Sciprog DS Lab

Enter the Matrix

David Leoni

Nov 18, 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 **5**
1.1 Slides . . . 5
1.2 Office hours . . . 5
1.3 Labs timetable . . . 5
1.4 Tutoring . . . 5
1.5 Resources . . . 6
1.6 Exams . . . 7
1.7 Acknowledgements . . . 11

2 Past Exams **13**
2.1 Data science . . . 13
2.2 2017-18 (QCB) . . . 277
2.3 2016-17 (QCB) . . . 277

3 Slides 2020/21 **279**
3.1 Part A . . . 279
3.2 Lab A.1 . . . 279
3.3 Lab A.2 . . . 280
3.4 Lab A.3 . . . 281
3.5 Lab A.4 . . . 281
3.6 Lab A.5 . . . 282
3.7 Lab A.6 . . . 282
3.8 Lab A.7 . . . 283
3.9 Lab A.8 . . . 283
3.10 Lab A.9 . . . 286
3.11 Lab A.10 . . . 287
3.12 Lab A.11 . . . 287
3.13 Lab A.12 . . . 288
3.14 Part B . . . 289
3.15 Lab B.1 . . . 289
3.16 Lab B.2 . . . 289
3.17 Lab B.3 . . . 290
3.18 Lab B.4 . . . 290

4 Commandments **291**
4.1 MOVED TO https://en.softpython.org/commandments.html . . . 291

5 **293**

6 Part A **295**
6.1 Installation . . . 295
6.2 Python basics solutions . . . 306
6.3 Strings solutions . . . 307
6.4 Lists solutions . . . 307
6.5 Tuples solutions . . . 307
6.6 Sets solutions . . . 307
6.7 Dictionaries solutions . . . 307
6.8 Control flow solutions . . . 308
6.9 Functions - solutions . . . 308
6.10 Error handling and testing solutions . . . 329
6.11 Matrices: list of lists solutions . . . 338
6.12 Matrices: Numpy solutions . . . 338
6.13 Data formats solutions . . . 338
6.14 Graph formats solutions . . . 338
6.15 Visualization solutions . . . 339
6.16 Pandas solutions . . . 339
6.17 Binary relations solutions . . . 339

7 Part B **341**
7.1 OOP . . . 341
7.2 OOP Matrix Challenge . . . 360
7.3 Complexity . . . 376
7.4 Sorting . . . 378
7.5 Sorting Challenge . . . 395
7.6 Linked lists . . . 404
7.7 Stacks . . . 425
7.8 Queues . . . 446
7.9 Binary Trees . . . 479
7.10 Generic Trees . . . 505
7.11 Graph algorithms . . . 526
7.12 Changelog . . . 557

8 Index **559**

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 announcement on Moodle[4]

[1] http://www.davidleoni.it
[2] https://creativecommons.org/licenses/by/4.0/
[3] https://didatticaonline.unitn.it/dol/course/view.php?id=25445
[4] https://didatticaonline.unitn.it/dol/mod/forum/discuss.php?d=164489

News

Friday 6 November 2020: published *midterm A solutions*

Tuesday 3 November 2020: published *midterm simulation solutions*

Thursday 22 October 2020: A TUTORING SERVICE for data science studentes has been set up, see announcement on Moodle[5]

> **WARNING: Part A of this website is being gradually improved and transfered to** en.softpython.org[6]
>
> **Keep an eye an news to see what has been transfered so far.** Whenever a page is moved, I will substitute the old page with a link.

MOVED SO FAR:

- introduction to Overview[7], Installation[8], Tools and scripts[9]
- basics to Basics[10]
- split strings into Strings1 - Introduction[11], Strings2 - Operators[12] and Strings3 - Methods[13] (a fourth page remains to be added)
- split lists into Lists1 - Introduction[14], Lists2 - Operators[15] and Lists3 - Methods[16] (a fourth page remains to be added)
- tuples to Tuples[17]
- sets to Sets[18]
- split dictionaries into Dictionaries1 - Introduction[19], Dictionaries2 - Operators[20] and Dictionaries3 - Methods[21] (a fourth and fifth pages remains to be added)
- split control flow into Control flow 1 - if[22], Control flow 2 - for[23] and Control flow 3 - while[24]
- Error handling and testing: moved exceptions and assert parts to Error handling and testing[25]
- Matrices: lists of lists to Matrices: lists of lists[26]

[5] https://didatticaonline.unitn.it/dol/mod/forum/discuss.php?d=182045
[6] https://en.softpython.org
[7] https://en.softpython.org/overview.html
[8] https://en.softpython.org/installation.html
[9] https://en.softpython.org/tools/tools-sol.html
[10] https://en.softpython.org/basics/basics-sol.html
[11] https://en.softpython.org/strings/strings1-sol.html
[12] https://en.softpython.org/strings/strings2-sol.html
[13] https://en.softpython.org/strings/strings3-sol.html
[14] https://en.softpython.org/lists/lists1-sol.html
[15] https://en.softpython.org/lists/lists2-sol.html
[16] https://en.softpython.org/lists/lists3-sol.html
[17] https://en.softpython.org/tuples/tuples-sol.html
[18] https://en.softpython.org/sets/sets-sol.html
[19] https://en.softpython.org/dictionaries/dictionaries1-sol.html
[20] https://en.softpython.org/dictionaries/dictionaries2-sol.html
[21] https://en.softpython.org/dictionaries/dictionaries3-sol.html
[22] https://en.softpython.org/control-flow/flow1-if-sol.html
[23] https://en.softpython.org/control-flow/flow2-for-sol.html
[24] https://en.softpython.org/control-flow/flow3-while-sol.html
[25] https://en.softpython.org/errors-and-testing/errors-and-testing-sol.html
[26] https://en.softpython.org/matrices-lists/matrices-lists-sol.html

- Part A Data formats (CSV) to SoftPython B-Data formats[27]
- Graph formats to Graph formats[28]
- Binary relations to Binary relations[29]
- Matrices Numpy to Matrices Numpy[30]
- Visualization to Visualization[31]
- Pandas to Pandas[32] and EURES job offers case study[33]

12 September 2020:

- Moved DS Lab website to sciprog.davidleoni.it[34]
- Simplified exercises structure

Old news

[27] https://en.softpython.org/formats/formats-sol.html
[28] https://en.softpython.org/graph-formats/graph-formats-sol.html
[29] https://en.softpython.org/binary-relations/binary-relations-sol.html
[30] https://en.softpython.org/matrices-numpy/matrices-numpy-sol.html
[31] https://en.softpython.org/visualization/visualization-sol.html
[32] https://en.softpython.org/pandas/pandas-sol.html
[33] https://en.softpython.org/pandas/eures-jobs-sol.html
[34] https://sciprog.davidleoni.it

CHAPTER

ONE

OVERVIEW

1.1 Slides

See *Slides page*

1.2 Office hours

To schedule a meeting, see here[35]

1.3 Labs timetable

For the regular labs timetable please see:

- Part A: Andrea Passerini's course site[36]
- Part B: Luca Bianco's course site[37]

1.4 Tutoring

A tutoring service for Scientific Programming has been set up, see Moodle announcement[38]

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

[35] http://www.davidleoni.it/office-hours
[36] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/
[37] https://sciproalgo2020.readthedocs.io/
[38] https://didatticaonline.unitn.it/dol/mod/forum/discuss.php?d=182045

1.5 Resources

- Source code[39] of these worksheets, (download HTML zip[40])

1.5.1 Part A Resources

- Part A Theory slides[41] by Andrea Passerini
- See References[42] on SoftPython website

1.5.2 Part B Resources

- Part B theory slides by Luca Bianco
- Problem Solving with Algorithms and Data Structures using Python[43] online book by Brad Miller and David Ranum
- Theory exercises (complexity, tree visits, graph visits) - by Alberto Montresor[44]
- LeetCode[45] (sort by easy difficulty)

1.5.3 Editors

- Visual Studio Code[46]: the course official editor.
- Spyder[47]: Seems like a fine and simple editor
- PyCharme Community Edition[48]
- Jupyter Notebook[49]: Nice environment to execute Python commands and display results like graphs. Allows to include documentation in Markdown format
- JupyterLab[50] : next and much better version of Jupyter, although as of Sept 2018 is still in beta
- PythonTutor[51], a visual virtual machine (*very useful*! can also be found in examples inside the book!)

[39] https://github.com/DavidLeoni/sciprog-ds
[40] https://github.com/DavidLeoni/sciprog-ds/archive/sciprog.davidleoni.it.zip
[41] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/
[42] http://en.softpython.org/references.html
[43] https://runestone.academy/runestone/static/pythonds/index.html
[44] https://drive.google.com/drive/folders/1RwjiSvIq60Z9mj_gCd5K2E6Bj9y1R0CL
[45] https://leetcode.com/problemset/all/
[46] https://code.visualstudio.com/
[47] https://pythonhosted.org/spyder/
[48] https://www.jetbrains.com/pycharm/download/#section=linux
[49] http://jupyter.org
[50] https://github.com/jupyterlab/jupyterlab
[51] http://www.pythontutor.com/visualize.html#mode=edit

1.5.4 Further readings

- Rule based design[52] 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)

1.6 Exams

Exams dates: see Moodle[53]

1.6.1 Past exams

- *Past exams page*

1.6.2 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[54]
- Andrea Passerini slides[55] and Luca Bianco slides
- Python 3 documentation[56]
  - In particular, Unittest docs[57]
  - If you need to look up some Python function, please start today learning how to search documentation on Python website.
- Part A: Think Python[58] book
- Part B: Problem Solving with Algorithms and Data Structures using Python[59] book

[52] https://www.researchgate.net/profile/Stef_Joosten/publication/327022933_Rule_Based_Design/links/5b7321be45851546c903234a/Rule-Based-Design.pdf
[53] https://didatticaonline.unitn.it/dol/course/view.php?id=25445
[54] https://sciprog.davidleoni.it
[55] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/
[56] https://docs.python.org/3/
[57] https://docs.python.org/3/library/unittest.html
[58] https://runestone.academy/runestone/static/thinkcspy/index.html
[59] https://runestone.academy/runestone/static/pythonds/index.html

1.6.3 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 or softpython**. 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, 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.

1.6.4 Grading

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)

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 percentage you get is usually inversely proportional to the amount of time I spend fixing 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.

1.6.5 Exams FAQ

First and foremost, I'm not the boss here, please refer to exam rules explained by Andrea Passerini[60] slides.

I add here some further questions I sometimes receive - luckily, answers are pretty easy to remember.

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 the grade of just that little 0.01?

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

[60] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/00-introcorso.pdf

1.6.6 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)
```

1.6.7 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

1.6.8 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)
```

1.7 Acknowledgements

This website and related courses were funded mainly by Department of Information Engineering and Computer Science (DISI)[61], University of Trento, and also Mathematics[62] and CIBIO[63] departments.

I wish also to thank Dr. Luca Bianco for the introductory material on Visual Studio Code and Python, and Dr. Alberto Montresor for having introduced me to first labs and slides on graphs.

All the material in this website is distributed with license CC-BY 4.0 International Attribution creativecommons.org/licenses/by/4.0/deed.en[64]

Basically, you can freely redistribute and modify the content, just remember to cite University of Trento and the present author.

Technical notes: all website pages are easily modifiable Jupyter notebooks, that were converted to web pages using NBSphinx[65] using template Jupman[66]. Text sources are on Github at address github.com/DavidLeoni/sciprog-ds[67]

[61] https://www.disi.unitn.it
[62] https://www.maths.unitn.it/en
[63] https://www.cibio.unitn.it/
[64] https://creativecommons.org/licenses/by/4.0/deed.en
[65] https://nbsphinx.readthedocs.io
[66] https://github.com/DavidLeoni/jupman
[67] https://github.com/DavidLeoni/sciprog-ds

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 Sim - Tue 13, Nov 2018

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solution

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.

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: ⊛⊛

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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

    """

    if len(lst1) > len(lst2):
        raise  ValueError("List 1 is bigger than list 2 ! lst_a = %s, lst_b = %s" %␣
↪(len(lst1), len(lst2)))
    j = 0
    for x in lst1:
        lst2[j] = x
        j += 1


try:
    fill(['a','b'], [None])
    raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
    "Test passed"

try:
    fill(['a','b','c'], [None,None])
    raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
    "Test passed"

L1 = []
R1 = []
fill(L1, R1)

assert L1 == []
assert R1 == []


L = []
R = ['x']
fill(L, R)

assert L == []
assert R == ['x']


L = ['a']
```

(continues on next page)

(continued from previous page)

```
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']
```

[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

    """
    raise Exception('TODO IMPLEMENT ME !')

try:
    fill(['a','b'], [None])
    raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
    "Test passed"

try:
```

(continues on next page)

(continued from previous page)

```
    fill(['a','b','c'], [None,None])
    raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
    "Test passed"

L1 = []
R1 = []
fill(L1, R1)

assert L1 == []
assert R1 == []


L = []
R = ['x']
fill(L, R)

assert L == []
assert R == ['x']


L = ['a']
R = ['x']
fill(L, R)

assert L == ['a']
assert R == ['a']


L = ['a']
R = ['x','y']
fill(L, R)

assert L == ['a']
assert R == ['a','y']

L = ['a','b']
R = ['x','y']
fill(L, R)

assert L == ['a','b']
assert R == ['a','b']

L = ['a','b']
R = ['x','y','z',]
fill(L, R)

assert L == ['a','b']
assert R == ['a','b','z']


L = ['a']
R = ['x','y','z',]
fill(L, R)

assert L == ['a']
assert R == ['a','y','z']
```

(continues on next page)

(continued from previous page)

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:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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
```

(continues on next page)

(continued from previous page)

```
    """


    n = len(chairs)
    m = len(chairs[0])

    if len(students) > n*m:
        raise ValueError("There are more students than chairs ! Students = %s, chairs␣
↪= %sx%s" % (len(students), n, m))

    i = 0
    j = 0
    students.sort()
    for s in students:
        chairs[i][j] = s

        if j == m - 1:
            j = 0
            i += 1
        else:
            j += 1



try:
    lab(['a','b'], [[None]])
    raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
    "Test passed"

try:
    lab(['a','b','c'], [[None,None]])
    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']
```

(continues on next page)

(continued from previous page)

```
            ]
assert r1 == None  # function is not meant to return anything (so returns None by␣
↪default)

m2 = [
        [None, None]
     ]
lab(['a'], m2)  # 1 student 2 chairs in one row

assert m2 == [
                ['a', None]
            ]


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

ss4 = ['b', 'a']
m4 = [
        [None, None]
     ]
lab(ss4, m4)  # 2 students 2 chairs in one row

assert m4 == [
                ['a','b']
            ]

assert ss4 == ['a', 'b']  # also modified input list as required by function text

m5 = [
        [None, None],
        [None, None]
     ]
lab(['b', 'c', 'a'], m5)  # 3 students 2x2 chairs

assert m5 == [
                ['a','b'],
                ['c', None]
            ]

m6 = [
        [None, None],
        [None, None]
     ]
lab(['b', 'd', 'c', 'a'], m6)  # 4 students 2x2 chairs

assert m6 == [
                ['a','b'],
                ['c','d']
            ]
```

(continues on next page)

(continued from previous page)

```
m7 = [
        [None, None, None],
        [None, None, None]
     ]
lab(['b', 'd', 'e', 'c', 'a'], m7)  # 5 students 3x2 chairs

assert m7 == [
                ['a','b','c'],
                ['d','e',None]
             ]

ss8 = ['b', 'd', 'e', 'g', 'c', 'a', 'h', 'f' ]
m8 = [
        [None, None, None],
        [None, None, None],
        [None, None, None],
        [None, None, None]
     ]
lab(ss8, m8)  # 8 students 3x4 chairs

assert m8 == [
                ['a',  'b',  'c'],
                ['d',  'e',  'f'],
                ['g',  'h',  None],
                [None, None, None],
             ]

assert ss8 == ['a','b','c','d','e','f','g','h']
```

[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)
```

(continues on next page)

(continued from previous page)

```
        # 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

    """
    raise Exception('TODO IMPLEMENT ME !')


try:
    lab(['a','b'], [[None]])
    raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
    "Test passed"

try:
    lab(['a','b','c'], [[None,None]])
    raise Exception("TEST FAILED: Should have failed before with a ValueError!")
except ValueError:
    "Test passed"


m0 = [
        [None]
     ]

r0 = lab([],m0)
assert m0 == [
                [None]
              ]
assert r0 == None  # function is not meant to return anything (so returns None by␣
↪default)


m1 = [
        [None]
     ]
r1 = lab(['a'], m1)

assert m1 == [
                ['a']
              ]
assert r1 == None  # function is not meant to return anything (so returns None by␣
↪default)

m2 = [
        [None, None]
     ]
```

(continues on next page)

(continued from previous page)

```
lab(['a'], m2)  # 1 student 2 chairs in one row

assert m2 == [
                ['a', None]
             ]


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

ss4 = ['b', 'a']
m4 = [
        [None, None]
     ]
lab(ss4, m4)  # 2 students 2 chairs in one row

assert m4 == [
                ['a','b']
             ]

assert ss4 == ['a', 'b']  # also modified input list as required by function text

m5 = [
        [None, None],
        [None, None]
     ]
lab(['b', 'c', 'a'], m5)  # 3 students 2x2 chairs

assert m5 == [
                ['a','b'],
                ['c', None]
             ]

m6 = [
        [None, None],
        [None, None]
     ]
lab(['b', 'd', 'c', 'a'], m6)  # 4 students 2x2 chairs

assert m6 == [
                ['a','b'],
                ['c','d']
             ]

m7 = [
        [None, None, None],
        [None, None, None]
     ]
lab(['b', 'd', 'e', 'c', 'a'], m7)  # 5 students 3x2 chairs
```

(continues on next page)

(continued from previous page)

```
assert m7 == [
                ['a','b','c'],
                ['d','e',None]
             ]

ss8 = ['b', 'd', 'e', 'g', 'c', 'a', 'h', 'f' ]
m8 = [
        [None, None, None],
        [None, None, None],
        [None, None, None],
        [None, None, None]
     ]
lab(ss8, m8)  # 8 students 3x4 chairs

assert m8 == [
                ['a',  'b',  'c'],
                ['d',  'e',  'f'],
                ['g',  'h',  None],
                [None, None, None],
             ]

assert ss8 == ['a','b','c','d','e','f','g','h']
```

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[68] and do read Falsehoods Programmers Believe About Phone Numbers[69]

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:

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

```
328 123 4567        ->  3281234567
0039 328 123 4567   ->  3281234567
0039 3771 1234567   ->  37711234567
```

REMEMBER: strings are immutable !!!!!

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

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

    p = phone.replace(' ', '')
    if p.startswith('0039'):
        p = p[4:]
    if p.startswith('+39'):
        p = p[3:]
    return p


assert canonical('+39 0461 123456') == '0461123456'
assert canonical('+390461123456') == '0461123456'
assert canonical('0039 0461 123456') == '0461123456'
assert canonical('00390461123456') == '0461123456'
assert canonical('003902123456') == '02123456'
assert canonical('003902120039') == '02120039'
assert canonical('0039021239') == '021239'
```

[4]:
```
def canonical(phone):
    """ RETURN the canonical version of phone as a string. See above for an␣
↪explanation.
    """
    raise Exception('TODO IMPLEMENT ME !')

assert canonical('+39 0461 123456') == '0461123456'
assert canonical('+390461123456') == '0461123456'
assert canonical('0039 0461 123456') == '0461123456'
assert canonical('00390461123456') == '0461123456'
assert canonical('003902123456') == '02123456'
assert canonical('003902120039') == '02120039'
assert canonical('0039021239') == '021239'
```

2.2 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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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)
    """

    c = canonical(phone)
    for m in mobile_prefixes:
        if c.startswith(m):
            return 'mobile'
    for p in province_prefixes:
        if c.startswith(p):
            return p
    return 'unrecognized'


assert prefix('0461123') == '0461'
assert prefix('+39 0461  4321') == '0461'
assert prefix('0039011 432434') == '011'
assert prefix('328 432434') == 'mobile'
assert prefix('+39340 432434') == 'mobile'
assert prefix('00666011 432434') == 'unrecognized'
assert prefix('12345') == 'unrecognized'
assert prefix('+39 123 12345') == 'unrecognized'
```

[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 !!
```

(continues on next page)

(continued from previous page)

```
        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)
    """
    raise Exception('TODO IMPLEMENT ME !')

assert prefix('0461123') == '0461'
assert prefix('+39 0461  4321') == '0461'
assert prefix('0039011 432434') == '011'
assert prefix('328 432434') == 'mobile'
assert prefix('+39340 432434') == 'mobile'
assert prefix('00666011 432434') == 'unrecognized'
assert prefix('12345') == 'unrecognized'
assert prefix('+39 123 12345') == 'unrecognized'
```

2.3 hist

Difficulty: ⊛⊛⊛

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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)
    """

    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
```

(continues on next page)

(continued from previous page)

```
            else:
                d[p] = 1
    return d


assert hist(['0461123']) == {'0461':1}
assert hist(['123']) == {'unrecognized':1}
assert hist(['328 123']) == {'mobile':1}
assert hist(['0461123','+390461123']) == {'0461':1}  # same canonicals, should be␣
↪counted only once
assert hist(['0461123', '+39 0461  4321']) == {'0461':2}
assert hist(['0461123', '+39 0461  4321', '0039011 432434']) == {'0461':2, '011':1}
assert hist(['+39   02 423', '0461123', '02 426', '+39 0461  4321', '0039328 1234567',
↪ '02 423', '02 424']) == {'0461':2, 'mobile':1,  '02':3}
```

[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)
    """
    raise Exception('TODO IMPLEMENT ME !')

assert hist(['0461123']) == {'0461':1}
assert hist(['123']) == {'unrecognized':1}
assert hist(['328 123']) == {'mobile':1}
assert hist(['0461123','+390461123']) == {'0461':1}  # same canonicals, should be␣
↪counted only once
assert hist(['0461123', '+39 0461  4321']) == {'0461':2}
assert hist(['0461123', '+39 0461  4321', '0039011 432434']) == {'0461':2, '011':1}
assert hist(['+39   02 423', '0461123', '02 426', '+39 0461  4321', '0039328 1234567',
↪ '02 423', '02 424']) == {'0461':2, 'mobile':1,  '02':3}
```

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[70]

You should obtain something like this:

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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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
```

[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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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']
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()
```

[8]:

2.1.2 Midterm - Fri 16 Nov 2018

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solution

Introduction

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[71]

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

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

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]
      ]
```

[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`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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
    """
```

(continues on next page)

(continued from previous page)

```
    if len(mata) != len(matb):
        raise ValueError("mata and matb have different row number a:%s b:%s!" %␣
↪(len(mata), len(matb)))


    n = len(mata)

    ret = []
    for i in range(n):
        row = []
        ret.append(row)
        for j in range(n):
            row.append(mata[i][j] or matb[i][j])
    return ret


try:
    union([[False],[False]], [[False]])
    raise Exception("Shouldn't arrive here !")
except ValueError:
    "test passed"

try:
    union([[False]], [[False],[False]])
    raise Exception("Shouldn't arrive here !")
except ValueError:
    "test passed"



ma1 =  [
            [False]
        ]
mb1 =  [
            [False]
      ]

assert union(ma1, mb1) == [
                           [False]
                         ]

ma2 =  [
            [False]
      ]
mb2 =  [
            [True]
      ]

assert union(ma2, mb2) == [
                           [True]
                         ]

ma3 =  [
            [True]
      ]
```

(continues on next page)

(continued from previous page)

```
mb3 =  [
            [False]
      ]

assert union(ma3, mb3) == [
                          [True]
                        ]


ma4 =  [
            [True]
      ]
mb4 =  [
            [True]
      ]

assert union(ma4, mb4) == [
                            [True]
                          ]

ma5 =  [
            [False, False, False],
            [False, False, False],
            [False, False, False]

       ]
mb5 =  [
            [True, False, True],
            [False, True, True],
            [False, False, False]
       ]

assert union(ma5, mb5) == [
                             [True, False, True],
                             [False, True, True],
                             [False, False, False]
                          ]

ma6 =  [
            [True, False, True],
            [False, True, True],
            [False, False, False]
      ]
mb6 =  [
            [False, False, False],
            [False, False, False],
            [False, False, False]

      ]

assert union(ma6, mb6) == [
                             [True, False, True],
                             [False, True, True],
                             [False, False, False]
                          ]

ma7 =  [
```

(continues on next page)

(continued from previous page)

```
            [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]

                        ]
```

[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
    """
    raise Exception('TODO IMPLEMENT ME !')

try:
    union([[False],[False]], [[False]])
    raise Exception("Shouldn't arrive here !")
except ValueError:
    "test passed"

try:
    union([[False]], [[False],[False]])
    raise Exception("Shouldn't arrive here !")
except ValueError:
    "test passed"



ma1 =  [
            [False]
         ]
mb1 =  [
            [False]
      ]

assert union(ma1, mb1) == [
                           [False]
                        ]

ma2 =  [
            [False]
```

(continues on next page)

(continued from previous page)

```
      ]
mb2 =  [
            [True]
      ]

assert union(ma2, mb2) == [
                          [True]
                        ]

ma3 =  [
            [True]
      ]
mb3 =  [
            [False]
      ]

assert union(ma3, mb3) == [
                          [True]
                        ]


ma4 =  [
            [True]
      ]
mb4 =  [
            [True]
      ]

assert union(ma4, mb4) == [
                            [True]
                          ]

ma5 =  [
            [False, False, False],
            [False, False, False],
            [False, False, False]

       ]
mb5 =  [
            [True, False, True],
            [False, True, True],
            [False, False, False]
       ]

assert union(ma5, mb5) == [
                             [True, False, True],
                             [False, True, True],
                             [False, False, False]
                          ]

ma6 =  [
            [True, False, True],
            [False, True, True],
            [False, False, False]
      ]
mb6 =  [
            [False, False, False],
```

(continues on next page)

(continued from previous page)

```
            [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)
```

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

data-jupman-hide="Hide">Show solution</a>

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

    n = len(mat)
    c = 0   # number of incoming edges found
    for j in range(len(mat)):      # go column by column
        for i in range(len(mat)):  # go row by row
            if mat[i][j]:
                c += 1
                break    # as you find first incoming edge, increment c and stop␣
↪search for that column
    return c == n




m1 =  [
          [False]
      ]

assert surjective(m1) == False


m2 =  [
          [True]
      ]

assert surjective(m2) == True

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

assert surjective(m3) == False


m4 =  [
          [False, True],
          [False, False],
      ]

assert surjective(m4) == False

m5 =  [
          [False, False],
          [True, False],
      ]

assert surjective(m5) == False

m6 =  [
          [False, False],
          [False, True],
```

(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
```

[11]:
```
def surjective(mat):
    """ RETURN True if provided graph mat as list of boolean lists is an
        nxn surjective binary relation, otherwise return False
    """
    raise Exception('TODO IMPLEMENT ME !')



m1 =  [
```

(continues on next page)

(continued from previous page)

```
        [False]
    ]

assert surjective(m1) == False


m2 =  [
        [True]
    ]

assert surjective(m2) == True

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

assert surjective(m3) == False


m4 =  [
        [False, True],
        [False, False],
    ]

assert surjective(m4) == False

m5 =  [
        [False, False],
        [True, False],
    ]

assert surjective(m5) == False

m6 =  [
        [False, False],
        [False, True],
    ]

assert surjective(m6) == False


m7 =  [
        [True, False],
        [True, False],
    ]

assert surjective(m7) == False

m8 =  [
        [True, False],
        [False, True],
    ]

assert surjective(m8) == True
```

(continues on next page)

(continued from previous page)

```
m9 =  [
         [True, True],
         [False, True],
     ]

assert surjective(m9) == True


m10 = [
        [True, True, False, False],
        [False, False,  False, True],
        [False, True, False, False],
        [False, True, False, False],

     ]
assert surjective(m10) == False

m11 = [
        [True, True, False, False],
        [False, False,  False, True],
        [False, True, True, False],
        [False, True, True, True],

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

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)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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'],
                      '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']
                                      }

    """


    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
```

(continues on next page)

(continued from previous page)

```
            if target not in db[key]:
                ret[key].append(target)
    return ret




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


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

da2 =  {
          'a': []
       }

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

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

da3 =  {
         'a': ['a']
       }
db3 =  {
          'a': []
       }

assert ediff(da3, db3) ==   {
                              'a': ['a']
                           }



da4 =  {
           'a': ['a']
       }
db4 =  {
           'a': ['a']
       }

assert ediff(da4, db4) == {
                           'a': []
                          }
da5 =  {
          'a':['b'],
          'b':[]
```

(continues on next page)

(continued from previous page)

```
         }
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':[]
         }

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'],
```

(continues on next page)

(continued from previous page)

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

db9 =  {
          'a':['c'],
          'b':['a','b', 'c'],
          'c':['a']
        }

assert ediff(da9, db9) == {
                            'a':['a'],
                            'b':[],
                            'c':['b','c']
                          }
```

[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'],
                      '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']
                                      }

    """
    raise Exception('TODO IMPLEMENT ME !')



da1 =  {
```

(continues on next page)

(continued from previous page)

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


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

da2 =   {
          'a': []
        }

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

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

da3 =   {
         'a': ['a']
        }
db3 =   {
          'a': []
        }

assert ediff(da3, db3) ==   {
                              'a': ['a']
                           }


da4 =   {
           'a': ['a']
        }
db4 =   {
           'a': ['a']
        }

assert ediff(da4, db4) == {
                           'a': []
                          }
da5 =   {
          'a':['b'],
          'b':[]
         }
db5 =   {
          'a':['b'],
          'b':[]
        }

assert ediff(da5, db5) == {
                          'a':[],
                          'b':[]
```

(continues on next page)

(continued from previous page)

```
                        }

da6 =  {
          'a':['b'],
          'b':[]
        }
db6 =  {
          'a':[],
          'b':[]
        }

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

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

assert ediff(da7, db7) == {
                            'a':['b'],
                            'b':[]
                          }


da8 =  {
          'a':['a','b'],
          'b':['a']
        }
db8 =  {
          'a':['a'],
          'b':['b']
        }

assert ediff(da8, db8) == {
                            'a':['b'],
                            'b':['a']
                          }

da9 =  {
          'a':['a','c'],
          'b':['b', 'c'],
          'c':['b','c']
        }

db9 =  {
          'a':['c'],
          'b':['a','b', 'c'],
          'c':['a']
        }
```

(continues on next page)

(continued from previous page)

```
assert ediff(da9, db9) == {
                            'a':['a'],
                            'b':[],
                            'c':['b','c']
                          }
```

2.1.3 Midterm - Thu 10, Jan 2019

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solution

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[72]

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:

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

```
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`?

