* its descendants
        must be new).

        - MUST run in O(n) where n is the number of nodes
        - a recursive method is acceptable.
    """

    raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python3 -m unittest gen_tree_test.CloneTest`

GT 2.7 rightmost

In the example above, the rightmost branch of `a` is given by the node sequence `a`,`d`,`n`

Implement this method:

```
def rightmost(self):
        """ RETURN a list containing the *data* of the nodes
            in the *rightmost* branch of the tree.

            Example:

            a
            ├─b
            ├─c
            │ └─e
            └─d
              ├─f
              ├─g
              │ ├─h
              │ └─i

            should give

            ['a','d','g','i']
        """
```

Testing: `python3 -m unittest gen_tree_test.RightmostTest`

GT 2.8 fill_left

Open `tree.py` and implement `fill_left` method:

```
def fill_left(self, stuff):
    """ MODIFIES the tree by filling the leftmost branch data
        with values from provided array 'stuff'

        - if there aren't enough nodes to fill, raise ValueError
        - root data is not modified
        - *DO NOT* use recursion

    """
```

Testing: `python3 -m unittest gen_tree_test.FillLeftTest`

Example:

[65]:
```
from gen_tree_test import gt
from gen_tree_sol import *
```

[66]:
```
t  = gt('a',
            gt('b',
                    gt('e',
                            gt('f'),
                            gt('g',
                                    gt('i')),
                    gt('h')),
            gt('c'),
            gt('d')))
```

[67]:
```
print(t)
```

```
a
└b
 ├e
 │├f
 │├g
 │└i
 │└h
 ├c
 └d
```

[68]:
```
t.fill_left(['x','y'])
```

[69]:
```
print(t)
```

```
a
└x
 ├y
 │├f
 │├g
 │└i
 │└h
 ├c
 └d
```

[70]:
```
t.fill_left(['W','V','T'])
print(t)
```

```
a
└W
 ├V
 │├T
 │├g
 │└i
 │└h
 ├c
 └d
```

GT 2.9 follow

Open `tree.py` and implement `follow` method:

```
def follow(self, positions):
        """
            RETURN an array of node data, representing a branch from the
            root down to a certain depth.
            The path to follow is determined by given positions, which
            is an array of integer indeces, see example.

            - if provided indeces lead to non-existing nodes, raise ValueError
            - IMPORTANT: *DO NOT* use recursion, use a couple of while instead.
            - IMPORTANT: *DO NOT* attempt to convert siblings to
                         a python list !!!! Doing so will give you less points!

        """
```

Example:

```
               level  01234

                      a
                      ├b
                      ├c
                      │└e
                      │ ├f
                      │ ├g
                      │ │└i
                      │ └h
                      └d

                     RETURNS
t.follow([])         [a]           root data is always present
t.follow([0])        [a,b]         b is the 0-th child of a
t.follow([2])        [a,d]         d is the 2-nd child of a
t.follow([1,0,2])    [a,c,e,h]     c is the 1-st child of a
                                   e is the 0-th child of c
                                   h is the 2-nd child of e
t.follow([1,0,1,0]) [a,c,e,g,i]   c is the 1-st child of a
                                   e is the 0-th child of c
                                   g is the 1-st child of e
                                   i is the 0-th child of g
```

Testing: `python3 -m unittest gen_tree_test.FollowTest`

GT 2.10 is_triangle

A *triangle* is a node which has *exactly* two children.

Let's see some example:

```
      a
    /   \
   /     \
  b ----- c
 /|\     /
d-e-f   g
       / \
      h---i
         /
        l
```

The tree above can also be represented like this:

```
a
├b
│├d
│├e
│└f
└c
 └g
  ├h
  └i
   └l
```

- node `a` is a triangle because has *exactly* two children `b` and `c`, note it doesn't matter if `b` or `c` have children)
- `b` is *not* a triangle (has 3 children)
- `c` and `i` are *not* triangles (have only 1 child)
- `g` is a triangle as it has *exactly* two children `h` and `i`
- `d`, `e`, `f`, `h` and `l` are not triangles, because they have zero children

Now implement this method:

```
def is_triangle(self, elems):
    """ RETURN True if this node is a triangle matching the data
        given by list elems.

        In order to match:
        - first list item must be equal to this node data
        - second list item must be equal to this node first child data
        - third list item must be equal to this node second child data

        - if elems has less than three elements, raises ValueError
    """
```

Testing: `python -m unittest gen_tree_test.IsTriangleTest`

Examples:

[71]:
```
from gen_tree_test import gt
```

[72]:
```
# this is the tree from the example above

tb = gt('b', gt('d', gt('e'), gt('f')))
tg = gt('g', gt('h'), gt('i', gt('l')))
ta = gt('a', tb, gt('c', tg))

ta.is_triangle(['a','b','c'])
```

