Sciprog DS Lab

Enter the Matrix

David Leoni

Jun 13, 2022

Copyright © 2022 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

Data Science Master @University of Trento - AA 2021/22

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

DS Labs:

- Thursdays 15.30-17.30 room B106
- Fridays 17.30-19.30 room A207

This year practicals will take place in person. This first part of the course will run from Thursday, September 23rd

Tutoring: by Gabriele Masina (gabriele.masina (guess what) studenti.unitn.it)

starting from Monday 15 november until Wednesday 15 december (included):

- Mondays: 9:30-11:30 A202
- Wednesdays: 9:30-11:30 A202

Complete timetable:

- Part A: Andrea Passerini's course site[3]
- Part B: Erik Dassi's course material is on Moodle[4]

Moodle: In the Moodle page of the course[5] you can find announcements and your repl links.

[1] http://www.davidleoni.it
[2] https://creativecommons.org/licenses/by/4.0/
[3] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/
[4] https://didatticaonline.unitn.it/dol/course/view.php?id=32585
[5] https://didatticaonline.unitn.it/dol/course/view.php?id=32585

Lab slides

News

Wed 08, Jun 2022: Published *exam solutions*

Wed 09, Feb 2022: Published *exam solutions*

Wed 12, Jan 2022: Published *exam solutions*

Thu 16, Dec 2021: Published *exam solutions*

Fri 12, Nov 2021: Published *exam solutions*

- I fixed the expected output and tests for cartels

5 November 2021: Published *midterm simulation solutions*:

- I fixed the tests
- to display filled nodes you also need to add the property `style='filled'`

24 Friday September lab is cancelled. Regular meetings will start again from Thursday 30 September

6 September 2021: Published *exam solutions*

12 July 2021: Published *exam solutions*

11 June 2021: Published *exam solutions*

12 September 2020:

- Moved DS Lab website to sciprog.davidleoni.it[6]
- Simplified exercises structure

Old news

[6] https://sciprog.davidleoni.it

CHAPTER

ONE

OVERVIEW

1.1 Office hours

To schedule a meeting, see here[7]

1.2 References

1.2.1 Part A References

- Part A Theory slides[8] by Andrea Passerini
- See References on SoftPython website[9]

1.2.2 Part B References page

1.2.3 Editors

- Visual Studio Code[10]: the course official editor.
- Spyder[11]: Seems like a fine and simple editor
- PyCharme Community Edition[12]
- Jupyter Notebook[13]: Nice environment to execute Python commands and display results like graphs. Allows to include documentation in Markdown format
- JupyterLab[14] : next and much better version of Jupyter, although as of Sept 2018 is still in beta
- PythonTutor[15], a visual virtual machine (*very useful*! can also be found in examples inside the book!)

[7] http://www.davidleoni.it/office-hours
[8] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/
[9] http://en.softpython.org/references.html
[10] https://code.visualstudio.com/
[11] https://pythonhosted.org/spyder/
[12] https://www.jetbrains.com/pycharm/download/#section=linux
[13] http://jupyter.org
[14] https://github.com/jupyterlab/jupyterlab
[15] http://www.pythontutor.com/visualize.html#mode=edit

1.3 Exams

1.3.1 Past exams

- *Past exams page*

1.3.2 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 :-)

Exams are open book: You will only be given online access to this documentation (you **can't** bring printed notes):

- Sciprog lab website[16]
- Andrea Passerini slides[17] and Erik Dassi slides
- Python 3 documentation[18]
  - In particular, Unittest docs[19]
  - If you need to look up some Python function, please start today learning how to search documentation on Python website.
- Part A: Think Python[20] book
- Part B: Problem Solving with Algorithms and Data Structures using Python[21] book

1.3.3 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 *References*

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 ⊛⊛⊛.

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:

[16] https://sciprog.davidleoni.it
[17] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/
[18] https://docs.python.org/3/
[19] https://docs.python.org/3/library/unittest.html
[20] https://runestone.academy/runestone/static/thinkcspy/index.html
[21] https://runestone.academy/runestone/static/pythonds/index.html

- 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.

1.3.4 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.

1.3.5 Exams FAQ

First and foremost, I'm not the boss here, please refer to exam rules explained by Andrea Passerini[22] slides.

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

Can I have additional retake just for me?

No way.

Can I have additional oral to increase the grade?

No way.

I have $\pi + \sqrt{7}$ 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*

[22] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/00-introcorso.pdf

1.3.6 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)
```

1.3.7 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

1.3.8 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)
```

1.4 Acknowledgements

This website and related courses were funded mainly by Department of Information Engineering and Computer Science (DISI)[23], University of Trento, and also Mathematics[24] and CIBIO[25] 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[26] Basically, you can freely redistribute and modify the content, just remember to cite Universit of Trento and the present author.

Technical notes: all website pages are easily modifiable Jupyter notebooks, that were converted to web pages using NBSphinx[27] using Jupman[28] template. Text sources are on Github: github.com/DavidLeoni/sciprog-ds[29]

[23] https://www.disi.unitn.it
[24] https://www.maths.unitn.it/en
[25] https://www.cibio.unitn.it/
[26] https://creativecommons.org/licenses/by/4.0/deed.en
[27] https://nbsphinx.readthedocs.io
[28] https://github.com/DavidLeoni/jupman
[29] 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

> **NOTICE: this part of the exam was ported to** softpython[30] **website**
>
> There you can find a more curated version (notice it may be longer than here)

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[31] and do read Falsehoods Programmers Believe About Phone Numbers[32]

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":

[30] https://en.softpython.org/projects/phone-calls/phone-calls-sol.html
[31] https://github.com/daviddrysdale/python-phonenumbers
[32] https://github.com/googlei18n/libphonenumber/blob/master/FALSEHOODS.md

```
+39 0461 123456
+390461123456
0039 0461 123456
00390461123456
```

These are canonicalized as the following:

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

REMEMBER: strings are immutable !!!!!

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

[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[33]

You should obtain something like this:

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

[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]
```

(continues on next page)

(continued from previous page)

```
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[34]

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

A1 union

⊛⊛ When we talk about the *union* of two graphs, we intend the graph having union of verteces of both graphs and having as edges the union of edges of both graphs. In this exercise, we have two graphs as list of lists with boolean edges. To simplify we suppose they have the same vertices but possibly different edges, and we want to calculate the union as a new graph.

For example, if we have a graph `ma` like this:

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

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

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

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
    """


    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 !")
```

(continues on next page)

(continued from previous page)

```
except ValueError:
    "test passed"



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

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

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

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

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

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


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

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

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

        ]
mb5 =  [
```

(continues on next page)

(continued from previous page)

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

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

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

      ]

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

ma7 =  [
            [True, False, False],
            [False, True, False],
            [True, False, False]
      ]

mb7 =  [
            [True, True, False],
            [False, False, True],
            [False, True, False]

      ]

assert union(ma7, mb7) == [
                             [True, True, False],
                             [False, True, True],
                             [True, True, False]

                          ]
```

[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)

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

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

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



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

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

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

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

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

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


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

(continues on next page)

(continued from previous page)

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

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

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

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

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

      ]

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

ma7 =  [
            [True, False, False],
            [False, True, False],
            [True, False, False]
      ]

mb7 =  [
            [True, True, False],
            [False, False, True],
            [False, True, False]

      ]

assert union(ma7, mb7) == [
                            [True, True, False],
                            [False, True, True],
```

(continues on next page)

(continued from previous page)

```
                                [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],
```

(continues on next page)

(continued from previous page)

```
        [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]
     ]
```

(continues on next page)

(continued from previous page)

```
assert surjective(m2) == True

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

assert surjective(m3) == False


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

assert surjective(m4) == False

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

assert surjective(m5) == False

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

assert surjective(m6) == False


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

assert surjective(m7) == False

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

assert surjective(m8) == True


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

assert surjective(m9) == True


m10 = [
```

(continues on next page)

(continued from previous page)

```
        [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 =  [
          [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],
```

(continues on next page)

(continued from previous page)

```
        [True, False],
    ]

assert surjective(m5) == False

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

assert surjective(m6) == False


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

assert surjective(m7) == False

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

assert surjective(m8) == True


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

assert surjective(m9) == True


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

    ]
assert surjective(m10) == False

m11 = [
       [True, True, False, False],
       [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:
```

(continues on next page)

(continued from previous page)

```
            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
            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) == {
```

(continues on next page)

(continued from previous page)

```
                            'a': []
                        }

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

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


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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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


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

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

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

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

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

[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
```

(continues on next page)

(continued from previous page)

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


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

da2 =  {
          'a': []
       }

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

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

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

(continues on next page)

(continued from previous page)

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


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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

2.1.3 Midterm - Thu 10, Jan 2019

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
```

(continues on next page)

(continued from previous page)

```
    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[35]

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

Introduction

B1 Theory

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

Given the following function:

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

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

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:

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

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

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

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

Testing: `python3 -m unittest gaps_test.GapsTest`

B3 Tasks stack

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

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

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

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

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

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

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

[3]:
```
from tasks_sol import *

do('a', subtasks)
```

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

(continues on next page)

(continued from previous page)

```
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.
```

(continues on next page)

(continued from previous page)

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

Testing: `python3 -m unittest tasks_test.DoTest`

B3.2 do_level

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

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

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

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

do_level('a', subtasks)
```

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

(continues on next page)

(continued from previous page)

```
            dictionary
    """
```

Testing: `python3 -m unittest exits_test.CpTest`

Example:

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

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

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

Basically, the dictionary above represents this visit tree:

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

(continues on next page)

(continued from previous page)

```
    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
```

(continues on next page)

(continued from previous page)

```
    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[36]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A

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

A.1 table_to_adj

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

[2]:
```
m0 =    [
            ['Identifier','Price','Quantity'],
            ['a',1,1],
            ['b',5,8],
            ['c',2,6],
            ['d',8,5],
            ['e',7,3]
        ]
```

where `a`, `b`, `c` etc are the row identifiers (imagine they represent items in a store), `Price` and `Quantity` are properties they might have. **NOTE**: here we put two properties, but they might have `n` properties !

We want to transform such table into a graph-like format as a dictionary of lists, which relates store items as keys to the properties they might have. To include in the list both the property identifier and its value, we will use tuples. So you need to write a function that transforms the above input into this:

[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)]
        }
```

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

<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],
            ['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)
```

(continues on next page)

(continued from previous page)

```
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)

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')],
```

(continues on next page)

(continued from previous page)

```
            '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

> **NOTICE: this part of the exam was ported to softpython[37] website**
>
> There you can find a more curated version (notice it may be longer than here)

Today we will analzye intercity bus network in GTFS format taken from dati.trentino.it[38], 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]
```

[37] https://en.softpython.org/projects/bus-speed/bus-speed-sol.html
[38] https://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs

[6]:
```
[OrderedDict([('', '1'),
              ('route_id', '76'),
              ('agency_id', '12'),
              ('route_short_name', 'B202'),
              ('route_long_name',
               'Trento-Sardagna-Candriai-Vaneze-Vason-Viote'),
              ('route_type', '3'),
              ('service_id', '22018091220190621'),
              ('trip_id', '000240274201809122019062 1'),
              ('trip_headsign', 'Trento-Autostaz.'),
              ('direction_id', '0'),
              ('arrival_time', '06:25:00'),
              ('departure_time', '06:25:00'),
              ('stop_id', '844'),
              ('stop_sequence', '2'),
              ('stop_code', '2620'),
              ('stop_name', 'Sardagna'),
              ('stop_desc', ''),
              ('stop_lat', '46.064848'),
              ('stop_lon', '11.09729'),
              ('zone_id', '2620.0')]),
 OrderedDict([('', '2'),
              ('route_id', '76'),
              ('agency_id', '12'),
              ('route_short_name', 'B202'),
              ('route_long_name',
               'Trento-Sardagna-Candriai-Vaneze-Vason-Viote'),
              ('route_type', '3'),
              ('service_id', '22018091220190621'),
              ('trip_id', '000240274201809122019062 1'),
              ('trip_headsign', 'Trento-Autostaz.'),
              ('direction_id', '0'),
              ('arrival_time', '06:26:00'),
              ('departure_time', '06:26:00'),
              ('stop_id', '5203'),
              ('stop_sequence', '3'),
              ('stop_code', '2620VD'),
              ('stop_name', 'Sardagna Civ. 22'),
              ('stop_desc', ''),
              ('stop_lat', '46.069494'),
              ('stop_lon', '11.095252'),
              ('zone_id', '2620.0')])]
```

Of interest to you are the fields `route_short_name`, `arrival_time`, and `stop_lat` and `stop_lon` which provide the geographical coordinates of the stop. Stops are already sorted in the file from earliest to latest.

Given a `route_short_name`, like `B202`, we want to plot the graph of bus velocity measured in **km/hours** at each stop. We define velocity at stop `n` as

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

where

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

and

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

We also set $velocity_0 = 0$

NOTE FOR TIME: When we say time in **hours**, it means that if you have the time as string `08:27:42`, its number in seconds since midnight is like:

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

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

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

[8]:
```
8.461666666666666
```

NOTE FOR SPACE: Unfortunately, we could not find the actual distance as road length done by the bus between one stop and the next one. So, for the sake of the exercise, we will take the *geo distance*, that is, we will calculate it using the line distance between the points of the stops, using their geographical coordinates. The function to calculate the `geo_distance` is already implemented :

[9]:
```
def geo_distance(lat1, lon1, lat2, lon2):
    """ Return the geo distance in kilometers
        between the points 1 and 2 at provided geographical coordinates.

    """
    # Shamelessly copied from https://stackoverflow.com/a/19412565

    from math import sin, cos, sqrt, atan2, radians

    # approximate radius of earth in km
    R = 6373.0

    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    return R * c
```

In the following we see the bus line `B102`, going from Sardagna to Trento. The graph should show something like the following.

We can see that as long as the bus is taking stops within Sardagna town, velocity (always intended as air-line velocity) is high, but when the bus has to go to Trento, since there are many twists and turns on the road, it takes a while to arrive even if in geo-distance Trento is near, so actual velocity decreases. In such case it would be much more convenient to take the cable car.

These type of graphs might show places in the territory where shortcuts such as cable cars, tunnels or bridges might be helpful for transportation.

<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):
    """
```

(continues on next page)

(continued from previous page)

```
        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
    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)
```

(continues on next page)

(continued from previous page)

```
    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
        - 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']
```

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[39]
- If you don't have unitn login: tell instructors and we will download your work manually

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

Part A - Bus network visualization

> **NOTICE: this part of the exam was ported to** softpython[40] **website**
>
> There you can find a more curated version (notice it may be longer than here)

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[41], 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[42] 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)
```

[40] https://en.softpython.org/projects/bus-network/bus-network-sol.html
[41] https://dati.trentino.it/dataset/trasporti-pubblici-del-trentino-formato-gtfs
[42] 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[43]
- 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

> **NOTICE: this part of the exam was ported to softpython[44] website**
>
> There you can find a more curated version (notice it may be longer than here)

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.

[43] http://examina.icts.unitn.it/studente
[44] https://en.softpython.org/projects/itea-real-estate/itea-real-estate-sol.html

Data provider: ITEA - dati.trentino.it[45]

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

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

    import csv
    with open('data/itea.csv', newline='',  encoding='latin-1',) as csvfile:
        reader = csv.DictReader(csvfile,  delimiter=';')
        lst = []
        for d in reader:
            lst.append(d)
    return lst


itea = load_itea()
```

IMPORTANT: look at the dataset by yourself !