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'],
```

(continues on next page)

(continued from previous page)

```
        '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'],
```

(continues on next page)

(continued from previous page)

```
        '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=[]
```

[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
        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
```

(continues on next page)

(continued from previous page)

```
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

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solution

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[73]
- 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`

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

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:

[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)]
        }
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
def table_to_adj(table):

    ret = {}
    headers = table[0]

    for row in table[1:]:
        lst = []
        for j in range(1, len(row)):
            lst.append((headers[j], row[j]))
        ret[row[0]] = lst
    return ret


m0 = [
        ['I','P','Q']
     ]
res0 = {}

assert res0 == table_to_adj(m0)

m1 =    [
            ['Identifier','Price','Quantity'],
            ['a',1,1],
            ['b',5,8],
            ['c',2,6],
            ['d',8,5],
```

(continues on next page)

(continued from previous page)

```
            ['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)
```

[4]:
```
def table_to_adj(table):
    raise Exception('TODO IMPLEMENT ME !')

m0 = [
        ['I','P','Q']
     ]
res0 = {}

assert res0 == table_to_adj(m0)
```

(continues on next page)

(continued from previous page)

```
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')],
            '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[74], 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', '0002402742018091220190621'),
              ('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', '0002402742018091220190621'),
              ('trip_headsign', 'Trento-Autostaz.'),
              ('direction_id', '0'),
              ('arrival_time', '06:26:00'),
              ('departure_time', '06:26:00'),
```

(continues on next page)

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

(continued from previous page)

```
                ('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)
```

(continues on next page)

(continued from previous page)

```
    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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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)
    """

    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)


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
    """

    stops = load_stops()

    %matplotlib inline
```

(continues on next page)

(continued from previous page)

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

[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)
    """
    raise Exception('TODO IMPLEMENT ME !')

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
```

(continues on next page)

(continued from previous page)

```
        - 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
    """
    raise Exception('TODO IMPLEMENT ME !')
```

```
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']
```

```
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']
```

(continues on next page)

(continued from previous page)

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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a>

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

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

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

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[75]
- If you don't have unitn login: tell instructors and we will download your work manually

[75] 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[76], 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[77] 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)
```

[76] https://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs
[77] 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:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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)

    """

    s = set()
    for diz in stops:
        s.add(diz['route_short_name'])
    ret = list(s)
    ret.sort()
    return ret
```

[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)

    """
    raise Exception('TODO IMPLEMENT ME !')
```

Example:

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

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

A2 to_int_min

Implement this function:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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)
    """

    hours = int(time_string[0:2])
    mins = int(time_string[3:5])
    return (hours * 60 + mins)
```

[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)
    """
    raise Exception('TODO IMPLEMENT ME !')
```

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',
```

(continues on next page)

(continued from previous page)

```
        '#d62728',
        '#9467bd',
        '#8c564b',
        '#e377c2',
        '#7f7f7f',
        '#bcbd22',
        '#17becf']

    return lst[i % 10]
```

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

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

Now implement this function:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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.
    """

    legend_edges = []
    i = 0
    routes = extract_routes(stops)

    for route_short_name in routes:
        legend_edges.append({
            'label': route_short_name,
            'color':get_color(i,len(routes))
        })
        i += 1
    return legend_edges
```

[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.
    """
    raise Exception('TODO IMPLEMENT ME !')
```

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

[15]:
```
[{'color': '#1f77b4', 'label': 'B201'},
 {'color': '#ff7f0e', 'label': 'B202'},
```

(continues on next page)

(continued from previous page)

```
 {'color': '#2ca02c', 'label': 'B211'},
 {'color': '#d62728', 'label': 'B217'},
 {'color': '#9467bd', 'label': 'B301'}]
```

A4 calc_nx

Implement this function:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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

    """


    G = nx.DiGraph()

    G.graph['graph']= {
                        'rankdir':'LR',  # horizontal layout ,

                      }

    G.name = '*************  calc_nx  SOLUTION '

    routes = extract_routes(stops)


    i = 0

    for route_short_name in routes:

        prev_diz = None

        for diz in stops:

            if diz['route_short_name'] == route_short_name:
```

(continues on next page)

(continued from previous page)

```
                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
```

[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

    """
    raise Exception('TODO IMPLEMENT ME !')
```

[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`:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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

    """


    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'
```

(continues on next page)

(continued from previous page)

```
    draw_nx(G, legend_edges=get_legend_edges())
```

[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

    """
    raise Exception('TODO IMPLEMENT ME !')
    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`:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

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

    """


    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)
```

(continues on next page)

(continued from previous page)

```
    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()
```

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

    """
    raise Exception('TODO IMPLEMENT ME !')
```

[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:
```

(continues on next page)

(continued from previous page)

```
        # 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)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a>

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 *

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)

        - 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)])
```

(continues on next page)

(continued from previous page)

```
  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
```

(continues on next page)

(continued from previous page)

```
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
 │├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
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 - Mon 10, Jun 2019

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

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

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[78]
- 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-2019-06-10.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[79]

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

[78] http://examina.icts.unitn.it/studente
[79] https://dati.trentino.it/dataset/patrimonio-immobiliare

[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', ''),
              ('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 ..)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
def calc_types_hist(db):


    tipologie = {}
    for diz in db:
        if diz['Tipologia'].startswith('ALLOGGIO'):
            chiave = 'ALLOGGIO'
        elif diz['Tipologia'].startswith('POSTO MACCHINA'):
            chiave = 'POSTO MACCHINA'
        else:
            chiave = diz['Tipologia']

        if chiave in tipologie:
            tipologie[chiave] += 1
        else:
            tipologie[chiave] = 1

    return tipologie


calc_types_hist(itea)
```

[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,
 'TETTOIA': 2,
 'ARCHIVIO ITEA': 10,
 'SALA / ATTIVITA SOCIALI': 45,
 'AREA URBANA': 6,
 'ASILO': 1,
 'CASERMA': 2,
 'LABORATORIO PER ARTI E MESTIERI': 3,
```

(continues on next page)

(continued from previous page)

```
 'MUSEO': 1,
 'SOFFITTA': 3,
 'AMBULATORIO': 1,
 'LEGNAIA': 3,
 'RUDERE': 1}
```

[4]:
```
def calc_types_hist(db):
    raise Exception('TODO IMPLEMENT ME !')

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,
 '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[80] and also in python documentation about sorting.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[5]:
```
def calc_types_series(hist):

    ret = []

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

(continues on next page)

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

(continued from previous page)

```
    ret.sort(key=lambda c: c[1],reverse=True)
    return ret[:10]


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)]
```

[5]:
```
def calc_types_series(hist):
    raise Exception('TODO IMPLEMENT ME !')

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)]
```

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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

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

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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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()
```

[7]:

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[81]

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' !

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

[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')]),
 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:

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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.

    """

    ret = {}
    counts = {}
    for diz in db:
        t = (diz['Stazione'], diz['Inquinante'])
        if t in ret:
            ret[t] += float(diz['Valore'])
            counts[t] += 1
        else:
            ret[t] = float(diz['Valore'])
            counts[t] = 1
```

(continues on next page)

(continued from previous page)

```
    for t in ret:
        ret[t] /= counts[t]
    return ret


calc_avg_pollution(air_quality)
```

[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,
 ('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}
```

[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.

    """
    raise Exception('TODO IMPLEMENT ME !')

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,
```

(continues on next page)

(continued from previous page)

```
 ('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,
 ('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)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a>

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.
```

(continues on next page)

(continued from previous page)

```
        - 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
 /|\     /
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
```

(continues on next page)

(continued from previous page)

```
        - 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')))
```

(continues on next page)

(continued from previous page)

```
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

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[82]
  - write pythonic code[83]
  - 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:

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

```
# 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[84]
- 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-2019-07-02.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[85]

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()
```

[84] http://examina.icts.unitn.it/studente
[85] https://dati.trentino.it/dataset/botteghe-storiche-del-trentino

> **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'),
              ('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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
def rank_categories(db, stopwords):

    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)


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)
```

(continues on next page)

(continued from previous page)

```
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)]
```

[4]:
```
def rank_categories(db, stopwords):
    raise Exception('TODO IMPLEMENT ME !')

stopwords = ['',
             'S.N.C.', 'SNC','S.A.S.', 'S.R.L.', 'S.C.A.R.L.', 'SCARL','S.A.S',
↪'COMMERCIALE','FAMIGLIA','COOPERATIVA',
             '-', '&', 'C.', 'ESERCIZIO',
             'IL', 'DE', 'DI','A', 'DA', 'E', 'LA', 'AL',  'DEL', 'ALLA', ]
categories = rank_categories(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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[6]:
```
# SOLUTION

%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()
```

[6]:

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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[7]:
```
def enrich(db, categories):

    ret = []

    for diz in db:
        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



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',
```

(continues on next page)

(continued from previous page)

```
  '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': '',
  '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']}]
```

[7]:
```
def enrich(db, categories):
    raise Exception('TODO IMPLEMENT ME !')


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': '',
  '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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[8]:
```
def dump(mat, q):

    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:
        raise ValueError("Couldn't fill the dump, %s tons remain!")
```

(continues on next page)

(continued from previous page)

```
m1 = [
    [5]
]

assert dump(m1,0) == []  # nothing to dump

m2 = [
    [4]
]

assert dump(m2,2) == [2]

m3 = [
    [5,4]
]

assert dump(m3,3) == [2, 1]


m3 = [
    [5,7,3]
]

assert dump(m3,3) == [2, 0, 1]


m5 = [
    [2,5],    # 5 2
    [4,3]     # 3 1

]

assert dump(m5,11) == [5,2,3,1]


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
```

[8]:
```
def dump(mat, q):
    raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
m1 = [
    [5]
]

assert dump(m1,0) == []  # nothing to dump

m2 = [
    [4]
]

assert dump(m2,2) == [2]

m3 = [
    [5,4]
]

assert dump(m3,3) == [2, 1]


m3 = [
    [5,7,3]
]

assert dump(m3,3) == [2, 0, 1]


m5 = [
    [2,5],    # 5 2
    [4,3]     # 3 1

]

assert dump(m5,11) == [5,2,3,1]


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
```

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)
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a>

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]
                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-blocks` 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:

[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|
        | 22|
        |1 1|

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

(continues on next page)

(continued from previous page)

```
        |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`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[ ]:

2.1.8 Exam - Mon 26, Aug 2019

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[86]
  - write pythonic code[87]
  - 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")
```

[86] https://sciprog.davidleoni.it/commandments.html
[87] 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-2019-08-26-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[88]
- 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[89]

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

[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()
```

[88] http://examina.icts.unitn.it/studente
[89] https://dati.trentino.it/dataset/personale-accademico-e-tecnico-amministrativo-dell-universita-di-trento

> **IMPORTANT**: look at the dataset !
>
> Here we show only first 2 rows, but to get a clear picture of the dataset you should explore it further.

The dataset contains a list of employees, each of whom may have one or more positions, in one or more university units. Each unit is identified by a code like `STO0000435`:

[2]:
```
unitn[:2]
```

[2]:
```
[{'givenName': 'NAME-1',
  'phone': ['0461 283752'],
  'identifier': 'eb9139509dc40d199b6864399b7e805c',
  'familyName': 'SURNAME-1',
  'positions': [{'unitIdentifier': 'STO0008929',
    'role': 'Staff',
    'unitName': 'Student Support Service: Economics, Law and International Studies'}]}
↪,
 {'givenName': 'NAME-2',
  'phone': ['0461 281521'],
  'identifier': 'b6292ffe77167b31e856d2984544e45b',
  'familyName': 'SURNAME-2',
  'positions': [{'unitIdentifier': 'STO0000435',
    'role': 'Associate professor',
    'unitName': 'Doctoral programme - Physics'},
   {'unitIdentifier': 'STO0000435',
    'role': 'Deputy coordinator',
    'unitName': 'Doctoral programme - Physics'},
   {'unitIdentifier': 'STO0008627',
    'role': 'Associate professor',
    'unitName': 'Department of Physics'}]}]
```

Department names can be very long, so when you need to display them you can use the function this `abbreviate`.

NOTE: function is already fully implemented, do *not* modify it.

[3]:
```
def abbreviate(unitName):

    abbreviations = {

        "Department of Psychology and Cognitive Science": "COGSCI",
        "Center for Mind/Brain Sciences - CIMeC":"CIMeC",
        "Department of Civil, Environmental and Mechanical Engineering":"DICAM",
        "Centre Agriculture Food Environment - C3A":"C3A",
        "School of International Studies - SIS":"SIS",
        "Department of Sociology and social research": "Sociology",
        "Faculty of Law": "Law",
        "Department of Economics and Management": "Economics",
        "Department of Information Engineering and Computer Science":"DISI",
        "Department of Cellular, Computational and Integrative Biology - CIBIO":"CIBIO
↪",
        "Department of Industrial Engineering":"DII"
    }
    if unitName in abbreviations:
        return abbreviations[unitName]
    else:
        return unitName.replace("Department of ", "")
```

Example:

[4]:
```
abbreviate("Department of Information Engineering and Computer Science")
```

[4]:
```
'DISI'
```

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',
 .
 .
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[5]:
```
def calc_uid_to_abbr(db):

    ret = {}
    for person in db:
        for position in person['positions']:
            uid = position['unitIdentifier']
            ret[uid] = abbreviate(position['unitName'])
    return ret


#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
```

[5]:
```
def calc_uid_to_abbr(db):
    raise Exception('TODO IMPLEMENT ME !')

#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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[6]:
```
def calc_prof_roles(db):

    hist = {}
    uid_to_abbr = calc_uid_to_abbr(db)

    for person in db:
        for position in person['positions']:

            role = position['role']
            uid = position['unitIdentifier']
            if 'professor'.lower() in role.lower():
                if uid in hist:
                    hist[uid] += 1
                else:
                    hist[uid] = 1

    ret = [(uid_to_abbr[x[0]],x[1]) for x in hist.items()]
    ret.sort(key=lambda c: c[1], reverse=True)
    return ret[:10]


#calc_prof_roles(unitn)
```

[6]:
```
def calc_prof_roles(db):
    raise Exception('TODO IMPLEMENT ME !')

#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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[8]:
```
%matplotlib inline
import matplotlib.pyplot as plt


def plot_profs(db):



    prof_roles = calc_prof_roles(db)

    xs = list(range(len(prof_roles)))
    xticks = [p[0] for p in prof_roles]
    ys = [p[1] for p in prof_roles]

    fig = plt.figure(figsize=(20,3))

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

    plt.title("Professor roles per department SOLUTION")
    plt.xticks(xs, xticks)

    plt.xlabel('departments')
    plt.ylabel('professor roles')

    plt.show()


#plot_profs(unitn)
```

[8]:
```
%matplotlib inline
import matplotlib.pyplot as plt


def plot_profs(db):
    raise Exception('TODO IMPLEMENT ME !')

#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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[10]:
```
def calc_roles(db):

    ret = {}
    for person in db:
        for position in person['positions']:
            uid = position['unitIdentifier']
            role = position['role']
            grouped_role = None
            if "professor" in role.lower():
                grouped_role = 'Professor'
            elif "research" in role.lower():
                grouped_role = 'Research'
            elif "teaching" in role.lower():
                grouped_role = 'Teaching'
            elif "guest" in role.lower():
                grouped_role = 'Guest'

            if grouped_role:
                if uid in ret:
                    if grouped_role in ret[uid]:
                        ret[uid][grouped_role] += 1
                    else:
                        ret[uid][grouped_role] = 1
                else:
                    diz = {}
                    diz[grouped_role] = 1
                    ret[uid] = diz

    return ret


#print(calc_roles(unitn)['STO0000001'])
#print(calc_roles(unitn)['STO0000006'])
#print(calc_roles(unitn)['STO0000012'])
#print(calc_roles(unitn)['STO0008629'])
```

[10]:
```
def calc_roles(db):
    raise Exception('TODO IMPLEMENT ME !')

#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[90]
- For on example on plotting side by side, see this[91]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[11]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_roles(db, abbrs):

    fig = plt.figure(figsize=(15,4))
    uid_to_abbr = calc_uid_to_abbr(db)

    for i in range(len(abbrs)):

            abbr = abbrs[i]
            roles = calc_roles(db)

            uid = None
```

(continues on next page)

[90] https://sciprog.davidleoni.it/visualization/visualization-sol.html#Pie-chart
[91] https://sciprog.davidleoni.it/visualization/visualization-sol.html#Showing-plots-side-by-side

(continued from previous page)

```
            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 )



#plot_roles(unitn, ['DISI','Sociology', 'COGSCI'])
```

[11]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_roles(db, abbrs):
    raise Exception('TODO IMPLEMENT ME !')


#plot_roles(unitn, ['DISI','Sociology', 'COGSCI'])
```

[12]:
```
# EXPECTED RESULT
plot_roles(unitn, ['DISI','Sociology', 'COGSCI'])
```

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    --
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[13]:
```
def calc_shared(db):

    ret = {}
    uid_to_people = {}

    uid_to_abbr = calc_uid_to_abbr(db)

    for person in db:

        for position in person['positions']:
            uid = position['unitIdentifier']
            if not uid in uid_to_people:
                uid_to_people[uid] = set()
            uid_to_people[uid].add(person['identifier'])

    uids = list(uid_to_people)

    ret = []
    for x in range(len(uids)):
```

(continues on next page)

(continued from previous page)

```
        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


#calc_shared(unitn)
```

[13]:
```
def calc_shared(db):
    raise Exception('TODO IMPLEMENT ME !')

#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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[15]:
```
import matplotlib.pyplot as plt

%matplotlib inline

def plot_shared(db):
```

(continues on next page)

(continued from previous page)

```
    uid_to_abbr = calc_uid_to_abbr(db)

    shared = calc_shared(db)
    xs = range(len(shared))

    xticks = [x[0] + "\n" + x[1] for x in shared]

    ys = [x[2] for x in shared]

    fig = plt.figure(figsize=(20,3))

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

    plt.title("SOLUTION")
    plt.xticks(xs, xticks)

    plt.xlabel('Department pairs')
    plt.ylabel('common employees')

    plt.show()


#plot_shared(unitn)
```

[15]:
```
import matplotlib.pyplot as plt

%matplotlib inline

def plot_shared(db):
    raise Exception('TODO IMPLEMENT ME !')

#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]))
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a>

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):
        """ 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:
              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 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

Scientific Programming - Data Science @ University of Trento

Introduction

This is only a simulation. By participating to it, you gain nothing, and you lose nothing

Part A - EURES Job Offers

MOVED TO en.softpython.org/pandas/eures-jobs-sol.html[92]

You will be redirected in 10 seconds …

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[1]:
```
# SOLUTION
```

[1]:

[ ]:

[1]:
```
#Please execute this cell
import jupman;
```

2.1.10 Midterm - Thu 07, Nov 2019

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

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

[92] https://en.softpython.org/pandas/eures-jobs-sol.html

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")
```

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:

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

```
# 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()
```

```
<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
```

(continues on next page)

[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

(continued from previous page)

```
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])

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
def is_leap(year):

    if year % 4 == 0:
        if year % 100 == 0:
            return year % 400 == 0
        else:
            return True
    else:
        return False



assert is_leap(4)    == True
assert is_leap(104)  == True
assert is_leap(204)  == True
assert is_leap(400)  == True
assert is_leap(1600) == True
assert is_leap(2000) == True
assert is_leap(2400) == True
assert is_leap(2000) == True
assert is_leap(2004) == True
assert is_leap(2008) == True
assert is_leap(2012) == True

assert is_leap(1)    == False
assert is_leap(5)    == False
assert is_leap(100)  == False
assert is_leap(200)  == False
assert is_leap(1700) == False
assert is_leap(1800) == False
assert is_leap(1900) == False
assert is_leap(2100) == False
assert is_leap(2200) == False
assert is_leap(2300) == False
assert is_leap(2500) == False
assert is_leap(2600) == False
```

[98] https://docs.microsoft.com/en-us/office/troubleshoot/excel/determine-a-leap-year

[4]:
```
def is_leap(year):
    raise Exception('TODO IMPLEMENT ME !')


assert is_leap(4)    == True
assert is_leap(104)  == True
assert is_leap(204)  == True
assert is_leap(400)  == True
assert is_leap(1600) == True
assert is_leap(2000) == True
assert is_leap(2400) == True
assert is_leap(2000) == True
assert is_leap(2004) == True
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
```

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']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[6]:
```
def full_date(text):

    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)


assert full_date("Giovedì 14 novembre 2019") == "2019-11-14"
assert full_date("Giovedì 7 novembre 2019") == "2019-11-07"
assert full_date("Giovedì pomeriggio 14 novembre 2019") == "2019-11-14"
assert full_date("sabato mattina 25 marzo 2017") == "2017-03-25"
assert full_date("Mercoledì 13 Novembre 2019") == "2019-11-13"
assert full_date("domenica 4 dicembre 1923") == "1923-12-04"
```

[6]:
```
def full_date(text):
    raise Exception('TODO IMPLEMENT ME !')

assert full_date("Giovedì 14 novembre 2019") == "2019-11-14"
assert full_date("Giovedì 7 novembre 2019") == "2019-11-07"
assert full_date("Giovedì pomeriggio 14 novembre 2019") == "2019-11-14"
assert full_date("sabato mattina 25 marzo 2017") == "2017-03-25"
assert full_date("Mercoledì 13 Novembre 2019") == "2019-11-13"
assert full_date("domenica 4 dicembre 1923") == "1923-12-04"
```

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']
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[8]:
```
def partial_date(text):

    if type(text) != str:
        return 'yyyy-mm-dd'

    year = 'yyyy'
    month = 'mm'
    day = 'dd'

    ntext = text.lower()
    ret = []
    words = ntext.split()

    if len(words) > 0:
        if words[0] == connective_from:
            i = 1
        else:
            i = 0
        if words[i] in days:
            i = i + 1
```

(continues on next page)

(continued from previous page)

```
            if words[i] in day_phases:
                i += 1
            day = "{:02d}".format(int(words[i]))
            i += 1
            if i < len(words):
                # 'e' case with double date
                if words[i] in months:
                    month = "{:02d}".format(months.index(words[i]) + 1)
                    i += 1
                    if i < len(words):
                        if words[i].isdigit():
                            year = "{:04d}".format(int(words[i]))

    return "%s-%s-%s" % (year, month, day)


# complete, uppercase day
assert partial_date("Giovedì 7 novembre 2019") == "2019-11-07"
assert partial_date("Giovedì 14 novembre 2019") == "2019-11-14"
# lowercase day
assert partial_date("mercoledì 13 novembre 2019") == "2019-11-13"
# lowercase, dayphase, missing month and year
assert partial_date("venerdì pomeriggio 15") == "yyyy-mm-15"
# single day, lowercase, no year
assert partial_date("venerdì 15 novembre") == "yyyy-11-15"

# no year,   hour / location to be discarded
assert partial_date("venerdì 5 aprile alle 20:30 in via degli Olmi 26 (Trento sud)")\
                    == "yyyy-04-05"

# two dates, 'and' connective ('e'), day phase morning/afternoon ('mattina'/
↪'pomeriggio')
assert partial_date("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019") \
                    == "yyyy-mm-15"

# two dates, begins with connective 'Da'
assert partial_date("Da lunedì 25 novembre a domenica 01 dicembre 2019") == "yyyy-11-
↪25"
assert partial_date("da giovedì 12 a domenica 15 dicembre 2019") == "yyyy-mm-12"
assert partial_date("da giovedì 9 a domenica 12 gennaio 2020") == "yyyy-mm-09"
assert partial_date("Da lunedì 04 a domenica 10 novembre 2019") == "yyyy-mm-04"
```

[8]:
```
def partial_date(text):
    raise Exception('TODO IMPLEMENT ME !')

# complete, uppercase day
assert partial_date("Giovedì 7 novembre 2019") == "2019-11-07"
assert partial_date("Giovedì 14 novembre 2019") == "2019-11-14"
# lowercase day
assert partial_date("mercoledì 13 novembre 2019") == "2019-11-13"
# lowercase, dayphase, missing month and year
assert partial_date("venerdì pomeriggio 15") == "yyyy-mm-15"
# single day, lowercase, no year
assert partial_date("venerdì 15 novembre") == "yyyy-11-15"
```

(continues on next page)

(continued from previous page)

```
# 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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[9]:
```
def parse_dates_and(text):

    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)
```

(continues on next page)

(continued from previous page)

```
# complete dates
assert parse_dates_and("lunedì 25 aprile 2018 e domenica 01 dicembre 2019") == ("2018-
↪04-25","2019-12-01")

# exactly two dates, day phase morning/afternoon ('mattina'/'pomeriggio')
assert parse_dates_and("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019") == (
↪"2019-11-15", "2019-11-16")

# first date missing year
assert parse_dates_and("lunedì 13 settembre e sabato 25 dicembre 2019") == ("2019-09-
↪13","2019-12-25")

# first date missing month and year
assert parse_dates_and("Giovedì 12 e domenica 15 dicembre 2019") == ("2019-12-12",
↪"2019-12-15")

assert parse_dates_and("giovedì 9 e domenica 12 gennaio 2020") == ("2020-01-09",
↪"2020-01-12")

assert parse_dates_and("lunedì 4 e domenica 10 novembre 2019") == ("2019-11-04","2019-
↪11-10")

# first missing month and year, second missing year
assert parse_dates_and("lunedì 4 e domenica 10 novembre") == ("yyyy-11-04","yyyy-11-10
↪")

# first missing month and year, second missing month and year
assert parse_dates_and("lunedì 4 e domenica 10") == ("yyyy-mm-04","yyyy-mm-10")
```

[9]:

```
def parse_dates_and(text):
    raise Exception('TODO IMPLEMENT ME !')


# complete dates
assert parse_dates_and("lunedì 25 aprile 2018 e domenica 01 dicembre 2019") == ("2018-
↪04-25","2019-12-01")

# exactly two dates, day phase morning/afternoon ('mattina'/'pomeriggio')
assert parse_dates_and("venerdì pomeriggio 15 e sabato mattina 16 novembre 2019") == (
↪"2019-11-15", "2019-11-16")

# 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")
```

(continues on next page)

(continued from previous page)

```
assert parse_dates_and("lunedì 4 e domenica 10 novembre 2019") == ("2019-11-04","2019-
↪11-10")

# first missing month and year, second missing year
assert parse_dates_and("lunedì 4 e domenica 10 novembre") == ("yyyy-11-04","yyyy-11-10
↪")

# first missing month and year, second missing month and year
assert parse_dates_and("lunedì 4 e domenica 10") == ("yyyy-mm-04","yyyy-mm-10")
```

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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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"),
```

(continues on next page)

[99] https://en.wikipedia.org/wiki/Functional_illiteracy

(continued from previous page)

```
    ("foreign policy","immigrants are stealing our jobs"),
]


def fake_news(subjects, antecedents,consequences):

    ret = []
    for subject in subjects:
        for ant in antecedents:
            for con in consequences:
                if ant[1] == con[0]:
                    ret.append(subject + ' ' + ant[0] + ', ' + con[1])
    return ret



#fake_news(db_subjects, db_antecedents, db_consequences)
```

[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):
    raise Exception('TODO IMPLEMENT ME !')


#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',
 '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.

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[100]
- If you don't have unitn login: tell instructors and we will download your work manually

Part B

B1 Theory

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

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

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')
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,
                bt(1),
                bt(7,
                    bt(5))),
        bt(9,
```

(continues on next page)

(continued from previous page)

```
                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']

[ ]:

[1]:

```
# Please execute this cell
import sciprog
```

2.1.12 Exam - Thu 23, Jan 2020

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

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

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[101]
- 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-2020-01-23.ipynb`

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

Metamath

Metamath[102] 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[103] up to the most basic axioms[104] 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:

- 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`,`$f`,`${`,`$}`, `$.`

[102] http://us.metamath.org
[103] http://us.metamath.org/mm_100.html
[104] http://us.metamath.org/mpeuni/mmtheorems1.html#mm5s

- 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[105]), 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 )'
        },
 .
 .
 .
}
```

[105] https://sciprog.davidleoni.it/formats/formats-sol.html#1.-line-files

A.1 Metamath db

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[2]:
```
def parse_db(filepath):

    ret = {}
    with open(filepath, encoding='utf-8') as f:
        line=f.readline().strip()
        while line != "":
            #print(line)

            if line.startswith('$('):
                label = ''
                keyword = '$('
                sequence = ''
            elif line.split()[0].startswith('${'):
                label = ''
                keyword = '${'
                sequence = ''
            elif line.split()[0].startswith('$}'):
                label = ''
                keyword = '$}'
                sequence = ''
            elif line.split()[0].startswith('$'):
                label = ''
                keyword = line.split()[0]
                sequence = line.split()[1][:-2].strip()
            else:
                label = line.split(' $')[0].strip()
                keyword = line.split()[1]
                if line.endswith('$.'):
                    sequence = line.split(keyword)[1][1:-2].strip()

            if label:
                ret[label] = {
                    'keyword' : keyword,
                    'sequence' : sequence
                }
                #print('   DEBUG: FOUND', label, ':', ret[label])
            #else:
                #print('   DEBUG: DISCARDED')
            line=f.readline().strip()
    return ret


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)
```

[2]:
```
def parse_db(filepath):
    raise Exception('TODO IMPLEMENT ME !')