[72]:
```
True
```

[73]:
```
ta.is_triangle(['b','c','a'])
```

[73]:
```
False
```

[74]:
```
tb.is_triangle(['b','d','e'])
```

[74]:
```
False
```

[75]:
```
tg.is_triangle(['g','h','i'])
```

[75]:
```
True
```

[76]:
```
tg.is_triangle(['g','i','h'])
```

[76]:
```
False
```

GT 2.11 has_triangle

Implement this method:

```
def has_triangle(self, elems):
    """ RETURN True if this node *or one of its descendants* is a triangle
        matching given elems. Otherwise, return False.

        - a recursive solution is acceptable
    """
```

Testing: `python -m unittest gen_tree_test.HasTriangleTest`

Examples:

[77]:
```
# example tree seen at the beginning

tb = gt('b', gt('d', gt('e'), gt('f')))
tg = gt('g', gt('h'), gt('i', gt('l')))
tc = gt('c', tg)
ta = gt('a', tb, tc)


ta.has_triangle(['a','b','c'])
```

[77]:
```
True
```

[78]:
```
ta.has_triangle(['a','c','b'])
```

[78]:
```
False
```

[79]:
```
ta.has_triangle(['b','c','a'])
```

[79]:
```
False
```

[80]:
```
tb.is_triangle(['b','d','e'])
```

[80]:
```
False
```

[81]:
```
tg.has_triangle(['g','h','i'])
```

[81]:
```
True
```

[82]:
```
tc.has_triangle(['g','h','i'])  # check recursion
```

[82]:
```
True
```

[83]:
```
ta.has_triangle(['g','h','i'])  # check recursion
```

[83]:
```
True
```

[ ]:
```

```

7.7 Graph algorithms

7.7.1 Download exercises zip

(before editing read whole introduction section 0.x)

Browse files online[360]

What to do

- unzip exercises in a folder, you should get something like this:

```
graph-algos
    graph-algos.ipynb
    graph.py
    graph_sol.py
    jupman.py
    sciprog.py
```

[360] https://github.com/DavidLeoni/sciprog-ds/tree/master/graph-algos

- open the editor of your choice (for example Visual Studio Code, Spyder or PyCharme), you will edit the files ending in `.py` files
- Go on reading this notebook, and follow instuctions inside.

7.7.2 Introduction

0.1 Graph theory

In short, a graph is a set of vertices linked by edges.

Longer version:

- Luca Bianco theory[361]
- Graphs on the book[362]
  - In particular, see Vocabulary and definitions[363]

[361] https://sciproalgo2019.readthedocs.io/slides/Lecture7.pdf
[362] https://interactivepython.org/runestone/static/pythonds/Graphs/toctree.html
[363] https://interactivepython.org/runestone/static/pythonds/Graphs/VocabularyandDefinitions.html

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

0.2 Directed graphs

In this worksheet we are going to use so called Directed Graphs (`DiGraph` for brevity), that is, graphs with *directed* edges: each edge can be pictured as an arrow linking source node `a` to target node `b`. With such an arrow, you can go from `a` to `b` but you cannot go from `b` to `a` unless there is another edge in the reverse direction.

- `DiGraph` for us can also have no edges or no verteces at all.
- Verteces for us can be anything, strings like *'abc'*, numbers like `3`, etc
- In our model, edges simply link vertices and have no weights
- `DiGraph` is represented as an adjacency list, mapping each vertex to the verteces it is linked to.

QUESTION: is `DiGraph` model good for dense or sparse graphs?

0.3 Serious graphs

In this worksheet we follow the *Do It Yourself* methodology and create graph classes from scratch for didactical purposes. Of course, in Python world you have alread nice libraries entirely devoted to graphs like networkx[364], you can also use them for visualizating graphs. If you have huge graphs to process you might consider big data tools like Spark GraphX[365] which is programmable in Python.

[364] https://networkx.github.io/
[365] http://spark.apache.org/graphx

0.4 Code skeleton

First off, download the exercises zip and look at the files:

- `graph.py` : the exercise to edit
- `graph_test.py`: the tests to run. Do not modify this file.

Before starting to implement methods in `DiGraph` class, read all the following sub sections (starting with '0.x')

0.5 Building graphs

IMPORTANT: All the functions in section 0 are already provided and you don't need to implement them !

For now, open a Python 3 interpreter and try out the `graph_sol` module :

[2]:
```
from graph_sol import *
```

0.5.1 Building basics

Let's look at the constructor `__init__` and `add_vertex`. They are already provided and you don't need to implement it:

```
class DiGraph:
    def __init__(self):
        # The class just holds the dictionary _edges: as keys it has the verteces, and
        # to each vertex associates a list with the verteces it is linked to.

        self._edges = {}

    def add_vertex(self, vertex):
        """ Adds vertex to the DiGraph. A vertex can be any object.

            If the vertex already exist, does nothing.
        """
        if vertex not in self._edges:
            self._edges[vertex] = []
```

You will see that inside it just initializes `_edges`. So the only way to create a `DiGraph` is with a call like

[3]:
```
g = DiGraph()
```

`DiGraph` provides an `__str__` method to have a nice printout:

[4]:
```
print(g)
```

```
DiGraph()
```

To draw a DiGraph, you can use `draw_dig` from `sciprog` module - in this case draw nothing as the graph is empty:

[5]:
```
from sciprog import draw_dig
draw_dig(g)
```

You can add then vertices to the graph like so:

[6]:
```
g.add_vertex('a')
g.add_vertex('b')
g.add_vertex('c')
```

[7]:
```
print(g)
```

```
a: []
b: []
c: []
```

To draw a DiGraph, you can use `draw_dig` from `sciprog` module:

[8]:
```
from sciprog import draw_dig
draw_dig(g)
```

Adding a vertex twice does nothing:

[9]:
```
g.add_vertex('a')
print(g)
```

```
a: []
b: []
c: []
```

Once you added the verteces, you can start adding directed edges among them with the method `add_edge`:

```
def add_edge(self, vertex1, vertex2):
    """ Adds an edge to the graph, from vertex1 to vertex2

        If verteces don't exist, raises an Exception.
        If there is already such an edge, exits silently.
    """

    if not vertex1 in self._edges:
        raise Exception("Couldn't find source vertex:" + str(vertex1))

    if not vertex2 in self._edges:
        raise Exception("Couldn't find target vertex:" + str(vertex2))

    if not vertex2 in self._edges[vertex1]:
        self._edges[vertex1].append(vertex2)
```

[10]:
```
g.add_edge('a', 'c')
print(g)
```

```
a: ['c']
b: []
c: []
```

[11]:
```
draw_dig(g)
```

[12]:
```
g.add_edge('a', 'b')
print(g)
```

```
a: ['c', 'b']
b: []
c: []
```

[13]:
```
draw_dig(g)
```

Adding an edge twice makes no difference:

[14]:
```
g.add_edge('a', 'b')
print(g)
```

```
a: ['c', 'b']
b: []
c: []
```

Notice a `DiGraph` can have self-loops too (also called *caps*):

[15]:
```
g.add_edge('b', 'b')
print(g)
```

```
a: ['c', 'b']
b: ['b']
c: []
```

[16]:
```
draw_dig(g)
```

0.5.2 dig()

`dig()` is a shortcut to build graphs, it is already provided and you don't need to implement it.

USE IT ONLY WHEN TESTING, *NOT* IN THE ``DiGraph`` CLASS CODE !!!!

First of all, remember to import it from `graph_test` package:

[17]:
```
from graph_test import dig
```

With empty dict prints the empty graph:

[18]:
```
print(dig({}))
```

```
DiGraph()
```

To build more complex graphs, provide a dictionary with pairs source vertex / target verteces list like in the following examples:

[19]:
```
print(dig({'a':['b','c']}))
```

```
a: ['b', 'c']
b: []
c: []
```

[20]:
```
print(dig({'a': ['b','c'],
           'b': ['b'],
           'c': ['a']}))
```

```
a: ['b', 'c']
b: ['b']
c: ['a']
```

0.6 Equality

Graphs for us are equal irrespectively of the order in which elements in adjacency lists are specified. So for example these two graphs will be considered equal:

[21]:
```
dig({'a': ['c', 'b']}) == dig({'a': ['b', 'c']})
```

[21]:
```
True
```

0.7 Basic querying

There are some provided methods to query the `DiGraph`: `adj`, `verteces`, `is_empty`

0.7.1 adj

To obtain the edges, you can use the method `adj(self, vertex)`. It is already provided and you don't need to implement it:

```
def adj(self, vertex):
    """ Returns the verteces adjacent to vertex.

        NOTE: verteces are returned in a NEW list.
        Modifying the list will have NO effect on the graph!
    """
    if not vertex in self._edges:
        raise Exception("Couldn't find a vertex " + str(vertex))

    return self._edges[vertex][:]
```

[22]:
```
lst = dig({'a': ['b', 'c'],
           'b': ['c']}).adj('a')
print(lst)
```

```
['b', 'c']
```

Let's check we actually get back a new list (so modifying the old one won't change the graph):

[23]:
```
lst.append('d')
print(lst)
```

```
['b', 'c', 'd']
```

[24]:
```
print(g.adj('a'))
```

```
['c', 'b']
```

NOTE: This technique of giving back copies is also called *defensive copying*: it prevents users from modifying the internal data structures of a class instance in an uncontrolled manner. For example, if we allowed them direct access to the internal verteces list, they could add duplicate edges, which we don't allow in our model. If instead we only allow users to add edges by calling `add_edge`, we are sure the constraints for our model will always remain satisfied.