Here we show only first 5 rows, but to get a clear picture of the dataset you need to study it a bit by yourself

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

[3]:
```
[OrderedDict([('Tipologia', 'ALTRO'),
              ('Proprietà', 'ITEA'),
              ('Indirizzo', "Codice unita': 30100049"),
              ('Frazione', ''),
              ('Comune', "BASELGA DI PINE'")]),
 OrderedDict([('Tipologia', 'ALLOGGIO'),
              ('Proprietà', 'ITEA'),
              ('Indirizzo', "Codice unita': 43100011"),
              ('Frazione', ''),
              ('Comune', 'TRENTO')]),
 OrderedDict([('Tipologia', 'ALLOGGIO'),
              ('Proprietà', 'ITEA'),
              ('Indirizzo', "Codice unita': 43100002"),
              ('Frazione', ''),
              ('Comune', 'TRENTO')]),
 OrderedDict([('Tipologia', 'ALLOGGIO'),
              ('Proprietà', 'ITEA'),
              ('Indirizzo', 'VIALE DELLE ROBINIE 26'),
              ('Frazione', ''),
              ('Comune', 'TRENTO')]),
 OrderedDict([('Tipologia', 'ALLOGGIO'),
              ('Proprietà', 'ITEA'),
              ('Indirizzo', 'VIALE DELLE ROBINIE 26'),
              ('Frazione', ''),
              ('Comune', 'TRENTO')])]
```

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

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[46] 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)

[46] 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.

[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[47]

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' !

[47] 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[48]
  - write pythonic code[49]
  - 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:

[48] https://sciprog.davidleoni.it/commandments.html
[49] 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[50]
- 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

NOTICE: this part of the exam was ported to softpython[51] website

There you can find a more curated version (notice it may be longer than here)

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[52]

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 = []
```

(continues on next page)

[50] http://examina.icts.unitn.it/studente
[51] https://en.softpython.org/formats/formats2-csv-sol.html?highlight=botteghe#CSV-Botteghe-storiche
[52] https://dati.trentino.it/dataset/botteghe-storiche-del-trentino

(continued from previous page)

```
        for d in reader:
            lst.append(d)
    return lst


botteghe = load_botteghe()
```

IMPORTANT: look at the dataset !

Here we show only first 5 rows, but to get a clear picture of the dataset you should explore it further.

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

[3]:
```
[OrderedDict([('Numero', '1'),
              ('Insegna', 'BAZZANELLA RENATA'),
              ('Indirizzo', 'Via del Lagorai'),
              ('Civico', '30'),
              ('Comune', 'Sover'),
              ('Cap', '38068'),
              ('Frazione/Località', 'Piscine di Sover'),
              ('Note', 'generi misti, bar - ristorante')]),
 OrderedDict([('Numero', '2'),
              ('Insegna', 'CONFEZIONI MONTIBELLER S.R.L.'),
              ('Indirizzo', 'Corso Ausugum'),
              ('Civico', '48'),
              ('Comune', 'Borgo Valsugana'),
              ('Cap', '38051'),
              ('Frazione/Località', ''),
              ('Note', 'esercizio commerciale')]),
 OrderedDict([('Numero', '3'),
              ('Insegna', 'FOTOGRAFICA TRINTINAGLIA UMBERTO S.N.C.'),
              ('Indirizzo', 'Largo Dordi'),
              ('Civico', '8'),
              ('Comune', 'Borgo Valsugana'),
              ('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)
```

(continues on next page)

(continued from previous page)

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

[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-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:

[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**

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[53]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - University of Trento staff

> **NOTICE: this part of the exam was ported to softpython[54] website**
>
> There you can find a more curated version (notice it may be longer than here)

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[55]

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

[53] http://examina.icts.unitn.it/studente
[54] https://en.softpython.org/projects/university-staff/university-staff-sol.html
[55] https://dati.trentino.it/dataset/personale-accademico-e-tecnico-amministrativo-dell-universita-di-trento

[1]:
```
import json

def load_data():
    with open('data/2019-06-30-persone-en-stripped.json', encoding='utf-8') as json_
↪file:
        data = json.load(json_file)
        return data

unitn = load_data()
```

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

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

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

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

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

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

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

    abbreviations = {

        "Department of Psychology and Cognitive Science": "COGSCI",
        "Center for Mind/Brain Sciences - CIMeC":"CIMeC",
        "Department of Civil, Environmental and Mechanical Engineering":"DICAM",
        "Centre Agriculture Food Environment - C3A":"C3A",
        "School of International Studies - SIS":"SIS",
        "Department of Sociology and social research": "Sociology",
        "Faculty of Law": "Law",
```

(continues on next page)

(continued from previous page)

```
        "Department of Economics and Management": "Economics",
        "Department of Information Engineering and Computer Science":"DISI",
        "Department of Cellular, Computational and Integrative Biology - CIBIO":"CIBIO
↪",
        "Department of Industrial Engineering":"DII"
    }
    if unitName in abbreviations:
        return abbreviations[unitName]
    else:
        return unitName.replace("Department of ", "")
```

Example:

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

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

A1 calc_uid_to_abbr

⊛ It will be useful having a map from department ids to their abbreviations, if they are actually present, otherwise to their original name. To implement this, you can use the previously defined function `abbreviate`.

```
{
 .
 .
 'STO0008629': 'DISI',
 'STO0008630': 'Sociology',
 'STO0008631': 'COGSCI',
 .
 .
 'STO0012897': 'Institutional Relations and Strategic Documents',
 .
 .
}
```

<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
```

(continues on next page)

(continued from previous page)

```
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'
```

(continues on next page)

(continued from previous page)

```
            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[56]

[56] https://sciprog.davidleoni.it/visualization/visualization-sol.html#Pie-chart

- For on example on plotting side by side, see this[57]

<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

            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'])
```

[57] https://sciprog.davidleoni.it/visualization/visualization-sol.html#Showing-plots-side-by-side

A4.1 calc_shared

⊛⊛⊛ We want to calculate the 10 *department pairs* that have the greatest number of people working in *both* departments (regardless of role), sorted in decreasing order.

For example, 'CIMeC' and 'COGSCI' have 23 people working in both departments, meaning each of these 23 people has at least a position at CIMeC and at least a position at COGSCI.

NOTE: in this case we are looking at number of actual people, *not* number of roles covered

- we do not want to consider Doctoral programmes
- we do not want to consider 'University of Trento' department (STO0000001)
- if your calculations display with swapped names (('COGSCI', 'CIMeC', 23) instead of ('CIMeC', 'COGSCI', 23)) it is not important, as long as they display just once per pair.

To implement this, we provide a sketch:

- build a dict which assigns unit codes to a `set` of *identifiers* of people that work for that unit
- to add elements to a set, use `.add` method
- to find common employees between two units, use set `.intersection` method (NOTE: it generates a *new* set)
- to check for all possibile unit couples, you will need a double `for` on a list of departments. To avoid double checking pairs (so not have both ('CIMeC', 'COGSCI', 23) and ('COGSCI', 'CIMeC', 23) in output), you can think like you are visiting the lower of a matrix (for the sake of the example here we put only 4 departments with random numbers).

```
           0       1       2       3
         DISI, COGSCI, CIMeC, DICAM
0 DISI    --      --      --     --
1 COGSCI  313     --      --     --
2 CIMeC   231     23      --     --
3 DICAM   12      13      123    --
```

<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 = {}
```

(continues on next page)

(continued from previous page)

```
    uid_to_abbr = calc_uid_to_abbr(db)

    for person in db:

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

    uids = list(uid_to_people)

    ret = []
    for x in range(len(uids)):
        uidx = uids[x]
        for y in range(x):
            uidy = uids[y]
            num = len(uid_to_people[uidx].intersection(uid_to_people[uidy]))
            if (num > 0) \
              and ("Doctoral programme" not in uid_to_abbr[uidx]) \
              and ("Doctoral programme" not in uid_to_abbr[uidy]) \
              and (uidx != '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):


    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 !!!
```

(continues on next page)

(continued from previous page)

```
        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),
```

(continues on next page)

(continued from previous page)

```
                            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[58]

<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]:
```
```

[ ]:
```
```

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

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**

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[59]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - Town events

NOTICE: this part of the exam was ported to softpython[60] website

There you can find a more curated version (notice it may be longer than here)

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[61]

License: Creative Commons Attribution 4.0[62]

[59] http://examina.icts.unitn.it/studente
[60] https://en.softpython.org/projects/town-events/town-events-sol.html
[61] https://dati.trentino.it/dataset/eventi-del-comune-di-trento
[62] http://creativecommons.org/licenses/by/4.0/deed.it

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):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   remoteId                     253 non-null    object
 1   published                    253 non-null    object
 2   modified                     253 non-null    object
 3   Priorità                     253 non-null    int64
 4   Evento speciale              0 non-null      float64
 5   Titolo                       253 non-null    object
 6   Titolo breve                 1 non-null      object
 7   Sottotitolo                  227 non-null    object
 8   Descrizione                  224 non-null    object
 9   Locandina                    16 non-null     object
 10  Inizio                       253 non-null    object
 11  Termine                      252 non-null    object
 12  Quando                       253 non-null    object
 13  Orario                       251 non-null    object
 14  Durata                       6 non-null      object
 15  Dove                         252 non-null    object
 16  lat                          253 non-null    float64
 17  lon                          253 non-null    float64
 18  address                      241 non-null    object
 19  Pagina web                   201 non-null    object
 20  Contatto email               196 non-null    object
 21  Contatto telefonico          196 non-null    object
 22  Informazioni                 62 non-null     object
 23  Costi                        132 non-null    object
 24  Immagine                     252 non-null    object
 25  Evento - manifestazione      252 non-null    object
 26  Manifestazione cui fa parte  108 non-null    object
 27  Tipologia                    252 non-null    object
 28  Materia                      252 non-null    object
 29  Destinatari                  24 non-null     object
 30  Circoscrizione               109 non-null    object
 31  Struttura ospitante          220 non-null    object
 32  Associazione                 1 non-null      object
 33  Ente organizzatore           0 non-null      float64
 34  Identificativo               0 non-null      float64
dtypes: float64(5), int64(1), object(29)
memory usage: 69.3+ KB
```

We will 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[63])

<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
```

(continues on next page)

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

(continued from previous page)

```
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
```

[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)
```

(continues on next page)

(continued from previous page)

```
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' ]
```

(continues on next page)

(continued from previous page)

```
             # 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
            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"
```

(continues on next page)

(continued from previous page)

```
# 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"

# 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)




# 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")

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[64] is reading and writing skills that are inadequate "to manage daily living and employment tasks that require reading skills beyond a basic level"

⊛⊛ Knowing that functional illiteracy is on the rise, a news agency wants to fire obsolete human journalists and attract customers by feeding them with automatically generated *fake news*. You are asked to develop the algorithm for producing the texts: while ethically questionable, the company pays well, so you accept.

Typically, a *fake news* starts with a real subject, a real fact (the *antecedent*), and follows it with some invented statement (the *consequence*). You are provided by the company three databases, one with subjects, one with antecedents and one of consequences. To each antecedent and consequence is associated a topic.

Write a function `fake_news` which takes the databases and RETURN a list holding strings with all possible combinations of subjects, antecedents and consequences where the topic of antecedent matches the one of consequence. See desired output for more info.

NOTE: Your code MUST work with *any* database

Expected output:

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

```
>>> fake_news(subjects, antecedents,consequences)

['Government passed fiscal reform, now spending is out of control',
 'Government passed fiscal reform, this increased taxes by 10%',
 'Government passed fiscal reform, this increased deficit by a staggering 20%',
 'Government passed fiscal reform, as a consequence our GDP has fallen dramatically',
 'Government passed jobs act, now spending is out of control',
 'Government passed jobs act, this increased taxes by 10%',
 'Government passed jobs act, this increased deficit by a staggering 20%',
 'Government passed jobs act, as a consequence our GDP has fallen dramatically',
 'Government regulated pollution emissions, businesses had to fire many employees',
 'Government regulated pollution emissions, businesses are struggling to meet law␣
↪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']
```

<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"),
```

(continues on next page)

(continued from previous page)

```
    ("restricted building in natural areas", "environment"),
    ("introduced more controls in agrifood production","environment"),
    ("changed immigration policy","foreign policy"),
]

db_consequences = [
    ("economy","now spending is out of control"),
    ("economy","this increased taxes by 10%"),
    ("economy","this increased deficit by a staggering 20%"),
    ("economy","as a consequence our GDP has fallen dramatically"),
    ("environment","businesses had to fire many employees"),
    ("environment","businesses are struggling to meet law requirements"),
    ("foreign policy","immigrants are stealing our jobs"),
]


def fake_news(subjects, antecedents,consequences):

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



fake_news(db_subjects, db_antecedents, db_consequences)
```

[10]:
```
['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%',
```

(continues on next page)

(continued from previous page)

```
 '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']
```

[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)
```

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[65]
- 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

[65] 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[66]
- 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`

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

Metamath

> **NOTICE: this part of the exam was ported to** softpython[67] **website**
>
> There you can find a more curated version (notice it may be longer than here)

Metamath[68] 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[69] up to the most basic axioms[70] 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 $.
    $}
```

[67] https://en.softpython.org/projects/metamath/metamath-sol.html
[68] http://us.metamath.org
[69] http://us.metamath.org/mm_100.html
[70] http://us.metamath.org/mpeuni/mmtheorems1.html#mm5s

Format description:

- Each row is a *statement*
- Words are separated by spaces. Each word that appears in a *statement* is called a *token*
- Tokens starting with dollar `$` are called *keywords*, you may have `$(`, `$)`, `$c`, `$v`, `$a`,`$f`,`${`,`$}`, `$.`
- Statements *may* be identified with a unique arbitrary *label*, which is placed at the beginning of the row. For example, `tt`, `weq`, `maj` are all labels (in the file there are more):
  - `tt $f term t $.`
  - `weq $a wff t = r $.`
  - `maj $e |- ( P -> Q ) $.`
- Some rows have no label, examples:
  - `$c 0 + = -> ( ) term wff |- $.`
  - `$v t r s P Q $.`
  - `$( State the axioms $)`
  - `${`
  - `$}`
- in each row, after the first dollar *keyword*, you *may* have an arbitratry *sequence* of characters terminated by a dollar followed by a dot `$.`. **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[71]), 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 )'
        },
 .
 .
 .
}
```

[71] 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`

[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")
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
```

(continues on next page)

(continued from previous page)

```
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',
                    bt('d'),
                    bt('e')),
            bt('c',
                    bt('f'),
                    bt('g')))

print(t)
```

```
a
├b
│ ├d
```

(continues on next page)