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
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
```

(continues on next page)

(continued from previous page)

```
10 9 a2          $a |- ( t + 0 ) = t
11 tt            $f term t
12 tze           $a term 0
13 11,12 tpl     $a term ( t + 0 )
14 tt            $f term t
15 13,14 weq     $a wff ( t + 0 ) = t
16 tt            $f term t
17 tze           $a term 0
18 16,17 tpl     $a term ( t + 0 )
19 tt            $f term t
20 18,19 weq     $a wff ( t + 0 ) = t
21 tt            $f term t
22 tt            $f term t
23 21,22 weq     $a wff t = t
24 20,23 wim     $a wff ( ( t + 0 ) = t -> t = t )
25 tt            $f term t
26 25 a2         $a |- ( t + 0 ) = t
27 tt            $f term t
28 tze           $a term 0
29 27,28 tpl     $a term ( t + 0 )
30 tt            $f term t
31 tt            $f term t
32 29,30,31 a1   $a |- ( ( t + 0 ) = t -> ( ( t + 0 ) = t -> t = t ) )
33 15,24,26,32 mp  $a |- ( ( t + 0 ) = t -> t = t )
34 5,8,10,33 mp  $a |- t = t
```

Each line represents a step of the proof. Last line is the final goal of the proof.

Each line contains, in order:

- a step number at the beginning, starting from 1 (`step_id`)
- *possibly* a list of other step_ids, separated by commas, like `29,30,31` - they are references to previous rows
- label of the `db_mm` statement referenced by the step, like `tt`, `tze`, `weq` - that label must have been defined somewhere in `db.mm` file
- statement type: a token starting with a dollar, like `$a`, `$f`
- a sequence of characters, like (for you they are just characters, **don't care about the meaning** !):
  - `term ( t + 0 )`
  - `|- ( ( t + 0 ) = t -> ( ( t + 0 ) = t -> t = t ) )`

Implement function `parse_proof`, which takes a `filepath` to the proof and RETURN a list of steps expressed as a dictionary, in this format (showing here only first 5 items):

NOTE: referenced `step_ids` are **integer** numbers and they are the original ones from the file, meaning they start **from one**.

```
[
    {'keyword': '$f',
     'label': 'tt',
     'sequence': 'term t',
     'step_ids': []},
    {'keyword': '$a',
     'label': 'tze',
     'sequence': 'term 0',
     'step_ids': []},
```

(continues on next page)

(continued from previous page)

```
    {'keyword': '$a',
     'label': 'tpl',
     'sequence': 'term ( t + 0 )',
     'step_ids': [1,2]},
    {'keyword': '$f',
     'label': 'tt',
     'sequence': 'term t',
     'step_ids': []},
    {'keyword': '$a',
     'label': 'weq',
     'sequence': 'wff ( t + 0 ) = t',
     'step_ids': [3,4]},
     .
     .
     .
]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
def parse_proof(filepath):

    ret = []

    with open(filepath, encoding='utf-8') as f:
        line=f.readline().strip()

        while line != "":

            step_id = int(line.split(' ')[0])
            label = line.split('$')[0].strip().split(' ')[-1]
            keyword = '$' + line.split('$')[1][:1]
            sequence = line.split('$')[1][2:]
            candidate_step_ids = line.split(' ')[1]

            if candidate_step_ids != label:
                step_ids = [int(x) for x in line.split(' ')[1].split(',')]
            else:
                step_ids = []
            #print('deps =', deps)

            ret.append( {
                'step_ids': step_ids,
                'sequence': sequence,
                'label': label,
                'keyword': keyword
            })

            line=f.readline().strip()
        return ret


proof = parse_proof('data/proof.txt')

assert proof[0] == {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids':␣
↪[]}
```

(continues on next page)

(continued from previous page)

```
assert proof[1] == {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids':
↪ []}
assert proof[2] == {'keyword': '$a',
                    'label': 'tpl',
                    'sequence': 'term ( t + 0 )',
                    'step_ids': [1, 2]}
assert proof[4] == {'keyword': '$a',
                    'label': 'weq',
                    'sequence': 'wff ( t + 0 ) = t',
                    'step_ids': [3,4]}
assert proof[33] == { 'keyword': '$a',
                      'label': 'mp',
                      'sequence': '|- t = t',
                      'step_ids': [5, 8, 10, 33]}

pprint(proof)
```

```
[{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
  'label': 'tpl',
  'sequence': 'term ( t + 0 )',
  'step_ids': [1, 2]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'weq',
  'sequence': 'wff ( t + 0 ) = t',
  'step_ids': [3, 4]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'weq', 'sequence': 'wff t = t', 'step_ids': [6, 7]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'a2',
  'sequence': '|- ( t + 0 ) = t',
  'step_ids': [9]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
  'label': 'tpl',
  'sequence': 'term ( t + 0 )',
  'step_ids': [11, 12]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'weq',
  'sequence': 'wff ( t + 0 ) = t',
  'step_ids': [13, 14]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
  'label': 'tpl',
  'sequence': 'term ( t + 0 )',
  'step_ids': [16, 17]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'weq',
  'sequence': 'wff ( t + 0 ) = t',
```

(continues on next page)

(continued from previous page)

```
 'step_ids': [18, 19]},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a',
 'label': 'weq',
 'sequence': 'wff t = t',
 'step_ids': [21, 22]},
{'keyword': '$a',
 'label': 'wim',
 'sequence': 'wff ( ( t + 0 ) = t -> t = t )',
 'step_ids': [20, 23]},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a',
 'label': 'a2',
 'sequence': '|- ( t + 0 ) = t',
 'step_ids': [25]},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
{'keyword': '$a',
 'label': 'tpl',
 'sequence': 'term ( t + 0 )',
 'step_ids': [27, 28]},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
{'keyword': '$a',
 'label': 'a1',
 'sequence': '|- ( ( t + 0 ) = t -> ( ( t + 0 ) = t -> t = t ) )',
 'step_ids': [29, 30, 31]},
{'keyword': '$a',
 'label': 'mp',
 'sequence': '|- ( ( t + 0 ) = t -> t = t )',
 'step_ids': [15, 24, 26, 32]},
{'keyword': '$a',
 'label': 'mp',
 'sequence': '|- t = t',
 'step_ids': [5, 8, 10, 33]}]
```

[4]:
```
def parse_proof(filepath):
    raise Exception('TODO IMPLEMENT ME !')

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',
```

(continues on next page)

(continued from previous page)

```
                        'label': 'mp',
                        'sequence': '|- t = t',
                        'step_ids': [5, 8, 10, 33]}

pprint(proof)
```

```
[{'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
  'label': 'tpl',
  'sequence': 'term ( t + 0 )',
  'step_ids': [1, 2]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'weq',
  'sequence': 'wff ( t + 0 ) = t',
  'step_ids': [3, 4]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'weq', 'sequence': 'wff t = t', 'step_ids': [6, 7]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'a2',
  'sequence': '|- ( t + 0 ) = t',
  'step_ids': [9]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
  'label': 'tpl',
  'sequence': 'term ( t + 0 )',
  'step_ids': [11, 12]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'weq',
  'sequence': 'wff ( t + 0 ) = t',
  'step_ids': [13, 14]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
  'label': 'tpl',
  'sequence': 'term ( t + 0 )',
  'step_ids': [16, 17]},
 {'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]},
```

(continues on next page)

(continued from previous page)

```
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'a2',
  'sequence': '|- ( t + 0 ) = t',
  'step_ids': [25]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a', 'label': 'tze', 'sequence': 'term 0', 'step_ids': []},
 {'keyword': '$a',
  'label': 'tpl',
  'sequence': 'term ( t + 0 )',
  'step_ids': [27, 28]},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$f', 'label': 'tt', 'sequence': 'term t', 'step_ids': []},
 {'keyword': '$a',
  'label': 'a1',
  'sequence': '|- ( ( t + 0 ) = t -> ( ( t + 0 ) = t -> t = t ) )',
  'step_ids': [29, 30, 31]},
 {'keyword': '$a',
  'label': 'mp',
  'sequence': '|- ( ( t + 0 ) = t -> t = t )',
  'step_ids': [15, 24, 26, 32]},
 {'keyword': '$a',
  'label': 'mp',
  'sequence': '|- t = t',
  'step_ids': [5, 8, 10, 33]}]
```

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`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[9]:
```
# write here
```

[9]:
```
# write here
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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")
```

(continues on next page)

(continued from previous page)

```
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
tpl :    term ( t + r )
tze :    term 0
a2 :     |- ( t + 0 ) = t
wim :    wff ( P -> Q )
mp :     |- Q
a1 :     |- ( t = r -> ( t = s -> r = s ) )
```

[10]:

```
tt :      term t
weq :     wff t = r
tpl :     term ( t + r )
tze :     term 0
a2 :      |- ( t + 0 ) = t
wim :     wff ( P -> Q )
mp :      |- Q
a1 :      |- ( t = r -> ( t = s -> r = s ) )
```

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 - Mon 10, Feb 2020

Scientific Programming - Data Science @ University of Trento

Download exercises and solution

Introduction

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

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[106]
- 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[107]® 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[108]. Princeton University. 2010

In Python there are specialized libraries to read WordNet like NLTK[109], 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)

[106] http://examina.icts.unitn.it/studente
[107] https://wordnet.princeton.edu/
[108] https://wordnet.princeton.edu/
[109] 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[110], 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)

[110] 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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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.
    """


    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 =  {
```

(continues on next page)

(continued from previous page)

```
                        'pointer_symbol': params[i ],     # '~'
                        'synset_offset': int(params[i + 1]),  # '00001930'
                        'pos': params[i + 2],            # 'n'
                        'source_target':params[i + 3],   # '0000'
                    }
                d['ptrs'].append(ptr)
                i += 4


            d['gloss'] = line.split('|')[1]

            ret[d['synset_offset']] = d
            i += 1
            line=f.readline()
        return ret
```

[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.
    """
    raise Exception('TODO IMPLEMENT ME !')
```

[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': '~',
```

(continues on next page)

(continued from previous page)

```
                    '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 '
                   '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',
```

(continues on next page)

(continued from previous page)

```
                    '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,
          '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',
```

(continues on next page)

(continued from previous page)

```
                     '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},
                    {'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 '
```

(continues on next page)

(continued from previous page)

```
                 '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',
                  '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',
```

(continues on next page)

(continued from previous page)

```
                    '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,
          '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',
```

(continues on next page)

(continued from previous page)

```
                    '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'},
                    {'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',
```

(continues on next page)

(continued from previous page)

```
            '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',
                      '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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
def to_adj(db):

    ret = {}

    for d in db.values():
        targets = []
        for ptr in d['ptrs']:
            if ptr['pointer_symbol'] == '@':
```

(continues on next page)

(continued from previous page)

```
                if ptr['synset_offset'] in db:
                    targets.append(db[ptr['synset_offset']]['words'][0]['word'])
                #else:
                #    targets.append(ptr['synset_offset'])
        ret[d['words'][0]['word']] = targets
    return ret


dogs_graph = to_adj(dogs_db)
from pprint import pprint
pprint(dogs_graph)
```

```
{'bulldog': ['working_dog'],
 'canine': [],
 'dalmatian': ['dog'],
 'dog': ['canine', 'domestic_animal'],
 'domestic_animal': [],
 'shepherd_dog': ['working_dog'],
 'wolf': ['canine'],
 'working_dog': ['dog']}
```

[4]:
```
def to_adj(db):
    raise Exception('TODO IMPLEMENT ME !')

dogs_graph = to_adj(dogs_db)
from pprint import pprint
pprint(dogs_graph)
```

```
{'bulldog': ['working_dog'],
 'canine': [],
 'dalmatian': ['dog'],
 'dog': ['canine', 'domestic_animal'],
 'domestic_animal': [],
 'shepherd_dog': ['working_dog'],
 'wolf': ['canine'],
 'working_dog': ['dog']}
```

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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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',
```

(continues on next page)

(continued from previous page)

```
    '%p':'Part meronym',
    '=':'Attribute',
    '+':'Derivationally related form',
    ';c':'Domain of synset - TOPIC',            # DISCARD
    '-c':'Member of this domain - TOPIC',       # DISCARD
    ';r':'Domain of synset - REGION',           # DISCARD
    '-r':'Member of this domain - REGION',      # DISCARD
    ';u':'Domain of synset - USAGE',            # DISCARD
    '-u':'Member of this domain - USAGE',       # DISCARD
    '\\': 'Pertainym (pertains to noun)'        # DISCARD
}

def draw_hist(db):

    hist = {}
    for d in db.values():
        for ptr in d['ptrs']:
            ps = ptr['pointer_symbol']
            if 'domain' not in relation_names[ps].lower() and ps != '\\':
                if ps in hist:
                    hist[ps] += 1
                else:
                    hist[ps] = 0

    pprint(hist)

    import numpy as np
    import matplotlib.pyplot as plt

    xs = list(range(len(hist.keys())))
    coords = [(x,hist[x]) for x in hist.keys()]
    coords.sort(key=lambda c: c[1], reverse=True)
    ys = [c[1] for c in coords]

    fig = plt.figure(figsize=(18,6))

    plt.bar(xs, ys,
            0.5,             # the width of the bars
            color='green',   # someone suggested the default blue color is depressing,
↪ so let's put green
            align='center')  # bars are centered on the xtick

    plt.title('Wordnet Relation frequency SOLUTION')
    xticks = [relation_names[c[0]].replace(' ', '\n') for c in coords]
    plt.xticks(xs,xticks)

    plt.show()
```

[6]:

```
relation_names = {
    '!':'Antonym',
    '@':'Hypernym',
    '@i':'Instance Hypernym',
    '~':'Hyponym',
```

(continues on next page)

(continued from previous page)

```
    '~i':'Instance Hyponym',
    '#m':'Member holonym',
    '#s':'Substance holonym',
    '#p':'Part holonym',
    '%m':'Member meronym',
    '%s':'Substance meronym',
    '%p':'Part meronym',
    '=':'Attribute',
    '+':'Derivationally related form',
    ';c':'Domain of synset - TOPIC',           # DISCARD
    '-c':'Member of this domain - TOPIC',      # DISCARD
    ';r':'Domain of synset - REGION',          # DISCARD
    '-r':'Member of this domain - REGION',     # DISCARD
    ';u':'Domain of synset - USAGE',           # DISCARD
    '-u':'Member of this domain - USAGE',      # DISCARD
    '\\': 'Pertainym (pertains to noun)'       # DISCARD
}

def draw_hist(db):
    raise Exception('TODO IMPLEMENT ME !')
```

[ ]:

```

```

[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[111], `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
```

[111] https://sciprog.davidleoni.it/queues/queues.html#3.-ItalianQueue

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
         ^    ^              ^              ^
         |    |              |              |
         | _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: {'y': Node(c,y),
               'x': Node(a,x),}
```

[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: {'y': Node(d,y),
               'x': Node(a,x),}
```

[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: {'y': Node(d,y),
               'x': Node(b,x),}
```

[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: {'z': Node(f,z),
               'y': Node(d,y),
               'x': Node(b,x),}
```

[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: {'z': Node(f,z),
               'y': Node(e,y),
               'x': Node(b,x),}
```

[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
```

(continues on next page)

(continued from previous page)

```
       _head: Node(a,x)
       _tail: Node(g,z)
      _tails: {'z': Node(g,z),
               'y': Node(e,y),
               'x': Node(b,x),}
```

[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: {'z': Node(h,z),
               'y': Node(e,y),
               'x': Node(b,x),}
```

[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: {'z': Node(h,z),
               'y': Node(e,y),
               'x': Node(b,x),}
```

[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: {'z': Node(h,z),
               'y': Node(i,y),
               'x': Node(b,x),}
```

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)
```

(continues on next page)

(continued from previous page)

```
       _tail: Node(h,z)
      _tails: {'z': Node(h,z),
               'y': Node(i,y),
               'x': Node(b,x),}
```

[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: {'z': Node(h,z),
               'y': Node(i,y),}
```

[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: {'z': Node(h,z),
               'y': Node(i,y),}
```

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 - Tue 16, Jun 2020

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-2020-06-16-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[112]
- 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[113]. 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:

- `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`

[112] http://examina.icts.unitn.it/studente
[113] https://zoom.us/

- `(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[114] as seen in class (there is no need to use pandas)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[2]:
```
import csv

def load(filepath):

    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:
                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


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'],
```

(continues on next page)

[114] https://sciprog.davidleoni.it/formats/formats-sol.html#2.-File-CSV

(continued from previous page)

```
 ['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']]
```

[2]:
```
import csv

def load(filepath):
    raise Exception('TODO IMPLEMENT ME !')

meeting_log = load('UserQos_12345678901.csv')

from pprint import pprint
pprint(meeting_log, width=150)
```

```
[['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'],
```

(continues on next page)

(continued from previous page)

```
 ['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'],
 ['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**

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
def duration(a, b):

    asp = a.split(':')
    ta = int(asp[0])*60+int(asp[1])
    bsp = b.split(':')
    tb = int(bsp[0])*60 + int(bsp[1])
    return tb - ta


assert duration('15:00','15:34') == 34
assert duration('15:00','17:34') == 120 + 34
assert duration('15:50','16:12') == 22
assert duration('09:55','11:06') == 5 + 60 + 6
assert duration('00:00','00:01') == 1
#assert duration('11:58','00:01') == 3  # no need to support this case !!
```

[4]:
```
def duration(a, b):
    raise Exception('TODO IMPLEMENT ME !')

assert duration('15:00','15:34') == 34
assert duration('15:00','17:34') == 120 + 34
assert duration('15:50','16:12') == 22
assert duration('09:55','11:06') == 5 + 60 + 6
assert duration('00:00','00:01') == 1
#assert duration('11:58','00:01') == 3  # no need to support this case !!
```

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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[5]:
```
def calc_stats(log):

    ret = {}

    last_sessions = {}

    first = True
    for session in log:
        if first:
            first = False
            continue
        date = session[2]
        participant = session[3]
        join_time = session[4]
        leave_time = session[5]
        reason = session[6]

        if participant not in ret:
            ret[participant] = {'max_gap': 0,
                                'gaps': 0,
                                'time_away':0
                               }

        if participant in last_sessions:
            last_leave_time = last_sessions[participant][5]
            gap = duration(last_leave_time, join_time)
            ret[participant]['max_gap'] = max(gap, ret[participant]['max_gap'])
            ret[participant]['gaps'] += 1
            ret[participant]['time_away'] += gap

        last_sessions[participant] = session
    return ret



stats = calc_stats(meeting_log)

# in case you had trouble implementing load function, use this:
#stats = calc_stats(EXPECTED_MEETING_LOG)

stats
```

[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}}
```

[5]:
```
def calc_stats(log):
    raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
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[115]

- participant names MUST be sorted in alphabetical order
- remember to put title, legend and axis labels

To test the function, you DON'T NEED to have correctly implemented previous functions

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[7]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def viz(stats):


    xs = np.arange(len(stats))
    ys_max_gap = []
    ys_time_away = []

    labels = list(sorted(stats.keys()))

    for participant in sorted(stats):
        pstats = stats[participant]
        ys_max_gap.append(pstats['max_gap'])
```

(continues on next page)

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

(continued from previous page)

```
        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()


viz(stats)

# in case you had trouble implementing calc_stats, use this:
#viz(EXPECTED_STATS)
```

[7]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def viz(stats):
    raise Exception('TODO IMPLEMENT ME !')

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)

---------------------------------------------------------------------------
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)
```

(continues on next page)

(continued from previous page)

```
     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
```

(continues on next page)

(continued from previous page)

```
  ├
  └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 - Fri 17, Jul 2020

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-2020-07-17-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[116]
- 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[117], which is then further specialized by various states (for example Italy's catalog is called ATECO[118])

Sections

A NACE code is subdivided in a hierarchical, four-level structure. The categories at the highest level are called *sections*, here they are:

[116] http://examina.icts.unitn.it/studente
[117] https://ec.europa.eu/eurostat/ramon/nomenclatures/index.cfm?TargetUrl=LST_NOM_DTL&StrNom=NACE_REV2&StrLanguageCode=EN&IntPcKey=&StrLayoutCode=HIERARCHIC
[118] https://www.istat.it/it/archivio/17888

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 0x7f114fce1240>
```

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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
def is_nace(word):
    """Given a word, RETURN True if the word is a NACE code, else otherwise"""

    # we could implement it also with regexes, here we use explicit methods:
    if len(word) == 1:
        return word.isalpha() and word.isupper()
    elif len(word) == 2:
        return word.isdigit()
    elif len(word) == 4:
        return word[:2].isdigit() and word[2] == '.' and word[3].isdigit()
    elif len(word) == 5:
        return word[:2].isdigit() and word[2] == '.' and word[3:].isdigit()
    else:
        return False


assert is_nace('0') == False
assert is_nace('01') == True
assert is_nace('A') == True   # this is a Section
assert is_nace('AA') == False
assert is_nace('a') == False
assert is_nace('01.2') == True
assert is_nace('01.20') == True
assert is_nace('03.25') == True
assert is_nace('02.753') == False
assert is_nace('300') == False
assert is_nace('5012') == False
```

[4]:
```
def is_nace(word):
    """Given a word, RETURN True if the word is a NACE code, else otherwise"""
    raise Exception('TODO IMPLEMENT ME !')

assert is_nace('0') == False
assert is_nace('01') == True
assert is_nace('A') == True   # this is a Section
assert is_nace('AA') == False
assert is_nace('a') == False
assert is_nace('01.2') == True
assert is_nace('01.20') == True
assert is_nace('03.25') == True
```

(continues on next page)

(continued from previous page)

```
assert is_nace('02.753') == False
assert is_nace('300') == False
assert is_nace('5012') == False
```

A1.2 extract_codes

Implement following function which extracts codes from `This item excludes` column cells. For examples, see asserts.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[5]:
```
def extract_codes(text):
    """Extracts all the NACE codes from given text (a single string),
       and RETURN a list of the codes

       - also extracts section letters
       - list must have *no* duplicates
    """

    ret = []

    words = [word.strip(';,.:()"\'') for word in text.replace('-',' ').split()]
    for i in range(len(words)):

        if i < len(words) - 1 \
            and words[i].lower() == 'section' \
            and len(words[i+1]) == 1 \
            and words[i+1][0].isalpha():

            if words[i+1] not in ret:
                ret.append(words[i+1])
        else:
            if is_nace(words[i]) and words[i] not in ret:
                ret.append(words[i])

    return ret


assert extract_codes('group 02.4') == ['02.4']
assert extract_codes('class 02.40') == ['02.40']
assert extract_codes('.') == []
assert extract_codes('exceeding 300 litres') == []
assert extract_codes('see 46.34') == ['46.34']
assert extract_codes('divisions 10 and 11') == ['10','11']
assert extract_codes('(10.20)') == ['10.20']
assert extract_codes('(30.1, 33.15)') == ['30.1', '33.15']
assert extract_codes('as outlined in groups 85.1-85.4, i.e.') == ['85.1','85.4']
assert extract_codes('see 25.99 see 25.99') == ['25.99']  # no duplicates
assert extract_codes('section A') == ['A']
assert extract_codes('in section G. Also') == ['G']
assert extract_codes('section F (Construction)') == ['F']
assert extract_codes('section A, section A') == ['A']
```

[5]:
```
def extract_codes(text):
    """Extracts all the NACE codes from given text (a single string),
       and RETURN a list of the codes

       - also extracts section letters
       - list must have *no* duplicates
    """
    raise Exception('TODO IMPLEMENT ME !')

assert extract_codes('group 02.4') == ['02.4']
assert extract_codes('class 02.40') == ['02.40']
assert extract_codes('.') == []
assert extract_codes('exceeding 300 litres') == []
assert extract_codes('see 46.34') == ['46.34']
assert extract_codes('divisions 10 and 11') == ['10','11']
assert extract_codes('(10.20)') == ['10.20']
assert extract_codes('(30.1, 33.15)') == ['30.1', '33.15']
assert extract_codes('as outlined in groups 85.1-85.4, i.e.') == ['85.1','85.4']
assert extract_codes('see 25.99 see 25.99') == ['25.99']  # no duplicates
assert extract_codes('section A') == ['A']
assert extract_codes('in section G. Also') == ['G']
assert extract_codes('section F (Construction)') == ['F']
assert extract_codes('section A, section A') == ['A']
```

[6]:
```
# MORE REALISTIC asserts:

t01 = """Agricultural activities exclude any subsequent processing of the
agricultural products (classified under divisions 10 and 11 (Manufacture of food
products and beverages) and division 12 (Manufacture of tobacco products)), beyond
that needed to prepare them for the primary markets. The preparation of products for
the primary markets is included here.

The division excludes field construction (e.g. agricultural land terracing,
drainage, preparing rice paddies etc.) classified in section F (Construction) and␣
↪buyers
and cooperative associations engaged in the marketing of farm products classified
in section G. Also excluded is the landscape care and maintenance,
which is classified in class 81.30.
"""
assert extract_codes(t01) == ['10','11','12','F','G','81.30']

t01_15 = """This class excludes:
- manufacture of tobacco products, see 12.00
"""
assert extract_codes(t01_15) == ['12.00']

t03 = """This division does not include building and repairing of ships and
boats (30.1, 33.15) and sport or recreational fishing activities (93.19).
Processing of fish, crustaceans or molluscs is excluded, whether at land-based
plants or on factory ships (10.20).
"""

assert extract_codes(t03) == ['30.1', '33.15','93.19','10.20']

t11_03 = """This class excludes:
- merely bottling and labelling, see 46.34 (if performed as part of wholesale)
and 82.92 (if performed on a fee or contract basis)
```

(continues on next page)

(continued from previous page)

```
"""
assert extract_codes(t11_03) == ['46.34', '82.92']


t01_64 = """This class excludes:
- growing of seeds, see groups 01.1 and 01.2
- processing of seeds to obtain oil, see 10.41
- research to develop or modify new forms of seeds, see 72.11
"""
assert extract_codes(t01_64) == ['01.1','01.2','10.41','72.11']

t02 = """Excluded is further processing of wood beginning with sawmilling and planing␣
↪of wood,
see division 16.
"""
assert extract_codes(t02) == ['16']

t09_90 = """This class excludes:
- operating mines or quarries on a contract or fee basis, see division 05, 07 or 08
- specialised repair of mining machinery, see 33.12
- geophysical surveying services, on a contract or fee basis, see 71.12
"""
assert extract_codes(t09_90) == ['05','07','08','33.12','71.12']
```

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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[7]:
```
def build_db(filepath):

    ret = {}
    import csv
    with open(filepath, encoding='utf-8', newline='') as f:
        my_reader = csv.DictReader(f, delimiter=',')
        for d in my_reader:
```

(continues on next page)

(continued from previous page)

```
            diz = {'description' : d['Description'],
                   'exclusions' : extract_codes(d['This item excludes'])}
            ret[d['Code']] = diz
    return ret


activities_db = build_db('NACE_REV2_20200628_213139.csv')
#activities_db
```

[7]:
```
def build_db(filepath):
    raise Exception('TODO IMPLEMENT ME !')

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`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[8]:
```
%matplotlib inline
def plot(db, level):

    import matplotlib.pyplot as plt


    coords = [(code, len(db[code]['exclusions'])) for code in db if len(code.replace(
↪'.','')) == level]
    coords.sort(key=lambda c: c[1], reverse=True)

    coords = coords[:10]

    xs = [c[0] for c in coords]
    ys = [c[1] for c in coords]

    fig = plt.figure(figsize=(13,6))  # width: 10 inches, height 3 inches


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

(continues on next page)

(continued from previous page)

```
    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])

    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()



#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)
```

[8]:
```
%matplotlib inline
```

(continues on next page)

(continued from previous page)

```
def plot(db, level):

    import matplotlib.pyplot as plt
    raise Exception('TODO IMPLEMENT ME !')