0.7.2 is_empty()

We can check if a `DiGraph` is empty. It is already provided and you don't need to implement it:

```
def is_empty(self):
    """  A DiGraph for us is empty if it has no verteces and no edges """

    return len(self._edges) == 0
```

[25]:
```
print(dig({}).is_empty())
```

```
True
```

[26]:
```
print(dig({'a':[]}).is_empty())
```

```
False
```

0.7.3 verteces()

To obtain the verteces, you can use the function `verteces`. (NOTE for Italians: method is called vert**ec**es, with two **e**s !!!). It is already provided and you don't need to implement it:

```
def verteces(self):
    """ Returns a set of the graph verteces. Verteces can be any object. """

    # Note dict keys() return a list, not a set. Bleah.
    # See http://stackoverflow.com/questions/13886129/why-does-pythons-dict-keys-
↪return-a-list-and-not-a-set
    return set(self._edges.keys())
```

[27]:
```
g = dig({'a': ['c', 'b'],
         'b': ['c']})
print(g.verteces())
```

```
{'a', 'c', 'b'}
```

Notice it returns a *set*, as verteces are stored as keys in a dictionary, so they are not supposed to be in any particular order. When you print the whole graph you see them vertically ordered though, for clarity purposes:

[28]:
```
print(g)
```

```

a: ['c', 'b']
b: ['c']
c: []
```

Verteces in the edges list are instead stored and displayed in the order in which they were inserted.

0.8 Blow up your computer

Try to call the already implemented function `graph_test.gen_graphs` with small numbers for `n`, like *1*, *2* , *3* , *4* …. Just with *2* we get back a lot of graphs:

```
def gen_graphs(n):
    """ Returns a list with all the possible 2^(n^2) graphs of size n

        Verteces will be identified with numbers from 1 to n
    """
```

[29]:
```
from graph_test import gen_graphs
print(gen_graphs(2))
```

```
[
1: []
2: []
,
1: []
2: [2]
,
1: []
2: [1]
,
1: []
2: [1, 2]
,
1: [2]
2: []
,
1: [2]
2: [2]
,
1: [2]
2: [1]
,
1: [2]
2: [1, 2]
,
1: [1]
2: []
,
1: [1]
2: [2]
,
1: [1]
2: [1]
,
1: [1]
2: [1, 2]
,
1: [1, 2]
2: []
,
1: [1, 2]
2: [2]
,
1: [1, 2]
```

(continues on next page)

(continued from previous page)

```
2: [1]
,
1: [1, 2]
2: [1, 2]
]
```

QUESTION: What happens if you call `gen_graphs(10)` ? How many graphs do you get back ?

7.7.3 1. Implement building

Enough for talking! Let's implement building graphs.

1.1 has_edge

Implement this method in `DiGraph`:

```
def has_edge(self, source, target):
    """  Returns True if there is an edge between source vertex and target vertex.
         Otherwise returns False.

        If either source, target or both verteces don't exist raises an Exception.
    """

    raise Exception("TODO IMPLEMENT ME!")
```

Testing: `python3 -m unittest graph_test.HasEdgeTest`

1.2 full_graph

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def full_graph(verteces):
    """ Returns a DiGraph which is a full graph with provided verteces list.

        In a full graph all verteces link to all other verteces (including themselves!
↪).
    """

    raise Exception("TODO IMPLEMENT ME!")
```

Testing: `python3 -m unittest graph_test.FullGraphTest`

1.3 dag

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def dag(verteces):
    """ Returns a DiGraph which is DAG (Directed Acyclic Graph) made out of provided␣
↪verteces list

        Provided list is intended to be in topological order.
        NOTE: a DAG is ACYCLIC, so caps (self-loops) are not allowed !!
    """

    raise Exception("TODO IMPLEMENT ME!")
```

Testing: `python3 -m unittest graph_test.DagTest`

1.4 list_graph

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def list_graph(n):
    """ Return a graph of n verteces displaced like a
        monodirectional list:  1 -> 2 -> 3 -> ... -> n

        Each vertex is a number i, 1 <= i <= n  and has only one edge connecting it
        to the following one in the sequence
        If n = 0, return the empty graph.
        if n < 0, raises an Exception.
    """

    raise Exception("TODO IMPLEMENT ME!")
```

Testing: `python3 -m unittest graph_test.ListGraphTest`

1.5 star_graph

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def star_graph(n):
    """ Returns graph which is a star with n nodes

        First node is the center of the star and it is labeled with 1. This node is␣
↪linked
        to all the others. For example, for n=4 you would have a graph like this:

                3
                ^
                |
           2 <- 1 -> 4

        If n = 0, the empty graph is returned
        If n < 0, raises an Exception
    """

    raise Exception("TODO IMPLEMENT ME!")
```

Testing: `python3 -m unittest graph_test.StarGraphTest`

1.6 odd_line

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def odd_line(n):
    """ Returns a DiGraph with n verteces, displaced like a line of odd numbers

        Each vertex is an odd number i, for  1 <= i < 2n. For example, for
        n=4 verteces are displaced like this:

        1 -> 3 -> 5 -> 7

        For n = 0, return the empty graph

    """
```

Testing: `python3 -m unittest graph_test.OddLineTest`

Example usage:

[30]:
```
odd_line(0)
```

[30]:
```
DiGraph()
```

[31]:
```
odd_line(1)
```

[31]:
```
1: []
```

[32]:
```
 odd_line(2)
```

[32]:
```
1: [3]
3: []
```

[33]:
```
odd_line(3)
```

[33]:
```
1: [3]
3: [5]
5: []
```

[34]:
```
odd_line(4)
```

[34]:
```
1: [3]
3: [5]
5: [7]
7: []
```

1.7 even_line

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def even_line(n):
    """ Returns a DiGraph with n verteces, displaced like a line of even numbers

        Each vertex is an even number i, for  2 <= i <= 2n. For example, for
        n=4 verteces are displaced like this:

        2 <- 4 <- 6 <- 8

        For n = 0, return the empty graph

    """
```