(continued from previous page)

```
│└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 solutions

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[72]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - Wordnet

NOTICE: this part of the exam was ported to softpython[73] website

There you can find a more curated version (notice it may be longer than here)

Open Jupyter and start editing this notebook `exam-2020-02-10.ipynb`

> WordNet[74]® 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[75]. Princeton University. 2010

In Python there are specialized libraries to read WordNet like NLTK[76], 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)

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[77], 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`.

[72] http://examina.icts.unitn.it/studente
[73] https://en.softpython.org/projects/wordnet/wordnet-sol.html
[74] https://wordnet.princeton.edu/
[75] https://wordnet.princeton.edu/
[76] https://www.nltk.org/howto/wordnet.html
[77] https://wordnet.princeton.edu/documentation/wndb5wn

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
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
```

(continues on next page)

(continued from previous page)

```
            d = {}

            params  = line.split('|')[0].split(' ')

            d['synset_offset'] = int(params[0])     # '00001740'
            d['lex_filenum'] = int(params[1])       # '03'
            d['ss_type'] = params[2]           # 'n'
            # WARNING: HERE THE STRING REPRESENT A NUMBER IN *HEXADECIMAL* FORMAT,
            #          AND WE WANT TO STORE AN *INTEGER*
            #          TO DO THE CONVERSION PROPERLY, YOU NEED TO USE int(my_string,␣
↪16)
            d['w_cnt'] = int(params[3], 16)        # 'b3' -> 179
            d['words'] = []
            i = 4
            for j in range(d['w_cnt']):
                wd = {
                      'word'  : params[i],      # 'entity'
                      'lex_id': int(params[i + 1],16), # '0'
                }
                d['words'].append(wd)
                i += 2
               #
            # WARNING: HERE THE STRING REPRESENT A NUMBER IN *DECIMAL* FORMAT,
            #          AND WE WANT TO STORE AN *INTEGER*
            #          TO DO THE CONVERSION PROPERLY, YOU NEED TO USE int(my_string)
            d['p_cnt'] = int(params[i])        # '003' -> 3
            d['ptrs'] = []
            i += 1
            for j in range(d['p_cnt']):
                ptr =  {
                          'pointer_symbol': params[i ],     # '~'
                          'synset_offset': int(params[i + 1]),  # '00001930'
                          'pos': params[i + 2],             # 'n'
                          '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': '~',
                     '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},
```

(continues on next page)

(continued from previous page)

```
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2086324},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2086723},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2116752},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2117748},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2117987},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2119787},
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2120985},
                    {'pointer_symbol': '%p',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2442560}],
          'ss_type': 'n',
          'synset_offset': 2085998,
          'w_cnt': 2,
          'words': [{'lex_id': 0, 'word': 'canine'},
                    {'lex_id': 0, 'word': 'canid'}]},
2086723: {'gloss': ' a member of the genus Canis (probably descended from the '
                   'common wolf) that has been domesticated by man since '
                   'prehistoric times; occurs in many breeds; "the dog barked '
                   'all night" \n',
          'lex_filenum': 5,
          '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',
```

(continues on next page)

(continued from previous page)

```
                    'source_target': '0000',
                    'synset_offset': 8011383},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 1325095},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2087384},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2087513},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2087924},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2088026},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2089774},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2106058},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2112993},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2113458},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2113610},
                   {'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',
```

(continues on next page)

(continued from previous page)

```
                    'synset_offset': 2114278},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2115149},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2115478},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2115987},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2116630},
                   {'pointer_symbol': '%p',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2161498}],
          'ss_type': 'n',
          'synset_offset': 2086723,
          'w_cnt': 3,
          'words': [{'lex_id': 0, 'word': 'dog'},
                    {'lex_id': 0, 'word': 'domestic_dog'},
                    {'lex_id': 0, 'word': 'Canis_familiaris'}]},
2106058: {'gloss': ' any of several breeds of usually large powerful dogs '
                   'bred to work as draft animals and guard and guide '
                   'dogs  \n',
          'lex_filenum': 5,
          'p_cnt': 16,
          'ptrs': [{'pointer_symbol': '@',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2086723},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2106493},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    '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': '~',
```

(continues on next page)

(continued from previous page)

```
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2110906},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2111074},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2111324},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2111699},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2111802},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2112043},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2112177},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2112339},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2112463},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2112613}],
          'ss_type': 'n',
          'synset_offset': 2106058,
          'w_cnt': 1,
          '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',
```

(continues on next page)

(continued from previous page)

```
                    'source_target': '0000',
                    'synset_offset': 2107903},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108064},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108157},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108293},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108507},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108682},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2108818},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2109034},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2109202},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2109314}],
          'ss_type': 'n',
          'synset_offset': 2107175,
          'w_cnt': 3,
          'words': [{'lex_id': 0, 'word': 'shepherd_dog'},
                    {'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},
```

(continues on next page)

(continued from previous page)

```
                    {'pointer_symbol': '~',
                     'pos': 'n',
                     'source_target': '0000',
                     'synset_offset': 2111567}],
          'ss_type': 'n',
          'synset_offset': 2111324,
          'w_cnt': 2,
          'words': [{'lex_id': 0, 'word': 'bulldog'},
                    {'lex_id': 0, 'word': 'English_bulldog'}]},
2112993: {'gloss': ' a large breed having a smooth white coat with black or '
                   'brown spots; originated in Dalmatia  \n',
          'lex_filenum': 5,
          'p_cnt': 2,
          'ptrs': [{'pointer_symbol': '@',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2086723},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2113184}],
          'ss_type': 'n',
          'synset_offset': 2112993,
          'w_cnt': 3,
          'words': [{'lex_id': 0, 'word': 'dalmatian'},
                    {'lex_id': 0, 'word': 'coach_dog'},
                    {'lex_id': 0, 'word': 'carriage_dog'}]},
2116752: {'gloss': ' any of various predatory carnivorous canine mammals of '
                   'North America and Eurasia that usually hunt in packs  \n',
          'lex_filenum': 5,
          'p_cnt': 7,
          'ptrs': [{'pointer_symbol': '@',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2085998},
                   {'pointer_symbol': '#m',
                    'pos': 'n',
                    'source_target': '0000',
                    'synset_offset': 2086515},
                   {'pointer_symbol': '~',
                    'pos': 'n',
                    'source_target': '0000',
                    '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',
```

(continues on next page)

(continued from previous page)

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

def draw_hist(db):

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

    pprint(hist)

    import numpy as np
```

(continues on next page)

(continued from previous page)

```
    import matplotlib.pyplot as plt

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

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

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

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

    plt.show()
```

[6]:

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

def draw_hist(db):
    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[78], `enqueue` can take $O(n)$: you will improve it by adding further indexing so it runs in $O(1)$

An `ItalianQueue` is modelled as a LinkedList with two pointers, a `_head` and a `_tail`:

- an element is enqueued scanning from `_head` until a matching group is found, in which case the element is inserted after (that is, at the right) of the matching group, otherwise the element is appended at the very end marked by `_tail`
- an element is dequeued from the `_head`

For this improved v2 version, you will use an additional dictionary `_tails` which associates to each group present in the queue the node at the tail of that group sequence. This way, instead of scanning you will be able to directly jump to insertion point.

```
class ItalianQueue:

    def __init__(self):
        """ Initializes the queue.

            - Complexity: O(1)
        """
        self._head = None
        self._tail = None
        self._tails = {}    #  <---- NEW  !
        self._size = 0
```

Example:

If we have the following situation:

```
data  :  a -> b -> c -> d -> e -> f -> g -> h
group :  x    x    y    y    y    z    z    z
         ^    ^              ^              ^
         |    |              |              |
         | _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
         ^    ^                   ^              ^
```

(continues on next page)

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

(continued from previous page)

```
        |     |                    |               |
        | _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
       _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)
```

(continues on next page)

(continued from previous page)

```
      _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)
       _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)
```

(continues on next page)

(continued from previous page)

```
      _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[79]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - Zoom surveillance

> **NOTICE: this part of the exam was ported to softpython[80] website**
>
> There you can find a more curated version (notice it may be longer than here)

A training center holds online courses with Zoom software[81]. 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.

[79] http://examina.icts.unitn.it/studente
[80] https://en.softpython.org/projects/zoom-surveillance/zoom-sol.html
[81] https://zoom.us/

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`
- `(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'],
```

(continues on next page)

(continued from previous page)

```
 ['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

<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 time24(t):
    """ Takes a time string like '06:27 PM' and outputs a string like 18:27
    """

    if t.endswith('AM'):
        if t.startswith('12:00'):
            return '00:00'
        else:
            return t.replace(' AM', '')
    else:
        if t.startswith('12:00'):
            return '12:00'

        h = '%0.d' % (int(t.split(':')[0]) + 12)

        return h + ':' + t.split(':')[1].replace(' PM','')


assert time24('12:00 AM') == '00:00'  # midnight
assert time24('01:06 AM') == '01:06'
assert time24('09:45 AM') == '09:45'
assert time24('12:00 PM') == '12:00'  # special case, it's actually midday
assert time24('01:27 PM') == '13:27'
assert time24('06:27 PM') == '18:27'
assert time24('10:03 PM') == '22:03'
```

[1]:
```
def time24(t):
    """ Takes a time string like '06:27 PM' and outputs a string like 18:27
    """
    raise Exception('TODO IMPLEMENT ME !')

assert time24('12:00 AM') == '00:00'  # midnight
assert time24('01:06 AM') == '01:06'
assert time24('09:45 AM') == '09:45'
assert time24('12:00 PM') == '12:00'  # special case, it's actually midday
```

(continues on next page)

(continued from previous page)

```
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[82] 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:

        my_reader = csv.reader(f, delimiter=',')
        next(my_reader)
        riga_meeting = next(my_reader)
        meeting_id = riga_meeting[0]
        topic = riga_meeting[1]
        meeting_date = riga_meeting[7]
        next(my_reader) # empty row
        next(my_reader) # second header
        ret.append(['meeting_id', 'topic','date', 'participant','join_time','leave_
↪time','reason'])
        for riga in my_reader:
            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'],
```

(continues on next page)

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

(continued from previous page)

```
 ['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']]
```

[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'],
```

(continues on next page)

(continued from previous page)

```
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '14:54', '15:03',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '15:12', '15:40',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '15:45', '16:00',
↪'Host closed the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Princess Toadstool', '13:56',
↪'15:33', 'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:05', '14:10',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:15', '14:29',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:33', '15:10',
↪'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '15:25', '15:54',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '15:55', '16:00',
↪'Host closed the meeting']]
```

[3]:
```
EXPECTED_MEETING_LOG = \
[['meeting_id', 'topic', 'date', 'participant', 'join_time', 'leave_time', 'reason'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Luigi', '13:54', '15:10',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Luigi', '15:12', '15:54',
↪'left the meeting'],
 ['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']]
```

(continues on next page)

(continued from previous page)

```
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 happened to the participant during the meeting

- `time_away` : the total time in minutes during which the participant has been disconnected during the meeting

To calculate gap durations, **we assume a meeting NEVER ends after midnight**

For the data format details, see `EXPECTED_STATS` below.

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

<a class="jupman-sol jupman-sol-toggler" onclick="jupman.toggleSolution(this);" data-jupman-show="Show solution" data-jupman-hide="Hide">Show solution</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]:
```
{'Luigi': {'max_gap': 2, 'gaps': 1, 'time_away': 2},
 'Mario': {'max_gap': 3, 'gaps': 4, 'time_away': 8},
 'Bowser': {'max_gap': 24, 'gaps': 3, 'time_away': 38},
```

(continues on next page)

(continued from previous page)

```
 'Princess Toadstool': {'max_gap': 0, 'gaps': 0, 'time_away': 0},
 'Wario': {'max_gap': 15, 'gaps': 4, 'time_away': 25}}
```

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


stats = calc_stats(meeting_log)

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

stats
```

[5]:
```
{'Luigi': {'max_gap': 2, 'gaps': 1, 'time_away': 2},
 'Mario': {'max_gap': 3, 'gaps': 4, 'time_away': 8},
 'Bowser': {'max_gap': 24, 'gaps': 3, 'time_away': 38},
 'Princess Toadstool': {'max_gap': 0, 'gaps': 0, 'time_away': 0},
 'Wario': {'max_gap': 15, 'gaps': 4, '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[83]

- 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):
```

(continues on next page)

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

(continued from previous page)

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

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

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

    width = 0.35
    fig, ax = plt.subplots(figsize=(10,3))
    rects1 = ax.bar(xs - width/2, ys_max_gap, width,
                    color='red', label='max gap')
    rects2 = ax.bar(xs + width/2, ys_time_away, width,
                    color='darkred', label='time_away')

    plt.xticks(xs, labels)

    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
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

```
    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)
     68
     69         ret = LinkedList()

ValueError: Negative values for end are not supported: -2
```

B3 BinaryTree prune_rec

Implement the method `prune_rec`:

```
def prune_rec(self, el):
    """ MODIFIES the tree by cutting all the subtrees that have their
        root node data equal to el. By 'cutting' we mean they are no longer linked
        by the tree on which prune is called.

        - if prune is called on a node having data equal to el, raises ValueError

        - MUST execute in O(n) where n is the number of nodes of the tree
        - NOTE: with big trees a recursive solution would surely
                exceed the call stack, but here we don't mind
    """
```

Testing: `python3 -m unittest bin_tree_test.PruneRecTest`

Example:

[17]:
```
from bin_tree_sol import *
from bin_tree_test import bt
```

[18]:
```
t = bt('a',
            bt('b',
                   bt('z'),
                   bt('c',
                        bt('d'),
                        bt('z',
                               None,
                               bt('e')))),
            bt('z',
                   bt('f'),
                   bt('z',
                          None,
                          bt('g'))))
```

[19]:
```
print(t)
```

```
a
├b
│├z
│└c
│ ├d
│ └z
│  ├
│  └e
└z
 ├f
 └z
  ├
  └g
```

[20]:
```
t.prune_rec('z')
```

[21]:
```
print(t)
```

```
a
├b
│├
│└c
│ ├d
│ └
└
```

[22]:
```
t.prune_rec('c')
```

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

```
a
├b
└
```

Trying to prune the root will throw a `ValueError`:

```
t.prune_rec('a')

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-27-f8e8fa8a97dd> in <module>()
----> 1 t.prune_rec('a')

ValueError: Tried to prune the tree root !
```

[ ]:
```

```

2.1.15 Exam - 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[84]
- If you don't have unitn login: tell instructors and we will download your work manually

Part A - NACE codes

> **NOTICE: this part of the exam was ported to softpython[85] website**
>
> There you can find a more curated version (notice it may be longer than here)

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?