#Uncomment *only* if you had problems with build_db
#from expected_db import activities_db

#1 Section
#2 Division
#3 Group
#4 Class
plot(activities_db, 2)
```

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')
```

(continues on next page)

(continued from previous page)

```
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


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
```

(continues on next page)

(continued from previous page)

```
        - 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 - Mon 24, Aug 2020

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[119]
- If you don't have unitn login: tell instructors and we will download your work manually

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

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[120]

> **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
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
```

(continues on next page)

[120] https://dati.trentino.it/dataset/prezziario-dei-lavori-pubblici-della-provincia-autonoma-di-trento

(continued from previous page)

```
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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
import math


#use this list to rmeove unneeded stuff
PUNCTUATION=[',','-','.','%']
UNTIL = 'fino'
BEYOND = 'oltre'

def extract_bounds(text):


    fixed_text = text
    for pun in PUNCTUATION:
        fixed_text = fixed_text.replace(pun, ' ')
    words = fixed_text.split()
    i = 0
    left = None
    right = None

    while i < len(words) and (not left or not right):

        if words[i].isdigit():
            if not left:
                left = int(words[i])
            elif not right:
                right = int(words[i])
        i += 1

    if not right:
        if BEYOND in text:
            right = math.inf
        else:
            right = left
            left = 0

    return (left,right)


assert extract_bounds('fino a mm 50.') == (0,50)
assert extract_bounds('oltre mm 50 fino a mm 100.') == (50,100)
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)
```

(continues on next page)

(continued from previous page)

```
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)
```

[4]:
```
import math


#use this list to rmeove unneeded stuff
PUNCTUATION=[',','-','.','%']
UNTIL = 'fino'
BEYOND = 'oltre'

def extract_bounds(text):
    raise Exception('TODO IMPLEMENT ME !')

assert extract_bounds('fino a mm 50.') == (0,50)
assert extract_bounds('oltre mm 50 fino a mm 100.') == (50,100)
assert extract_bounds('oltre mm 200.') == (200, math.inf)
assert extract_bounds('da diametro 63 mm a diametro 127 mm') == (63, 127)
assert extract_bounds('fino a 10 KW') ==  (0,10)
assert extract_bounds('oltre 156 fino a 184 KW') ==  (156,184)
assert extract_bounds('fino a 170 A, avviamento elettrico') == (0,170)
assert extract_bounds('oltre 170 A fino a 250 A, avviamento elettrico') == (170, 250)
assert extract_bounds('oltre 300 A, avviamento elettrico')  == (300, math.inf)
assert extract_bounds('tetti piani o con bassa pendenza - fino al 10%') == (0,10)
assert extract_bounds('tetti a media pendenza - oltre al 10% e fino al 45%') == (10,
↪45)
assert extract_bounds('tetti ad alta pendenza - oltre al 45%') == (45, math.inf)
```

A2 extract_product

Write a function that given a `filename`, a `code` and a `unit`, parses the csv until it finds the corresponding code and RETURNS **one** dictionary with relevant information for that product

- *Prezzo* (*price*) **must be converted to float**
- **implement the parsing with a** `csv.DictReader`, see example[121]
- **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
```

(continues on next page)

[121] https://sciprog.davidleoni.it/formats/formats-sol.html#Reading-as-dictionaries

(continued from previous page)

```
8   A.02.40.0010.050  oltre mm 200.                Noli e trasporti  6.84
9   A.02.40.0020      GRUPPO ELETTROGENO           NaN               NaN
10  A.02.40.0020.010  fino a 10 KW                 Noli e trasporti  8.77
11  A.02.40.0020.020  oltre 10 fino a 13 KW        Noli e trasporti  9.94
```

A call to

```
pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm'))
```

Must produce:

```
{'category': 'Noli e trasporti',
 'code': 'A.02.40.0010',
 'description': 'POMPA COMPLETA DI MOTORE',
 'measure_unit': 'mm',
 'models': [{'bounds': (0, 50),        'price': 2.21, 'subcode': '010'},
            {'bounds': (50, 100),      'price': 3.36, 'subcode': '020'},
            {'bounds': (100, 150),     'price': 4.42, 'subcode': '030'},
            {'bounds': (150, 200),     'price': 5.63, 'subcode': '040'},
            {'bounds': (200, math.inf),'price': 6.84, 'subcode': '050'}]}
```

Notice that if we append `subcode` to `code` (with a dot) we obtain the full product code.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[6]:
```
import csv
from pprint import pprint

def extract_product(filename, code, measure_unit):


    c = 0
    with open(filename, encoding='latin-1', newline='') as f:
        my_reader = csv.DictReader(f, delimiter=',')    # Notice we now used DictReader
        for d in my_reader:

            if d['Codice Prodotto'] == code:
                ret = {}
                ret['description'] = d['Descrizione Breve Prodotto']
                ret['code'] = code
                ret['measure_unit'] = measure_unit
                ret['models'] = []

            if d['Codice Prodotto'].startswith(code + '.'):
                ret['category'] = d['Categoria']
                subdiz = {}
                subdiz['price'] = float(d['Prezzo'])
                subdiz['subcode'] = d['Codice Prodotto'][len(code)+1:]
                subdiz['bounds'] = extract_bounds(d['Descrizione Breve Prodotto'])
                ret['models'].append(subdiz)

    return ret


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') == \
```

(continues on next page)

(continued from previous page)

```
    {'category': 'Noli e trasporti',
     'code': 'A.02.40.0010',
     'description': 'POMPA COMPLETA DI MOTORE',
     'measure_unit': 'mm',
     'models': [{'bounds': (0, 50),        'price': 2.21, 'subcode': '010'},
                {'bounds': (50, 100),      'price': 3.36, 'subcode': '020'},
                {'bounds': (100, 150),     'price': 4.42, 'subcode': '030'},
                {'bounds': (150, 200),     'price': 5.63, 'subcode': '040'},
                {'bounds': (200, math.inf),'price': 6.84, 'subcode': '050'}]}

#pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0020', 'KW'))
#pprint(extract_product('EPPAT-2018-new-compact.csv', 'B.02.10.0042', 'mm'))
#pprint(extract_product('EPPAT-2018-new-compact.csv','B.30.10.0010', '%'))
```

```
{'category': 'Noli e trasporti',
 'code': 'A.02.40.0010',
 'description': 'POMPA COMPLETA DI MOTORE',
 'measure_unit': 'mm',
 'models': [{'bounds': (0, 50), 'price': 2.21, 'subcode': '010'},
            {'bounds': (50, 100), 'price': 3.36, 'subcode': '020'},
            {'bounds': (100, 150), 'price': 4.42, 'subcode': '030'},
            {'bounds': (150, 200), 'price': 5.63, 'subcode': '040'},
            {'bounds': (200, inf), 'price': 6.84, 'subcode': '050'}]}
```

[6]:
```
import csv
from pprint import pprint

def extract_product(filename, code, measure_unit):
    raise Exception('TODO IMPLEMENT ME !')

pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm'))
assert extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm') == \
    {'category': 'Noli e trasporti',
     'code': 'A.02.40.0010',
     'description': 'POMPA COMPLETA DI MOTORE',
     'measure_unit': 'mm',
     'models': [{'bounds': (0, 50),        'price': 2.21, 'subcode': '010'},
                {'bounds': (50, 100),      'price': 3.36, 'subcode': '020'},
                {'bounds': (100, 150),     'price': 4.42, 'subcode': '030'},
                {'bounds': (150, 200),     'price': 5.63, 'subcode': '040'},
                {'bounds': (200, math.inf),'price': 6.84, 'subcode': '050'}]}

#pprint(extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0020', 'KW'))
#pprint(extract_product('EPPAT-2018-new-compact.csv', 'B.02.10.0042', 'mm'))
#pprint(extract_product('EPPAT-2018-new-compact.csv','B.30.10.0010', '%'))
```

```
{'category': 'Noli e trasporti',
 'code': 'A.02.40.0010',
 'description': 'POMPA COMPLETA DI MOTORE',
 'measure_unit': 'mm',
 'models': [{'bounds': (0, 50), 'price': 2.21, 'subcode': '010'},
            {'bounds': (50, 100), 'price': 3.36, 'subcode': '020'},
            {'bounds': (100, 150), 'price': 4.42, 'subcode': '030'},
            {'bounds': (150, 200), 'price': 5.63, 'subcode': '040'},
            {'bounds': (200, inf), 'price': 6.84, 'subcode': '050'}]}
```

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**.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[7]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def plot_product(product):


    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()



product = extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm')
#product = extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0020', 'KW')
#product = extract_product('EPPAT-2018-new-compact.csv', 'B.02.10.0042', 'mm')
#product = extract_product('EPPAT-2018-new-compact.csv','B.30.10.0010', '%')

plot_product(product)
```

[7]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

def plot_product(product):
    raise Exception('TODO IMPLEMENT ME !')


product = extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0010', 'mm')
#product = extract_product('EPPAT-2018-new-compact.csv', 'A.02.40.0020', 'KW')
#product = extract_product('EPPAT-2018-new-compact.csv', 'B.02.10.0042', 'mm')
#product = extract_product('EPPAT-2018-new-compact.csv','B.30.10.0010', '%')

plot_product(product)
```

POMPA COMPLETA DI MOTORE (A.02.40.0010) SOLUTION
Price (€)
0
1
2
3
4
5
6
7
0 - 50
50 - 100
100 - 150
150 - 200
>200
mm

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
```

[14]:
```
tasks = bt('i',
                bt('d',
                        bt('b',
                                bt('a')),
                        bt('c')),
                bt('h',
                        bt('f',
                                None,
                                bt('e')),
                        bt('g')))
```

[15]:
```
print(tasks)
```

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

[16]:
```
tasks.schedule_rec()
```

[16]:
```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

2.1.17 Midterm Sim - Mon 02, Nov 2020

Scientific Programming - Data Science @ University of Trento

Download exercises and solutions

This simulation gives you NO credit whatsoever: If you do everything wrong, you lose nothing. If you do everything correct, you gain nothing

What to do

1) Download `sciprog-ds-2020-11-02-exam.zip` and extract it on your desktop.
2) Rename `sciprog-ds-2020-11-02-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2020-11-02-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. 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

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

Part A - Galactic Love

Open Jupyter and start editing this notebook `exam-2020-11-02.ipynb`

Since today is a pseudo-exam, you are going to do pseudo-science!

The company Astro Logic provides horoscopes to thousands of loyal customers, who each day require a number of divinitions. The most requested is whether or not they should engage in love affairs with a potential partner, who is chosen according to rigourous criteria like his/her astrological sign. You are then hired to devise a fancy visualization which given two astrological signs and their love compatibility, displays the constellations of their signs close when the the compatibility is high and far away when compatibility is low.

parse_stars

Let's start with real astronomical data. You are given a database of constellations called `stars.csv` (we slightly tweaked it for this occasion - original data source: Space Telescope Science Institute[123])

[1]:
```
import pandas as pd

stars_df = pd.read_csv('stars.csv', encoding='UTF-8')
stars_df[0:32]
```

[1]:

	constellation	type	ra	dec	description
0	Andromeda	0	3717	2539	move gamma 1
1	Andromeda	1	2091	2137	draw beta
2	Andromeda	1	1179	1851	draw delta
3	Andromeda	1	251	1745	draw alpha
4	Andromeda	0	1716	1405	move eta
5	Andromeda	1	1420	1456	draw zeta
6	Andromeda	1	1156	1758	draw epsilon
7	Andromeda	1	1179	1851	draw delta
8	Andromeda	1	1106	2023	draw pi
9	Andromeda	1	512	2320	draw theta
10	Andromeda	1	42544	2596	draw iota
11	Andromeda	1	42612	2660	draw kappa
12	Andromeda	1	42526	2787	draw lambda
13	Andromeda	0	42544	2596	move iota
14	Andromeda	1	41457	2539	draw omicron
15	Andromeda	0	1106	2023	move pi
16	Andromeda	1	2091	2137	draw beta
17	Andromeda	1	1702	2309	draw mu
18	Andromeda	1	1494	2464	draw nu
19	Andromeda	1	2085	2834	draw phi
20	Andromeda	1	2939	2917	draw 51
21	Andromeda	-1	0	0	NaN
22	Antlia	0	17077	-2157	move epsilon
23	Antlia	2	18814	-1864	dotted alpha
24	Antlia	2	19701	-2228	dotted iota
25	Antlia	-1	0	0	NaN
26	Apus	0	26635	-4742	move alpha
27	Apus	2	29803	-4733	dotted gamma
28	Apus	2	30092	-4651	dotted beta
29	Apus	2	29410	-4721	dotted delta 1
30	Apus	2	29803	-4733	dotted gamma
31	Apus	-1	0	0	NaN

[123] https://github.com/mperrin/misc_astro

You will have to parse it so to obtain a dictionary which maps each constellation to its stars, expressed as a list of lists of points type and coordinates.

Since later we will need to show points in a 2d chart, you will have to transform the coordinates obtained from the data (right ascension and declination in degrees) as follows:

$x = \frac{15}{1800} ra$

$y = \frac{dec}{60}$

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[2]:
```
import csv

def parse_stars(filename):


    ret = {}
    with open(filename, encoding='utf-8', newline='') as f:
        my_reader = csv.reader(f, delimiter=',')
        next(my_reader) # skips header
        constellation = ''
        d = None
        for row in my_reader:
            if row[0] != constellation:
                constellation = row[0]
                stars = []
                ret[constellation] = stars
            coords = [0]* 3
            coords[0] = int(row[1])
            coords[1] = int(row[2]) * (1.0 / 1800) * 15
            coords[2] = int(row[3]) * (1.0 / 60)
            stars.append(coords)

    return ret


stars_db = parse_stars('stars.csv')
```

[2]:
```
import csv

def parse_stars(filename):
    raise Exception('TODO IMPLEMENT ME !')

stars_db = parse_stars('stars.csv')
```

You can find the complete output in `expected_stars_db.py`

Excerpt:

```
{'Andromeda': [
                  [0, 30.974999999999998, 42.31666666666666],
                  [1, 17.425, 35.61666666666667],
                  [1, 9.825000000000001, 30.849999999999998],
```

(continues on next page)

(continued from previous page)

```
                    [1, 2.091666666666667, 29.083333333333332],
                    [0, 14.3, 23.416666666666668],
                    [1, 11.833333333333332, 24.266666666666666],
                    [1, 9.633333333333333, 29.3],
                    [1, 9.825000000000001, 30.849999999999998],
                    [1, 9.216666666666667, 33.71666666666667],
                    [1, 4.266666666666667, 38.666666666666664],
                    [1, 354.5333333333333, 43.266666666666666],
                    [1, 355.09999999999997, 44.333333333333336],
                    [1, 354.3833333333333, 46.45],
                    [0, 354.5333333333333, 43.266666666666666],
                    [1, 345.475, 42.31666666666666],
                    [0, 9.216666666666667, 33.71666666666667],
                    [1, 17.425, 35.61666666666667],
                    [1, 14.183333333333334, 38.483333333333334],
                    [1, 12.45, 41.06666666666666],
                    [1, 17.375, 47.233333333333334],
                    [1, 24.491666666666667, 48.61666666666667],
                    [-1, 0.0, 0.0]
                ],
 'Antlia':    [
                    [0, 142.30833333333334, -35.95],
                    [2, 156.78333333333333, -31.066666666666666],
                    [2, 164.175, -37.13333333333333],
                    [-1, 0.0, 0.0]
             ],
  .
  .
  .
}
```

plot_stars 1

Write a function `plot_stars` to plot constellations

> **WARNING: DO NOT use GraphViz!**
>
> Even if we are making plots which look like networks, for these visualizations you just need basic matplotlib (and some creativity ;-)

> **WARNING: for now, ignore the `new_center` parameter**

A point type can either be:

- `0`: start a new line not connected with the previous one
- `1`: connect previous point with a straight segment
- `2`: connect previous point with a dotted segment (draw it with `linestyle=':'` parameter)
- `-1`: last point, ignore

Available colorschemes are `'M'`, `'F'`, or `'R'` (red)

- to set a black background, set `plt.rcParams['axes.facecolor'] = 'black'`
- to get a nice glowing effect for the lines, draw twice: once with a thick line and dark color, and once with a thin line with a bright color. You can find the colors in `color_schemes`. To set them in `plt.plot` call, use `linewidth` (sets width in pixels) and `color` parameter, note `color` takes a **single** parameter
- draw stars as white dots, setting `markersize=6`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[3]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

color_schemes = {
    'M': ('blue', '#039dfc'),
    'F': ('purple', 'pink'),
    'R': ('darkred', 'red')
}

def plot_stars(constellation_name, color_scheme, stars, new_center=None):

    plt.rcParams['axes.facecolor'] = 'black'

    color1, color2 = color_schemes[color_scheme]

    point_list = stars[constellation_name]
    points = np.asarray(point_list)
    drawtype = points[:,0]
    ra_degrees = points[:-1,1]
    dec_degrees = points[:-1,2]

    if new_center:
        xbounds = (np.min(ra_degrees), np.max(ra_degrees))
        ybounds = (np.min(dec_degrees), np.max(dec_degrees))
        halfx = (xbounds[1]-xbounds[0])/2
        halfy = (ybounds[1]-ybounds[0])/2

        ra_degrees -= xbounds[0] + halfx - new_center[0]
        dec_degrees -= ybounds[1] - halfy - new_center[1]

    for i in range(0, len(drawtype)-1):
        if drawtype[i] == 0 or drawtype[i] == -1:
            continue

        xs = ra_degrees[i - 1:(i)+1]
        ys = dec_degrees[i - 1:(i)+1]
        plt.plot(xs,ys, linewidth=8, linestyle=':' if drawtype[i] ==2 else "-",␣
↪color=color1)
        plt.plot(xs,ys, linewidth=3, linestyle=':' if drawtype[i] ==2 else "-",␣
↪color=color2)
        plt.plot(xs, ys, 'o', markersize=6, color='white')



from pprint import pprint
pprint(stars_db['Libra'])
```

(continues on next page)

(continued from previous page)

```
plot_stars('Libra', 'M', stars_db)
```

```
[[0, 226.01666666666665, -25.266666666666666],
 [1, 222.71666666666667, -16.03333333333333],
 [1, 229.25, -9.366666666666667],
 [1, 233.875, -14.783333333333333],
 [1, 222.71666666666667, -16.03333333333333],
 [0, 233.875, -14.783333333333333],
 [1, 234.25, -28.133333333333333],
 [1, 234.65833333333333, -29.766666666666666],
 [-1, 0.0, 0.0]]
```

[3]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

color_schemes = {
    'M': ('blue', '#039dfc'),
    'F': ('purple', 'pink'),
    'R': ('darkred', 'red')
}

def plot_stars(constellation_name, color_scheme, stars, new_center=None):
    raise Exception('TODO IMPLEMENT ME !')

from pprint import pprint
pprint(stars_db['Libra'])
plot_stars('Libra', 'M', stars_db)
```

```
[[0, 226.01666666666665, -25.266666666666666],
 [1, 222.71666666666667, -16.03333333333333],
 [1, 229.25, -9.366666666666667],
 [1, 233.875, -14.783333333333333],
 [1, 222.71666666666667, -16.03333333333333],
 [0, 233.875, -14.783333333333333],
 [1, 234.25, -28.133333333333333],
 [1, 234.65833333333333, -29.766666666666666],
```

(continues on next page)

(continued from previous page)

```
[-1, 0.0, 0.0]]
```

[4]:
```
stars_db['Cancer']    # has type-2  dotted points
```

[4]:
```
[[0, 131.66666666666669, 28.75],
 [2, 130.8166666666667, 21.466666666666665],
 [2, 131.16666666666666, 18.15],
 [2, 134.61666666666667, 11.85],
 [0, 131.16666666666666, 18.15],
 [2, 124.125, 9.183333333333334],
 [-1, 0.0, 0.0]]
```

[18]:
```
plot_stars("Cancer", 'F', stars_db)
```

[6]:
```
plot_stars("Cygnus", 'M', stars_db)  # mixed segment types
```

plot_stars 2 - new_center

Change the previous function `plot_stars` so it accepts a new argument `new_center`, which is either `None` or a tuple of coordinates where the constellation should be centered:

- be precise in determining the boundaries of the constellation
- **DO NOT** assume the constallation has a fixed width nor height (so no constants in code!)

[7]:
```
fig = plt.figure(figsize=(30,7))
plt.xlim(0,360)
plt.ylim(-40,40)
plot_stars('Gemini', 'F',stars_db, new_center=None)  # no translation
```

[8]:
```
fig = plt.figure(figsize=(30,7))
plt.xlim(0,360)
plt.ylim(-40,40)
plot_stars('Gemini', 'F',stars_db, new_center=(300, -20))  # centered in 300, -20
```

parse_zodiac

You are given a file `zodiac.csv`. For each sign, the table contains astrological information and affinity with other signs, expressed as a relation matrix:

[9]:
```
import pandas as pd
df = pd.read_csv('zodiac.csv', encoding='UTF-8')
df[:4]
```

[9]:
```
  Constellation  House Glyph Symbol                  Dates Element  \
0         Aries      1     ♈    Ram  21 March\n-\n20 April    Fire
1        Taurus      2     ♉   Bull    21 April\n-\n21 May   Earth
2        Gemini      3     ♊  Twins     22 May\n-\n21 June     Air
3        Cancer      4     ♋   Crab    22 June\n-\n21 July   Water

      Quality Ruling Planet Day/Night  Aries   …    Gemini  Cancer  Leo  \
0    Cardinal          Mars       Day    NaN   …       4.0     NaN  5.0
1       Fixed         Venus     Night    NaN   …       NaN     4.0  NaN
2     Mutable       Mercury       Day    4.0   …       NaN     NaN  4.0
3   Cardinal           Moon     Night    NaN   …       NaN     NaN  NaN

   Virgo  Libra  Scorpius  Sagittarius  Capricornus  Aquarius  Pisces
0    NaN    NaN       NaN          5.0          NaN       4.0     NaN
1    5.0    NaN       NaN          NaN          5.0       NaN     4.0
2    NaN    5.0       NaN          NaN          NaN       5.0     NaN
3    4.0    NaN       5.0          NaN          NaN       NaN     5.0

[4 rows x 21 columns]
```

Parse the table so to get a a dictionary of dictionaries, with some selected data:

- affinities are in the scale 1-5, normalize them to floats 0.0-1.0
- dates contain `\n` , normalize them so to have dates separated by a dash as in `21 March-20 April`

NOTE: To parse the file, a `csv.reader` is sufficient, it's not necessary to use pandas - even if data seem to span multiple lines because of the `\n` in dates, note they are bounded by `"` so rows will be correctly parsed by `csv.reader`

You can find the complete output in `expected_zodiac_db.py`

```
{
 'Aquarius': {
              'affinities': {
                              'Aries': 0.8,
```

(continues on next page)

(continued from previous page)

```
                                'Gemini': 1.0,
                                'Libra': 1.0,
                                'Sagittarius': 0.8
                              },
               'dates': '21 January-18 February',
               'glyph': '♒',
               'house': 11
              },
 'Aries':     {
               'affinities': {
                                'Aquarius': 0.8,
                                'Gemini': 0.8,
                                'Leo': 1.0,
                                'Sagittarius': 1.0
                              },
               'dates': '21 March-20 April',
               'glyph': '♈',
               'house': 1
              },
  .
  .
  .
}
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[10]:
```
import csv

def parse_zodiac(filename):

    with open(filename, encoding='utf-8', newline='') as f:
        my_reader = csv.reader(f, delimiter=',')
        header = next(my_reader)

        ret = {}
        for row in my_reader:
            workd = {}
            workd['glyph'] = row[2]
            workd['house'] = int(row[1])
            workd['dates'] = row[4].replace('\n','')
            ret[row[0]] = workd
            workd['affinities'] = {}
            for j in range(9,len(row)):
                if row[j] != '':
                    workd['affinities'][header[j]] = int(row[j])/5
        return ret


zodiac_db = parse_zodiac('zodiac.csv')

from pprint import pprint
#pprint(zodiac_db, width=100)
assert zodiac_db['Aries']['dates'] == '21 March-20 April'
assert zodiac_db['Aries']['affinities'] == {'Aquarius': 0.8, 'Gemini': 0.8, 'Leo': 1.
↪0, 'Sagittarius': 1.0}
```

(continues on next page)

(continued from previous page)

```
assert zodiac_db['Aries']['glyph'] == '♈'
assert zodiac_db['Aries']['house'] == 1
assert zodiac_db['Gemini']['dates'] == '22 May-21 June'
assert zodiac_db['Gemini']['affinities'] == {'Aquarius': 1.0, 'Aries': 0.8, 'Leo': 0.
↪8, 'Libra': 1.0}
assert zodiac_db['Gemini']['glyph'] == '♊'
assert zodiac_db['Gemini']['house'] == 3
```

[10]:
```
import csv

def parse_zodiac(filename):
    raise Exception('TODO IMPLEMENT ME !')

zodiac_db = parse_zodiac('zodiac.csv')

from pprint import pprint
#pprint(zodiac_db, width=100)
assert zodiac_db['Aries']['dates'] == '21 March-20 April'
assert zodiac_db['Aries']['affinities'] == {'Aquarius': 0.8, 'Gemini': 0.8, 'Leo': 1.
↪0, 'Sagittarius': 1.0}
assert zodiac_db['Aries']['glyph'] == '♈'
assert zodiac_db['Aries']['house'] == 1
assert zodiac_db['Gemini']['dates'] == '22 May-21 June'
assert zodiac_db['Gemini']['affinities'] == {'Aquarius': 1.0, 'Aries': 0.8, 'Leo': 0.
↪8, 'Libra': 1.0}
assert zodiac_db['Gemini']['glyph'] == '♊'
assert zodiac_db['Gemini']['house'] == 3
```

plot_love

In `stars.csv` we inserted the special (fake!) constellation of `'Love'`: given the importance, we placed it at the center of the galaxy, positioned at `x=180` degrees and `y=0`. If you try to plot it now, you should get something like this:

[11]:
```
# 'Aries',    'Taurus',    'Gemini',       'Cancer',       'Leo',       'Virgo',
# 'Libra',   'Scorpius',   'Sagittarius', 'Capricornus', 'Aquarius', 'Pisces'

fig = plt.figure(figsize=(30,7))
plt.xlim(0,360)
plot_stars('Gemini','F', stars_db)
plot_stars('Aquarius','M', stars_db)
plot_stars('Love','R', stars_db)     # fake!
```

Given two astrological signs, place them on the same y=0 axis as the heart and make them symmetrically closer or farther from it according to their astrological affinity, also displaying their name and astrological glyph:

- **REMEMBER** title and xlabels !
- you can reuse previosly defined `plot_stars` function
- constellations x centers should go from 50 to 150 degrees (and symmetrically, from -50 to -150)
- **BUT you will have to display reversed ticks**: 100 50 0 for positive (and symmetrically 0 50 100 for negative)

For drawing text:

- For increasing text size in `title`, `xticks`, `xlabel`, `text` calls, you can use `fontsize=20` parameter (for glyphs you will need a bigger number)
- for text inside the chart use use `plt.text(x,y,"some text")`
- the glyph must be drawn bigger than the sign name, so you will need a separate call to `plt.text`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[12]:
```
def plot_love(f_sign, m_sign, stars, zodiac):

    fig = plt.figure(figsize=(30,7))  # 30 inches large by 7 high
    plt.xlim(-175,175)



    if m_sign in zodiac[f_sign]['affinities']:
        coeff = zodiac[f_sign]['affinities'][m_sign]
    else:
        coeff = 0.0

    plt.title('GALACTIC LOVE', fontsize=22)
    xs = np.array([50,100,150])
    plt.xlabel('AFFINITY', fontsize=19)
    plt.xticks(np.hstack((-xs,xs)), np.hstack((150-np.abs(xs), 150-np.abs(xs))),␣
↪fontsize=18)
    plt.ylim(-19,19)
    plot_stars('Love', 'R',stars, new_center=(0,0))
    prox = (1.0 - coeff)*100+25+25
    plot_stars(m_sign, 'M',stars, new_center=(prox,0))
    plot_stars(f_sign, 'F', stars, new_center=(-prox,0))

    plt.text(+prox,-16, m_sign,  fontsize=19,  fontweight='bold', color='white')
    plt.text(+prox-11,-16.5, zodiac[m_sign]['glyph'], fontsize=45,  fontweight='bold',
↪ color='white')

    plt.text(-prox,-16, f_sign, fontsize=19,  fontweight='bold', color='white')
    plt.text(-prox-11,-16.5, zodiac[f_sign]['glyph'], fontsize=45,  fontweight='bold',
↪ color='white')


plot_love('Gemini','Aquarius', stars_db, zodiac_db)  # 1.0 affinity
```

[12]:
```
def plot_love(f_sign, m_sign, stars, zodiac):

    fig = plt.figure(figsize=(30,7))  # 30 inches large by 7 high
    plt.xlim(-175,175)

    raise Exception('TODO IMPLEMENT ME !')

plot_love('Gemini','Aquarius', stars_db, zodiac_db)  # 1.0 affinity
```

[13]:
```
plot_love('Gemini','Leo', stars_db, zodiac_db)  # 0.8 affinity
```

[14]:
```
plot_love('Gemini','Taurus', stars_db, zodiac_db) # 0.0 affinity
```

[15]:
```
plot_love('Taurus','Capricornus', stars_db, zodiac_db)  # 1.0 affinity
```

[16]:
```
plot_love('Leo','Libra', stars_db, zodiac_db)  # 0.8 affinity
```

[17]:
```
plot_love('Taurus','Scorpius', stars_db, zodiac_db)  # 0.0 affinity
```

[ ]:

2.1.18 Midterm A - Fri 06, Nov 2020

Scientific Programming - Data Science @ University of Trento

Download exercises and solutions

Taking part to this exam erases any vote you had before

1) Download `sciprog-ds-2020-11-06-exam.zip` and extract it on your desktop.
2) Rename `sciprog-ds-2020-11-06-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2020-11-06-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 zip and send to examina.icts.unitn.it/studente[124]

Music Sequencer

Open Jupyter and start editing this notebook `exam-2020-11-06.ipynb`

ABC[125] is a popular format to write music notation in plain text files, you can see an example by openining `tunes.abc` with a text editor. A music sequencer is an editor software which typically displays notes as a matrix: you will parse simplified abc tunes and display their melodies in a matrix.