Testing: `python3 -m unittest graph_test.EvenLineTest`

Example usage:

[35]:
```
even_line(0)
```

[35]:
```
DiGraph()
```

[36]:
```
even_line(1)
```

[36]:
```
2: []
```

[37]:
```
 even_line(2)
```

[37]:
```
2: []
4: [2]
```

[38]:
```
even_line(3)
```

[38]:
```
2: []
4: [2]
6: [4]
```

1.8 quads

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def quads(n):
    """ Returns a DiGraph with 2n verteces, displaced like a strip of quads.

        Each vertex is a number i,  1 <= i <= 2n.
        For example, for n = 4, verteces are displaced like this:

        1 -> 3 -> 5 -> 7
        ^    |    ^    |
        |    ;    |    ;
        2 <- 4 <- 6 <- 8
```

(continues on next page)

(continued from previous page)

```
        where

          ^                                          |
          |  represents an upward arrow,   while    ;  represents a downward arrow

    """
```

Testing: `python3 -m unittest graph_test.QuadsTest`

Example usage:

[39]:
```
quads(0)
```

[39]:
```
DiGraph()
```

[40]:
```
quads(1)
```

[40]:
```
1: []
2: [1]
```

[41]:
```
quads(2)
```

[41]:
```
1: [3]
2: [1]
3: [4]
4: [2]
```

[42]:
```
quads(3)
```

[42]:
```
1: [3]
2: [1]
3: [5, 4]
4: [2]
5: []
6: [4, 5]
```

[43]:
```
quads(4)
```

[43]:
```
1: [3]
2: [1]
3: [5, 4]
4: [2]
5: [7]
6: [4, 5]
7: [8]
8: [6]
```

1.9 pie

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def pie(n):
    """
        Returns a DiGraph with n+1 verteces, displaced like a polygon with a perimeter
        of n verteces progressively numbered from 1 to n.
        A central vertex numbered zero has outgoing edges to all other verteces.

        For n = 0, return the empty graph.
        For n = 1, return vertex zero connected to node 1, and node 1 has a self-loop.

    """
```

Testing: `python3 -m unittest graph_test.PieTest`

Example usage:

For `n=5`, the function creates this graph:

[44]:
```
pie(5)
```

[44]:
```
0: [1, 2, 3, 4, 5]
1: [2]
2: [3]
3: [4]
4: [5]
5: [1]
```

Degenerate cases:

[45]:
```
pie(0)
```

[45]:
```
DiGraph()
```

[46]:
```
pie(1)
```

[46]:
```
0: [1]
1: [1]
```

1.10 Flux Capacitor

A *Flux Capacitor* is a plutonium-powered device that enables time travelling. During the 80s it was installed on a Delorean car and successfully used to ride humans back and forth across centuries:

In this exercise you will build a Flux Capacitor model as a Y-shaped `DiGraph`, created according to a parameter `depth`. Here you see examples at different depths:

Implement this function **outside** the class definition. It is **not** a method of `DiGraph` !

```
def flux(depth):
    """ Returns a DiGraph with 1 + (d * 3) numbered verteces displaced like a Flux␣
↪Capacitor:

        - from a central node numbered 0, three branches depart
        - all edges are directed outward
        - on each branch there are 'depth' verteces.
        - if depth < 0, raises a ValueError

        For example, for depth=2 we get the following graph (suppose arrows point␣
↪outward):

              4          5
               \        /
                1      2
                 \    /
                   0
                   |
                   3
                   |
                   6
```

Testing: `python3 -m unittest graph_test.FluxTest`

Example usage:

[47]:
```
flux(0)
```

[47]:
```
0: []
```

[48]:
```
flux(1)
```

[48]:
```
0: [1, 2, 3]
1: []
2: []
```

(continues on next page)

(continued from previous page)

```
3: []
```

[49]:
```
flux(2)
```

[49]:
```
0: [1, 2, 3]
1: [4]
2: [5]
3: [6]
4: []
5: []
6: []
```

[50]:
```
 flux(3)
```

[50]:
```
0: [1, 2, 3]
1: [4]
2: [5]
3: [6]
4: [7]
5: [8]
6: [9]
7: []
8: []
9: []
```

7.7.4 2. Manipulate graphs

You will now implement some methods to manipulate graphs.