[84] http://examina.icts.unitn.it/studente
[85] https://en.softpython.org/projects/nace/nace-sol.html

If you are in Europe, you will be presented with a catalog of economic activites you can choose from called NACE[86], which is then further specialized by various states (for example Italy's catalog is called ATECO[87])

Sections

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

Detail	
+ A	AGRICULTURE, FORESTRY AND FISHING Detail
+ B	MINING AND QUARRYING Detail
+ C	MANUFACTURING Detail
+ D	ELECTRICITY, GAS, STEAM AND AIR CONDITIONING SUPPLY Detail
+ E	WATER SUPPLY; SEWERAGE, WASTE MANAGEMENT AND REMEDIATION ACTIVITIES Detail
+ F	CONSTRUCTION Detail
+ G	WHOLESALE AND RETAIL TRADE; REPAIR OF MOTOR VEHICLES AND MOTORCYCLES Detail
+ H	TRANSPORTATION AND STORAGE Detail
+ I	ACCOMMODATION AND FOOD SERVICE ACTIVITIES Detail
+ J	INFORMATION AND COMMUNICATION Detail
+ K	FINANCIAL AND INSURANCE ACTIVITIES Detail
+ L	REAL ESTATE ACTIVITIES Detail
+ M	PROFESSIONAL, SCIENTIFIC AND TECHNICAL ACTIVITIES Detail
+ N	ADMINISTRATIVE AND SUPPORT SERVICE ACTIVITIES Detail
+ O	PUBLIC ADMINISTRATION AND DEFENCE; COMPULSORY SOCIAL SECURITY Detail
+ P	EDUCATION Detail
+ Q	HUMAN HEALTH AND SOCIAL WORK ACTIVITIES Detail
+ R	ARTS, ENTERTAINMENT AND RECREATION Detail
+ S	OTHER SERVICE ACTIVITIES Detail
+ T	ACTIVITIES OF HOUSEHOLDS AS EMPLOYERS; UNDIFFERENTIATED GOODS- AND SERVICES-PRODUCING ACTIVITIES OF HOUSEHOLDS FOR OWN USE Detail
+ U	ACTIVITIES OF EXTRATERRITORIAL ORGANISATIONS AND BODIES Detail

Section detail

If you drill down in say, section `M`, you will find something like this:

The first two digits of the code identify the *division*, the third digit identifies the *group*, and the fourth digit identifies the *class*:

[86] https://ec.europa.eu/eurostat/ramon/nomenclatures/index.cfm?TargetUrl=LST_NOM_DTL&StrNom=NACE_REV2&StrLanguageCode=EN&IntPcKey=&StrLayoutCode=HIERARCHIC

[87] https://www.istat.it/it/archivio/17888

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[88]
- If you don't have unitn login: tell instructors and we will download your work manually

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

Part A - Prezzario

> **NOTICE: this part of the exam was ported to softpython[89] website**
>
> There you can find a more curated version (notice it may be longer than here)

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[90]

> **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
```

(continues on next page)

[89] https://en.softpython.org/projects/price-catalog/price-catalog-sol.html
[90] https://dati.trentino.it/dataset/prezziario-dei-lavori-pubblici-della-provincia-autonoma-di-trento

(continued from previous page)

```
           Categoria  Prezzo
0   NaN               NaN
1   Noli e trasporti  109.09
2   NaN               NaN
3   NaN               NaN
4   Noli e trasporti  2.21
5   Noli e trasporti  3.36
6   Noli e trasporti  4.42
7   Noli e trasporti  5.63
8   Noli e trasporti  6.84
9   NaN               NaN
10  Noli e trasporti  8.77
11  Noli e trasporti  9.94
12  Noli e trasporti  14.66
13  Noli e trasporti  15.62
14  Noli e trasporti  16.40
15  Noli e trasporti  28.53
16  Noli e trasporti  44.06
17  Noli e trasporti  50.86
18  Noli e trasporti  55.88
19  Noli e trasporti  80.47
20  Noli e trasporti  94.00
21  NaN               NaN
```

Pompa completa a motore Example

If we look at the dataset, in some cases we can spot a pattern like the following (rows 3 to 8 included):

[3]:
```
df[3:12]
```

[3]:
```
     Codice Prodotto   Descrizione Breve Prodotto         Categoria Prezzo
3   A.02.40.0010      POMPA COMPLETA DI MOTORE     NaN               NaN
4   A.02.40.0010.010  fino a mm 50.                Noli e trasporti  2.21
5   A.02.40.0010.020  oltre mm 50 fino a mm 100.   Noli e trasporti  3.36
6   A.02.40.0010.030  oltre mm 100 fino a mm 150.  Noli e trasporti  4.42
7   A.02.40.0010.040  oltre mm 150 fino a mm 200.  Noli e trasporti  5.63
8   A.02.40.0010.050  oltre mm 200.                Noli e trasporti  6.84
9   A.02.40.0020      GRUPPO ELETTROGENO           NaN               NaN
10  A.02.40.0020.010  fino a 10 KW                 Noli e trasporti  8.77
11  A.02.40.0020.020  oltre 10 fino a 13 KW        Noli e trasporti  9.94
```

We see the first column holds product codes. If two rows share a code prefix, they belong to the same product type. As an example, we can take product `A.02.40.0010`, which has `'POMPA COMPLETA A MOTORE'` as description ('Descrizione Breve Prodotto' column). The first row is basically telling us the product type, while the following rows are specifying several products of the same type (notice they all share the `A.02.40.0010` prefix code until `'GRUPPO ELETTROGENO'` excluded). Each description specifies a range of values for that product: *fino a* means *until to* , and *oltre* means *beyond*.

Notice that:

- first row has only one number
- intermediate rows have two numbers
- last row of the product series (row 8) has only one number and contains the word *oltre* (*beyond*) (in some other cases, last row of product series may have two numbers)

A1 extract_bounds

Write a function that given a Descrizione Breve Prodotto **as a single string** extracts the range contained within as a tuple.

If the string contains only one number `n`:

- if it contains `UNTIL` ('fino') it is considered a first row with bounds `(0,n)`
- if it contains `BEYOND` ('oltre') it is considered a last row with bounds `(n, math.inf)`

DO NOT use constants like measure units 'mm', 'KW', etc in the code

<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)
```

(continues on next page)

(continued from previous page)

```
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)
```

[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[91]
- **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

(continues on next page)

[91] https://en.softpython.org/formats/formats2-csv-sol.html#Reading-as-dictionaries

(continued from previous page)

```
7   A.02.40.0010.040  oltre mm 150 fino a mm 200.  Noli e trasporti  5.63
8   A.02.40.0010.050  oltre mm 200.                Noli e trasporti  6.84
9   A.02.40.0020      GRUPPO ELETTROGENO           NaN               NaN
10  A.02.40.0020.010  fino a 10 KW                 Noli e trasporti  8.77
11  A.02.40.0020.020  oltre 10 fino a 13 KW        Noli e trasporti  9.94
```

A call to

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

Must produce:

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

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

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

(continues on next page)

(continued from previous page)

```
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'}]}
```

[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[92]
- If you don't have unitn login: tell instructors and we will download your work manually

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

Part A - Galactic Love

> **NOTICE: this part of the exam was ported to** softpython[93] **website**
>
> There you can find a more curated version (notice it may be longer than here)

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[94])

[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

(continues on next page)

[93] https://en.softpython.org/projects/galactic-love/galactic-love-sol.html
[94] https://github.com/mperrin/misc_astro

(continued from previous page)

```
26          Apus     0  26635 -4742      move alpha
27          Apus     2  29803 -4733    dotted gamma
28          Apus     2  30092 -4651     dotted beta
29          Apus     2  29410 -4721  dotted delta 1
30          Apus     2  29803 -4733    dotted gamma
31          Apus    -1      0     0             NaN
```

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

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],
                  [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)
```

(continues on next page)

(continued from previous page)

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



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

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

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

[4]:
```
[[0, 131.66666666666669, 28.75],
 [2, 130.816666666667, 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[95]

Music Sequencer

> **NOTICE: this part of the exam was ported to softpython[96] website**
>
> There you can find a more curated version (notice it may be longer than here)

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

ABC[97] 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>

[95] http://examina.icts.unitn.it/studente
[96] https://en.softpython.org/projects/music-sequencer/music-sequencer-sol.html
[97] http://abcnotation.com/wiki/abc:standard:v2.1#rrhythm

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

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

    ret = []
    for note in notes:
        n = ord(note[0])-ord('A')
        if len(note) == 1:
            ret.append((n,2))
        else:
            ret.append((n,2*int(note[1])))

    return ret


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

assert parse_melody("||") == []
assert parse_melody("|A|") == [(0,2)]
assert parse_melody("|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,
```

(continues on next page)

(continued from previous page)

```
  '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'}]
```

[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
```

(continues on next page)

(continued from previous page)

```
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)]
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]
```

(continues on next page)

(continued from previous page)

```
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                                            <==>']
```

[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            <==>        <>                        ',
```

(continues on next page)

(continued from previous page)

```
 'F                                                  ',
 'G                                              <==>']
```

[6]:
```
from pprint import pprint
melody2 =  "|F2  G4     |E4      E F|A2  B2  D2 |D3    E3   |C3    C3   |"
exp2 =    ['A                          <==>                                ',
           'B                              <==>                            ',
           'C                                                  <====><====>',
           'D                                  <==><====>                  ',
           'E             <======><>                      <====>           ',
           'F<==>                    <>                                    ',
           'G    <======>                                                  ']

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']))
```

(continues on next page)

(continued from previous page)

```
    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])
```

```
[ ]:
```

2.1.19 Midterm B - Wed 16, Dec 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-2020-12-16-exam.zip` and extract it on your desktop.
2) Rename `sciprog-ds-2020-12-16-FIRSTNAME-LASTNAME-ID` folder: put your name, lastname an id number, like `sciprog-ds-2020-12-16-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. Each 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[98]
- If you don't have unitn login: tell instructors and we will download your work manually

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

Now Open Visual Studio Code and start editing the folder on your desktop

For running tests: open *Accessories -> Terminal*

B1 Theory

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

B1.1. complexity

Given a list `L` of $n \geq 3$ integer elements, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

```
def my_fun(L):
    for i in range(3, len(L)):
        k = 0
        R = L[i]
        tmp = []
        while k < 3:
            if k % 2 == 1:
                R = R - L[k]
            else:
                R = R + L[k]
            k += 1
        tmp.append(R)

    return sum(tmp)
```

B1.2 dag topsort

What is the topological sorting of a directed acyclic graph (DAG)? Briefly describe an algorithm to compute it and provide a possible topological view of the following DAG.

B2 LinkedList pivot

```
def pivot(self):
    """
        Selects first node data as pivot, and then MOVES before the pivot
        all the nodes which have data value STRICTLY LESS (<) than the pivot.
        Finally, RETURN the number of moved nodes.

        IMPORTANT:
        - *DO NOT* create new nodes
        - nodes < than pivot must be in the reversed order they were found
        - nodes >= than pivot will maintain the original order
        - MUST EXECUTE in O(n), where n is the list size
    """
```

Testing: `python3 -m unittest linked_list_test.PivotTest`

Example:

[2]:
```
from linked_list_sol import *
from linked_list_test import to_ll

ll = to_ll([7, 12, 1, 3, 8, 9, 6, 4, 7, 2, 10])
```

[3]:
```
print(ll)
```

```
LinkedList: 7,12,1,3,8,9,6,4,7,2,10
```

[4]:
```
res = ll.pivot()
```

[5]:
```
res   # there were 5 elements strictly less than pivot 7
```

[5]:
```
5
```

Note elements < 7 are in reverse order in which they were found, elements ≥ 7 are in the original order

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

```
LinkedList: 2,4,6,3,1,7,12,8,9,7,10
```

B3 swap_stack

Open `bin_tree.py` and implement this method:

```
def swap_stack(self, a, b):
    """ Given two elements a and b, locates the two nodes where they are contained
        and swaps *only the data* in the found nodes.

        - if a or b are not found, raise LookupError

        - IMPORTANT 1: assume tree is NOT ordered
        - IMPORTANT 2: assume all nodes have different data
        - Implement it with a while and a stack
        - MUST execute in O(n) where n is the size of the tree
    """
```

Testing: `python3 -m unittest bin_tree_test.SwapStackTest`

Example:

[7]:
```
from bin_tree_test import bt

t = bt('a',
                    bt('b',
                            bt('c'),
                            bt('d')),
                    bt('e',
                            None,
                            bt('f',
                                    bt('g'))))
```

[8]:
```
draw_bt(t)
```

[9]:
```
t.swap_stack('f', 'b')
```

[10]:
```
draw_bt(t)
```

B4 family_sum_rec

Open `bin_tree.py` and implement this method:

```
def family_sum_rec(self):
    """ MODIFIES the tree by adding to each node data its *original* parent and␣
↪children data

        - MUST execute in O(n) where n is the size of the tree
        - A recursive implementation is acceptable
        - HINT: you will probably want to define a helper function
    """
```

Testing: `python3 -m unittest bin_tree_test.FamilySumRec`

Example:

[11]:
```
from bin_tree_test import bt

t = bt(5,
         bt(1,
              bt(4,
                   None,
                   bt(3)),
              bt(2)),
         bt(9,
              bt(11)))
```

[12]:
```
draw_bt(t)
```

[13]:
```
t.family_sum_rec()
draw_bt(t)
```

You need to sum to each node data its original parent data + original left child data + original right child data , for example:

- Root: 15 = 5 + 0 + 1 + 9
- left child of root: 12 = 1 + 5 + 4 + 2
- right child of root: 25 = 9 + 5 + 11 + 0
- leftmost grandchild of root: 8 = 4 + 1 + 0 + 3

[ ]:

2.1.20 Exam - Tue 14, Jan 2021

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

Part A - Witchcraft

> **NOTICE: this part of the exam was ported to** softpython[99] **website**
>
> There you can find a more curated version (notice it may be longer than here)

The early sixteenth century saw a dramatic rise in awareness and terror of witchcraft in the troubled lands of early modern Scotland[100]: thousands of people were executed, imprisoned, tortured, banished, and had lands and possessions confiscated. Persecution took place in courts of law: you shall analyze the evidence gathered during those dark days.

[99] https://en.softpython.org/projects/witchcraft/witchcraft-sol.html
[100] https://www.youtube.com/watch?v=4s9Hd8onAKQ

Data source: Julian Goodare, Lauren Martin, Joyce Miller and Louise Yeoman, 'The Survey of Scottish Witchcraft' http://www.shca.ed.ac.uk/witches/ (archived January 2003, accessed '11/1/2016').

In particular, we took WDB_Case.csv as published on data.world[101], which contains cases brought against suspected witches, along with annotations by researchers, mostly as boolean fields.

The dataset has lots of columns, we show here only the relevant ones `Case_date`, `CaseCommonName`, `Suspects_text` and an excerpt of the many boolean columns:

[1]:
```
import pandas as pd
case_df = pd.read_csv('WDB_Case.csv', encoding='UTF-8')
sel_case_df = case_df[['Case_date', 'CaseCommonName', 'Suspects_text', 'Demonic_p',
↪'Demonic_s', 'Maleficium_p','Maleficium_s', 'WitchesMeeting']]
sel_case_df[1986:1994]
```

[1]:
```
       Case_date      CaseCommonName  Suspects_text  Demonic_p  Demonic_s  \
1986   29/6/1649   3 unnamed witches            3.0          0          0
1987   19/8/1590      Leslie,William            NaN          0          0
1988        1679  McGuffock,Margaret            NaN          0          0
1989        1679        Rae,Grissell            NaN          0          0
1990        1679         Howat,Jonet            NaN          0          0
1991  15/10/1673       McNicol,Janet            NaN          1          1
1992    4/6/1674      Clerk,Margaret            NaN          0          0
1993   29/7/1675       Hendrie,Agnes            NaN          0          1