1. parse_melody

Write a function which given a melody as a string of notes translates it to a list of tuples:

```
>>> parse_melody("|A4      C2  E2 |C4      E D C2 |C3    B3    G2 |")
[(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
```

Each melody note is followed by its duration. If no duration number is specified, we assume it is one.

Each tuple first element represents a note as a number from `0` (`A`) to `6` (`G`) and the second element is the note length in the sequencer. We assume our sequencer has a resolution of two beats per note, so for us a note `A` would have length `2`, a note `A2` a length `4`, a note `A3` a length `6` and so on.

- **DO NOT** care about spaces nor bars `|`, they have no meaning at all
- To get a character position, use `ord` python function

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[1]:
```
def parse_melody(melody):

    notes = melody.replace('|',' ').split()

    ret = []
    for note in notes:
        n = ord(note[0])-ord('A')
```

(continues on next page)

[124] http://examina.icts.unitn.it/studente
[125] http://abcnotation.com/wiki/abc:standard:v2.1#rrhythm

(continued from previous page)

```
        if len(note) == 1:
            ret.append((n,2))
        else:
            ret.append((n,2*int(note[1])))

    return ret


from pprint import pprint
melody1 = "|A4      C2  E2 |C4      E D C2 |C3    B3    G2 |"
pprint(parse_melody(melody1) )

assert parse_melody("||") == []
assert parse_melody("|A|") == [(0,2)]
assert parse_melody("|A3|") == [(0,6)]
assert parse_melody("|A B|") == [(0,2), (1,2)]
assert parse_melody("|C D|") == [(2,2), (3,2)]
assert parse_melody(" | G    F  |    ") == [(6,2), (5,2)]
assert parse_melody("|D|B|") == [(3,2), (1,2)]
assert parse_melody("|D3 E4|") == [(3,6),(4,8)]
assert parse_melody("|F|A2 B|") == [(5,2),(0,4),(1,2)]
assert parse_melody("|A4      C2  E2 |C4      E D C2 |C3    B3    G2 |") == \
       [(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6,␣
↪4)]
```

```
[(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
```

[1]:

```
def parse_melody(melody):
    raise Exception('TODO IMPLEMENT ME !')

from pprint import pprint
melody1 = "|A4      C2  E2 |C4      E D C2 |C3    B3    G2 |"
pprint(parse_melody(melody1) )

assert parse_melody("||") == []
assert parse_melody("|A|") == [(0,2)]
assert parse_melody("|A3|") == [(0,6)]
assert parse_melody("|A B|") == [(0,2), (1,2)]
assert parse_melody("|C D|") == [(2,2), (3,2)]
assert parse_melody(" | G    F  |    ") == [(6,2), (5,2)]
assert parse_melody("|D|B|") == [(3,2), (1,2)]
assert parse_melody("|D3 E4|") == [(3,6),(4,8)]
assert parse_melody("|F|A2 B|") == [(5,2),(0,4),(1,2)]
assert parse_melody("|A4      C2  E2 |C4      E D C2 |C3    B3    G2 |") == \
       [(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6,␣
↪4)]
```

```
[(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
```

2. parse_tunes

An `.abc` file is a series of key:value fields. Keys are always one character long. Anything after a `%` is a comment and must be ignored

File `tunes.abc` excerpt:

```
[2]: with open("tunes.abc", encoding='utf-8') as f: print(''.join(f.readlines()[0:17]))
```

```
%abc-2.1
H:Tune made in a dark algorithmic night    % history and origin in header, so␣
↪replicated in all tunes!
O:Trento

X:1                       % index
T:Algorave                % title
C:The Lord of the Loop    % composer
M:4/4                     % meter
K:C                       % key
|A4      C2  E2 |C4      E D C2 |C3    B3    G2 |   % melodies can also have a comment

X:2
T:Transpose Your Head
C:Matrix Queen
M:3/4
K:G
|F2  G4     |E4      E F|A2  B2  D2 |D3    E3   |C3    C3   |
```

First lines (3 in the example) are the file header, separated by tunes with a blank line.

- first line must always be ignored
- fields specified in the file header must be copied in *all* tunes

After the first blank line, there is the first tune:

- `X` is the tune index, convert it to integer
- `M` is the meter, convert it to a tuple of two integers
- `K` is the last field of metadata
- melody line has no field key, it always follows line with `K` and it immediately begins with a pipe: convert it to list by calling `parse_melody`

Following tunes are separated by blank lines

Write a function `parse_tunes` which parses the file and outputs a list of dictionaries, one per tune. Use provided `field_names` to obtain dictionary keys. Full expected db is in `expected_db.py` file.

> **DO NOT write hundreds of `ifs`**
>
> Special keys are listed above, all others should be treated in a generic way

> **DO NOT assume header always contains `'origin'` and `'history'`**
>
> It can contain *any* field, which has to be then copied in all the tunes.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[3]:
```
field_names = {
    'O':'origin',
    'H':'history',
    'X':'index',
    'T':'title',
    'C':'composer',
    'M':'meter',
    'K':'key',
}

def parse_tunes(filename):

    with open('tunes.abc', encoding='utf-8') as f:
        f.readline()  # skips %abc-2.1
        tunes = []
        common = {}
        for line in f:
            #pprint(line)
            clean_line = line.split('%')[0].strip()
            if clean_line == '':
                tune = common.copy()
                tunes.append(tune)
            elif len(tunes) > 0:
                if 'key' in tune:
                    tune['melody'] = parse_melody(clean_line)
                else:
                    k,v = clean_line.split(':')
                    if k == 'X':    # index
                        vp = int(v)
                    elif k == 'M':  # meter
                        s = v.split('/')
                        vp = (int(s[0]), int(s[1]))
                    else:
                        vp = v
                    tune[field_names[k]] = vp
            else: # header
                couple = clean_line.split(':')
                common[field_names[couple[0]]] = couple[1]

        return tunes


tunes_db = parse_tunes('tunes.abc')
pprint(tunes_db[:2],width=150)
```

```
[{'composer': 'The Lord of the Loop',
  'history': 'Tune made in a dark algorithmic night',
  'index': 1,
  'key': 'C',
  'melody': [(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6),␣
↪(6, 4)],
  'meter': (4, 4),
  'origin': 'Trento',
  'title': 'Algorave'},
```

(continues on next page)

(continued from previous page)

```
 {'composer': 'Matrix Queen',
  'history': 'Tune made in a dark algorithmic night',
  'index': 2,
  'key': 'G',
  'melody': [(5, 4), (6, 8), (4, 8), (4, 2), (5, 2), (0, 4), (1, 4), (3, 4), (3, 6),␣
↪(4, 6), (2, 6), (2, 6)],
  'meter': (3, 4),
  'origin': 'Trento',
  'title': 'Transpose Your Head'}]
```

[3]:
```
field_names = {
    'O':'origin',
    'H':'history',
    'X':'index',
    'T':'title',
    'C':'composer',
    'M':'meter',
    'K':'key',
}

def parse_tunes(filename):
    raise Exception('TODO IMPLEMENT ME !')

tunes_db = parse_tunes('tunes.abc')
pprint(tunes_db[:2],width=150)
```

```
[{'composer': 'The Lord of the Loop',
  'history': 'Tune made in a dark algorithmic night',
  'index': 1,
  'key': 'C',
  'melody': [(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6),␣
↪(6, 4)],
  'meter': (4, 4),
  'origin': 'Trento',
  'title': 'Algorave'},
 {'composer': 'Matrix Queen',
  'history': 'Tune made in a dark algorithmic night',
  'index': 2,
  'key': 'G',
  'melody': [(5, 4), (6, 8), (4, 8), (4, 2), (5, 2), (0, 4), (1, 4), (3, 4), (3, 6),␣
↪(4, 6), (2, 6), (2, 6)],
  'meter': (3, 4),
  'origin': 'Trento',
  'title': 'Transpose Your Head'}]
```

[4]:
```
assert tunes_db[0]['history']=='Tune made in a dark algorithmic night'
assert tunes_db[0]['origin']=='Trento'
assert tunes_db[0]['index']==1
assert tunes_db[0]['title']=='Algorave'
assert tunes_db[0]['composer']=='The Lord of the Loop'
assert tunes_db[0]['meter']==(4,4)
assert tunes_db[0]['key']== 'C'
assert tunes_db[0]['melody']==\
[(0, 8), (2, 4), (4, 4), (2, 8), (4, 2), (3, 2), (2, 4), (2, 6), (1, 6), (6, 4)]
```

(continues on next page)

(continued from previous page)

```
assert tunes_db[1]['history']=='Tune made in a dark algorithmic night'
assert tunes_db[1]['origin']=='Trento'
assert tunes_db[1]['index']==2
assert tunes_db[1]['title']=='Transpose Your Head'
assert tunes_db[1]['composer']=='Matrix Queen'
assert tunes_db[1]['meter']==(3,4)
assert tunes_db[1]['key']== 'G'
assert tunes_db[1]['melody']==\
[(5, 4), (6, 8), (4, 8), (4, 2), (5, 2), (0, 4), (1, 4), (3, 4), (3, 6), (4, 6), (2,␣
↪6), (2, 6)]
```

3. sequencer

Write a function `sequencer` which takes a melody in text format and outputs a matrix of note events, as a list of strings.

The rows are all the notes on keyboard (we assume 7 notes without black keys) and the columns represent the duration of a note.

- a note start is marked with < character, a sustain with = character and end with >
- **HINT 1**: call `parse_melody` to obtain notes as a list of tuples (if you didn't manage to implement it copy expected list from `expected_db.py`)
- **HINT 2**: build first a list of list of characters, and only at the very end convert to a list of strings
- **HINT 3**: try obtaining the note letters for first column by using `ord` and `chr`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[5]:
```
def sequencer(melody):

    notes = parse_melody(melody)
    length = 0
    for note in notes:
        length += note[1]
    ret = []

    work = [[' ']*(length+1) for i in range(7)]

    for i in range(7):
        work[i][0] = chr(ord('A')+i)

    j = 0
    for n,d in notes:
        work[n][j+1] = '<'
        eqlen = d-2
        work[n][j+2 : j+eqlen+2] = '=' * eqlen   # cool slice writing
        work[n][j+d] = '>'
        j += d
    return [''.join(w) for w in work]


from pprint import pprint
melody1 =  "|A4      C2  E2 |C4      E D C2 |C3    B3    G2  |"
exp1 = [
           'A<======>                                        ',
```

(continues on next page)

(continued from previous page)

```
            'B                                          <====>    ',
            'C         <==>    <======>    <==><====>            ',
            'D                            <>                      ',
            'E             <==>        <>                         ',
            'F                                                    ',
            'G                                              <==>']

res1 = sequencer(melody1)
print('  ' + melody1)
print()
pprint(res1)
assert res1 == exp1
```

```
  |A4      C2  E2 |C4      E D C2 |C3    B3    G2  |

['A<======>                                          ',
 'B                                      <====>    ',
 'C        <==>    <======>    <==><====>           ',
 'D                            <>                   ',
 'E            <==>        <>                       ',
 'F                                                 ',
 'G                                            <==>']
```

[5]:
```
def sequencer(melody):
    raise Exception('TODO IMPLEMENT ME !')

from pprint import pprint
melody1 =  "|A4      C2  E2 |C4      E D C2 |C3    B3    G2  |"
exp1 = [
            'A<======>                                          ',
            'B                                          <====>    ',
            'C         <==>    <======>    <==><====>            ',
            'D                            <>                      ',
            'E             <==>        <>                         ',
            'F                                                    ',
            'G                                              <==>']

res1 = sequencer(melody1)
print('  ' + melody1)
print()
pprint(res1)
assert res1 == exp1
```

```
  |A4      C2  E2 |C4      E D C2 |C3    B3    G2  |

['A<======>                                          ',
 'B                                      <====>    ',
 'C        <==>    <======>    <==><====>           ',
 'D                            <>                   ',
 'E            <==>        <>                       ',
 'F                                                 ',
 'G                                            <==>']
```

[6]:
```
from pprint import pprint
melody2 =  "|F2  G4     |E4      E F|A2  B2  D2 |D3    E3   |C3    C3   |"
```

(continues on next page)

(continued from previous page)

```
exp2 =    ['A                        <==>                                ',
           'B                            <==>                            ',
           'C                                                <====><====>',
           'D                                <==><====>                  ',
           'E             <======><>                     <====>          ',
           'F<==>                   <>                                  ',
           'G    <======>                                                ']

res2 = sequencer(melody2)
print('  ' + melody2)
print()
pprint(res2)
assert res2 == exp2
```

```
  |F2  G4     |E4      E F|A2  B2  D2 |D3    E3   |C3    C3   |

['A                        <==>                                ',
 'B                            <==>                            ',
 'C                                                <====><====>',
 'D                                <==>><====>                 ',
 'E             <======><>                     <====>          ',
 'F<==>                   <>                                  ',
 'G    <======>                                                ']
```

4. plot_tune

Make it fancy: write a function which takes a tune dictionary from the db and outputs a plot

- use beats as xs, remembering the shortest note has two beats
- to increase thickness, use `linewidth=5` parameter

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[7]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_tune(tune):


    j = 0
    for n,d in tune['melody']:
        r = 6 - n
        xs = [j, j+d-1]
        ys = [r, r]
        j += d
        plt.plot(xs, ys, linewidth=5)

    plt.title('%s \nby \n %s' %(tune['title'], tune['composer']))
    plt.ylim(-0.5,6.5)
    plt.yticks(range(7), [chr(ord('G')-i) for i in range(7)])
    plt.xlabel('beats')
    plt.ylabel('keyboard')


plot_tune(tunes_db[0])
```

[7]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_tune(tune):
    raise Exception('TODO IMPLEMENT ME !')

plot_tune(tunes_db[0])
```

[ ]:

2.2 2017-18 (QCB)

See QCB master past exams on sciprolab2 Github[126]

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[127]

WARNING: keep in mind that 2016-17 exams are for Python 2 - in this course we use Python 3

```
[ ]:
```

[126] https://github.com/DavidLeoni/sciprolab2/tree/master/overlay/_static
[127] https://davidleoni.github.io/algolab/past-exams.html

CHAPTER

THREE

SLIDES 2020/21

old 2019/20 slides

3.1 Part A

3.2 Lab A.1

Wednesday 23 Sep 2020

3.2.1 Links

lab site: sciprog.davidleoni.it[128]

- *Installation* from sciprog, with links to relevant SoftPython stuff
- Tools and Scripts on SoftPython[129] : describes Jupyter and other things are described more in detail
- Python basics[130] : hopefully we will see something (try doing them at home anyway)

What I expect

- if you don't program in Python, you don't learn Python
- you don't learn Python if you don't program in Python
- to be a successful data scientist, you must know programming
- **Exercise**: now put the right priorities in your TODO list ;-)

[128] https://sciprog.davidleoni.it
[129] https://en.softpython.org/tools/tools-sol.html
[130] https://en.softpython.org/basics/basics-sol.html

Course contents

- Hands-on approach

Part A - python intro

- logic basics
- discrete structures basics
- python basics
- data cleaning
- format conversion (matrices, tables, graphs, …)
- visualization (matplotlib, graphviz)
- some analytics (with pandas)
- focus on correct code, don't care about performance
- plus: some software engineering wisdom

Part A exams:

There will always be some practical structured exercise.

Examples:

- analizing employees of University of Trento[131]
- visualizing intercity bus network[132]
- extracting categories from workshops of Trentino[133]

Sometimes, there can also be a more abstract thing with matrices / relations, (i.e. surjective relation[134])

DO read the exam rules[135]

Part B - algorithms

- going from theory taught by Prof. Luca Bianco to Python 3 implementation
- performance matters
- few Python functions

3.3 Lab A.2

Monday 28 Sep 2020

Worksheets

- *Finish Basics* (NOTE: redownload it, was updated)
- Strings1 - Introduction[136]
- Strings2 - Operators[137]

[131] https://sciprog.davidleoni.it/exams/2019-08-26/solutions/exam-2019-08-26-sol.html#Part-A---University-of-Trento-staff
[132] https://sciprog.davidleoni.it/exams/2019-02-13/solutions/exam-2019-02-13-sol.html#Part-A---Bus-network-visualization
[133] https://sciprog.davidleoni.it/exams/2019-07-02/solutions/exam-2019-07-02-sol.html#A1-Botteghe-storiche
[134] https://sciprog.davidleoni.it/exams/2018-11-16/solutions/exam-2018-11-16-sol.html#A2-surjective
[135] https://sciprog.davidleoni.it/#Exams
[136] https://en.softpython.org/strings/strings1-sol.html
[137] https://en.softpython.org/strings/strings2-sol.html

- Strings3 - Methods[138]
- (a fourth page remains to be added)

References:

- Andrea Passerini - Introduction to Python slides[139]
- Andrea Passerini - data structures slides[140]

3.4 Lab A.3

Wednesday 30 Set 2020

Worksheets:

- Lists1 - Introduction[141]
- Lists2 - Operators[142]
- Lists3 - Methods[143]
- (a fourth page remains to be added)

References: Andrea Passerini - data structures slides[144]

For Realtime stuff during the lab, see this repl:

https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21

3.5 Lab A.4

Monday 5 Oct 2020

Worksheets (now English links are finally online!)

- Tuples[145]
- Sets[146]
- Dictionaries1 - Introduction[147]
- Dictionaries2 - Operators[148]
- Dictionaries3 - Methods[149]
- (a fourth and fifth pages remain to be added)

References:

[138] https://en.softpython.org/strings/strings3-sol.html
[139] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A01-introduction.pdf
[140] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A02-datastructures.pdf
[141] https://en.softpython.org/lists/lists1-sol.html
[142] https://en.softpython.org/lists/lists2-sol.html
[143] https://en.softpython.org/lists/lists3-sol.html
[144] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A02-datastructures.pdf
[145] https://en.softpython.org/tuples/tuples-sol.html
[146] https://en.softpython.org/sets/sets-sol.html
[147] https://en.softpython.org/dictionaries/dictionaries1-sol.html
[148] https://en.softpython.org/dictionaries/dictionaries2-sol.html
[149] https://en.softpython.org/dictionaries/dictionaries3-sol.html

- Andrea Passerini - data structures slides[150]
- Real Python sets[151]

For realtime stuff during the lab, see this repl:

https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21

I copy here last weekend announcment:

I added here some links to exercises: https://en.softpython.org/references.html

- Beginners: try W3Resources
- Intermediate: exercises from LeetCode and first tutorial from Software Carpentry
- Math-oriented people: try Introduction to Scientific Programming with Python by Joakim Sundnes

3.6 Lab A.5

Wednesday 7 Oct 2020

- Worksheets (now English links are finally online!)
  - Control Flow 1 - ifs[152]
  - Control Flow 2 - for[153]
  - Control Flow 3 - while[154]
  - Errors and testing[155] Note: SoftPython tutorial only talks about quick testing with `assert` which is fine for Part A of this course. In Part B we will use more sophisticated testing with `unittest`
  - References: Andrea Passerini - complex statements[156]

For realtime stuff during the lab, see this repl:

https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21

3.7 Lab A.6

Monday 12 Oct 2020

Worksheets

- More on functions
- Matrices - list of lists[157]
- Errors and testing[158] Note: SoftPython tutorial only talks about quick testing with `assert` which is fine for Part A of this course. In Part B we will use more sophisticated testing with `unittest`

References:

[150] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A02-datastructures.pdf
[151] https://realpython.com/python-sets/
[152] https://en.softpython.org/control-flow/flow1-if-sol.html
[153] https://en.softpython.org/control-flow/flow2-for-sol.html
[154] https://en.softpython.org/control-flow/flow3-while-sol.html
[155] https://en.softpython.org/errors-and-testing/errors-and-testing-sol.html
[156] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A03-controlflow.pdf
[157] https://en.softpython.org/matrices-lists/matrices-lists-sol.html
[158] https://en.softpython.org/errors-and-testing/errors-and-testing-sol.html

- Andrea Passerini - complex statements[159]
- Andrea Passerini - functions[160]

For realtime stuff during the lab, see this repl:

https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21

3.8 Lab A.7

Wednesday 14 Oct 2020

Worksheets

- Sequences[161] (already translated)
- Data formats[162] (already translated)

References:

- Andrea Passerini - Modules and programs[163]

For realtime stuff during the lab, see this repl:

https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21

3.9 Lab A.8

Monday 19 Oct 2020

Worksheets:

- Graph formats[164]
  - graph matrices, adjacency lists (we will use networkx next lab)
- Binary relations[165] (finish at home)

NOTE: In Part A we will limit ourselves to graph formats and see some theory in separate binary relations notebook, while in Part B of the course will focus on graph algorithms[166]

For realtime stuff during the lab:

- sciprog repl[167]
- graphviz repl[168]
- graph drawings and challenges[169]

[159] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A03-controlflow.pdf
[160] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A04-functions.pdf
[161] https://en.softpython.org/sequences/sequences-sol.html
[162] https://en.softpython.org/formats/formats-sol.html
[163] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A05-programs.pdf
[164] https://en.softpython.org/graph-formats/graph-formats-sol.html
[165] https://en.softpython.org/binary-relations/binary-relations-sol.html
[166] https://sciprog.davidleoni.it/graph-algos/graph-algos.html
[167] https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21
[168] https://repl.it/@DavidLeoniWork/softgraphviz
[169] https://docs.google.com/presentation/d/1rnSWMikHWAgxyVpFWI1NouB7iQiZNYUKwgF9IHL0MdU/edit?usp=sharing

3.9.1 How to see graphviz on repl.it

Since we're online, I came up with this solution

NOTE: this is a shaky experiment only to share code during the lab, but you won't need if you do the worksheets above alone.

1. Go to this NEW repl: https://repl.it/@DavidLeoniWork/softgraphviz
2. Fork the project (**DO NOT** copy paste files in your projects, you **really** need to fork !)

3. Run the script - you will have to wait some time until it loads all the packages:

NOTE: this will activate a *server,* which means from now on the server will be kept running and you shouldn't need to click on Run again (unless of course something is seriously compromised)

4. Try making changes to `example.py` and then, instead of hitting the usual Run in repl, click the Reload button. Reload will run the script `example.py`, which at the end regenerates the `output.dot` file used in the visualization

NOTE: that after a while of inactivity, the server might stop and repl will need to reload all the packages.

3.10 Lab A.9

Wednesday 21 Oct 2020

EXAM - Save the date:

- Simulation: Monday 2 November, 14:30-16:30 (online)
- Midterm Part A: Friday 6 November, 11.30-13.30 (online) - please login 15 mins before and allocate some time after the end in case we have technical problems

Simulation attendance is recommended!: during the simulation and the exam you will connect online to a linux environment (but you **do not** need to install linux!), so the simulation is your chance to try the system.

You will receive official communication with details in a short time. For now **please read** exam guidelines from previous years[170]

WORKSHEETS:

Today we focus on graphs as adjacency lists, networkx

- Graph formats[171]
- Binary relations[172]

NOTE: In Part A we will limit ourselves to graph formats and see some theory in separate binary relations notebook, while in Part B of the course will focus on graph algorithms[173]

[170] https://sciprog.davidleoni.it/#Exams
[171] https://en.softpython.org/graph-formats/graph-formats-sol.html
[172] https://en.softpython.org/binary-relations/binary-relations-sol.html
[173] https://sciprog.davidleoni.it/graph-algos/graph-algos.html

For realtime stuff during the lab:

- sciprog repl[174]
- graphviz repl[175] (to use, see instructions of *previous lab*)
- graph drawings and challenges[176]

3.11 Lab A.10

Monday 26 Oct 2020

Worksheets:

Today we focus on numpy and matplotlib

- Matrices - Numpy[177] - actually we don't do that much of numpy, it's more an introduction to things needed for visualization
- Visualization with matplotlib[178]
- The AlgoRythm Opera Challenge[179]

NOTE 1: Visualization worksheet is different from Italian version! Contains also some statistics on networks.

NOTE 2: after the lab I put on softpython a revised version of the The AlgoRythm Opera Challenge. In particular even if you were at the lab, make sure to try this new last part of the challenge[180], with changed variables: most probably you will discover there is some constant in your code which should be a variable instead.

References:

- Andrea Passerini Numpy and Matplotlib slides[181]

For realtime stuff during the lab:

- sciprog repl[182]
- graph drawings and challenges[183]

3.12 Lab A.11

Wednesday 28 Oct 2020

Worksheets:

- Matrices - Numpy images (new!)[184] - was proposed as a challenge during the lab, now on the website you can find a revised version with solutions. Notice images may look like an application, but actually they are matrices, only much nicer to see!

References:

[174] https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21
[175] https://repl.it/@DavidLeoniWork/softgraphviz
[176] https://docs.google.com/presentation/d/1rnSWMikHWAgxyVpFWI1NouB7iQiZNYUKwgF9IHL0MdU/edit?usp=sharing
[177] https://en.softpython.org/matrices-numpy/matrices-numpy-sol.html
[178] https://en.softpython.org/visualization/visualization-sol.html
[179] https://en.softpython.org/visualization/algorythm-opera-chal.html
[180] https://en.softpython.org/visualization/algorythm-opera-chal.html#Final-result-2
[181] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A08-numpy.pdf
[182] https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21
[183] https://docs.google.com/presentation/d/1rnSWMikHWAgxyVpFWI1NouB7iQiZNYUKwgF9IHL0MdU/edit?usp=sharing
[184] https://en.softpython.org/matrices-numpy/numpy-images-sol.html

- Andrea Passerini Numpy and Matplotlib slides[185]

For realtime stuff during the lab:

- sciprog repl[186]
- graph drawings and challenges[187]

3.13 Lab A.12

Wednesday 4 Nov 2020

Worksheets:

- Pandas tutorial[188]
- Pandas case study: EURES job offers[189]
- Pandas: Italian Poets challenge[190]
- Exercises from Edabit, I put a selection on SoftPython References page[191] (if you don't see it refresh the page with ctrl-R or ctrl-F5)

Note there won't be pandas during midterm for part A, at most I will use it only to show some data. For you, it could be useful to quickly gain an understanding of dataset contents.

References:

- Andrea Passerini Pandas slides[192]

For realtime stuff during the lab:

- sciprog repl[193]
- graph drawings and challenges[194]

[185] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A08-numpy.pdf
[186] https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21
[187] https://docs.google.com/presentation/d/1rnSWMikHWAgxyVpFWI1NouB7iQiZNYUKwgF9IHL0MdU/edit?usp=sharing
[188] https://en.softpython.org/pandas/pandas-sol.html
[189] https://en.softpython.org/pandas/eures-jobs-sol.html
[190] https://en.softpython.org/pandas/italian-poets-chal.html
[191] https://en.softpython.org/references.html#Edabit
[192] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/slides/A07-pandas.pdf
[193] https://repl.it/@DavidLeoniWork/sciprog-ds-lab-2020-21
[194] https://docs.google.com/presentation/d/1rnSWMikHWAgxyVpFWI1NouB7iQiZNYUKwgF9IHL0MdU/edit?usp=sharing

3.14 Part B

3.15 Lab B.1

Monday 9 Nov 2020

- Remember that from now on we only use Visual Studio Code, possibly with the Live Share extension, so get familiar with it by reading *installation*
- *Testing with unittest*
- *OOP* (probably only first part)

3.16 Lab B.2

Wednesday 11 Nov 2020

Worksheets:

- *OOP* Finished ComplexNumber, try doing Multiset on your own.
- *OOP Matrix Challenge*: (updated after the lab, please use this version). Try solving this for next week. Tests are not provided, you are invited to collaboratively write at least some in the new 'REPL Test Playground'. You can access the link on Moodle[195]

[195] https://didatticaonline.unitn.it/dol/mod/url/view.php?id=721545

3.17 Lab B.3

Monday 16 Nov 2020

Worksheets:

- *Complexity*: the invisible enemy
- *Sorting Challenge*
- *Sorting* Selection sort and Insertion sort

3.18 Lab B.4

Monday 16 Nov 2020

Worksheets:

- *Complexity*: the invisible enemy
- *Sorting Challenge* - I added Partitocracy challenge (a kind of quicksort)
- *Sorting* in lab we will see quicksort and mergesort, finish the rest at home

```
[ ]:
```

CHAPTER

FOUR

COMMANDMENTS

4.1 MOVED TO https://en.softpython.org/commandments.html

You will be redirected in 10 seconds

```
[ ]:
```

CHAPTER

FIVE

CHAPTER

SIX

PART A

6.1 Installation

You will need to install several pieces of software to get a working Python 3 programming environment. In this section we will install everything that we are going to need in the next few weeks.

There are many ways to write and execute Python code:

- Python interpreter[196] (command line)
- Visual Studio Code[197] (editor, good debugger)
- Jupyter[198] (notebook, good for experimentation and writing reports)
- Google Colab[199] (online, sort-of collaborative)
- repl.it[200] (online, collaborative)
- Python Tutor[201] (online, visual debugger, only for short code)

Python 3 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 Jupyter for Part A of the course and Visual Studio Code for Part B.

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 3.8

1. Install Anaconda for Python 3.8[202] (anaconda installer will ask you to install also visual studio code, so accept the kind offer)
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[203]. Downloads for all platforms can be found here[204]
3. Now you can read all of Installation on SoftPython[205] **EXCEPT**:
    - For Mac users: for this course you **don't** need to install homebrew, just install Anaconda
    - For everybody: You don't need to read *Projects with virtual environments* paragraph (although it's certainly useful when you have many Python projects on your pc!)