2.1 remove_vertex

```
def remove_vertex(self, vertex):
    """ Removes the provided vertex  and returns it

        If the vertex is not found, raises an Exception.
    """
```

Testing: `python3 -m unittest graph_test.RemoveVertexTest`

2.2 transpose

```
def transpose(self):
    """ Reverses the direction of all the edges

        - MUST perform in O(|V|+|E|)
             Note in adjacency lists model we suppose there are only few edges per␣
↪node,
             so if you end up with an algorithm which is O(|V|^2) you are ending up␣
↪with a
             complexity usually reserved for matrix representations !!
```

(continues on next page)

(continued from previous page)

```
        NOTE: this method changes in-place the graph: does **not** create a new␣
↪instance
              and does *not* return anything !!

        NOTE: To implement it *avoid* modifying the existing _edges dictionary (would
              probably more problems than anything else).
              Instead, create a new dictionary, fill it with the required
              verteces and edges ad then set _edges to point to the new dictionary.
    """
```

Testing: `python3 -m unittest graph_test.TransposeTest`

2.3 has_self_loops

```
def has_self_loops(self):
    """ Returns True if the graph has any self loop (a.k.a. cap), False otherwise """
```

Testing: `python3 -m unittest graph_test.HasSelfLoopsTest`

2.4 remove_self_loops

```
def remove_self_loops(self):
    """ Removes all of the self-loops edges (a.k.a. caps)

        NOTE: Removes just the edges, not the verteces!
    """
```

Testing: `python3 -m unittest graph_test.RemoveSelfLoopsTest`

2.5 undir

```
def undir(self):
    """ Return a *NEW* undirected version of this graph, that is, if an edge a->b␣
↪exists in this graph,
        the returned graph must also have both edges  a->b and b->a

        *DO NOT* modify the current graph, just return an entirely new one.
    """
```

Testing: `python3 -m unittest graph_test.UndirTest`

7.7.5 3. Query graphs

You can query graphs the *Do it yourself* way with Depth First Search (DFS) or Breadth First Search (BFS).

Let's make a simple example:

[51]:
```
g = dig({'a': ['a','b', 'c'],
         'b': ['c'],
         'd': ['e']})

from sciprog import draw_dig
draw_dig(g)
```

[52]:
```
g.dfs('a')
```

```
DEBUG:  Stack is: ['a']
DEBUG:  popping from stack: a
DEBUG:    not yet visited
DEBUG:    Scheduling for visit: a
DEBUG:    Scheduling for visit: b
DEBUG:    Scheduling for visit: c
DEBUG:  Stack is : ['a', 'b', 'c']
DEBUG:  popping from stack: c
DEBUG:    not yet visited
DEBUG:  Stack is : ['a', 'b']
DEBUG:  popping from stack: b
DEBUG:    not yet visited
DEBUG:    Scheduling for visit: c
DEBUG:  Stack is : ['a', 'c']
DEBUG:  popping from stack: c
DEBUG:    already visited!
DEBUG:  popping from stack: a
DEBUG:    already visited!
```

Compare it wirh the example for the bfs :

[53]:
```
draw_dig(g)
```

[54]:
```
g.bfs('a')
```

```
DEBUG:  Removed from queue: a
DEBUG:    Found neighbor: a
DEBUG:      already visited
DEBUG:    Found neighbor: b
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Found neighbor: c
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['b', 'c']
DEBUG:  Removed from queue: b
DEBUG:    Found neighbor: c
DEBUG:      already visited
DEBUG:    Queue is: ['c']
DEBUG:  Removed from queue: c
DEBUG:    Queue is: []
```

Predictably, results are different.

3.1 distances()

Implement this method of `DiGraph`:

```
def distances(self, source):
    """
    Returns a dictionary where the keys are verteces, and each vertex v is associated
    to the *minimal* distance in number of edges required to go from the source
    vertex to vertex v. If node is unreachable, the distance will be -1

    Source has distance zero from itself
    Verteces immediately connected to source have distance one.

    - if source is not a vertex, raises an LookupError
    - MUST execute in O(|V| + |E|)
    - HINT: implement this using bfs search.
    """
```

If you look at the following graph, you can see an example of the distances to associate to each vertex, supposing that the `source` is `a`. Note that `a` iself is at distance zero from itself and also that unreachable nodes like `f` and `g` will be at

distance -1

[55]:
```
import sciprog
sciprog.draw_nx(sciprog.show_distances())
```

`distances('a')` called on this graph would return a map like this:

```
{
  'a':0,
  'b':1,
  'c':1,
  'd':2,
  'e':3,
  'f':-1,
  'g':-1,

}
```

3.2 equidistances()

Implement this method of `DiGraph`:

```
def equidistances(self, va, vb):
    """ RETURN a dictionary holding the nodes which
        are equidistant from input verteces va and vb.
        The dictionary values will be the distances of the nodes.

        - if va or vb are not present in the graph, raises LookupError
        - MUST execute in O(|V| + |E|)
        - HINT: To implement this, you can use the previously defined distances()␣
↪method
    """
```

Example:

[56]:
```
G = dig({'a': ['b','e'],
         'b': ['d'],
         'c': ['d'],
         'd': ['f'],
         'e': ['d','b'],
         'f': ['g','h'],
         'g': ['e']})
draw_dig(G, options={'graph':{'size':'15,3!', 'rankdir':'LR'}})
```

Consider `a` and `g`, they both:

- can reach `e` in one step
- can reach `d` in two steps
- can reach `f` in three steps
- can reach `h` in four steps
- `c` is unreachable by both `a` and`g`,so it won't be present in the output
- `b` is reached from `a` in one step, and from `g` in two steps, so it won't be included in the output

[57]:
```
G.equidistances('a','g')
```

[57]:
```
{'e': 1, 'd': 2, 'f': 3, 'h': 4}
```

3.3 Play with dfs and bfs

Create small graphs (like linked lists a->b->c, triangles, mini-full graphs, trees - you can also use the functions you defined to create graphs like `full_graph`, `dag`, `list_graph`, `star_graph`) and try to predict the visit sequence (verteces order, with discovery and finish times) you would have running a dfs or bfs. Then write tests that assert you actually get those sequences when running provided dfs and bfs

3.4 Exits graph

There is a place nearby Trento called Silent Hill, where people always study and do little else. Unfortunately, one day an unethical biotech AI experiment goes wrong and a buggy cyborg is left free to roam in the building. To avoid panic, you are quickly asked to devise an evacuation plan. The place is a well known labyrinth, with endless corridors also looping into cycles. But you know you can model this network as a digraph, and decide to represent crossings as nodes. When a crossing has a door to leave the building, its label starts with letter `e`, while when there is no such door the label starts with letter `n`.

In the example below, there are three exits `e1`, `e2`, and `e3`. Given a node, say `n1`, you want to tell the crowd in that node the **shortest** paths leading to the three exits. To avoid congestion, one third of the crowd may be told to go to `e2`, one third to reach `e1` and the remaining third will go to `e3` even if they are farther than `e2`.

In Python terms, we would like to obtain a dictionary of paths like the following, where as keys we have the exits and as values the shortest sequence of nodes from `n1` leading to that exit

```
{
    'e1': ['n1', 'n2', 'e1'],
    'e2': ['n1', 'e2'],
    'e3': ['n1', 'e2', 'n3', 'e3']
}
```

[58]:
```
from sciprog import draw_dig
from graph_sol import *
from graph_test import dig
```

[59]:
```
G = dig({'n1':['n2','e2'],
         'n2':['e1'],
         'e1':['n1'],
         'e2':['n2','n3', 'n4'],
         'n3':['e3'],
         'n4':['n1']})
draw_dig(G)
```

You will solve the exercise in steps, so open `exits_sol.py` and proceed reading the following points.

3.4.1 Exits graph cp

Implement this method

```
def cp(self, source):
    """ Performs a BFS search starting from provided node label source and
        RETURN a dictionary of nodes representing the visit tree in the
        child-to-parent format, that is, each key is a node label and as value
        has the node label from which it was discovered for the first time

        So if node "n2" was discovered for the first time while
        inspecting the neighbors of "n1", then in the output dictionary there
        will be the pair "n2":"n1".

        The source node will have None as parent, so if source is "n1" in the
        output dictionary there will be the pair  "n1": None
```

(continues on next page)

(continued from previous page)

```
        - MUST execute in O(|V| + |E|)
        - NOTE: This method must *NOT* distinguish between exits
                and normal nodes, in the tests we label them n1, e1 etc just
                because we will reuse in next exercise
        - NOTE: You are allowed to put debug prints, but the only thing that
                matters for the evaluation and tests to pass is the returned
                dictionary
    """
```

Testing: `python3 -m unittest graph_test.CpTest`

Example:

[60]:
```
G.cp('n1')
```

```
DEBUG:  Removed from queue: n1
DEBUG:    Found neighbor: n2
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Found neighbor: e2
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['n2', 'e2']
DEBUG:  Removed from queue: n2
DEBUG:    Found neighbor: e1
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['e2', 'e1']
DEBUG:  Removed from queue: e2
DEBUG:    Found neighbor: n2
DEBUG:      already visited
DEBUG:    Found neighbor: n3
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Found neighbor: n4
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['e1', 'n3', 'n4']
DEBUG:  Removed from queue: e1
DEBUG:    Found neighbor: n1
DEBUG:      already visited
DEBUG:    Queue is: ['n3', 'n4']
DEBUG:  Removed from queue: n3
DEBUG:    Found neighbor: e3
DEBUG:      not yet visited, enqueueing ..
DEBUG:    Queue is: ['n4', 'e3']
DEBUG:  Removed from queue: n4
DEBUG:    Found neighbor: n1
DEBUG:      already visited
DEBUG:    Queue is: ['e3']
DEBUG:  Removed from queue: e3
DEBUG:    Queue is: []
```

[60]:
```
{'n1': None,
 'n2': 'n1',
 'e2': 'n1',
 'e1': 'n2',
 'n3': 'e2',
 'n4': 'e2',
 'e3': 'n3'}
```

Basically, the dictionary above represents this visit tree:

```
   n1
  /   \
n2     e2
 \    /  \
 e1   n3  n4
      |
      e3
```

3.4.2 Exit graph exits

Implement this function. **NOTE**: the function is external to class DiGraph.

```
def exits(cp):
    """
        INPUT: a dictionary of nodes representing a visit tree in the
        child-to-parent format, that is, each key is a node label and
        as value has its parent as a node label. The root has
        associated None as parent.

        OUTPUT: a dictionary mapping node labels of exits to a list
                of node labels representing the the shortest path from
                the root to the exit (root and exit included)

        - MUST execute in O(|V| + |E|)
    """
```