      Maleficium_p  Maleficium_s  WitchesMeeting
1986             0             0               0
1987             0             1               0
1988             0             1               0
1989             0             0               0
1990             0             0               0
1991             0             1               1
1992             0             0               0
1993             0             0               1
```

A1 parse_bool_cols

Open Jupyter and start editing this notebook `exam-2021-01-14.ipynb`

Since boolean columns are so many, as a first step you will build a recognizer for them.

- Consider a column as boolean if **ALL** of its values are either `0` or `1`
- Parse with CSV DictReader[102]

> **WARNING: Have you carefully read the text above?**
>
> Most students during the exam didn't, and wrote algorithms which declare a column as boolean as soon as a `0` or a `1` are found and manually discarded columns which didn't fit their flawed logic (like `NamedIndividual`).

To avoid getting bad grades for simple exercises, always ask yourself:

1. Am I sure about the results of my algorithm *without* looking at the expected solution ? In this case, it should be obvious that if you don't scan **all** cells in a column and you still declare it's boolean you are basically resorting to being lucky.

[101] https://data.world/history/scottish-witchcraft
[102] https://en.softpython.org/formats/formats-sol.html#Reading-as-dictionaries

2. Am I putting *constants* in the code (like `'NamedIndividual'`)? Whenever you have such urge, please ask first for permission
3. Is the exercise open to interpretation, maybe because it has so many possible weird cases and relative assertions, or is the text pretty clear? In this case the scope was quite definite, so you are expected to find a generic solution which could work with *any* dataset.

Example (for full output see expected_bool_cols.py):

```
>>> bool_cols = get_bool_cols('WDB_Case.csv')
>>> print('Found', len(bool_cols), 'cols. EXCERPT:', ' '.join(bool_cols[:17]), '...')
Found 77 cols. EXCERPT:
AdmitLesserCharge AggravatingDisease AnimalDeath AnimalIllness ClaimedBewitched␣
↪ClaimedNaturalCauses ClaimedPossessed CommunalSex Consulting_p Consulting_s Cursing␣
↪Dancing DemonicPact Demonic_p Demonic_possess_p Demonic_possess_s Demonic_s ...
```

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

[2]:

```
def get_bool_cols(filename):
    """RETURN a sorted list of all the names of boolean columns"""


    import csv
    with open(filename, encoding='utf-8', newline='') as f:

        cols = set(next(csv.DictReader(f, delimiter=',')).keys())

    with open(filename, encoding='utf-8', newline='') as f:
        my_reader = csv.DictReader(f, delimiter=',')    # Notice we now used DictReader
        for diz in my_reader:
            for k in diz:

                if not (diz[k] == '0' or diz[k] == '1'):
                    if k in cols:
                        # Note: for an optimal algorithm we could also stop scanning
                        #       the rows for this column
                        cols.remove(k)

        return sorted(cols)



bool_cols = get_bool_cols('WDB_Case.csv')
print('Found', len(bool_cols), 'cols.', 'EXCERPT: ',)
print( ' '.join(bool_cols[:17]), '...')

from expected_bool_cols import expected_bool_cols
if len(bool_cols) != len(expected_bool_cols):
    print('ERROR! different lengths: bools_cols: %s    expected_bools_cols: %s' %␣
↪(len(bool_cols), len(expected_bool_cols)))
else:
    for i in range(len(expected_bool_cols)):
        if bool_cols[i] != expected_bool_cols[i]:
            print('ERROR at index', i, ':')
            print('    ACTUAL:', repr(bool_cols[i]))
            print('  EXPECTED:', repr(expected_bool_cols[i]))
```

```
Found 77 cols. EXCERPT:
AdmitLesserCharge AggravatingDisease AnimalDeath AnimalIllness ClaimedBewitched␣
↪ClaimedNaturalCauses ClaimedPossessed CommunalSex Consulting_p Consulting_s Cursing␣
↪Dancing DemonicPact Demonic_p Demonic_possess_p Demonic_possess_s Demonic_s …
```

[2]:

```
def get_bool_cols(filename):
    """RETURN a sorted list of all the names of boolean columns"""
    raise Exception('TODO IMPLEMENT ME !')

bool_cols = get_bool_cols('WDB_Case.csv')
print('Found', len(bool_cols), 'cols.', 'EXCERPT: ',)
print( ' '.join(bool_cols[:17]), '...')

from expected_bool_cols import expected_bool_cols
if len(bool_cols) != len(expected_bool_cols):
    print('ERROR! different lengths: bools_cols: %s    expected_bools_cols: %s' %␣
↪(len(bool_cols), len(expected_bool_cols)))
else:
    for i in range(len(expected_bool_cols)):
        if bool_cols[i] != expected_bool_cols[i]:
            print('ERROR at index', i, ':')
            print('    ACTUAL:', repr(bool_cols[i]))
            print('  EXPECTED:', repr(expected_bool_cols[i]))
```

A2 fix_date

Implement `fix_date`, which takes a possibly partial date as a string `d/m/yyyy` and RETURN a string formatted as `mm/dd/yyyy`. If data is missing, omits it in the output as well, see examples.

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

[3]:

```
def fix_date(d):

    if '/' in d:
        case_date = d.split('/')
        if len(case_date) == 3:
            d = case_date[1]
            m = case_date[0]
            y = case_date[2]
            return "%02d/%02d/%04d" % (int(d),int(m),int(y))
        elif len(case_date) == 2:
            m = case_date[0]
            y = case_date[1]
            return "%02d/%04d" % (int(m),int(y))
    return d


assert fix_date('2/8/1649') == '08/02/1649'
assert fix_date('25/4/1627') == '04/25/1627'
assert fix_date('6/11/1629') == '11/06/1629'
assert fix_date('12/1649') == '12/1649'
```

(continues on next page)

(continued from previous page)

```
assert fix_date('7/1652') == '07/1652'
assert fix_date('1560') == '1560'
assert fix_date('') == ''
# Note there is a damned extra space in the dataset (Oswald,Katharine)
assert fix_date('13/11/ 1629') == '11/13/1629'
```

[3]:
```
def fix_date(d):
    raise Exception('TODO IMPLEMENT ME !')

assert fix_date('2/8/1649') == '08/02/1649'
assert fix_date('25/4/1627') == '04/25/1627'
assert fix_date('6/11/1629') == '11/06/1629'
assert fix_date('12/1649') == '12/1649'
assert fix_date('7/1652') == '07/1652'
assert fix_date('1560') == '1560'
assert fix_date('') == ''
# Note there is a damned extra space in the dataset (Oswald,Katharine)
assert fix_date('13/11/ 1629') == '11/13/1629'
```

A3 parse_db

Given a CSV of cases, outputs a list of dictionaries (parse with CSV DictReader[103]), each representing a case with these fields:

- `name`: the isolated name of the witch taken from `CaseCommonName` column if parseable, otherwise the full cell content
- `surname`: the isolated surname of the witch taken from `CaseCommonName` column if parseable, otherwise empty string
- `case_date`: `Case_date` column corrected with `fix_date`
- `suspects`: number of suspects as **integer**, to be taken from the column `Suspects_text`. If column is empty, use `1`

`primary`, `secondary` and `tags` fields are to be filled with names of **boolean** columns for which the corresponding cell is marked with `'1'` according to these criteria:

- `primary`: a **single string** as follows: if a column ending with `_p` is marked `'1'`, this field contains that column name without the `'_p'`. If column name is `'NotEnoughInfo_p'` or in other cases, use `None`.
- `secondary`: **sorted** column names ending with `_s`. If col name is `'NotEnoughInfo_s'` or it's already present as `primary`, it's discarded. Remove trailing `_s` from values in the list.
- `tags`: **sorted** column names which are not primary nor secondary

Example (full output is in expected_cases_db.py):

```
>>> cases_db = parse_db('WDB_Case.csv')
>>> cases_db[1991:1994]
[{'primary': 'Demonic',
  'secondary': ['ImplicatedByAnother', 'Maleficium', 'UNorthodoxRelPract'],
  'tags': ['DevilPresent', 'UnorthodoxReligiousPractice', 'WitchesMeeting'],
  'name': 'Janet',
```

(continues on next page)

[103] https://en.softpython.org/formats/formats2-csv-sol.html#Reading-as-dictionaries

(continued from previous page)

```
  'surname': 'McNicol',
  'suspects': 1,
  'case_date': '10/15/1673'},
 {'primary': None,
  'secondary': [],
  'tags': [],
  'name': 'Margaret',
  'surname': 'Clerk',
  'suspects': 1,
  'case_date': '06/04/1674'},
 {'primary': None,
  'secondary': ['Demonic'],
  'tags': ['Dancing', 'DevilPresent', 'Singing', 'WitchesMeeting'],
  'name': 'Agnes',
  'surname': 'Hendrie',
  'suspects': 1,
  'case_date': '07/29/1675'}]
```

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

[4]:

```
import csv

def parse_db(filename):


    characterisations = [c[:-2] for c in bool_cols if c.endswith('_p')]
    other_bool_fields = [c for c in bool_cols if not (c.endswith('_p') or c.endswith(
↪'_s'))]

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

        ret = []
        found = {}
        for diz in my_reader:
            work = {'primary': None,
                    'secondary':[],
                    'tags':[],
                   }
            for c in characterisations:
                if diz[c + '_p'] == '1':
                    if c != 'NotEnoughInfo':
                        work['primary'] = c
                if diz[c + '_s'] == '1':
                    if c != 'NotEnoughInfo':
                        if c != work['primary']:
                            work['secondary'].append(c)
                cn = diz['CaseCommonName']
                if ',' in cn:
                    spli = cn.split(',')
                    work['name'] = spli[-1].strip()
                    if len(spli) == 2:
                        work['surname'] = spli[0].strip()
                    else:
```

(continues on next page)

(continued from previous page)

```
                        work['surname'] = ','.join(spli[:-1]).strip()

                else:
                    work['name'] = cn.strip()
                    work['surname'] = ''

                sus = diz['Suspects_text']
                if sus:
                    work['suspects'] = int(sus)
                else:
                    work['suspects'] = 1

            for c in other_bool_fields:
                if diz[c] == '1':
                    work['tags'].append(c)

            work['case_date'] = fix_date(diz['Case_date'])
            ret.append(work)

        return ret


cases_db = parse_db('WDB_Case.csv')

cases_db[1991:1994]
```

[4]:
```
[{'primary': 'Demonic',
  'secondary': ['ImplicatedByAnother', 'Maleficium', 'UNorthodoxRelPract'],
  'tags': ['DevilPresent', 'UnorthodoxReligiousPractice', 'WitchesMeeting'],
  'name': 'Janet',
  'surname': 'McNicol',
  'suspects': 1,
  'case_date': '10/15/1673'},
 {'primary': None,
  'secondary': [],
  'tags': [],
  'name': 'Margaret',
  'surname': 'Clerk',
  'suspects': 1,
  'case_date': '06/04/1674'},
 {'primary': None,
  'secondary': ['Demonic'],
  'tags': ['Dancing', 'DevilPresent', 'Singing', 'WitchesMeeting'],
  'name': 'Agnes',
  'surname': 'Hendrie',
  'suspects': 1,
  'case_date': '07/29/1675'}]
```

[4]:
```
import csv

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

cases_db = parse_db('WDB_Case.csv')
```

(continues on next page)

(continued from previous page)

```
cases_db[1991:1994]
```

[5]:
```
# TESTS
assert cases_db[0]['primary'] == None
assert cases_db[0]['secondary'] == []
assert cases_db[0]['name'] == '3 unnamed witches'
assert cases_db[0]['surname'] == ''
assert cases_db[0]['suspects'] == 3  # int !
assert cases_db[0]['case_date'] == '08/02/1649'

assert cases_db[1]['primary'] == None
assert cases_db[1]['secondary'] ==  ['ImplicatedByAnother']
assert cases_db[1]['tags'] ==  []
assert cases_db[1]['name'] ==  'Cristine'
assert cases_db[1]['surname'] ==  'Kerington'
assert cases_db[1]['suspects'] ==  1    # Suspects_text is '', we put 1
assert cases_db[1]['case_date'] == '05/08/1591'

assert cases_db[1991]['primary'] == 'Demonic'
#NOTE: since 'Demonic' is already 'primary', we removed it from 'secondary'
assert cases_db[1991]['secondary'] == ['ImplicatedByAnother', 'Maleficium',
↪'UNorthodoxRelPract']
assert cases_db[1991]['tags'] == ['DevilPresent', 'UnorthodoxReligiousPractice',
↪'WitchesMeeting']
assert cases_db[1991]['name'] == 'Janet'
assert cases_db[1991]['surname'] == 'McNicol'
assert cases_db[1991]['suspects'] == 1    #  Suspects_text is '', we put 1
assert cases_db[1991]['case_date'] == '10/15/1673'

assert cases_db[0]['case_date'] == '08/02/1649'  # 2/8/1649
assert cases_db[1143]['case_date'] == '1560'     # 1560
assert cases_db[924]['case_date'] ==  '07/1652'  # 7/1652
assert cases_db[491]['suspects'] == 15           # 15

#composite name
assert cases_db[249]['name'] == 'Francis'     # "Stewart, Earl of Bothwell,Francis"
assert cases_db[249]['surname'] == 'Stewart, Earl of Bothwell'

from expected_cases_db import expected_cases_db
from pprint import pprint
for i in range(len(expected_cases_db)):
    if cases_db[i] != expected_cases_db[i]:
        print('ERROR at index %s!' % i)
        print('ACTUAL:')
        pprint(cases_db[i])
        print('EXPECTED:', )
        pprint(expected_cases_db[i])
if len(cases_db) != len(expected_cases_db):
    print('ERROR! different lengths: cases_db: %s    expected_cases_db: %s' %␣
↪(len(cases_db), len(expected_cases_db)))
assert cases_db == expected_cases_db
```

A4 plot_cases

Given the previously computed db, plot the number of cases per year.

- plot the ticks with 10 years intervals, according to the actual data (**DO NOT use constants like** `1560` **!!**)
- careful some cases have no year

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

[6]:

```
def plot_cases(db):

    import numpy as np
    import matplotlib.pyplot as plt
    from collections import Counter

    years = [int(diz['case_date'].split('/')[-1]) for diz in db if diz['case_date']]
    hist = Counter(years)
    xs = sorted(hist.keys())
    ys = [hist[x] for x in xs]

    fig = plt.figure(figsize=(18,5))
    plt.plot(xs, ys, color='purple')

    plt.title("Witchcraft cases per year SOLUTION")
    plt.xlabel('Year')
    plt.ylabel('Number of cases')

    tks = np.arange(min(xs), max(xs), 10)
    plt.xticks(tks, tks)

    plt.show()


plot_cases(cases_db)
```

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

plot_cases(cases_db)
```