[196] https://www.python.org/
[197] https://code.visualstudio.com/
[198] https://jupyter.org/
[199] https://colab.research.google.com/
[200] https://repl.it/languages/python3
[201] http://www.pythontutor.com/visualize.html#mode=edit
[202] https://www.anaconda.com/download/
[203] https://code.visualstudio.com/
[204] https://code.visualstudio.com/Download
[205] https://en.softpython.org/installation.html

4. **Jupyter and course material format** is described in detail in Tools and Scripts on SoftPython[206], read it!
5. Finally, you can get familiar with Visual Studio Code by reading what follows. Even if we will use it only in Part B, read it anyway as it's useful for many development tasks and supports a lot languages.

6.1.1 Visual Studio Code

Visual Studio Code[207] 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:

[206] https://en.softpython.org/tools/tools-sol.html
[207] https://code.visualstudio.com/

You can find useful information on this tool here[208]. 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.

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.

[1]:
```
"""
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)
```

[208] https://code.visualstudio.com/docs#vscode

```
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.

```
1   """
2   This is the first example of Python script.
3   """
4   a = 10 # variable a
5   b = 33 # variable b
6   c = a / b # variable c holds the ratio
7
8   # Let's print the result to screen.
9   printt("a:", a, " b:", b, " a/b=", c)
10
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    Filter by t
example1.py  4
[pylint] E0602:Undefined variable 'printt' (9, 1)
[pylint] C0103:Invalid constant name "a" (4, 1)
[pylint] C0103:Invalid constant name "b" (5, 1)
[pylint] C0103:Invalid constant name "c" (6, 1)
```

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*):

[2]:
```
"""
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 value 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.

6.1.2 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.

[3]:
```
""" 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

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.

6.1.3 VS Code - collaborative coding

Visual Studio Code offers a cool way to collaborate in realtime on a project you are working on. Let's see how to set it up.

1. Install: First make sure you have Live Share Extension Pack[209] installed. It should already be included in VS Code 2019, if not got to extensions and install it:

[209] https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare-pack

Let's share some code - as example, we will use a project taken from *Object Oriented Programming* chapter.

2. open a folder: Once you downloaded the code, open the folder where you put it with `File->Open Folder` (avoid `Open Workspace` for simplicity reasons). You should end up with something like this:

> **CAREFUL ABOUT WHAT YOU SHARE !!!**
>
> Once you start sharing, anything in that folder will be accessible and writable by others, we do not want to see your embarassing desktop/home folder !

3. Sign in Now you need to sign in, for this we refer to the the official How To[210]

NOTE: in order to share your folder, you will need a Github account[211], so please if you don't have it already create one now

4. Share the folder: See the official How To[212]

6.1.4 VS Code - Sharing test runs

In Part B we will also heavily use unit tests (see Unittest tutorial[213]), so it's a good idea to enabling sharing of test runs by installing Test Explorer Live Share[214] extension:

> **IT MAY NOT WORK!**
>
> It's often troublesome enough having working tests on your own system, so I expect sharing to be even more problematic. As of Nov 2020, I've never tried this thing before, so let's make a try and if if doesn't work let's just use good old screen sharing.

[210] https://docs.microsoft.com/en-us/visualstudio/liveshare/use/vscode#sign-in
[211] https://github.com
[212] https://docs.microsoft.com/en-us/visualstudio/liveshare/use/vscode#share-a-project
[213] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-Unittest
[214] https://marketplace.visualstudio.com/items?itemName=hbenl.vscode-test-explorer-liveshare

[ ]:

```

```

6.2 Python basics solutions

6.2.1 MOVED TO en.softpython.org/basics/basics-sol.html[215]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[1]:
```
# SOLUTION
```

[1]:

[ ]:

[215] https://en.softpython.org/basics/basics-sol.html

6.3 Strings solutions

6.3.1 MOVED TO https://en.softpython.org/#strings

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

6.4 Lists solutions

6.4.1 MOVED TO https://en.softpython.org/#lists

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

6.5 Tuples solutions

6.5.1 MOVED TO https://en.softpython.org/tuples/tuples-sol.html

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

```
[ ]:
```

6.6 Sets solutions

6.6.1 MOVED TO https://en.softpython.org/sets/sets-sol.html

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

6.7 Dictionaries solutions

6.7.1 MOVED TO https://en.softpython.org/#dictionaries

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[ ]:

6.8 Control flow solutions

6.8.1 MOVED TO https://en.softpython.org/#control-flow

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

6.9 Functions - solutions

6.9.1 Download exercises zip

Browse files online[216]

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[217]
- Thinking in Python, Chapter 3, Functions[218]
- Thinking in Python, Chapter 6, Fruitful functions[219] **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
```

[216] https://github.com/DavidLeoni/sciprog-ds/tree/master/functions
[217] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf
[218] http://greenteapress.com/thinkpython2/html/thinkpython2004.html
[219] http://greenteapress.com/thinkpython2/html/thinkpython2007.html

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `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`

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):
```

(continues on next page)

(continued from previous page)

```
    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)
        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)
```

(continues on next page)

(continued from previous page)

```
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()
```

(continues on next page)

(continued from previous page)

```
    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)"""
    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
```

(continues on next page)

(continued from previous page)

```
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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show answer" data-jupman-hide="Hide">Show answer</a>

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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[44]:
```
# write here

def sum2(x,y):
    return x + y
```

[44]:
```
# write here
```

[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 ")
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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)
```

[47]:
```
# write here
```

[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`

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[51]:
```
# write here
def comparer(x,y):
    if x > y:
        return '>'
    elif x < y:
        return '<'
    else:
        return '=='
```

[51]:
```
# write here
```

[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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[59]:
```
# write here
def even(x):
    return x % 2 == 0
```

[59]:
```
# write here
```

[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.

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[65]:
```
# write here

def gre(x,y):
    if x > y:
        return x
    else:
        return y
```

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

[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'
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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')
```

[71]:
```
# write here
```

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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[72]:
```
# write here

def sphere_volume(radius):
    print((4/3)*3.14*(radius**3))
```

[72]:
```
# write here
```

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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[73]:
```
# write here

def ciri(name):
    if name == "Cirillo":
        return True
    else:
        return False
```

[73]:
```
# write here
```

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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[74]:
```
# write here

def age(year):
    return 2019 - year
```

[74]:
```
# write here
```

6.9.6 Verify comprehension

Following exercises require you to know:

> **ATTENTION**
>
> Following exercises require you to know:
>
> Complex statements: Andrea Passerini slides A03[220]
>
> Tests with asserts[221]: Following exercises contain automated tests to help you spot errors. To understand how to do them, read before Error handling and testing[222]

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:
            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
```

[220] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[221] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-asserts
[222] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html

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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[76]:
```
def final_price(n):

    if n == 0:
        return 0
    else:
        return n* 24.95*0.6 + 3 +(n-1)*0.75


assert final_price(10) == 159.45
assert final_price(0) == 0
```

[76]:
```
def final_price(n):
    raise Exception('TODO IMPLEMENT ME !')

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
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[77]:
```
def arrival_time(n,m):

    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)


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'
```

[77]:
```
def arrival_time(n,m):
    raise Exception('TODO IMPLEMENT ME !')

assert arrival_time(0,0) == '6:52'
assert arrival_time(2,2) == '7:22'
assert arrival_time(2,5) == '7:44'
assert arrival_time(8,5) == '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')
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[92]:
```
# write here

def apply_borders(f, seq):
    return ( f(seq[0]),   f(seq[-1]) )
```

[92]:
```
# write here
```

[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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[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
```

[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 ...
```

[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

- **Error handling was** moved to softpython[223]

According to the part of the course you are following, we will review two kinds of tests:

- **Part A testing with asserts**: moved to SoftPython[224]
- **Part B testing with unittest**: read this notebook

[223] https://en.softpython.org/errors-and-testing/errors-and-testing-sol.html
[224] https://en.softpython.org/errors-and-testing/errors-and-testing-sol.html#Testing-with-asserts

6.10.1 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[225]. 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[226] (note it can also execute tests made with unittest, so if your visualstudio code for some reason doesn't work with unittest, you can try setting pytest as test framework)

So let's give unittest a try. Suppose you have a file called `file_test.py` like this:

[2]:
```
import unittest

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
    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 chapter[227]), 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)`

[225] https://docs.python.org/3/library/unittest.html
[226] https://docs.pytest.org/
[227] http://interactivepython.org/runestone/static/pythonds/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html

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:

[3]:
```
jupman.show_run(MyTest)
```

```
F
======================================================================
FAIL: test_long_list (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-2-ec89a972a892>", 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]

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 putting the following code at the very end of the file you are editing:

```
unittest.main()
```

Then simply run your file with:

```
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 very 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:

[9]:
```
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
```

[5]:
```
jupman.show_run(MyTest)
```

```
FF
======================================================================
FAIL: test_long_list (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-4-306d9f1c7777>", 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-4-306d9f1c7777>", 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 0.003s

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:

[10]:
```
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
```

And call yet again the command:

```
python -m unittest file_test
```

[7]:
```
jupman.show_run(MyTest)
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.002s

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[228] (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[229]:

- If `even_numbers(3.5)` actually raises `TypeError` exception, nothing happens
- If `even_numbers(3.5)` does not raise `TypeError` exception, with raises `AssertionError`

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`

[228] https://docs.python.org/3/library/exceptions.html#TypeError
[229] https://docs.python.org/3/library/exceptions.html#TypeError

2. Select unittest:

3. Select `. root directory` (we assume tests are in the folder that you've opened):

4. Select `*test*.py 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:

In the bottom bar, you should see a recap of tests run (right side of the picture):

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 updating 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.

> **Spend time using the console !!!!**
>
> During exams VSCode testing might not work, so please be prepared to use the console

6.10.2 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, when 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: it might fail in one run and succeed in another just because the system scheduled differently the code execution! Functional programming frameworks like Spark[230] 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[231] (will talk more about it in class)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[230] https://spark.apache.org
[231] https://docs.google.com/presentation/d/1hTHty5aML9WDDTvkflvvdGDh0AfZwV_8ZEr10-rUPVA

6.11 Matrices: list of lists solutions

6.11.1 Moved to https://en.softpython.org/matrices-lists/matrices-lists-sol.html

(Redirecting in 10 seconds …)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

6.12 Matrices: Numpy solutions

6.12.1 Moved to https://en.softpython.org/matrices-numpy/matrices-numpy-sol.html

(Redirecting in 10 seconds …)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

6.13 Data formats solutions

6.13.1 MOVED TO https://en.softpython.org/formats/formats-sol.html

(Redirecting in 10 seconds …)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

6.14 Graph formats solutions

6.14.1 Moved to https://en.softpython.org/graph-formats/graph-formats-sol.html

(Redirecting in 10 seconds …)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

```
[ ]:
```

6.15 Visualization solutions

6.15.1 Moved to https://en.softpython.org/visualization/visualization-sol.html

(Redirecting in 10 seconds …)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

6.16 Pandas solutions

6.16.1 Moved to en.softpython.org/pandas/pandas-sol.html[232]

You will be redirected in 10 seconds …

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

```
[ ]: # SOLUTION
```

```
[ ]:
```

6.17 Binary relations solutions

6.17.1 MOVED TO https://en.softpython.org/binary-relations/binary-relations-sol.html

(Will redirect in 10 seconds)

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[232] https://en.softpython.org/pandas/pandas-sol.html

CHAPTER

SEVEN

PART B

7.1 OOP

7.1.1 Download exercises zip

Browse files online[233]

7.1.2 What to do

- unzip exercises in a folder, you should get something like this:

```
oop
   ComplexNumber.py
   ComplexNumber_test.py
   ComplexNumber_sol.py
   multiset.py
   multiset_test.py
   multiset_sol.py
   matrix.py
   oop.ipynb
   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:
  - Andrea Passerini - Programming paradagims, Object-Oriented Python[234]
- Object Oriented programming on the the book[235] (In particular, Fraction class[236], in this course we won't focus on inheritance)

[233] https://github.com/DavidLeoni/sciprog-ds/tree/master/oop
[234] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/handouts/A09-oop.pdf
[235] https://runestone.academy/runestone/books/published/pythonds/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html
[236] https://runestone.academy/runestone/books/published/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:

[1]:
```
c = [3.0, 5.0]
```

Or we could use a tuple:

[2]:
```
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:

[3]:
```
c = {'real': 3.0, 'imaginary': 5.0}
```

[4]:
```
print(c)
```
```
{'imaginary': 5.0, 'real': 3.0}
```

[5]:
```
print(c['real'])
```
```
3.0
```

[6]:
```
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:

[7]:
```
def complex_number(real, imaginary):
    d = {}
    d['real'] = real
```

(continues on next page)

(continued from previous page)

```
    d['imaginary'] = imaginary
    return d
```

[8]:
```
c = complex_number(3.0, 5.0)
```

[9]:
```
print(c)
```

```
{'imaginary': 5.0, 'real': 3.0}
```

To do something with our dictionary, we would then define functions, like for example `complex_str` to show them nicely:

[10]:
```
def complex_str(cn):
    return str(cn['real']) + " + " + str(cn['imaginary']) + "i"
```

[11]:
```
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!

[12]:
```
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'])
```

[13]:
```
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):

[14]:
```
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)}
```

[15]:
```
print(log(c,2))
```

```
{'imaginary': 1.4865195378735334, 'real': 1.5849625007211563}
```

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:

[16]:
```
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:

[17]:
```
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__`:

[18]:
```
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:

[19]:
```
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:

[20]:
```
c = ComplexNumber(3.0,5.0)
print(c)
```

```
<__main__.ComplexNumber object at 0x7f54942ae2b0>
```

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:

!!!!!! VIII COMMANDMENT[237] : YOU SHALL NEVER EVER REASSIGN `self` !!!!!!!

Since self is a kind of dictionary, you might be tempted to do like this:

[21]:
```
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:

[22]:
```
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
```

init 2.2.5) Python automatically returns from `__init__` the special dictionary `self`

[237] https://en.softpython.org/commandments.html#VIII-COMMANDMENT

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

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

```
<__main__.ComplexNumber object at 0x7f54942ae2b0>
```

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:

[24]:
```
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`:

[25]:
```
print(c.real)
```

```
3.0
```

[26]:
```
print(c.imaginary)
```

```
5.0
```

If we want, we can also change them:

[27]:
```
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:

[28]:
```
import unittest
import math

class ComplexNumber:

    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary
```

(continues on next page)

(continued from previous page)

```
    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)
```

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:

[29]:
```
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)`:

[30]:
```
import math

class ComplexNumber:

    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary
```

(continues on next page)

(continued from previous page)

```
    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))
```

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`:

[31]:
```
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`

[32]:
```
print(logarithm)
```

```
<__main__.ComplexNumber object at 0x7f54942aed30>
```

To see if the method worked and we got back we got back a different complex number, we can print the single fields:

[33]:
```
print(logarithm.real)
```

```
1.0986122886681098
```

[34]:
```
print(logarithm.imaginary)
```

```
1.0303768265243125
```

2.3.3 __str__ for printing

As we said, printing is not so informative:

[35]:
```
print(ComplexNumber(3.0, 5.0))
```

```
<__main__.ComplexNumber object at 0x7f54942aefd0>
```

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__` !

[36]:
```
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__[238] obviously state you MUST return a string. **Do read them!**

[37]:
```
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:

[38]:
```
print(c)
```

```
3.0 + 5.0i
```

[39]:
```
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:

[40]:
```
c.__str__()
```

[40]:
```
'3.0 + 5.0i'
```

EXERCISE: There is another method for getting a string representation of a Python object, called `__repr__`. Read carefully __repr__ documentation[239] 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[240]

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
```

[238] https://docs.python.org/3/reference/datamodel.html#object.__str__
[239] https://docs.python.org/3/reference/datamodel.html#object.__repr__
[240] 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`

```
[41]: ComplexNumber(1,2) == ComplexNumber(1,2)
```

```
[41]: 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[241] 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:

[241] 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[242]
  - Messing with hashing[243]

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:

[242] http://jcalderone.livejournal.com/32837.html
[243] 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:

[42]:
```
    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.

[43]:
```
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`

4. Challenges

Have a look at the OOP Matrix Challenge[244]

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[ ]:

7.2 OOP Matrix Challenge

7.2.1 Download exercises zip

Browse files online[245]

We want to model operations on IMMUTABLE matrices, the object oriented way.

You have a generic interface called `Matrix` in file `matrix.py`, and you have to implement two descendants of the interface, `DenseMatrix` and `SparseMatrix`.

IMPORTANT: work in group and share tests!

You will make a lot of mistakes, some hard to debug: help yourself by writing test cases, we also invite you to share the tests (not the solutions!) with your fellow students.

Please note there are many cornercases, even if stuff *seems* to work there will probably occasions where it fails - you won't probably be able to find them all on your own.

7.2.2 What to do

- unzip exercises in a folder, you should get something like this:

```
oop
   matrix.py
   oop-matrix-chal.ipynb
   .
   .
```

You can ignore the notebook, and edit `.py` files in Visual Studio Code.

[244] http://sciprog.davidleoni.it/oop/oop-matrix-chal.html
[245] https://github.com/DavidLeoni/sciprog-ds/tree/master/oop

7.2.3 DenseMatrix

Let's begin with the simpler implementation, which is `DenseMatrix`: you will code it by storing data inside lists of lists.

Create a new file `dense_matrix.py` and declare a class `DenseMatrix` which inherits from `Matrix`. Copy inside all the methods definitions from `Matrix`.

```
from matrix import Matrix

class DenseMatrix(Matrix):
    """ A Matrix represented internally as a list of lists

        When creating it from list of lists, perform a DEEP COPY of the input
        and store it inside a private field you will call  _cells (use ONE underscore)
    """
```

7.2.4 Constructors and printing

Generally first methods we implement are the constructor and methods to show the class content such as `__str__` and `__repr__`. All methods are documented in the `matrix.py` interface, let's see now the constructor. Copy paste the following inside your `DenseMatrix` class.

```
def __init__(self, *args):
        """Initializes the Matrix with either:

            One argument: a list of lists
            Three arguments: number of rows
                             number of columns
                             a list of cell triplets (row_index,col_index,value)
                            - if number of rows or columns is less than indeces found
                              in triplets, raise Value Error

            In both cases, provided data must allow the creation of a matrix with
            at least one row and a column, otherwise raise ValueError
        """
        raise Exception("Should be implemented in a descendant of Matrix!")
```

Constructor as list of lists

We see the the constructor takes the special `*args` argument, which means it can accept a variable number of arguments. To check for the two cases, you might write something like this:

```
class DenseMatrix(Matrix):
    def __init__(self, *args):

        if len(args) == 1:   # list of lists
            rows = args[0]
            # do something ...
        elif len(args) == 3:
            n,m,triplets = args
            # do something
        else:
            raise ValueError('Expected one or three args!')
```

You can start by handling only the one argument case, which we will interpret as a list of lists. You can store a **deep** copy of the list of lists in a private field you can call `_cells`. Why should we store a deep copy? It's a form a so-called defensive-programming: if we store a copy, it's less likely that a user of our class will later mess with the instance internal data. If we store the data in a field beginning with one underscore _, we will signal external users we do not want them to directly access it (notice if they really want they will still be able to do it)

[1]:
```
from dense_matrix_chal_sol import *
from sparse_matrix_chal_sol import *
```

[2]:
```
dma = DenseMatrix([ [0,6,8,0],
                    [0,0,0,5],
                    [9,0,7,0] ])
```

[3]:
```
dma
```

[3]:
```
DenseMatrix([[0, 6, 8, 0], [0, 0, 0, 5], [9, 0, 7, 0]])
```

[4]:
```
type(DenseMatrix)
```

[4]:
```
type
```

str and repr

Implement `__str__` and `__repr__`

```
def __str__(self):
    """ RETURN a nice human-readable formatted string, possibly like this:

          Matrix  [ [5,2,6,3],
                    [8,4,7,4],
                    [2,1,9,8] ]

          - substitute Matrix with the descendant class name
          - NOTE: sometimes this representation is impractical (i.e. sparse matrices
                  with large n/m), in that case use another format of your choice
    """
    raise Exception("Should be implemented in a descendant of Matrix!")

def __repr__(self):
    """ RETURN one-line string representing a Python expression which would recreate␣
↪the atrix
    """
    raise Exception("Should be implemented in a descendant of Matrix!")
```

[5]:
```
str(dma)
```

[5]:
```
'DenseMatrix [ [0, 6, 8, 0]\n               [0, 0, 0, 5]\n               [9, 0, 7, 0] ]'
```

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

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

[7]:
```
repr(dma)
```

[7]: 'DenseMatrix([[0, 6, 8, 0], [0, 0, 0, 5], [9, 0, 7, 0]])'

[8]:
```
dma
```

[8]: DenseMatrix([[0, 6, 8, 0], [0, 0, 0, 5], [9, 0, 7, 0]])

constructor as list of triplets

Now extend the constructor to handle also the list of triplets case:

[9]:
```
dmb = DenseMatrix(3,4, [(0,1,6),(0,2,8),(1,3,5),(2,0,9),(2,2,7)])
```

[10]:
```
dmb
```

[10]: DenseMatrix([[0, 6, 8, 0], [0, 0, 0, 5], [9, 0, 7, 0]])

[11]:
```
print(dmb)
```

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

7.2.5 shape

Implement `shape` method:

```
def shape(self):
    """RETURN the number of rows and columns as a tuple
    """
```

[12]:
```
dma.shape()  # NOTE the ()
```

[12]: (3, 4)

7.2.6 Brackets operator

Override the square bracket operator by reimplementing the method `__getitem__`

IMPORTANT: Read carefully Python documentation[246]!!

```
def __getitem__(self, key):
    """Overrides default bracket access behaviour.
       key is whatever the user passes within the brackets, in our case
       we want user to be able to write

       my_mat[2,5]  to access element at row 2 and column 5

       NOTE 1: if the user types 2,5 inside the brackets, Python will actually
             generate a TUPLE (2,5)  so that is the key type you should
             accept (a tuple of *exactly two* integers)
       NOTE 2: Since this method signature is defined by Python documention:
```

(continues on next page)

[246] https://docs.python.org/3/reference/datamodel.html#object.__getitem__

(continued from previous page)

```
        https://docs.python.org/3/reference/datamodel.html#object.__getitem__

      you MUST write it respecting ALL the indications of the docs, in particular
      think also about  what should happen in undesired scenarios when
      the user enters a key of wrong type, wrong value, etc as described
      in the documentation
    """
```

[13]:
```
print(dma)
```

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

[14]:
```
dma[0,1]
```

[14]:
```
6
```

[15]:
```
dma[0,2]
```

[15]:
```
8
```

Negative indeces should also be supported:

[16]:
```
dma[0,-3]
```

[16]:
```
6
```

7.2.7 nonzero

```
def nonzero(self):
        """Return a list of triplets (row index, column index, value) of non-zero␣
↪cells,
           in no particular order.
        """
```

Since we stored the rows in a private field `_cells`, we have to offer users other ways to access data. For example, by calling `nonzero()` method users should be able to obtain non-zero values, as a list of triplets with row index, column index and value of cells. You might wonder what the utility is, after we already implemented the brackets operator: the reason will become clearer when you start implementing methods that need to visit a generic `Matrix`, like the following ones.

[17]:
```
print(dma)
```

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

[18]:
```
dma.nonzero()
```

[18]:
```
[(0, 1, 6), (0, 2, 8), (1, 3, 5), (2, 0, 9), (2, 2, 7)]
```

7.2.8 isclose

Implement this method:

> **DO NOT MAKE ASSUMPTIONS ABOUT `other` CLASS!**
>
> In the implementation, use `other` like if it were a generic `Matrix`, it might not always be a `DenseMatrix`. So the only way to access elements will be either by using the brackets operator or by calling `nonzero()`

```
def isclose(self, other, delta):
        """ RETURN True if each cell in this matrix is within a delta distance
            from other Matrix. RETURN False if any cell couple differs more than␣
↪delta.

            - if matrices have different dimensions, raise ValueError
        """
```

[19]:
```
print(dma)
```

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

[20]:
```
dmd = DenseMatrix([ [0,      5.9,    8,    0],
                    [0,        0, 0.1,    5],
                    [9.15, -0.1,    7, 0.1] ])

dma.isclose(dmd, 0.2)
```

[20]:
```
True
```

[21]:
```
dma.isclose(dmd, 0.05)
```

[21]:
```
False
```

7.2.9 Equality

Override == operator:

[22]:
```

dma = DenseMatrix([ [0,6,8,0],
                    [0,0,0,5],
                    [9,0,7,0] ])

dmb = DenseMatrix(3,4, [(0,1,6),(0,2,8),(1,3,5),(2,0,9),(2,2,7)])

dma == dmb
```

[22]:
```
True
```

Notice in this case, since we want in general to check equality *to the last bit,* we will want to check that `other` is actually a `DenseMatrix`, otherwise we will return `False`

[23]:
```
dma == 'ciao'
```

[23]:
```
False
```

Equality with a different shape should just return `False`, without raising exceptions:

[24]:
```
dma == DenseMatrix([[2,4],
                    [9,3]])
```

[24]:
```
False
```

Equality with different numbers:

[25]:
```
dmc = DenseMatrix([ [1,2,3,4],
                    [6,7,8,9],
                    [10,11,12,13] ])
dma == dmc
```

[25]:
```
False
```

7.2.10 Sum

Override + operator so can sum other `Matrix` instances:

> **DO NOT MAKE ASSUMPTIONS ABOUT `other` CLASS!**
>
> In the implementation, use `other` like if it were a generic `Matrix`, it might not always be a `DenseMatrix`. So the only way to access elements will be either by using the brackets operator or by calling `nonzero()`

[26]:
```
dma = DenseMatrix([ [0,6,8,0],
                    [0,0,0,5],
                    [9,0,7,0] ])

dmc = DenseMatrix([ [1,2,3,4],
                    [6,7,8,9],
                    [10,11,12,13] ])
```

[27]:
```
print(dma + dmc)
```

```
DenseMatrix [ [1, 8, 11, 4]
              [6, 7, 8, 14]
              [19, 11, 19, 13] ]
```

Note `dma` and `dmc` where **not** modified:

[28]:
```
print(dma)
```

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

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

```
DenseMatrix [ [1, 2, 3, 4]
              [6, 7, 8, 9]
              [10, 11, 12, 13] ]
```

7.2.11 Multiplication

Override `*` operator so we can multiply by a scalar:

[30]:
```
print(dma * 2)
```

```
DenseMatrix [ [0, 12, 16, 0]
              [0, 0, 0, 10]
              [18, 0, 14, 0] ]
```

> **DO NOT MAKE ASSUMPTIONS ABOUT `other` CLASS!**
>
> In the implementation, use `other` like if it were a generic `Matrix`, it might not always be a `DenseMatrix`. So the only way to access elements will be either by using the brackets operator or by calling `nonzero()`

Again multiplication by a scalar does **not** modify the original:

[31]:
```
print(dma)
```

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

Multiplying a scalar by a matrix should also work:

[32]:
```
print(2 * dma)
```

```
DenseMatrix [ [0, 12, 16, 0]
              [0, 0, 0, 10]
              [18, 0, 14, 0] ]
```

Again original `dma` was not changed:

[33]:
```
print(dma)
```

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

Matrix vector multiplication

Next implement support for multiplication of a matrix by a properly dimensioned vector **on the right**, which must have the same number of rows as the matrix number of columns:

```
     matrix   vector      result

\j    012
i  0  aaa        v0       a00*v0 + a01*v1 + a02*v2
   1  aaa        v1       a10*v0 + a11*v1 + a12*v2
   2  aaa        v2       a20*v0 + a21*v1 + a22*v2
   3  aaa                 a30*v0 + a31*v1 + a32*v2
```

The result will have the same number of rows as the matrix.

DO NOT MAKE ASSUMPTIONS ABOUT `other` CLASS!

In the implementation, use `other` like if it were a generic `Matrix`, it might not always be a `DenseMatrix`. So the only way to access elements will be either by using the brackets operator or by calling `nonzero()`

[34]:
```
dmveca = DenseMatrix([[1],[2],[3],[4]])
```

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

```
DenseMatrix [ [1]
              [2]
              [3]
              [4] ]
```

[36]:
```
print(dma)
```

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

[37]:
```
print(dma * dmveca)
```

```
DenseMatrix [ [36]
              [20]
              [30] ]
```

Multiplication by a vector does **not** modify the originals:

[38]:
```
print(dmveca)
```

```
DenseMatrix [ [1]
              [2]
              [3]
              [4] ]
```

Notice dimensions must be compatible, this should **not** work:

```
>>> print(dma*DenseMatrix([[1],[2]]))

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-bc850efec5f1> in <module>()
----> 1 print(dma*DenseMatrix([[1],[2]]))

TypeError: unsupported operand type(s) for *: 'DenseMatrix' and 'DenseMatrix'
```

Vector matrix multiplication

Next implement support for multiplication of a matrix by a properly dimensioned horizontal vector **on the left**, which must have the same number of columns as the matrix number of rows:

```
  ector matrix result

\j  012   0123 0123
i   vvv 0 aaaa rrrr
```

(continues on next page)

(continued from previous page)

```
        1 aaaa
        2 aaaa


res = a00*v0+a10*v1+a20*v2, a01*v0+a11*v1+a21*v2, a02*v0+a12*v1+a22*v2,␣
↪a03*v0+a13*v1+a23*v2
```

The result will have the same number of columns as the matrix.

> **DO NOT MAKE ASSUMPTIONS ABOUT `other` CLASS!**
>
> In the implementation, use `other` like if it were a generic `Matrix`, it might not always be a `DenseMatrix`. So the only way to access elements will be either by using the brackets operator or by calling `nonzero()`

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

```
DenseMatrix [ [0, 6, 8, 0]
              [0, 0, 0, 5]
              [9, 0, 7, 0] ]
```

[40]:
```
dmvecb = DenseMatrix([[1,2,3]])
```

[41]:
```
print(dmvecb * dma)
```

```
DenseMatrix [ [27, 6, 29, 10] ]
```

Matrix matrix multiplication

This would be the general problem but for the purposes of this challenge, it is **not mandatory** to support multiplication by a matrix, **even if they have compatible dimensions:**

```
>>> print(dma*DenseMatrix([[3,2],
                           [3,7],
                           [2,8],
                           [9,3],]))

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-ed1ad440c418> in <module>()
      2                        [3,7],
      3                        [2,8],
----> 4                        [9,3],]))

TypeError: unsupported operand type(s) for *: 'DenseMatrix' and 'DenseMatrix'
```

7.2.12 SparseMatrix

If you've arrived so far, congratulations, that was quite a journey. But only half of it! We've seen matrices as lists of lists for a long time, let's now see if we can save some space. Many real-world matrices have often most cells set to zero, so it would be convenient to avoid storing in memory those cells, and reserve space only for non-zero valued cells. There are many ways to do it (see Wikipedia[247]), in this challenge we will adopt the simplest which is the so-called Dictionary of Keys[248] (DOK in short). DOK consists of a dictionary that maps (row, column)-pairs to the value of the elements. Elements that are missing from the dictionary are assumed to be zero.

Now we can see the real reason why we supported also a constructor with triplets: if the goal of using a SparseMatrix is to avoid creating huge tables in memory, initializing it with a list of lists would defeat the purpose! In what follows, sometimes we may still use the list of lists initialization, but that would only be for clarity purposes.

7.2.13 Sparse constructors and printing

Create a new file `sparse_matrix.py` and copy inside this with the method definitions you find in `matrix.py` as done before.