Testing: `python3 -m unittest graph_test.ExitsTest`

Example:

[61]:
```
# as example we can use the same dictionary outputted by the cp call in the previous␣
↪exercise

visit_cp = { 'e1': 'n2',
             'e2': 'n1',
             'e3': 'n3',
             'n1': None,
             'n2': 'n1',
             'n3': 'e2',
             'n4': 'e2'
           }
exits(visit_cp)
```

[61]:
```
{'e1': ['n1', 'n2', 'e1'], 'e2': ['n1', 'e2'], 'e3': ['n1', 'e2', 'n3', 'e3']}
```

3.5 connected components

Implement `cc`:

```
def cc(self):
    """ Finds the connected components of the graph, returning a dict object
        which associates to the verteces the corresponding connected component
        number id, where 1 <= id <= |V|

        IMPORTANT:  ASSUMES THE GRAPH IS UNDIRECTED !
```

(continues on next page)

(continued from previous page)

```
                    ON DIRECTED GRAPHS, THE RESULT IS UNPREDICTABLE !

        To develop this function, implement also ccdfs

        HINT: store 'counter' as field in Visit object
    """
```

Which in turn uses the FUNCTION `ccdfs`, also to implement INSIDE the method cc:

```
def ccdfs(counter, source, ids):
    """
        Performs a DFS from source vertex

        HINT: Copy in here the method from DFS and adapt it as needed
        HINT: store the connected component id in VertexLog objects
    """
```

Testing: `python3 -m unittest graph_test.CCTest`

NOTE: In tests, to keep code compact graphs are created a call to `udig()`

[62]:
```
from graph_test import udig

udig({'a': ['b'],
      'c': ['d']})
```

[62]:
```
a: ['b']
b: ['a']
c: ['d']
d: ['c']
```

which makes sure the resulting graph is undirected as CC algorithm requires (so if there is one edge `a->b` there will also be another edge `b->a`)

3.6 has_cycle

Implement `has_cycle` method for directed graphs:

```python

```
def has_cycle(self):
    """ Return True if this directed graph has a cycle, return False otherwise.

        - To develop this function, implement also has_cycle_rec(u) inside this method
        - Inside has_cycle_rec, to reference variables of has_cycle you need to
          declare them as nonlocal like
             nonlocal clock, dt, ft
        - MUST be able to also detect self-loops
    """```
```

and also `has_cycle_rec` inside `has_cycle`:

```
def has_cycle_rec(u):
    raise Exception("TODO IMPLEMENT ME !")
```

Testing: `python3 -m unittest graph_test.HasCycleTest`
```

3.7 top_sort

Look at Montresor slides on topological sort[366]

Keep in mind two things:

- topological sort works on DAGs, that is, Directed *Acyclic* Graphs
- given a graph, there can be more than one valid topological sort
- it works also on DAGs having disconnected components, in which case the nodes of one component can be interspersed with the nodes of other components at will, provided the order within nodes belonging to the same component is preserved.

EXERCISE: Before coding, try by hand to find all the topological sorts of the following graphs. For all them, you will find the solutions listed in the tests.

[63]:
```
G = dig({'a':['c'],
         'b':['c']})
draw_dig(G)
```

[64]:
```
G = dig({'a':['b'], 'c':[]})
draw_dig(G)
```

[65]:
```
G = dig({'a':['b'], 'c':['d']})
draw_dig(G)
```

[366] http://disi.unitn.it/~montreso/sp/slides/B04-grafi.pdf

[66]:
```
G = dig({'a':['b','c'], 'b':['d'], 'c':['d']})
draw_dig(G)
```

[67]:
```
G = dig({'a':['b','c','d'], 'b':['e'], 'c':['e'], 'd':['e']})
draw_dig(G)
```

[68]:
```
G = dig({'a':['b','c','d'], 'b':['c','d'], 'c':['d'], 'd':[]})
draw_dig(G)
```

Now implement this method:

```
def top_sort(self):
    """ RETURN a topological sort of the graph. To implement this code,
        feel free to adapt Montresor algorithm

        - implement  Stack S  as a list
        - implement  visited  as a set
        - NOTE: differently from Montresor code, for tests to pass
                you will need to return a reversed list. Why ?
    """
```

Testing: `python3 -m unittest graph_test.TopSortTest`

Note: in tests there is the method `self.assertIn(el,elements)` which checks `el` is in `elements`. We use it because for a graph there a many valid topological sorts, and we want the test independent from your particular implementation .

[ ]:

7.8 Changelog

Scientific Programming Data Science Lab

https://sciprog.davidleoni.it

7.8.1 1.0 September 2020

- upgraded to Jupman 3, changed folder structure
- moved webste to sciprog.davidleoni.it
- building with Github Actions

7.8.2 0.1, September 2018

Site is born

CHAPTER

EIGHT

INDEX