Part B

B1.1 Theory - Complexity

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

1. 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)
            b = my_fun(L2) + min(L2)
            return max(a, b)
```

B1.2 Theory - Graph

Briefly describe what a BFS visit of a graph is. Please provide the order of visited nodes of a possible BFS visit on the following graph (starting from node 'a'):

B2 Bank

WARNING: AN UPDATED VERSION OF THIS EXERCISE CAN BE FOUND HERE[104]

A bank stores all its transactions in a log: possible types of transactions are opening an account, transfering some amount of money, withdrawal, etc. For simplicity, we represent each transaction type with a character, and in the log which is a Python list we put only such characters (we don't care about their meaning).

The bank knows some sequences of events are extremely rare, and probably denote suspicious activity. For simplicitly, we assume such sequences only contain two transactions and represent them in tuples called *biseqs*.

NOTE: a biseq can occur multiple times, but you can consider the number of repetitions negligible with respect to log size.

THESE EXERCISES ARE ABOUT CUSTOM INDEXING, NOT STRINGS: DO NOT convert everything to a string, use regexes, nor use string methods like `.index()`, `.replace()`, etc

B2.1 constructor, log and pos

Implement following methods, adding some appropriate indexing data structure.

BEFORE IMPLEMENTING, DOUBLE CHECK COMPLEXITY REQUIREMENTS!

The idea is that given *an entire biseq* (not just the first character !), we want to find all of its (rare) occurrences *very fast* in $O(1)$

Since performance is important here, passing tests does *not* imply your work is correct!

[104] https://sciprog.davidleoni.it/indexing/indexing.html#2.-Exercise---Bank

```
class Bank:

    def __init__(self):
        """ Initializes an empty bank

            Add a data structure of your choice for indexing the log
        """
        self._trans = []
        raise Exception("TODO IMPLEMENT ME!")

    def log(self, transaction):
        """ Appends transaction to the log

            - REMEMBER to also update the index
            - MUST EXECUTE IN O(1)
        """
```

```
def pos(self, biseq):
    """ RETURN a NEW list with all the indeces where the sequence of
        two transactions can be found

        - MUST execute in O(1)
    """
```

Testing: `python3 -m unittest bank_test.LogTest`

Example:

[7]:
```
from bank_sol import *

bank = Bank()
print(bank)
```

```
Bank:
```

[8]:
```
bank.log('a')
```

[9]:
```
print(bank)
```

```
Bank: a
```

[10]:
```
print(bank.pos( ('a','b') ))  # <-- a biseq is a tuple
```

```
[]
```

[11]:
```
bank.log('b')
```

[12]:
```
print(bank.pos( ('a','b') ))
```

```
[0]
```

[13]:
```
bank.log('a')
bank.log('a')
bank.log('a')
bank.log('a')
bank.log('b')
print(bank)
```

```
Bank: a,b,a,a,a,a,b
```

[14]:
```
print(bank.pos(('a','b')))
```

```
[0, 5]
```

[15]:
```
print(bank.pos(('a','a')))
```

```
[2, 3, 4]
```

B2.2 revert

```
def revert(self):
    """ Completely eliminates last transaction and RETURN it

        - if bank is empty, raises IndexError

        - REMEMBER to update any index referring to it
        - *MUST* EXECUTE IN O(1)
    """
```

Testing: `python3 -m unittest RevertTest`

Example:

[16]:
```
bank = Bank()
bank.log('a')
bank.log('b')
bank.log('c')
bank.log('a')
bank.log('b')
print(bank.pos(('a','b')))
```

```
[0, 3]
```

[17]:
```
bank.revert()
```

[17]:
```
'b'
```

[18]:
```
print(bank)
```

```
Bank: a,b,c,a
```

[19]:
```
print(bank.pos(('a','b')))
```

```
[0]
```

B2.3 max_interval

Typically, there is a biseq which triggers a period of suspicious activity, and there is another biseq which ends it. So given a start and an end biseq, the bank wants a report of all the transactions that happened in between:

```
def max_interval(self, bi_start, bi_end):
    """ RETURN a list with all the transactions occurred between
        the *largest* interval among bi-sequences bi_start and bi_end

        - bi_start and bi_end are EXCLUDED
        - if bi_start / bi_end are not found, or if bi_end is before/includes bi_
↪start,
          raise LookupError

        - DO *NOT* MODIFY the data structure
        - MUST EXECUTE IN O(k) where k is the length of the *largest* interval
          you can return
        - consider number of repetitions a negligible size
    """
```

Testing: `python -m unittest bank_test.MaxIntervalTest`

Example:

[20]:
```
bank = Bank()
bank.log('c')
bank.log('d')
bank.log('c')
bank.log('a')
bank.log('b') # <--- b
bank.log('e') #      e
bank.log('f') #      --- f     |
bank.log('a') #          a     |
bank.log('f') #          f     | k
bank.log('c') #          c     |
bank.log('b') #      --- b     |
bank.log('a') # <--- a
bank.log('f') #      f
bank.log('b')
bank.log('e')
bank.log('l')
bank.max_interval( ('b','e'), ('a','f') )
```

[20]:
```
['f', 'a', 'f', 'c', 'b']
```

[ ]:
```

```

2.1.21 Exam - Wed 10, Feb 2021

Scientific Programming - Data Science @ University of Trento

Download exercises and solutions

Part A - Wikispeedia

> **NOTICE: this part of the exam was ported to softpython[105] website**
>
> There you can find a more curated version (notice it may be longer than here)

Wikispeedia[106] is a fun game where you are given (or can choose) a source and a target Wikipedia page, and you are asked to reach target page by only clicking links you find along the pages you visit. These click paths provide valuable information regarding human behaviour and the semantic connection between different topics. You will analyze a dataset of such paths.

Data source: https://snap.stanford.edu/data/wikispeedia.html

- Robert West and Jure Leskovec: Human Wayfinding in Information Networks. 21st International World Wide Web Conference (WWW), 2012.
- Robert West, Joelle Pineau, and Doina Precup: Wikispeedia: An Online Game for Inferring Semantic Distances between Concepts. 21st International Joint Conference on Artificial Intelligence (IJCAI), 2009.

Open Jupyter and start editing this notebook `exam-2021-02-10.ipynb`

[2]:
```
import pandas as pd
import numpy as np
pd.options.display.max_colwidth = None
df = pd.read_csv('paths_finished.tsv', encoding='UTF-8', skiprows=16, header=None,␣
↪sep='\t')
```

Each row of the dataset paths_finished.tsv is a user *session*, where user navigates from start page to end page. Columns are `hashedIpAddress`, `timestamp`, `durationInSec`, `path` and `rating`.

We define a **session group** as all sessions which have same start page and same end page, for example all these paths start with `Linguistics` and end in `Rome`:

[3]:
```
df[5890:5902]
```

[3]:
```
                     0           1    2  \
5890  2f8c281d5e0b0e93  1248913182  106
5891  389b67fa365b727a  1249604227   89
5892  0299542414c3f20a  1257970576   94
5893  2b6e83d366a7514d  1260188882  113
5894  0d57c8c57d75e2f5  1282028286  153
5895  0d57c8c57d75e2f5  1295244051   62
5896  772843f73d9cf93d  1307880434  177
5897  654430d34a08a0f5  1339755238   81
5898  12470aee3d5ad152  1344167616   40
5899  6365b049395c53ce  1345421532   67
5900  05786cb24102850d  1347671980   65
5901  369a3f7e77700217  1349302559   56

                                                                    3  \
5890            Linguistics;Philosophy;Aristotle;Ancient_Greece;Italy;Rome
5891                      Linguistics;Language;English_language;Latin;Rome
5892                               Linguistics;Philosophy;Plato;Latin;Rome
```

(continues on next page)

[105] https://en.softpython.org/projects/wikispeedia/wikispeedia-sol.html
[106] https://dlab.epfl.ch/wikispeedia/play/

(continued from previous page)

```
5893                     Linguistics;Philosophy;Thomas_Aquinas;Italy;Rome
5894  Linguistics;Language;Spanish_language;Vulgar_Latin;Roman_Empire;Rome
5895      Linguistics;Philosophy;Socrates;Ancient_Greece;Ancient_Rome;Rome
5896      Linguistics;Language;German_language;Latin_alphabet;<;Italy;Rome
5897                 Linguistics;Philosophy;Augustine_of_Hippo;Italy;Rome
5898                              Linguistics;Philosophy;Plato;Italy;Rome
5899                                Linguistics;Culture;Ancient_Rome;Rome
5900                     Linguistics;Language;English_language;Latin;Rome
5901                     Linguistics;Language;English_language;Latin;Rome

        4
5890  3.0
5891  2.0
5892  NaN
5893  2.0
5894  NaN
5895  1.0
5896  NaN
5897  2.0
5898  NaN
5899  1.0
5900  2.0
5901  NaN
```

In this other session group, all sessions start with `Pikachu` and end with `Sun`:

[4]:
```
df[45121:45138]
```

[4]:
```
                      0           1   2  \
45121  0d57c8c57d75e2f5  1278403914  17
45122  0d57c8c57d75e2f5  1278403938  42
45123  0d57c8c57d75e2f5  1278403972  17
45124  0d57c8c57d75e2f5  1278403991   8
45125  0d57c8c57d75e2f5  1278404007   9
45126  0d57c8c57d75e2f5  1278404048   8
45127  0d57c8c57d75e2f5  1278404061  16
45128  0d57c8c57d75e2f5  1278404065   8
45129  0d57c8c57d75e2f5  1278404070   8
45130  0d57c8c57d75e2f5  1278404089   8
45131  0d57c8c57d75e2f5  1278404095   7
45132  0d57c8c57d75e2f5  1278404101   9
45133  0d57c8c57d75e2f5  1278404117   6
45134  0d57c8c57d75e2f5  1278404124  10
45135  0d57c8c57d75e2f5  1278404129   9
45136  0d57c8c57d75e2f5  1278404142   7
45137  0d57c8c57d75e2f5  1278404145   6

                                          3    4
45121  Pikachu;North_America;Earth;Planet;Sun  1.0
45122               Pikachu;Tree;Sunlight;Sun  NaN
45123       Pikachu;Tree;Plant;<;Sunlight;Sun  1.0
45124               Pikachu;Tree;Sunlight;Sun  NaN
45125               Pikachu;Tree;Sunlight;Sun  1.0
45126               Pikachu;Tree;Sunlight;Sun  NaN
45127               Pikachu;Tree;Sunlight;Sun  NaN
45128               Pikachu;Tree;Sunlight;Sun  NaN
45129               Pikachu;Tree;Sunlight;Sun  NaN
```

(continues on next page)

(continued from previous page)

```
45130                Pikachu;Tree;Sunlight;Sun  NaN
45131                Pikachu;Tree;Sunlight;Sun  NaN
45132                Pikachu;Tree;Sunlight;Sun  NaN
45133                Pikachu;Tree;Sunlight;Sun  NaN
45134                Pikachu;Tree;Sunlight;Sun  NaN
45135                Pikachu;Tree;Sunlight;Sun  NaN
45136                Pikachu;Tree;Sunlight;Sun  NaN
45137                Pikachu;Tree;Sunlight;Sun  NaN
```

A1 filter_back

Whenever a user clicks the *Back* button, she navigates back one page. This fact is tracked in the data by the presence of a `'<'` symbol. Write a function which RETURN a NEW path without pages which were navigated back.

NOTE: you can have duplicates even without presence of <, because a user might end up to a previous page just by following circular links. Don't misuse search methods, I'm watching you }:-{

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

[5]:
```
def filter_back(path):

    ret = []

    for page in path:
        if page == '<':
            if len(ret) > 0:
                ret.pop()
        else:
            ret.append(page)
    return ret


assert filter_back([]) == []
assert filter_back(['alfa']) == ['alfa']
assert filter_back(['beta','alfa','charlie']) == ['beta','alfa','charlie']

assert filter_back(['charlie', 'tango','<']) == ['charlie']
inp = ['charlie', 'tango','<']
assert filter_back(inp) == ['charlie']  # new
assert inp == ['charlie', 'tango','<']
assert filter_back(['alfa', 'beta', 'charlie','<','<','delta']) == ['alfa','delta']
assert filter_back(['alfa', 'beta', 'charlie','delta','eagle','<','<','golf','<','<',
↪'hotel']) \
       == ['alfa','beta','hotel']

# circular paths
assert filter_back(['alfa','beta','alfa','alfa','beta']) ==  ['alfa','beta','alfa',
↪'alfa','beta']
assert filter_back(['alfa','beta','alfa','<','charlie','charlie','delta','charlie','<
↪','charlie','delta']) \
       == ['alfa','beta','charlie','charlie','delta','charlie','delta']
```

[5]:
```
def filter_back(path):
    raise Exception('TODO IMPLEMENT ME !')

assert filter_back([]) == []
assert filter_back(['alfa']) == ['alfa']
assert filter_back(['beta','alfa','charlie']) == ['beta','alfa','charlie']

assert filter_back(['charlie', 'tango','<']) == ['charlie']
inp = ['charlie', 'tango','<']
assert filter_back(inp) == ['charlie']  # new
assert inp == ['charlie', 'tango','<']
assert filter_back(['alfa', 'beta', 'charlie','<','<','delta']) == ['alfa','delta']
assert filter_back(['alfa', 'beta', 'charlie','delta','eagle','<','<','golf','<','<',
↪'hotel']) \
       == ['alfa','beta','hotel']

# circular paths
assert filter_back(['alfa','beta','alfa','alfa','beta']) ==  ['alfa','beta','alfa',
↪'alfa','beta']
assert filter_back(['alfa','beta','alfa','<','charlie','charlie','delta','charlie','<
↪','charlie','delta']) \
       == ['alfa','beta','charlie','charlie','delta','charlie','delta']
```

A2 load_db

Load the **tab**-separated file paths_finished.tsv with a CSV Reader. The file has some rows to skip and no column names: parse it and RETURN a list of dictionaries, with `hashedIpAddress`, `timestamp`, `durationInSec`, `path` and `rating` as fields:

- `path`: convert it with `filter_back` function
- `timestamp` and `durationInSec`: convert to integer
- `rating`: convert to integer, if `NULL`, set it to `None`

<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

def load_db(filename):



    with open(filename, encoding='utf-8', newline='') as f:
        my_reader = csv.reader(f, delimiter='\t')

        ret = []

        for row in my_reader:
            if len(row) < 2:
                continue

            ses = { "hashedIpAddress": row[0],
                    "timestamp" : int(row[1]),
                    "durationInSec" : int(row[2]),
                    "path" : filter_back(row[3].split(';')),
```

(continues on next page)

(continued from previous page)

```
                    "rating": int(row[4]) if row[4] != 'NULL' else None
            }

            ret.append(ses)

        print('Parsed', len(ret), 'sessions')

        return ret


sessions_db = load_db('paths_finished.tsv')

sessions_db[:2]
```

```
Parsed 51318 sessions
```

[6]:
```
[{'hashedIpAddress': '6a3701d319fc3754',
  'timestamp': 1297740409,
  'durationInSec': 166,
  'path': ['14th_century',
   '15th_century',
   '16th_century',
   'Pacific_Ocean',
   'Atlantic_Ocean',
   'Accra',
   'Africa',
   'Atlantic_slave_trade',
   'African_slave_trade'],
  'rating': None},
 {'hashedIpAddress': '3824310e536af032',
  'timestamp': 1344753412,
  'durationInSec': 88,
  'path': ['14th_century',
   'Europe',
   'Africa',
   'Atlantic_slave_trade',
   'African_slave_trade'],
  'rating': 3}]
```

[6]:
```
import csv

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

sessions_db = load_db('paths_finished.tsv')

sessions_db[:2]
```

```
Parsed 51318 sessions
```