```
from matrix import Matrix

class SparseMatrix(Matrix):
    """ A SparseMatrix is a matrix which contains very few non-zero elements.

        In this implementation, storage is modelled as a dictionary of keys (DOK in␣
↪short),
        where each key is a tuple of cell coordinates which maps to a non-zero cell␣
↪value.

        NOTE: *NEVER* store zero-valued cells.
    """
```

Ffollow the same implementation order as before, so start with the constructor `__init__`, `__str__` and `__repr__` overrides. Note this time our class implementation will have a **dictionary** as private `_cells` field.

With sparse matrices we can have a big 30x20 matrix and occupy very little memory:

[42]:
```
sma = SparseMatrix(30,20, [(26,17,3),(24,11,5)])
```

You can save the shape inside a private `_shape` field.

Since shape can be much bigger than atcual data, with `__str__` you must show the internal dictionary **and** the shape:

[43]:
```
print(sma)
```

```
SparseMatrix {
    (24, 11): 5,
    (26, 17): 3
}
shape: (30, 20)
```

`__repr__` will show the triplets constructor:

[44]:
```
repr(sma)
```

[247] https://en.wikipedia.org/wiki/Sparse_matrix
[248] https://en.wikipedia.org/wiki/Sparse_matrix#Dictionary_of_keys_(DOK)

[44]:
```
'SparseMatrix(30,20, [(24, 11, 5), (26, 17, 3)])'
```

[45]:
```
sma
```

[45]:
```
SparseMatrix(30,20, [(24, 11, 5), (26, 17, 3)])
```

7.2.14 Sparse shape

Implement `shape()`:

[46]:
```
sma.shape()    # note the round brackets
```

[46]:
```
(30, 20)
```

7.2.15 Sparse Brackets operator

Override the square bracket operator by reimplementing the method `__getitem__`

> **CAREFUL!**
>
> Even more than before, you have to super-carefully read Python documentation[249]!!
>
> To respect Python documentation, this time you can't rely on built-in errors and features of lists as you probably did in the `DenseMatrix` case, so you will have to write **a lot** of `if` that check for various possible errors (bad types, values, etc) and raise the appropriate errors (`IndexError`, `TypeError`, `KeyError`, etc)

Implement brackets operator:

[47]:
```
sma[24,11]
```

[47]:
```
5
```

[48]:
```
sma[26,17]
```

[48]:
```
3
```

[49]:
```
sma[29,3]    # shouldn't explode
```

[49]:
```
0.0
```

Support negative numbers:

[50]:
```
sma[-6,11]
```

[50]:
```
5
```

[51]:
```
sma[29,-2]
```

[51]:
```
0.0
```

[249] https://docs.python.org/3/reference/datamodel.html#object.__getitem__

7.2.16 Sparse nonzero

This is very important, as it allows iterating the data structure in an efficient way - depending on the shape, trying all the couples `i`, `j` with the square brackets might be prohibitively costly.

[52]:
```
sma.nonzero()
```

[52]:
```
[(24, 11, 5), (26, 17, 3)]
```

7.2.17 Sparse isclose

Be careful to write generic code, and **avoid iterating all possible i,j couples** - you should only look into nonzero ones.

[53]:
```
print(sma)
```

```
SparseMatrix {
    (24, 11): 5,
    (26, 17): 3
}
shape: (30, 20)
```

[54]:
```
sma.isclose(SparseMatrix(30,20, [(24, 11, 4.9), (26, 17, 3.1)]), 0.2)
```

[54]:
```
True
```

[55]:
```
sma.isclose(SparseMatrix(30,20, [(24, 11, 4.9), (26, 17, 3.1)]), 0.05)
```

[55]:
```
False
```

[56]:
```
sma.isclose(SparseMatrix(30,20, [(24, 11, 4.9), (26, 17, 3.1), (13,18,0.5)]), 0.2)
```

[56]:
```
False
```

[57]:
```
sma.isclose(SparseMatrix(30,20, [(24, 11, 4.9), (26, 17, 3.1), (13,18,0.1)]), 0.2)
```

[57]:
```
True
```

[58]:
```
sma.isclose(SparseMatrix(30,20, [(24, 11, 4.9), (26, 17, 3.1), (13,18,-0.1)]), 0.2)
```

[58]:
```
True
```

The interesting part comes when we try `isclose` with *another* class, like `DenseMatrix`. If you wrote general enough code, only assuming `other` is a generic `Matrix` the following should work without particular effort:

[59]:
```
smp = SparseMatrix([[1,     0,    3],
                    [4,     5,    0]])
dmp = DenseMatrix( [[1.1,  0.1, 2.9],
                    [4.05, 4.9, -0.05]])
```

[60]:
```
smp.isclose(dmp, 0.2)
```

[60]:
```
True
```

[61]:
```
smp.isclose(dmp, 0.05)
```

[61]:
```
False
```

[62]:
```
dmp.isclose(smp, 0.2)
```
[62]:
```
True
```

[63]:
```
dmp.isclose(smp, 0.05)
```
[63]:
```
False
```

7.2.18 Sparse equality

Be careful to write generic code, and **avoid iterating all possible i,j couples** - you should only look into nonzero ones.

[64]:
```
sma
```
[64]:
```
SparseMatrix(30,20, [(24, 11, 5), (26, 17, 3)])
```

[65]:
```
smb = SparseMatrix(30,20, [(24, 11, 5), (26, 17, 3)])
```

[66]:
```
sma == sma
```
[66]:
```
True
```

[67]:
```
sma == smb
```
[67]:
```
True
```

[68]:
```
sma == SparseMatrix(30,20, [(24, 11, 5), (26, 17, 123)])
```
[68]:
```
False
```

[69]:
```
sma == SparseMatrix(30,20, [(24, 11, 5), (26, 17, 3), (5,10,6)])
```
[69]:
```
False
```

Equality among different types should always be `False`, even if they descend from the same ancestor:

[70]:
```
SparseMatrix([[1,2],[3,4]]) == DenseMatrix([[1,2],[3,4]])
```
[70]:
```
False
```

[71]:
```
DenseMatrix([[1,2],[3,4]]) == SparseMatrix([[1,2],[3,4]])
```
[71]:
```
False
```

7.2.19 Sparse sum

Be careful to write generic code, and **avoid iterating all possible i,j couples** - you should only look into nonzero ones.

[72]:
```
smq = SparseMatrix([[1,2,0],
                    [4,0,6]])
dmq = DenseMatrix([[0,8,0],
                   [0,7,3]])
```

[73]:
```
print(smq + smq)
```

```
SparseMatrix {
    (0, 1): 4,
    (1, 2): 12,
    (1, 0): 8,
    (0, 0): 2
}
shape: (2, 3)
```

Again the interesting bit is when you try summing among different classes:

[74]:
```
print(smq + dmq)
```

```
SparseMatrix {
    (0, 1): 10,
    (1, 2): 9,
    (1, 0): 4,
    (0, 0): 1,
    (1, 1): 7
}
shape: (2, 3)
```

[75]:
```
print(dmq + smq)
```

```
DenseMatrix [ [1, 10, 0.0]
              [4, 7.0, 9] ]
```

7.2.20 Sparse multiplication

As always, be careful writing general code, and avoid iterating all possible i,j couples - you should only look into nonzero ones.

Sparse matrix vector multiplication

[76]:
```
smq = SparseMatrix([[1,0,3],
                    [4,5,0]])

smvecq = SparseMatrix([[1],
                       [0],
                       [3]])
print(smq * smvecq)
```

```
SparseMatrix {
    (1, 0): 4.0,
    (0, 0): 10
}
shape: (2, 1)
```

Let's try mixing types:

[77]:
```
dmq = DenseMatrix([[1,0,3],
                   [4,5,0]])

dmvecq = DenseMatrix([[1],
                      [0],
```

(continues on next page)

(continued from previous page)

```
                        [3]])
print(smq * dmvecq)
```

```
SparseMatrix {
    (1, 0): 4,
    (0, 0): 10
}
shape: (2, 1)
```

[78]:
```
print(dmq * smvecq)
```

```
DenseMatrix [ [10.0]
              [4.0] ]
```

Sparse vector matrix multiplication

[79]:
```
smvecr = SparseMatrix([[1,2,0]])

smr = SparseMatrix([ [0, 6, 8, 0],
                     [0, 0, 0, 5],
                     [9, 0, 7, 0] ])

print(smvecr * smr)
```

```
SparseMatrix {
    (0, 1): 6,
    (0, 3): 10,
    (0, 2): 8.0
}
shape: (1, 4)
```

Let's mix types:

[80]:
```
dmvecr = DenseMatrix([[1,2,0]])

dmr = DenseMatrix([ [0, 6, 8, 0],
                    [0, 0, 0, 5],
                    [9, 0, 7, 0] ])

print(smvecr * dmr)
```

```
SparseMatrix {
    (0, 1): 6,
    (0, 3): 10,
    (0, 2): 8.0
}
shape: (1, 4)
```

[81]:
```
print(dmvecr * smr)
```

```
DenseMatrix [ [0, 6, 8, 10] ]
```

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

```
[ ]:
```

7.3 Complexity

Browse files online[250]

7.3.1 Introduction

References

- Luca Bianco's Introduction to algorithms slides[251]
- Luca Bianco's Algorithms and complexity slides[252]
- Python DS Chapter 2.6: Algorithm analysis[253]

7.3.2 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[254]:

Operation	Big-O Efficiency	Operation	Big-O Efficiency
index []	$O(1)$	`in` (contains)	$O(n)$
index assignment	$O(1)$	get slice `[x:y]`	$O(k)$
append	$O(1)$	del slice	$O(n)$
pop()	$O(1)$	set slice	$O(n+k)$
pop(i)	$O(n)$	reverse	$O(n)$
extend(lb)	$O(m)$	la + lb	$O(n+m)$
insert(i,item)	$O(n)$	sort	$O(n \log n)$
del operator	$O(n)$	multiply	$O(nk)$
iteration	$O(n)$		

Fast or not?

```
x = ["a", "b", "c"]

x[2]
x[2] = "d"
x.append("d")
x.insert(0, "d")
x[3:5]
x.sort()
```

[250] https://github.com/DavidLeoni/sciprog-ds/tree/master/complexity
[251] https://sciproalgo2020.readthedocs.io/en/latest/slides/Lecture1.pdf
[252] https://sciproalgo2020.readthedocs.io/en/latest/slides/Lecture2.pdf
[253] https://runestone.academy/runestone/books/published/pythonds/AlgorithmAnalysis/toctree.html
[254] http://interactivepython.org/runestone/static/pythonds/AlgorithmAnalysis/Lists.html

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]
```

7.3.3 TODO WILL PUT EXERCISES HERE

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</a>

[4]:
```
# SOLUTION
```

[4]:

[ ]:

7.4 Sorting

7.4.1 Download exercises zip

Browse files online[255]

7.4.2 Introduction

We will see several algorithms for sorting and discuss their time and space complexity along the way.

BEFORE CONTINUING, READ FIRST THE WORKSHEET ON COMPLEXITY[256]

References

- Luca Bianco's Introduction to algorithms slides[257]
- Luca Bianco's Algorithms and complexity slides[258]
- Luca Bianco's Complexity and sorting[259]
- Python DS Chapter 2.6: Algorithm analysis[260]

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.

[255] https://github.com/DavidLeoni/sciprog-ds/tree/master/sorting
[256] https://sciprog.davidleoni.it/complexity-sol.html
[257] https://sciproalgo2020.readthedocs.io/en/latest/slides/Lecture1.pdf
[258] https://sciproalgo2020.readthedocs.io/en/latest/slides/Lecture2.pdf
[259] https://sciproalgo2020.readthedocs.io/en/latest/slides/Lecture3.pdf
[260] https://runestone.academy/runestone/books/published/pythonds/AlgorithmAnalysis/toctree.html

7.4.3 Exercises

7.4.4 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`

[2]:
```
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` ?

VII COMMANDMENT[261]: **You shall also write on paper!**

[261] https://en.softpython.org/commandments.html#VII-COMMANDMENT

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:

VI COMMANDMENT[262] **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, 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)

[262] https://en.softpython.org/commandments.html#VI-COMMANDMENT

- 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[263] 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:

[3]:
```
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:

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
$i=0$	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

[263] https://sciprog.davidleoni.it/testing/testing.html#Error-kinds

- 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`

[4]:
```
def selection_sort(A):
```

(continues on next page)

(continued from previous page)

```
    """ 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[264] 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`.

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
```

7.4.5 2 Insertion sort

Insertion sort is a basic sorting algorithm. This animation[265] 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?

[264] http://interactivepython.org/runestone/static/pythonds/SortSearch/TheSelectionSort.html
[265] https://github.com/DavidLeoni/sciprog-ds/blob/master/sorting/img/insertion-sort-example.gif

- 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
```

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
    """
```

7.4.6 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]`:

```
[5]: [9,7,8][-1]
```

```
[5]: 8
```

Reversing a list

REMEMBER: `.reverse()` method MODIFIES the list it is called on and returns `None` !

[6]:
```
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:

[7]:
```
lst
```

[7]:
```
[8, 7, 9]
```

If you want to reversed version of a list without actually changing it, you can use `reversed` function:

[8]:
```
lst = [9,7,8]
reversed(lst)
```

[8]:
```
<list_reverseiterator at 0x7f2aec1b97f0>
```

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`:

[9]:
```
lst = [9,7,8]
list(reversed(lst))
```

[9]:
```
[8, 7, 9]
```

Notice `lst` itself was not changed:

[10]:
```
lst
```

[10]:
```
[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

[11]:
```

A = [9,7,8]
x = A.pop()
```

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

```
[9, 7]
```

[13]:
```
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`

[14]:
```
lst = [9,5,6,7]
del lst[2]      # internal delete is O(n)
```

[15]:
```
lst
```

[15]:
```
[9, 5, 7]
```

Costly internal pop

You can remove an internal element with `pop(i)`

[16]:
```
lst = [9,5,6,7]

lst.pop(2)  # internal pop is O(n)
```

[16]:
```
6
```

[17]:
```
lst
```

[17]:
```
[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:

[18]:
```
from merge_sort_sol import *

merge1([3,6,9,13,15], [2,4,8,9])
```

[18]:
```
[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
```

(continues on next page)

(continued from previous page)

```
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
```

7.4.7 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):
```

(continues on next page)

(continued from previous page)

```
    """
        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
```

7.4.8 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`

7.4.9 6 SwapArray

> **NOTE: This exercise was given at an exam. Solving it could have been quite easy, if students had just read** the book[266] (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.

[19]:
```
from swap_array_sol import *
```

To create a `SwapArray`, just call it passing a python list:

[20]:
```
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()`

[21]:
```
sarr.get(0)
```

[21]:
```
7
```

[22]:
```
sarr.get(1)
```

[22]:
```
8
```

[23]:
```
sarr.get_last()
```

[23]:
```
6
```

You can know the size in $O(1)$ with `size()` method:

[24]:
```
sarr.size()
```

[24]:
```
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:

[25]:
```
sarr = SwapArray([7,8,6,3])
print(sarr)
```

```
SwapArray: [7, 8, 6, 3]
```

[266] https://runestone.academy/runestone/static/pythonds/SortSearch/sorting.html

[26]:
```
sarr.swap_next(2)
print(sarr)
```

```
SwapArray: [7, 8, 3, 6]
```

[27]:
```
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:

[28]:
```
is_sorted(SwapArray([8,5,6]))
```

[28]:
```
False
```

[29]:
```
is_sorted(SwapArray([5,6,6,8]))
```

[29]:
```
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:

[30]:
```
sarr = SwapArray([7, 9, 6, 5, 8])
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[31]:
```
max_to_right(sarr,4)  # 4 is an *index*
print(sarr)
```

```
SwapArray: [7, 6, 5, 8, 9]
```

[32]:
```
sarr = SwapArray([7, 9, 6, 5, 8])
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[33]:
```
max_to_right(sarr,3)
print(sarr)
```

```
SwapArray: [7, 6, 5, 9, 8]
```

[34]:
```
sarr = SwapArray([7, 9, 6, 5, 8])
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[35]:
```
max_to_right(sarr,1)
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[36]:
```
sarr = SwapArray([7, 9, 6, 5, 8])
print(sarr)
```

```
SwapArray: [7, 9, 6, 5, 8]
```

[37]:
```
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:

[38]:
```
sar = SwapArray([8,4,2,4,2,7,3])
```

[39]:
```
swapsort(sar)
```

[42]:
```
print(sar)
```

```
SwapArray: [2, 2, 3, 4, 4, 7, 8]
```

7.4.10 Challenges

See Sorting challenges[267] worksheet

[ ]:
```

```

7.5 Sorting Challenge

7.5.1 Download exercises zip

Browse files online[268]

[267] https://sciprog.davidleoni.it/sorting/sorting-chal.html
[268] https://github.com/DavidLeoni/sciprog-ds/tree/master/sorting

7.5.2 Crime parade

A bank robbery has been committed, and witnesses gave police the description of a gangster as a series of m features. For us all features are just numbers, you do not need to care about their range:

```
# height weight beard moustache eye color (RGB)
[ 1.70,  80,    0,    1,        190,150,230 ]
```

The police then caught a bunch of n usual suspects and gathered them in the waiting room of the police station. Witnesses will be asked to recognize the actual gangster among a lineup of criminals who are deemed most similar in features to the description. To fill the lineup, the policeman will:

- go to the waiting room and select the criminal most distant from the description
- bring the criminal at the right of the lineup
- repeat until there are no more criminals in the waiting room

Write a function `lineup` which takes a dictionary of gangster nicknames mapped to their features and a suspect description as a list of features. To implement it, create two lists `lineup_names` and `lineup_sims` of **fixed length set equal to the number of criminals.** Fill the lists with empty strings. Then implement the algorithm as a kind of selection sort, where you keep filling the lists cells always writing into the first available cell found from the left, printing at each step what happens as in the example. Note our version would not be an in-place algorithm, as we are creating new lists to host the data.

- you can assume max name length is `9`

a) Implement this support function to calculate distance among two criminals (it's the square root of sum of squared differences):

```
def distance(features1, features2):
    """ Takes two lists, each having m features as floats
        and RETURN  a float which is the vector distance among
        the two features

        - MUST RUN IN O(m) WHERE m IS THE NUMBER OF FEATURES
    """
```

b) implement main `lineup` function:

```
def lineup(waiting_room, description):
    """ RETURN a NEW list of the criminals names sorted by similarity distance with␣
↪description,
                from most distant to least distant
        PRINT all the passages
        MODIFY waiting_room so it's empty at the end
    """
```

DO NOT recalculate distance each time you have to compare suspects with the description, that would be an unacceptable extra $O(nm)$ cost. Instead, save somewhere the distance of each suspect so you can get it fast later

DO NOT change length of lists (so no `.insert`, `.remove()`, `.pop()`, `.append`,`.extend`, ...), you can only set cell values of `lineup_names` and `lineup_sims`!

DO NOT call `min`, `max`, `sum`

DO NOT call `sort()` nor `sorted()` - seems obvious, but during exams there were people who called it in exercises about sorting (they didn't get a good grade)

Testing: `python3 -m unittest sorting_chal_test.TestLineup`

[1]:
```
from sorting_chal_sol import *

waiting_room = {  # height weight beard moustache eye color (RGB)
     'The Saint':   [1.70,    90, 1,    0,        250,100,190],
     'Goliath':     [2.00,   120, 0,    0,        210,230, 30],
     'Revolvers':   [1.80,    70, 1,    1,         40,120, 30],
     'Mammoth':     [1.60,    60, 1,    0,        110,230, 30],
     'Razor':       [1.80,   110, 1,    0,        130,230, 20],
     'Shadow':      [1.60,    70, 1,    0,        190,230,140],
     'Moneybags':   [1.60,    80, 1,    0,        210,230,220],
     'Bomber':      [1.70,   100, 0,    0,        140,110,170],
     'Rifleman':    [1.60,    90, 1,    1,        110, 20, 40],
     'The Ghost':   [1.60,    50, 0,    0,        150, 70, 90]
}

lineup(waiting_room, [1.70,   80, 0,    1,        190,150,230 ])
```

```
Waiting room: Mammoth,Razor,Revolvers,Goliath,The Ghost,Rifleman,Bomber,Shadow,
↪Moneybags,The Saint
Most distant is Revolvers with distance 251.99 , moving him into the lineup

|   251.99|         |         |         |         |         |         |         |    ␣
↪    |         |
|Revolvers|         |         |         |         |         |         |         |    ␣
↪    |         |

Waiting room: Mammoth,Razor,Goliath,The Ghost,Rifleman,Bomber,Shadow,Moneybags,The␣
↪Saint
Most distant is Rifleman with distance 243.93 , moving him into the lineup

|   251.99|   243.93|         |         |         |         |         |         |    ␣
↪    |         |
|Revolvers| Rifleman|         |         |         |         |         |         |    ␣
↪    |         |

Waiting room: Mammoth,Razor,Goliath,The Ghost,Bomber,Shadow,Moneybags,The Saint
Most distant is Razor with distance 234.53 , moving him into the lineup

|   251.99|   243.93|   234.53|         |         |         |         |         |    ␣
↪    |         |
|Revolvers| Rifleman|    Razor|         |         |         |         |         |    ␣
↪    |         |

Waiting room: Mammoth,Goliath,The Ghost,Bomber,Shadow,Moneybags,The Saint
Most distant is Mammoth with distance 230.66 , moving him into the lineup

|   251.99|   243.93|   234.53|   230.66|         |         |         |         |    ␣
↪    |         |
|Revolvers| Rifleman|    Razor|  Mammoth|         |         |         |         |    ␣
↪    |         |

Waiting room: Goliath,The Ghost,Bomber,Shadow,Moneybags,The Saint
Most distant is Goliath with distance 220.00 , moving him into the lineup

|   251.99|   243.93|   234.53|   230.66|   220.00|         |         |         |    ␣
↪    |         |
|Revolvers| Rifleman|    Razor|  Mammoth|  Goliath|         |         |         |    ␣
↪    |         |
```

(continues on next page)

(continued from previous page)

```
Waiting room: The Ghost,Bomber,Shadow,Moneybags,The Saint
Most distant is The Ghost with distance 168.82 , moving him into the lineup

|   251.99|   243.93|   234.53|   230.66|   220.00|   168.82|         |         |
↪    |         |
|Revolvers| Rifleman|    Razor|  Mammoth|  Goliath|The Ghost|         |         |
↪    |         |

Waiting room: Bomber,Shadow,Moneybags,The Saint
Most distant is Shadow with distance 120.84 , moving him into the lineup

|   251.99|   243.93|   234.53|   230.66|   220.00|   168.82|   120.84|         |
↪    |         |
|Revolvers| Rifleman|    Razor|  Mammoth|  Goliath|The Ghost|   Shadow|         |
↪    |         |

Waiting room: Bomber,Moneybags,The Saint
Most distant is Bomber with distance 90.01 , moving him into the lineup

|   251.99|   243.93|   234.53|   230.66|   220.00|   168.82|   120.84|    90.01|
↪    |         |
|Revolvers| Rifleman|    Razor|  Mammoth|  Goliath|The Ghost|   Shadow|   Bomber|
↪    |         |

Waiting room: Moneybags,The Saint
Most distant is The Saint with distance 88.33 , moving him into the lineup

|   251.99|   243.93|   234.53|   230.66|   220.00|   168.82|   120.84|    90.01|
↪88.33|         |
|Revolvers| Rifleman|    Razor|  Mammoth|  Goliath|The Ghost|   Shadow|   Bomber|The
↪Saint|         |

Waiting room: Moneybags
Most distant is Moneybags with distance 83.08 , moving him into the lineup

|   251.99|   243.93|   234.53|   230.66|   220.00|   168.82|   120.84|    90.01|
↪88.33|    83.08|
|Revolvers| Rifleman|    Razor|  Mammoth|  Goliath|The Ghost|   Shadow|   Bomber|The
↪Saint|Moneybags|
```

[1]:
```
['Revolvers',
 'Rifleman',
 'Razor',
 'Mammoth',
 'Goliath',
 'The Ghost',
 'Shadow',
 'Bomber',
 'The Saint',
 'Moneybags']
```

7.5.3 McFat's

The fastfood chain MacFat's recently opened new stores around main attractions of each major Italian city: even if not economically profitable (obviously, you can't compete with Italian food), the flashy store signs will get featured in the thousands pictures turists take each day, and will appear in the many reports tv networks give daily from main squares.

Each store can hold up to `n` people in a queue. McFat's rewards obese clients by assigning them a priority according to their weight, so overweights clients can go past thinner ones. If a client has the same weight as another one, he/she will queue just behind the client who arrived first.

Write a function `mcfats` which shows a queue of a list of clients, where each client is represented as an integer indicating his/her weight. In the function, create two zero-filled lists `arrival_times` and `weigths` of **fixed length set equal to the number of clients.** Then implement the algorithm as a kind of insertion sort, where you keep filling the lists cells from the left, printing at each step what happens as in the example. Note our version would not be an in-place algorithm, as we are creating new lists to host the data.

```
def mcfats(clients):
    """ RETURN a NEW sorted list with the client weights, from smallest to greatest
        PRINT all the passages
        DO *NOT* MODIFY clients
    """
```

- First client arrives at time `1`
- Display zeros as empty spaces

DO NOT change length of lists (so no `.insert`, `.remove()`, `.pop()`, `.append`,`.extend`, ...), you can only set cell values of `arrival_times` and `weigths`!