[6]:
```
[{'hashedIpAddress': '6a3701d319fc3754',
  'timestamp': 1297740409,
  'durationInSec': 166,
  'path': ['14th_century',
   '15th_century',
   '16th_century',
   'Pacific_Ocean',
```

(continues on next page)

(continued from previous page)

```
   'Atlantic_Ocean',
   'Accra',
   'Africa',
   'Atlantic_slave_trade',
   'African_slave_trade'],
  'rating': None},
 {'hashedIpAddress': '3824310e536af032',
  'timestamp': 1344753412,
  'durationInSec': 88,
  'path': ['14th_century',
   'Europe',
   'Africa',
   'Atlantic_slave_trade',
   'African_slave_trade'],
  'rating': 3}]
```

[7]:
```
# TESTING
from pprint import pprint
from expected_db import expected_db
for i in range(0, len(expected_db)):
    if expected_db[i] != sessions_db[i]:
        print('\nERROR at index', i, ':')
        print('  ACTUAL:')
        pprint(sessions_db[i])
        print('  EXPECTED:')
        pprint(expected_db[i])
        break
```

A3 calc_stats

Write a function which takes the sessions db and RETURN a NEW dictionary which maps sessions groups expressed as tuples (start, end) to a dictionary of statistics about them

- dictionary key: tuple with start,end page
- `sessions`: the number of sessions in that group
- `avg_len`: the average length (as number of edges) of all paths in the group
- `pages`: the total number of **DISTINCT** pages found among all sessions in that group
- `freqs`: a dictionary which maps edges found in all sessions of that group to their count
- `max_freq`: the highest count among all freqs

<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 calc_stats(sessions):

    ret = {}

    for session in sessions:
        path = session['path']

        t = (path[0], path[-1])
        if t in ret:
```

(continues on next page)

(continued from previous page)

```
            ret[t]['avg_len'] += len(path) - 1
            ret[t]['sessions'] += 1

            ret[t]['pages_set'].update(path)

        else:
            ret[t] = { 'avg_len' : len(path) - 1,
                       'sessions' : 1,
                       'pages_set' : set(path),
                       'freqs' : {}}

        freqs = ret[t]['freqs']

        for i in range(1, len(path)):
            seg = tuple([path[i-1], path[i]])
            if seg in freqs:
                freqs[seg] += 1
            else:
                freqs[seg] = 1

    for t in ret:
        ret[t]['avg_len'] = ret[t]['avg_len'] / ret[t]['sessions']
        ret[t]['pages'] = len(ret[t]['pages_set'])
        freq_vs = ret[t]['freqs'].values()
        ret[t]['max_freq'] = max(freq_vs) if len(freq_vs) > 0 else 0
        del ret[t]['pages_set']

    return ret


stats_db = calc_stats(sessions_db)
```

[8]:
```
def calc_stats(sessions):
    raise Exception('TODO IMPLEMENT ME !')

stats_db = calc_stats(sessions_db)
```

[9]:
```
# TESTING
from pprint import pprint
from expected_stats_db import expected_stats_db
for t in expected_stats_db:
    if not t in stats_db:
        print('\nERROR: missing key for session group', t)
        break
    elif expected_stats_db[t] != stats_db[t]:
        print('\nERROR at key for session group', t, ':')
        print('  ACTUAL:')
        pprint(stats_db[t])
        print('  EXPECTED:')
        pprint(expected_stats_db[t])
        break
```

Output excerpt:

```
[10]: pprint({ k:stats_db[k] for k in [('Linguistics', 'Rome'), ('Pikachu', 'Sun')]})
```

```
{('Linguistics', 'Rome'): {'avg_len': 4.166666666666667,
                           'freqs': {('Ancient_Greece', 'Ancient_Rome'): 1,
                                     ('Ancient_Greece', 'Italy'): 1,
                                     ('Ancient_Rome', 'Rome'): 2,
                                     ('Aristotle', 'Ancient_Greece'): 1,
                                     ('Augustine_of_Hippo', 'Italy'): 1,
                                     ('Culture', 'Ancient_Rome'): 1,
                                     ('English_language', 'Latin'): 3,
                                     ('German_language', 'Italy'): 1,
                                     ('Italy', 'Rome'): 5,
                                     ('Language', 'English_language'): 3,
                                     ('Language', 'German_language'): 1,
                                     ('Language', 'Spanish_language'): 1,
                                     ('Latin', 'Rome'): 4,
                                     ('Linguistics', 'Culture'): 1,
                                     ('Linguistics', 'Language'): 5,
                                     ('Linguistics', 'Philosophy'): 6,
                                     ('Philosophy', 'Aristotle'): 1,
                                     ('Philosophy', 'Augustine_of_Hippo'): 1,
                                     ('Philosophy', 'Plato'): 2,
                                     ('Philosophy', 'Socrates'): 1,
                                     ('Philosophy', 'Thomas_Aquinas'): 1,
                                     ('Plato', 'Italy'): 1,
                                     ('Plato', 'Latin'): 1,
                                     ('Roman_Empire', 'Rome'): 1,
                                     ('Socrates', 'Ancient_Greece'): 1,
                                     ('Spanish_language', 'Vulgar_Latin'): 1,
                                     ('Thomas_Aquinas', 'Italy'): 1,
                                     ('Vulgar_Latin', 'Roman_Empire'): 1},
                           'max_freq': 6,
                           'pages': 19,
                           'sessions': 12},
 ('Pikachu', 'Sun'): {'avg_len': 3.0588235294117645,
                      'freqs': {('Earth', 'Planet'): 1,
                                ('North_America', 'Earth'): 1,
                                ('Pikachu', 'North_America'): 1,
                                ('Pikachu', 'Tree'): 16,
                                ('Planet', 'Sun'): 1,
                                ('Sunlight', 'Sun'): 16,
                                ('Tree', 'Sunlight'): 16},
                      'max_freq': 16,
                      'pages': 7,
                      'sessions': 17}}
```

A4 plot_network

Given a sessions group (start_page, target_page), we want to display the pages clicked by users for all its sessions. Since some sessions share edges, we will show their click frequency.

- Set edges attributes `'weight'`, `'label'` as $freq$, `'penwidth'` as $5\frac{freq}{max_freq}$ and `'color'` as `'#45daed'`
- Only display edges (and pages connected by such edges) for which the click count is *strictly greater* than the given `threshold`
- **NOTE:** when applying a threshold, it's fine if some nodes don't appear linked to either source or target

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

[11]:
```
from sciprog import draw_nx
import networkx as nx

def plot_network(stats, source_page, target_page, threshold=0):


    G = nx.DiGraph()

    st = stats[(source_page, target_page)]

    for p1,p2 in st['freqs']:

        d = st['freqs'][(p1,p2)]

        if d > threshold:

            G.add_node(p2)
            G.add_edge(p1, p2)

            G[p1][p2]['weight'] = d
            G[p1][p2]['label'] = d
            G[p1][p2]['penwidth']= 5*d/st['max_freq']
            G[p1][p2]['color']= '#45daed'


    draw_nx(G

               )


plot_network(stats_db,'Linguistics', 'Rome')
```

```
Image saved to file:  expected-Linguistics-Rome-0.png
```

[11]:
```
from sciprog import draw_nx
import networkx as nx

def plot_network(stats, source_page, target_page, threshold=0):
    raise Exception('TODO IMPLEMENT ME !')

plot_network(stats_db,'Linguistics', 'Rome')
```

```
Image saved to file:  expected-Linguistics-Rome-0.png
```

[12]:
```
plot_network(stats_db, 'Batman', 'Bible', 0)  # default threshold zero, big graph
```

```
Image saved to file:  expected-Batman-Bible-0.png
```

[13]:
```
plot_network(stats_db, 'Batman', 'Bible', 1)    # we take only edges > 1
```

```
Image saved to file:  expected-Batman-Bible-1.png
```

Part B

- Open Visual Studio Code and start editing the folder on your desktop

B1.1 Theory - Complexity

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

Given a list L of n elements, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

[15]:
```
def my_fun(L):
    T = []
    N = len(L)
    for i in range(N):
        cnt = 0
        for j in range(N):
            if  L[i] > L[j]:
                cnt += 1
        T.append(cnt)

    return T
```

B1.2 Theory - BST

Briefly answer the following questions: what is the Tree data structure. What is a Binary Search Tree (BST)? What can we use a BST for?

B2 Reconstruct BinaryTree

Open `bin_tree.py` and implement the following function (note it's **external** to the class!)

```
def reconstruct(root, iterator):
    """ Takes a root (i.e. 'a') and a sequence of tuples (node, branch, subnode)
        *in no particular order*
        i.e. ('b','R','c'), ('a','L','b'), ('a','R','b') …

        and RETURN a NEW BinaryTree reconstructed from such tuples

        - node and subnode are represented as node data
        - a branch is indicated by either 'L' or 'R'

        - MUST perform in O(n) where n is the length of the stream
          produced by the iterator
        - NOTE: you can read the sequence only once (you are given an iterator)
        - in case a branch is repeated (i.e. ('a','L','b') and ('a','L','c'))
          the new definition replaces old one
    """
```

Testing: `python3 -m unittest bin_tree_test.ReconstructTest`

Example:

[17]:
```
from bin_tree_sol import *
from bin_tree_test import bt

# note we explicitly pass an iterator just to make sure the implementation reads the␣
↪sequence only once
t = reconstruct('a', iter([('e','L','g'), ('h','R','i'), ('b','R','d'),
                           ('a','L','b'), ('d','L','f'), ('a','R','c'),
                           ('e','R','h'), ('c', 'L', 'e')]))
```

[18]:
```
from sciprog import draw_bt
draw_bt(t)
```

B3 Marvelous

Open `generic_tree.py` and edit this method:

```
def marvelous(self):
    """ MODIFIES each node data replacing it with the concatenation
        of its leaves data

        - MUST USE a recursive solution
        - assume node data is always a string
    """
```

Testing: `python3 -m unittest generic_tree_test.MarvelousTest`

Example: **NOTE: leaves may be any character!** Here we show them as uppercase but they might also be lowercase.

[20]:
```
from gen_tree_sol import *
from gen_tree_test import gt

t = gt('a',
          gt('b',
                gt('M'),
                gt('A'),
                gt('R')),
          gt('c',
                gt('e',
                      gt('V'),),
                gt('E')),
          gt('d',
```

(continues on next page)

(continued from previous page)

```
                gt('L'),
                gt('f',
                      gt('O'),
                      gt('U'),),
                gt('S')))
```

[21]:
```
print(t)
```

```
a
├b
│├M
│├A
│└R
├c
│├e
││└V
│└E
└d
 ├L
 ├f
 │├O
 │└U
 └S
```

[22]:
```
t.marvelous()
print(t)
```

```
MARVELOUS
├MAR
│├M
│├A
│└R
├VE
│├V
││└V
│└E
└LOUS
 ├L
 ├OU
 │├O
 │└U
 └S
```

[ ]:
```

```

2.1.22 Exam - Fri 11, Jun 2021

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

Part A - Trans-Atlantic Slave Trade

> **NOTICE: this part of the exam was ported to** softpython[107] **website**
>
> There you can find a more curated version (notice it may be longer than here)

Open Jupyter and start editing this notebook `exam-2021-06-11.ipynb`

Two centuries ago the shipping of enslaved Africans across the Atlantic was morally indistinguishable from shipping sugar or textiles. This migration experience covers an era of very dramatic shifts in perceptions of good and evil, which provided the Americas with a crucial labor force for their own economic development.

Data provider: The Trans-Atlantic Slave Trade Database. 2020. SlaveVoyages. https://www.slavevoyages.org. You are enouraged to explore the dataset with the very interesting online tool[108] they built, in particular check out the Maps[109] and Timelapse[110] tabs.

Data license:

- Historical data: The Trans-Atlantic Slave Trade Database. 2020. SlaveVoyages. https://www.slavevoyages.org (accessed June 9, 2020). License: Public domain (use restrictions do not apply).
- Imputed data: Estimates. 2020. SlaveVoyages. https://slavevoyages.org/assessment/estimates (accessed June 9, 2020). License: Creative Commons Attribution-Noncommercial 3.0 United States License.

A1 read_trade

Each line in slave-trade.csv represents a ship voyage from a purchase place to a landing place. Parse it with a csv reader and output a list of dictionaries, one per voyage according to the output excerpt.

- Each ship has a nation flag `NATINIMP`
- Each voyage has purchase place code `MJBYPTIMP` and a landing place code `MJSLPTIMP` with five digits format `xyzvt` that indicate a specific town: you **MUST** save more generic codes of the form `xyz00` which indicate broader regions.
- **WARNING 1**: convert to int **only** `VOYAGEID` and `YEARAM`, leave `MJBYPTIMP` and `MJSLPTIMP` as strings
- **WARNING 2:** some codes in `slave-trade.csv` have a space instead of a number, in those cases save code `00000`

[2]:
```
import pandas as pd
import numpy as np
df = pd.read_csv('slave-trade.csv', encoding='UTF-8')
df[df.VOYAGEID.isin([1, 2024, 2393, 4190])]
```

[107] https://en.softpython.org/projects/slave-trade/slave-trade-sol.html
[108] https://www.slavevoyages.org/voyage/database
[109] https://www.slavevoyages.org/voyage/database#maps
[110] https://www.slavevoyages.org/voyage/database#timelapse

[2]:
```
      VOYAGEID  YEARAM         NATINIMP MJBYPTIMP MJSLPTIMP
0            1    1817  Portugal/Brazil     60820     50299
2000      2024    1840           U.S.A.     60615     31399
2361      2393    1829    Spain/Uruguay     60212
4000      4190    1854           U.S.A.     60515     31301
```

Region labels: For each location you need to also save its label, which you can find in separate file region-codes.csv (load the file with a csv reader)

- **WARNING 1:** in `region-codes.csv` there are **only** codes in format `xyz00`
- **WARNING 2:** some region codes are missing, in those cases place label `'unknown'`

[3]:
```python
import pandas as pd
dfr = pd.read_csv('region-codes.csv', encoding='UTF-8', dtype=str)
dfr[dfr.Value.isin(['60800','60600','31300','50200','60500'])]
```

[3]:
```
    Value                                      Region
47  31300                                        Cuba
84  50200                                       Bahia
92  60500                                Bight of Benin
93  60600  Bight of Biafra and Gulf of Guinea islands
95  60800   Southeast Africa and Indian Ocean islands
```

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

[4]:
```python
import csv

def read_voyages(slave_trade_csv, region_codes_csv):


    with open(region_codes_csv, encoding='utf-8', newline='') as fregions:
        my_reader = csv.DictReader(fregions, delimiter=',')

        regions_db = {}
        for d in my_reader:
            regions_db[d['Value']] = d['Region']

    with open(slave_trade_csv, encoding='utf-8', newline='') as f:
        my_reader = csv.DictReader(f, delimiter=',')

        ret = []

        for d in my_reader:

            voyage = {}
            voyage['id'] = int(d['VOYAGEID'])
            voyage['year'] = int(d['YEARAM'])
            voyage['flag'] = d['NATINIMP']

            if  d['MJBYPTIMP'].strip():
                pur_reg = d['MJBYPTIMP'][:3] + '00'
                voyage['purchase_id'] = pur_reg
                if pur_reg in regions_db:
                    voyage['purchase_label'] = regions_db[pur_reg]
                else:
                    voyage['purchase_label'] = 'unknown'
```

(continues on next page)

(continued from previous page)