DO NOT call `min`, `max`, `sum`

DO NOT call `sort()` nor `sorted()` - seems obvious, but during exams there were people who called it in exercises about sorting (they didn't get a good grade)

Testing: `python3 -m unittest sorting_chal_test.TestMcFats`

[2]:
```
from sorting_chal_sol import *
```

[3]:
```
#      9, 8, 7, 6, 5, 4, 3, 2, 1   # arrival time
cs = [80,90,50,60,80,60,80,50,70]
res = mcfats(cs)
print(res)
assert  cs == [80,90,50,60,80,60,80,50,70]  # don't sort original
assert res == [50,50,60,60,70,80,80,80,90]  # return a new sorted version
```

```
weight 70 kg arrived at time 1
will insert in slot = 0
no people to shift

slot:          | 0| 1| 2| 3| 4| 5| 6| 7| 8|
arrival_times:| 1|  |  |  |  |  |  |  |  |
weights:       |70|  |  |  |  |  |  |  |  |

weight 50 kg arrived at time 2
will insert in slot = 0
will shift  1 people starting from slot 0

slot:          | 0| 1| 2| 3| 4| 5| 6| 7| 8|
```

(continues on next page)

(continued from previous page)

```
arrival_times:| 2| 1|  |  |  |  |  |  |
weights:      |50|70|  |  |  |  |  |  |

weight 80 kg arrived at time 3
will insert in slot = 2
no people to shift

slot:         | 0| 1| 2| 3| 4| 5| 6| 7| 8|
arrival_times:| 2| 1| 3|  |  |  |  |  |
weights:      |50|70|80|  |  |  |  |  |

weight 60 kg arrived at time 4
will insert in slot = 1
will shift  2 people starting from slot 1

slot:         | 0| 1| 2| 3| 4| 5| 6| 7| 8|
arrival_times:| 2| 4| 1| 3|  |  |  |  |
weights:      |50|60|70|80|  |  |  |  |

weight 80 kg arrived at time 5
will insert in slot = 3
will shift  1 people starting from slot 3

slot:         | 0| 1| 2| 3| 4| 5| 6| 7| 8|
arrival_times:| 2| 4| 1| 5| 3|  |  |  |
weights:      |50|60|70|80|80|  |  |  |

weight 60 kg arrived at time 6
will insert in slot = 1
will shift  4 people starting from slot 1

slot:         | 0| 1| 2| 3| 4| 5| 6| 7| 8|
arrival_times:| 2| 6| 4| 1| 5| 3|  |  |
weights:      |50|60|60|70|80|80|  |  |

weight 50 kg arrived at time 7
will insert in slot = 0
will shift  6 people starting from slot 0

slot:         | 0| 1| 2| 3| 4| 5| 6| 7| 8|
arrival_times:| 7| 2| 6| 4| 1| 5| 3|  |
weights:      |50|50|60|60|70|80|80|  |

weight 90 kg arrived at time 8
will insert in slot = 7
no people to shift

slot:         | 0| 1| 2| 3| 4| 5| 6| 7| 8|
arrival_times:| 7| 2| 6| 4| 1| 5| 3| 8|
weights:      |50|50|60|60|70|80|80|90|

weight 80 kg arrived at time 9
will insert in slot = 5
will shift  3 people starting from slot 5

slot:         | 0| 1| 2| 3| 4| 5| 6| 7| 8|
arrival_times:| 7| 2| 6| 4| 1| 9| 5| 3| 8|
```

(continues on next page)

(continued from previous page)

```
weights:      |50|50|60|60|70|80|80|80|90|

[50, 50, 60, 60, 70, 80, 80, 80, 90]
```

7.5.4 Partitocracy

A democracy degenerates into a partitocracy when political parties keep splitting the power among themselves without actually allowing citizens to have their say. Parties pretend there is a lively democracy by being very litigious, and keep creating factions this way:

1 - at the beginning there are all the people, each with a number indicating his/her political inclination:

```
We, the people: [6, 3, 5, 7, 5, 7, 2, 9, 1, 4, 8]
```

2 - the people elect a leader - as thoughtful criteria they of course choose the first guy who passes by:

```
- elect our new leader: 6
```

3 - heated arguments around the leader begin: people leaning left of him/her decide to create their own party:

```
- create a new left party: [3, 5, 5, 2, 1, 4]
```

while right-leaning people (or with equal orientation) create their own party:

```
- create a new right party: [7, 7, 9, 8]
```

Naturally, both parties recursively degenerate into their own partitocracies.

4 - the leader feels ignored by the original voters, so in protest creates his/her own party of one person

5 - a new partitocracy is produced, by joining left partitocracy with the one-person party of the leader and the right partitocracy:

```
- join them in a new partitocracy: [[[[[1]], [2]], [3], [[[4]], [5], [[5]]]], [6],␣
↪[[7], [[7], [[[8]], [9]]]]]
```

The result is of course a mess of burocratic structures. If we `pprint` with `width=10` the result, the subdivision is slighlty clearer:

```
[[[[[1]],     <---- left partitocracy
   [2]],
  [3],
  [[[4]],
   [5],
   [[5]]]],
 [6],         <---- leader
 [[7],        <---- right partitocracy
  [[7],
   [[[8]],
    [9]]]]]
```

Implement the following function, which is a kind of quick sort, although this version does not modify input data so it is not *in-place:*

```
def partitocracy(people, level=0):
    """Takes a list of integers and:

       PRINT the process, starting each line  with a number
             of spaces proportional to recursion level parameter
       RETURN a partitocracy as a NEW list of nested lists

       - a recursive implementation is acceptable
    """
```

MUST execute in $O(n \log n)$ where n is the size of the input list

DO NOT modify the input list

ONLY USE `.append` method to change lists size

DO NOT append empty lists to a partitocracy

DO NOT call `min`, `max`, `sum`

DO NOT call `sort()` nor `sorted()` - seems obvious, but during exams there were people who called it in exercises about sorting (they didn't get a good grade)

Testing: `python3 -m unittest sorting_chal_test.TestPartitocracy`

[4]:
```
from sorting_chal_sol import partitocracy

italy = [6,3,5,7,5,7,2,9,1,4,8]
res = partitocracy(italy)
from pprint import pprint
print()
print()
print('Result pprint is:')
print()
pprint(res, width=10)
```

```
We, the people: [6, 3, 5, 7, 5, 7, 2, 9, 1, 4, 8]
- elect our new leader: 6
- create a new left party: [3, 5, 5, 2, 1, 4]
    We, the people: [3, 5, 5, 2, 1, 4]
    - elect our new leader: 3
    - create a new left party: [2, 1]
        We, the people: [2, 1]
        - elect our new leader: 2
        - create a new left party: [1]
            We, the people: [1]
            - elect our new leader: 1
            - can't create a left party
            - can't create a right party
            - join them in a new partitocracy: [[1]]
        - can't create a right party
        - join them in a new partitocracy: [[[1]], [2]]
    - create a new right party: [5, 5, 4]
        We, the people: [5, 5, 4]
        - elect our new leader: 5
        - create a new left party: [4]
            We, the people: [4]
            - elect our new leader: 4
            - can't create a left party
            - can't create a right party
```

(continues on next page)

(continued from previous page)

```
            - join them in a new partitocracy: [[4]]
        - create a new right party: [5]
            We, the people: [5]
            - elect our new leader: 5
            - can't create a left party
            - can't create a right party
            - join them in a new partitocracy: [[5]]
        - join them in a new partitocracy: [[[4]], [5], [[5]]]
    - join them in a new partitocracy: [[[[1]], [2]], [3], [[[4]], [5], [[5]]]]
- create a new right party: [7, 7, 9, 8]
    We, the people: [7, 7, 9, 8]
    - elect our new leader: 7
    - can't create a left party
    - create a new right party: [7, 9, 8]
        We, the people: [7, 9, 8]
        - elect our new leader: 7
        - can't create a left party
        - create a new right party: [9, 8]
            We, the people: [9, 8]
            - elect our new leader: 9
            - create a new left party: [8]
                We, the people: [8]
                - elect our new leader: 8
                - can't create a left party
                - can't create a right party
                - join them in a new partitocracy: [[8]]
            - can't create a right party
            - join them in a new partitocracy: [[[8]], [9]]
        - join them in a new partitocracy: [[7], [[[8]], [9]]]
    - join them in a new partitocracy: [[7], [[7], [[[8]], [9]]]]
- join them in a new partitocracy: [[[[[1]], [2]], [3], [[[4]], [5], [[5]]]], [6],␣
↪[[7], [[7], [[[8]], [9]]]]]


Result pprint is:

[[[[[1]],
   [2]],
  [3],
  [[[4]],
   [5],
   [[5]]]],
 [6],
 [[7],
  [[7],
   [[[8]],
    [9]]]]]
```

[ ]:

[ ]:

7.6 Linked lists

7.6.1 Download exercises zip

Browse files online[269]

7.6.2 0 Introduction

In these exercises, you will be implementing several versions of a `LinkedList`, improving its performances with each new version.

References

- theory slides[270](Monodirectional list)
- LinkedList Abstract Data Type[271] on the book
- Implementing LinkedListLinkedLists[272] 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.

[269] https://github.com/DavidLeoni/sciprog-ds/tree/master/linked-lists
[270] https://sciproalgo2019.readthedocs.io/slides/Lecture4.pdf
[271] http://interactivepython.org/runestone/static/pythonds/BasicDS/TheUnorderedListAbstractDataType.html
[272] http://interactivepython.org/runestone/static/pythonds/BasicDS/ImplementinganUnorderedListLinkedLists.html

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:

[5]:
```
ll
```

[5]:
```
<linked_list_sol.LinkedList at 0x7f89885b3438>
```

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 `Nodes`:

[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 $O(n)$.

7.6.3 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[273] 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 *never* supposed to 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 !)

VII COMMANDMENT[274]: **You shall also draw lists on paper**

Helps a lot avoiding mistakes!

VIII COMMANDMENT[275] **You shall never ever reassing `self` !**

Never write horrors such as this:

[273] https://www.python.org/dev/peps/pep-0008/#id45
[274] https://en.softpython.org/commandments.html#VII-COMMANDMENT
[275] https://en.softpython.org/commandments.html#VIII-COMMANDMENT

[12]:
```
class LinkedList:
    def my_method(self):
        self = {'my_field':666}     # SIN
```

VI COMMANDMENT[276]: **You shall use `return` command only if you see written RETURN in function description!**

If there is no `return` in the description, it 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.

7.6.4 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[277]) 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.

[276] https://en.softpython.org/commandments.html#VI-COMMANDMENT
[277] https://git-scm.com

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')
    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
```

7.6.5 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
```

7.6.6 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!

7.6.7 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`

7.6.8 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`

6.3 Slice

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 cloning_test.SliceTest`

Example:

[17]:
```
from cloning_sol import *
```

[18]:
```
la = LinkedList()
la.add('g')
la.add('f')
la.add('e')
la.add('d')
la.add('c')
la.add('b')
la.add('a')
```

[19]:
```
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:

[20]:
```
lb = la.slice(2,5)
```

[21]:
```
print(lb)
```

```
LinkedList: c,d,e
```

Note original `LinkedList` is still intact:

[22]:
```
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`:

[23]:
```
print(la.slice(5,3))
```

```
LinkedList:
```

If `start` is greater than available nodes, you get an empty `LinkedList`:

[24]:
```
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:

[25]:
```
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)

---------------------------------------------------------------------------
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
```

7.6.9 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: **

[27]:
```
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
```

[28]:
```
print(ul.occurrences('a'))
```

```
2
```

[29]:
```
print(ul.occurrences('c'))
```

```
1
```

[30]:
```
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`

[31]:
```
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
```

[32]:
```
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`

[33]:
```
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`

7.14 couple_sort

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 more_test.CoupleSortTest`

[34]:
```
from more_sol import *
from more_test import to_ll
```

[35]:
```
ll = to_ll([4,3,5,2,6,7,6,3,2,4,5,3,2])
```

[36]:
```
print(ll)
```

```
LinkedList: 4,3,5,2,6,7,6,3,2,4,5,3,2
```

[37]:
```
ll.couple_sort()
```

[38]:
```
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.

7.6.10 8 Last exercises

For these exercises, you also need to take care about attributes `size` and `_last` pointer.

8.1 rotate

⊛⊛ Now implement this method:

> Remember to update also `_last` and `_size` attributes !!!

```
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
```

(continues on next page)

(continued from previous page)

```
        - WARNING: DO *NOT* swap node data or create nodes, I want you to
                   change existing node links !!
    """
```

Testing: `python3 -m unittest last_test.RotateTest`

Example:

[40]:
```
from last_sol import *
```

[41]:
```
ll = LinkedList()
ll.add('d')
ll.add('c')
ll.add('b')
ll.add('a')
print(ll)
```

```
LinkedList: a,b,c,d
```

[42]:
```
ll.rotate()
```

[43]:
```
print(ll)
```

```
LinkedList: d,a,b,c
```

8.2 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 last_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:

[44]:
```
ll = LinkedList()
ll.add('h')
ll.add('g')
ll.add('f')
ll.add('e')
ll.add('d')
ll.add('c')
ll.add('b')
ll.add('a')
print(ll)
```

```
LinkedList: a,b,c,d,e,f,g,h
```

[45]:
```
ll.rotaten(0)  # changes nothing
```

[46]:
```
print(ll)
```

```
LinkedList: a,b,c,d,e,f,g,h
```

[47]:
```
ll.rotaten(3)
```

[48]:
```
print(ll)
```

```
LinkedList: f,g,h,a,b,c,d,e
```

[49]:
```
ll.rotaten(8)  # changes nothing
```

[50]:
```
print(ll)
```

```
LinkedList: f,g,h,a,b,c,d,e
```

[51]:
```
ll.rotaten(5)
```

[52]:
```
print(ll)
```

```
LinkedList: a,b,c,d,e,f,g,h
```

[53]:
```
ll.rotaten(11)  # 11 = 8 + 3 , only rotates 3 nodes
```

[54]:
```
print(ll)
```

```
LinkedList: f,g,h,a,b,c,d,e
```

8.3 plus_one

Supposing the list can only hold digits from `0` to `9` as node data, implement this method:

```
def plus_one(self):
    """ MODIFIES the 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:
```

(continues on next page)

(continued from previous page)

```
            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 last_test.PlusOneTest`

Example:

[55]:
```
from last_sol import *

dl = LinkedList()

dl.add(9)
dl.add(9)
dl.add(7)
dl.add(3)
dl.add(9)
dl.add(2)

print(dl)
```

```
LinkedList: 2,9,3,7,9,9
```

[56]:
```
dl.last()
```

[56]:
```
9
```

[57]:
```
dl.plus_one()
```

[58]:
```
print(dl)
```

```
LinkedList: 2,9,3,8,0,0
```

7.7 Stacks

7.7.1 Download exercises zip

Browse files online[278]

[278] https://github.com/DavidLeoni/sciprog-ds/tree/master/stacks

7.7.2 0. Introduction

References

- theory: https://sciproalgo2019.readthedocs.io/slides/Lecture5.pdf
- stack definition on the book[279]

and following sections :

- Stack Abstract Data Type[280]
- Implementing a Stack in Python[281]
- Simple Balanced Parenthesis[282]
- 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

7.7.3 1. CappedStack

You will try to implement a so called *capped* stack, which has a limit called *cap* over which elements are discarded.

[279] http://interactivepython.org/runestone/static/pythonds/BasicDS/WhatisaStack.html
[280] http://interactivepython.org/runestone/static/pythonds/BasicDS/TheStackAbstractDataType.html
[281] http://interactivepython.org/runestone/static/pythonds/BasicDS/ImplementingaStackinPython.html
[282] http://interactivepython.org/runestone/static/pythonds/BasicDS/SimpleBalancedParentheses.html

- 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`

7.7.4 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`

7.7.5 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

7.7.6 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]
```

7.7.7 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`

7.7.8 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`

[ ]:

```

```

7.8 Queues

7.8.1 Download exercises zip

Browse files online[283]

7.8.2 Introduction

In these exercises, you will be implementing several queues.

- See theory slides[284]
- See Queue Abstract Data Type[285] on the book
- See Implementing a Queue in Python[286] 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.

[283] https://github.com/DavidLeoni/sciprog-ds/tree/master/queues
[284] https://sciproalgo2019.readthedocs.io/slides/Lecture5.pdf
[285] http://interactivepython.org/runestone/static/pythonds/BasicDS/WhatIsaQueue.html
[286] http://interactivepython.org/runestone/static/pythonds/BasicDS/ImplementingaQueueinPython.html

7.8.3 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`

7.8.4 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:

Stack and queues | Queue

Queue based on circular array – Pseudocode

QUEUE

```
A                    % Elements
size                 % Current size
head                 % Head of the queue
cap                  % Maximum size

Queue(self, dim)
    self.A ← new int[0 ... dim − 1]
    self.cap ← dim
    self.head ← 0
    self.size ← 0

top()
    if size > 0 then
        return A[head]

dequeue()
    if size > 0 then
        temp ← A[head]
        head ← (head + 1)%cap
        size ← size − 1
        return temp

enqueue(v)
    if size < cap then
        A[(head + size)%cap] ← v
        size ← size + 1

size()
    return size

isEmpty()
    return size = 0
```

Alberto Montresor (UniTN) | SP - Data structure | 2017/11/21 | 62 / 69

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
```

7.8.5 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[287]) 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 make it fast

Improve `enqueue` adn `dequeue` so they work both 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

SOLUTION: This exercise was proposed during this exam[288]

[287] https://git-scm.com
[288] https://sciprog.davidleoni.it/exams/2020-02-10/solutions/exam-2020-02-10-sol.html#B2-ItalianQueue-v2

7.8.6 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`

7.8.7 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 : []
```

7.8.8 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`

7.8.9 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]:
```
[]
```

7.8.10 8. 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:

[90]:
```
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
```

(continues on next page)

(continued from previous page)

```
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)
```

[91]:
```
from office_queue_sol import *
SERVICES = { 'x':5,    # minutes
             'y':20,
             'z':30
           }


oq = OfficeQueue(SERVICES)
print(oq)
```

```
OfficeQueue:
```

[92]:
```
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')
```

[93]:
```
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
```

[94]:
```
oq.size()
```

[94]:
```
12
```

Total wait time can be accessed from outside with the method `wait_time()`:

[95]:
```
oq.wait_time()
```

[95]:
```
155
```

> **ATTENTION**: you only need to implement the methods `time_to_service` and `split`
>
> **DO NOT** touch other methods.

8.1 - 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
```

[96]:
```
SERVICES = { 'x':5,    # minutes
             'y':20,
             'z':30
           }

oq = OfficeQueue(SERVICES)
print(oq)
```

```
OfficeQueue:
```

[97]:
```
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:

[98]:
```
oq.time_to_service()
```

[98]:
```
{'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)

[99]:
```
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
```

[100]:
```
print(oq.wait_time())
```

```
25
```

[101]:
```
oq.time_to_service()    # note z is set to total wait time
```

[101]:
```
{'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
```

(continues on next page)

(continued from previous page)

```
        - MUST run in O(n) where n is the size of the queue.
    """
```

Testing: `python3 -m unittest office_queue_test.TestTimeToService`

8.2 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


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`

[ ]:

```

```

7.9 Binary Trees

7.9.1 Download exercises zip

(before editing read whole introduction section)

Browse files online[289]

7.9.2 0. Introduction

Here we only deal with binary trees, for generic trees see separate notebook[290].

What to do

- unzip exercises in a folder, you should get something like this:

```
trees
   bin-trees.ipynb
   gen-trees.ipynb
   bin_tree_test.py
   bin_tree.py
   bin_tree_sol.py
   gen_tree_test.py
   gen_tree.py
   gen_tree_sol.py
   jupman.py
   sciprog.py
```

[289] https://github.com/DavidLeoni/sciprog-ds/tree/master/trees
[290] https://sciprog.davidleoni.it/trees/gen-trees.html

- 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.

7.9.3 0. Introduction

0.1 References

See

- Luca Bianco theory here[291]
- Trees on the book[292]
  - In particular, Vocabulary and definitions[293]

0.2 Terminology - relations

[291] https://sciproalgo2020.readthedocs.io/slides/Lecture6.pdf
[292] https://interactivepython.org/runestone/static/pythonds/Trees/toctree.html
[293] https://interactivepython.org/runestone/static/pythonds/Trees/VocabularyandDefinitions.html

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

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.

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')

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 *
```

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.

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'
```

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
 └
```

7.9.4 1. Insertions

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`

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`

7.9.5 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.

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
```

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`

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
  └
```

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
```

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'
```

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))'
```

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`

2.8 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:

[30]:
```
t = bt(3, bt(3), bt(3, bt(3, bt(3, None, bt(3)))))
print(t)
```

```
3
├3
└3
 ├3
 │├3
 │└
 │└3
 │└
 └
```

[31]:
```
t.univalued_rec()
```

[31]:
```
True
```

[32]:
```
t = bt(2, bt(3), bt(6, bt(3, bt(3, None, bt(3)))))
print(t)
```

```
2
├3
└6
 ├3
 │├3
 │└
 └
```

[33]:
```
t.univalued_rec()
```

[33]:
```
False
```

2.9 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`

2.10 sum_leaves_rec

We will now go looking for leaves, that is, nodes with no children:

⊛⊛ 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:

[34]:
```
t = bt(3,
        bt(10,
                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
```

[34]:
```
18
```

2.11 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.

```
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:

[35]:
```
from bin_tree_sol import *
from bin_tree_test import bt
```

[36]:
```
tasks = bt('i',
                bt('d',
                        bt('b',
                                bt('a')),
                        bt('c')),
                bt('h',
                        bt('f',
                                None,
                                bt('e')),
                        bt('g')))
```

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

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

(continues on next page)

(continued from previous page)

```
| └e
└g
```

[38]:
```
tasks.schedule_rec()
```

[38]:
```
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

7.9.6 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.

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`

3.3 height_stack

The idea of this function is not that different from the Tasks do_level exercise[294] 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`

[294] https://sciprog.davidleoni.it/stacks/stacks.html#5.2-do_level

3.3 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:

[39]:
```
t = bt('a',
            bt('b',
                    bt('c'),
                    bt('d',
                            None,
                            bt('e'))),
            bt('f',
                    bt('g',
                            bt('h')),
                    bt('i')))
t.leaves_stack()
```

[39]:
```
['c', 'e', 'h', 'i']
```

3.4 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).

7.9.7 4. Modifying the tree

4.1 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:

[40]:
```
t = bt(7)
print(t)
```

```
7
```

[41]:
```
t.bin_insert_rec(3)
print(t)
```

```
7
├3
└
```

[42]:
```
t.bin_insert_rec(6)
print(t)
```

```
7
├3
│├
│└6
└
```

[43]:
```
t.bin_insert_rec(2)
print(t)
```

```
7
├3
│├2
│└6
└
```

[44]:
```
t.bin_insert_rec(12)
print(t)
```

```
7
├3
│├2
│└6
└12
```

[45]:
```
t.bin_insert_rec(14)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├
 └14
```

[46]:
```
t.bin_insert_rec(13)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├
 └14
  ├13
  └
```

[47]:
```
t.bin_insert_rec(8)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├8
 └14
  ├13
  └
```

[48]:
```
t.bin_insert_rec(11)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├8
 │├
 │└11
 └14
```

(continues on next page)

(continued from previous page)

```
  ├13
  └
```

[49]:
```
t.bin_insert_rec(9)
print(t)
```

```
7
├3
│├2
│└6
└12
 ├8
 │├
 │└11
 │ ├9
 │ └
 └14
  ├13
  └
```

4.2 add_row

```
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:

[50]:
```
from bin_tree_sol import *
from bin_tree_test import bt

t = bt('a',
            bt('b',
                    bt('d'),
                    bt('e')),
            bt('c',
                    bt('f'),
                    bt('g')))

print(t)
```

```
a
├b
│├d
│└e
└c
 ├f
 └g
```

[51]:
```
t.add_row(['h','i','j','k','l'])
```

[52]:
```
print(t)
```

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

4.3 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:

[53]:
```
from bin_tree_sol import *
from bin_tree_test import bt
```

[54]:
```
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'))))
```

[55]:
```
print(t)
```

```
a
├b
│├z
│└c
│ ├d
│ └z
│  ├
│  └e
└z
 ├f
 └z
  ├
  └g
```

[56]:
```
t.prune_rec('z')
```

[57]:
```
print(t)
```

```
a
├b
│├
│└c
│ ├d
│ └
└
```

[58]:
```
t.prune_rec('c')
```

[59]:
```
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 !
```

7.9.8 Further resources

See Trees exercises[295] on LeetCode (sort by easy difficulty), for example:

- univalued[296]
- same_tree[297] (give recursive solution)
- Sum range of BST[298]

[ ]:

```

```

7.10 Generic Trees

7.10.1 Download exercises zip

(before editing read whole introduction sections 0.x)

Browse files online[299]

7.10.2 0. Introduction

Before tackling generic trees please see first binary trees[300]

What to do

- unzip exercises in a folder, you should get something like this:

```
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
```

(continues on next page)

[295] https://leetcode.com/tag/tree/
[296] https://leetcode.com/problems/univalued-binary-tree/
[297] https://leetcode.com/problems/same-tree/
[298] https://leetcode.com/problems/range-sum-of-bst/
[299] https://github.com/DavidLeoni/sciprog-ds/tree/master/trees
[300] https://sciprog.davidleoni.it/trees/bin-trees.html

(continued from previous page)

```
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.

7.10.3 0. Introduction

0.1 References

See

- Luca Bianco Generic Tree theory[301]
- Trees on the book[302]
  - In particular, Vocabulary and definitions[303]

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.

[301] https://sciproalgo2020.readthedocs.io/slides/Lecture6.pdf
[302] https://interactivepython.org/runestone/static/pythonds/Trees/toctree.html
[303] https://interactivepython.org/runestone/static/pythonds/Trees/VocabularyandDefinitions.html

- 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.

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.

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')

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:

[2]:
```
from gen_tree_sol import *
```

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:

[3]:
```
tblah = GenericTree("blah")
tn = GenericTree(5)
```

Note that with the provided constructor you can't pass children.

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:

[4]:
```

ta = GenericTree('a')
print(ta)    # 'a' is the root
```

```
a
```

[5]:
```
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
```

[6]:
```
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
```

[7]:
```
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
```

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.
```

[8]:
```
# 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
```

[9]:
```
# 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
```

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:

[10]:
```
# 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
```

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
```

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.

7.10.4 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 !

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 ?

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`

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:

[11]:
```
tb = gt('b')
ta = gt('a', tb, gt('c'))
print(ta)
```

```
a
├b
└c
```

[12]:
```
tx = gt('x', gt('y'))
print(tx)
```

```
x
└y
```

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

1.4 insert_siblings

Testing: `python3 -m unittest tree_test.InsertSiblingsTest`

1.5 detach_child

QUESTION: does a detached child have still any parent or sibling ?

Testing: `python3 -m unittest tree_test.DetachChildTest`

1.6 detach_sibling

Testing: `python3 -m unittest tree_test.DetachSiblingTest`

1.7 detach

Testing: `python3 -m unittest tree_test.DetachTest`

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

7.10.5 2 Implement more complex functions

After you understood well and implemented the previous methods, you can continue with the following ones:

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:

[14]:
```
ta = gt('a', gt('b', gt('c')))
print(ta)
```

```
a
└b
 └c
```

[15]:
```
print(ta.grandchildren())
```

```
['c']
```

[16]:
```
ta = gt('a', gt('b'))
print(ta)
```

```
a
└b
```

[17]:
```
print(ta.grandchildren())
```

```
[]
```

[18]:
```
ta = gt('a', gt('b', gt('c'), gt('d')), gt('e', gt('f')) )
print(ta)
```

```
a
├b
│ ├c
│ └d
└e
  └f
```

[19]:
```
print(ta.grandchildren())
```

```
['c', 'd', 'f']
```

2.2 Zig Zag

Here you will be visiting a generic tree in various ways.

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']
```

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']
```

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']
```

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:

[20]:
```
td = gt('d')
tb = gt('b')
ta = gt('a', tb,  gt('c', td), gt('e'))
print(ta)
```

```
a
├b
├c
│└d
└e
```

[21]:
```
print(td.uncles())
```

```
['b', 'e']
```

[22]:
```
print(tb.uncles())
```

```
[]
```

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

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`

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`

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`

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:

[23]:
```
from gen_tree_test import gt
from gen_tree_sol import *
```

[24]:
```
t  = gt('a',
            gt('b',
                    gt('e',
                            gt('f'),
                            gt('g',
                                    gt('i')),
                    gt('h')),
            gt('c'),
            gt('d')))
```

[25]:
```
print(t)
```

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

[26]:
```
t.fill_left(['x','y'])
```

[27]:
```
print(t)
```

```
a
└x
 ├y
 │├f
 │├g
 │└i
 │└h
 ├c
 └d
```

[28]:
```
t.fill_left(['W','V','T'])
print(t)
```

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

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`

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:

[29]:
```
from gen_tree_test import gt
```

[30]:
```
# 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'])
```

[30]:
```
True
```

[31]:
```
ta.is_triangle(['b','c','a'])
```

[31]:
```
False
```

[32]:
```
tb.is_triangle(['b','d','e'])
```

[32]:
```
False
```

[33]:
```
tg.is_triangle(['g','h','i'])
```

[33]:
```
True
```

[34]:
```
tg.is_triangle(['g','i','h'])
```

[34]:
```
False
```

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:

[35]:
```
# 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'])
```

[35]:
```
True
```

[36]:
```
ta.has_triangle(['a','c','b'])
```

[36]:
```
False
```

[37]:
```
ta.has_triangle(['b','c','a'])
```

[37]:
```
False
```

[38]:
```
tb.is_triangle(['b','d','e'])
```

[38]:
```
False
```

[39]:
```
tg.has_triangle(['g','h','i'])
```

[39]:
```
True
```

[40]:
```
tc.has_triangle(['g','h','i'])  # check recursion
```

[40]:
```
True
```

[41]:
```
ta.has_triangle(['g','h','i'])  # check recursion
```

[41]:
```
True
```

[ ]:
```

```

7.11 Graph algorithms

7.11.1 Download exercises zip

(before editing read whole introduction section 0.x)

Browse files online[304]

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
```

[304] 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.

7.11.2 Introduction

0.1 Graph theory

In short, a graph is a set of vertices linked by edges.

Longer version:

- Luca Bianco theory[305]
- Graphs on the book[306]
  - In particular, see Vocabulary and definitions[307]

[305] https://sciproalgo2019.readthedocs.io/slides/Lecture7.pdf
[306] https://interactivepython.org/runestone/static/pythonds/Graphs/toctree.html
[307] 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[308], you can also use them for visualizating graphs. If you have huge graphs to process you might consider big data tools like Spark GraphX[309] which is programmable in Python.

[308] https://networkx.github.io/
[309] 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 ?

7.11.3 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: []
```

7.11.4 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`

7.11.5 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[310]

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)
```

[310] 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 .

[ ]:

```

```

7.12 Changelog

Scientific Programming Data Science Lab

https://sciprog.davidleoni.it

7.12.1 1.0 September 2020

- upgraded to Jupman 3, changed folder structure
- moved webste to sciprog.davidleoni.it
- building with Github Actions

7.12.2 0.1, September 2018

Site is born

CHAPTER

EIGHT

INDEX