```
            else:
                voyage['purchase_id'] = '00000'
                voyage['purchase_label'] = 'unknown'

            if d['MJSLPTIMP'].strip():
                lan_reg = d['MJSLPTIMP'][:3] + '00'
                voyage['landing_id'] = lan_reg
                if lan_reg in regions_db:
                    voyage['landing_label'] = regions_db[lan_reg]
                else:
                    voyage['landing_label'] = 'unknown'
            else:
                voyage['landing_id'] = '00000'
                voyage['landing_label'] = 'unknown'

            ret.append(voyage)

        return ret


voyages_db = read_voyages('slave-trade.csv', 'region-codes.csv')

print('OUTPUT EXCERPT:')
from pprint import pformat
print('[\n' +',\n'.join([pformat(voyages_db[vid]) for vid in [0,2000,2361, 4000]]) +
↪',\n  .\n  .\n]')
```

```
OUTPUT EXCERPT:
[
{'flag': 'Portugal/Brazil',
 'id': 1,
 'landing_id': '50200',
 'landing_label': 'Bahia',
 'purchase_id': '60800',
 'purchase_label': 'Southeast Africa and Indian Ocean islands',
 'year': 1817},
{'flag': 'U.S.A.',
 'id': 2024,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60600',
 'purchase_label': 'Bight of Biafra and Gulf of Guinea islands',
 'year': 1840},
{'flag': 'Spain/Uruguay',
 'id': 2393,
 'landing_id': '00000',
 'landing_label': 'unknown',
 'purchase_id': '60200',
 'purchase_label': 'Sierra Leone',
 'year': 1829},
{'flag': 'U.S.A.',
 'id': 4190,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60500',
 'purchase_label': 'Bight of Benin',
 'year': 1854},
```

(continues on next page)

(continued from previous page)

```
  .
  .
]
```

[4]:
```
import csv

def read_voyages(slave_trade_csv, region_codes_csv):
    raise Exception('TODO IMPLEMENT ME !')

voyages_db = read_voyages('slave-trade.csv', 'region-codes.csv')

print('OUTPUT EXCERPT:')
from pprint import pformat
print('[\n' +',\n'.join([pformat(voyages_db[vid]) for vid in [0,2000,2361, 4000]]) +
↪',\n  .\n  .\n]')
```

```
OUTPUT EXCERPT:
[
{'flag': 'Portugal/Brazil',
 'id': 1,
 'landing_id': '50200',
 'landing_label': 'Bahia',
 'purchase_id': '60800',
 'purchase_label': 'Southeast Africa and Indian Ocean islands',
 'year': 1817},
{'flag': 'U.S.A.',
 'id': 2024,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60600',
 'purchase_label': 'Bight of Biafra and Gulf of Guinea islands',
 'year': 1840},
{'flag': 'Spain/Uruguay',
 'id': 2393,
 'landing_id': '00000',
 'landing_label': 'unknown',
 'purchase_id': '60200',
 'purchase_label': 'Sierra Leone',
 'year': 1829},
{'flag': 'U.S.A.',
 'id': 4190,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60500',
 'purchase_label': 'Bight of Benin',
 'year': 1854},
  .
  .
]
```

[5]:
```
# TESTING
from pprint import pformat; from expected_db import expected_db
for i in range(0, len(expected_db)):
    if expected_db[i] != voyages_db[i]:
        print('\nERROR at index', i, ':')
```

(continues on next page)

(continued from previous page)

```
        print('  ACTUAL:\n', pformat(voyages_db[i]))
        print('  EXPECTED:\n', pformat(expected_db[i]))
        break
```

A2 Deportation

For each link purchase -> landing place, count in how many voyages it was present, then draw result in networkx.

- as edge `weight` use a normalized value from 0.0 to 1.0 (maximal count found in the graph)
- show only edges with weight greater or equal to `min_weight`
- to display the graph from right to left, set `G.graph['graph']= {'rankdir':'RL'}`
- for networkx attributes see this example[111]

<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 networkx as nx
from sciprog import draw_nx

def show_deportation(voyages, min_weight):


    edges = {}

    for v in voyages_db:
        t = (v['purchase_id'], v['landing_id'])
        if t in edges:
            edges[t] += 1
        else:
            edges[t] = 1



    G = nx.DiGraph()
    G.graph['graph']= {'rankdir':'RL'}     # force horiz right to left layout

    mx = max(edges.values())
    for v in voyages_db:
        s = edges[(v['purchase_id'], v['landing_id'])]
        r = s / mx
        if r >= min_weight:
            G.add_node(v['purchase_id'], fontcolor='black', color='brown', label=v[
↪'purchase_label'])
            G.add_node(v['landing_id'], fontcolor='darkorange', color='orange',␣
↪label=v['landing_label'])
            G.add_edge(v['purchase_id'],v['landing_id'],
                       color="blue",
                       penwidth= 5 * r,
                       weight=r)

    draw_nx(G)
```

(continues on next page)

[111] https://en.softpython.org/graph-formats/graph-formats-sol.html#Fancy-networkx-graphs

(continued from previous page)

```
show_deportation(voyages_db, 0.09)
#show_deportation(voyages_db, 0.06)
```

```
COUNTS EXCERPT SOLUTION:
{
  ('60800', '50200') : 48,
  ('60700', '50200') : 1301,
  ('60700', '50400') : 2770,
  ('60800', '50400') : 443,
  ('60900', '50400') : 196,
    .
    .
}
```

[6]:
```
import networkx as nx
from sciprog import draw_nx

def show_deportation(voyages, min_weight):
    raise Exception('TODO IMPLEMENT ME !')
show_deportation(voyages_db, 0.09)
```

(continues on next page)

(continued from previous page)

```
#show_deportation(voyages_db, 0.06)
```

```
COUNTS EXCERPT SOLUTION:
{
  ('60800', '50200') : 48,
  ('60700', '50200') : 1301,
  ('60700', '50400') : 2770,
  ('60800', '50400') : 443,
  ('60900', '50400') : 196,
    .
    .
}
```

Barbados
Jamaica
Bahia
Americas
Saint-Domingue
Cuba
Pernambuco
unknown
Spanish Circum-Caribbean
Southeast Brazil
Amazonia
Gold Coast
Bight of Biafra and Gulf of Guinea islands
Bight of Benin
Other Africa
West Central Africa and St. Helena
Senegambia and off shore Atlantic
Southeast Africa and Indian Ocean islands

A3 The time to stop

Given a nation flag, plot inside `draw_time` the number of voyages per year done by ships belonging to that flag.

DO NOT call `plt.show` nor `plt.figure`

- we show some counts example but to calculate the data feel free to use any method you want
- **to associate a plot to a label, use i.e.** `plt.plot(xs, ys, label='France')`

<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 draw_time(voyages, flag):


    plt.title("Slave trade voyages SOLUTION")
    plt.xlabel('years')
    plt.ylabel('voyages')

    counts = {}
    for v in voyages:
        if v['flag'] == flag:
            y = v['year']
            if y in counts:
                counts[y] += 1
            else:
                counts[y] = 1



    xs = sorted(counts.keys())
    ys = [counts[x] for x in xs]
    plt.plot(xs, ys, label=flag)


fig = plt.figure(figsize=(15,6))
draw_time(voyages_db, 'France')
draw_time(voyages_db, 'U.S.A.')
draw_time(voyages_db, 'Great Britain')
plt.legend()
plt.show()
```

```
France COUNTS EXCERPT SOLUTION:
{
        1816:7,
       1819:30,
       1821:59,
       .
       .
              }
U.S.A. COUNTS EXCERPT SOLUTION:
{
        1821:1,
        1827:1,
        1837:2,
       .
```

(continues on next page)

(continued from previous page)

```
        .
                }
Great Britain COUNTS EXCERPT SOLUTION:
{
         1810:1,
         1809:1,
         1811:1,
        .
        .
                }
```

[7]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def draw_time(voyages, flag):
    raise Exception('TODO IMPLEMENT ME !')

fig = plt.figure(figsize=(15,6))
draw_time(voyages_db, 'France')
draw_time(voyages_db, 'U.S.A.')
draw_time(voyages_db, 'Great Britain')
plt.legend()
plt.show()
```

```
France COUNTS EXCERPT SOLUTION:
{
         1816:7,
        1819:30,
        1821:59,
        .
        .
                }
U.S.A. COUNTS EXCERPT SOLUTION:
{
         1821:1,
         1827:1,
```

(continues on next page)

(continued from previous page)

```
        1837:2,
       .
       .
              }
Great Britain COUNTS EXCERPT SOLUTION:
{
        1810:1,
        1809:1,
        1811:1,
       .
       .
              }
```

Part B

- Open Visual Studio Code and start editing the folder on your desktop

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.

```python
def my_fun(L):
    T = []
    N = len(L)
    for i in range(N//2):
        k = 0
        tmp = 0
        while k < 4:
```

(continues on next page)

(continued from previous page)

```
            tmp += L[i]*k
            k = k + 1
        T.insert(0,tmp)

    return T
```

B1.2 Graph

What is a depth first search (DFS) of a graph? Please answer briefly and then provide a possible DFS (illustrating the reasoning behind the answer) of the following graph starting from the node a

B2 - is_heap_stack

Open `bin_tree.py` and implement:

```
def is_heap_stack(self):
    """ A tree is a min heap if each node data is less or equal than its children␣
↪data.
        RETURN True if the tree is a min heap, False otherwise

        - DO *NOT* use recursion
        - implement it with a while and a stack (as a python list)
        - MUST run in O(n), where n is the tree size
    """
```

Testing: `python3 -m unittest bin_tree_test`

[10]:
```
from bin_tree_test import bt
from bin_tree_sol import *
t = bt(7,
        bt(9,
                bt(10,
                        bt(40)),
                bt(14)),
```

(continues on next page)

(continued from previous page)

```
        bt(20,
                None,
                bt(30,
                        bt(50),
                        bt(90))))

t.is_heap_stack()
```

[10]:
```
True
```

[11]:
```
t = bt(7,
        bt(9,
                bt(10,
                        bt(40)),
                bt(14)),
        bt(20,
                None,
                bt(30,
                        bt(11),
                        bt(90))))
t.is_heap_stack()
```

[11]:
```
False
```

B3 - sepel

Open `linked_list` and implement this method:

```
def sepel(self, el):
    """ Separates this list into two lists:

        - this list will have all nodes without el as data
        - the other list will contain all nodes with el as data

        - IMPORTANT: DO *NOT* create new nodes, REUSE existing ones!!
        - MUST execute in O(n), where n is the length of the list
    """
```

Testing: `python3 -m unittest linked_list_test`

Example:

[13]:
```
from linked_list_sol import *
la = LinkedList()
la.add('c')
la.add('e')
la.add('c')
la.add('d')
la.add('c')
la.add('c')
la.add('b')
la.add('a')
la.add('c')
```

[14]:
```
print(la)
```

```
LinkedList: c,a,b,c,c,d,c,e,c
```

[15]:
```
lb = la.sepel('c')
```

[16]:
```
print(la)
```

```
LinkedList: a,b,d,e
```

[17]:
```
print(lb)
```

```
LinkedList: c,c,c,c,c
```

2.1.23 Exam - Mon 12, Jul 2021

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

Part A - DOOM

Open Jupyter and start editing this notebook `exam-2021-07-12.ipynb`

The Union Aerospace Corporation (UAC) is the largest inter-planetary corporate entity in existence. While drilling Mars soil, UAC discovered that what human legends call Hell, is actually a dimension reachable from a portal buried in Mars ground. UAC promptly sends space marines into the portal to extract demonic creatures, which now intends to use as military weapons. You are asked to develop a software to map and simulate the entities inside the Mars base. The offer looks questionable, but they pay well, so you accept.

Historical note: DOOM[112] is a glorious videogame by Id Software which first introduced somewhat credible 3d graphics in the early 90s, plus lots of gore. Over time, a number of fans made extensions for it, and even serious projects were born like VizDoom[113] which allows developing AI bots that play DOOM using visual information.

A1 parse_map

Map data is provided in doom-map.udmf, in the UDMF text format (complete specification[114] is long, but we will only use a small subset of it)

We are interested in 3 categories of objects: *vertex*, *linedef*, and *thing*. Each of these objects has a numeric integer `id`, progressively numbered according to its position in the file starting from zero.

- a *vertex* holds coordinates float `x` ad `y`.
- a *linedef* describes a map segment and holds references to the ids of two verteces `v1` and `v2`
- a *thing* can be an entity in the map, like a monster or a space marine. A thing has float `x` and `y` coordinates, and an attribute called `type` (as int)
- ignore all other categories (like sidedef and sector) and attributes (sidefront, etc)
- whatever follows a `//` is a comment

Parse it **one line at a time** like this[115] and output a dictionary as in expected_map.py file.

[112] https://www.britannica.com/topic/Doom
[113] http://vizdoom.cs.put.edu.pl/
[114] https://github.com/coelckers/gzdoom/blob/master/specs/udmf.txt
[115] https://en.softpython.org/formats/formats-sol.html#1.-line-files

- **DO NOT** assume number of blank lines is constant
- **DO NOT** assume number of parameters is constant
- **DO NOT** perform mega `.replace` on the whole file (i.e. to make it look like a JSON)

<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_map(filepath):


    print("Parsing map", filepath)

    thing_counter = -1
    linedef_counter = -1
    vertex_counter = -1
    parsing = "" # "", "thing", "vertex", "linedef"
    things = []
    linedefs = []
    verteces = []

    with open(filepath, encoding='utf-8') as f:

        for line in f:
            pline = line.strip()
            if pline.startswith('//'):
                continue

            if pline.startswith("namespace="):
                continue

            if not pline:
                continue

            if pline == '{':
                continue

            if pline.startswith('thing'):
                thing_counter += 1
                parsing = "thing"
                thing = {}
            elif pline.startswith('vertex'):
                vertex_counter += 1
                parsing = "vertex"
                vertex = {}
            elif pline.startswith('linedef'):
                linedef_counter += 1
                parsing = "linedef"
                linedef = {}
            elif pline == '}':
                if parsing == "thing":
                    things.append(thing)
                elif parsing == "linedef":
                    linedefs.append(linedef)
                elif parsing == "vertex":
                    verteces.append(vertex)
                parsing=""
```

(continues on next page)

(continued from previous page)

```
            if parsing == "thing":
                thing["id"] = thing_counter
                i = pline.find('=')
                attr = pline[:i]
                value = pline[i+1:].rstrip(';')
                if attr in ['x','y'] :
                    thing[attr] = float(value)
                elif attr in ['type']:
                    thing[attr] = int(value)
            elif parsing == "linedef":
                linedef["id"] = linedef_counter
                i = pline.find('=')
                attr = pline[:i]
                value = pline[i+1:].rstrip(';')
                if attr in ['v1','v2'] :
                    linedef[attr] = int(value)
            elif parsing == "vertex":
                vertex["id"] = vertex_counter
                i = pline.find('=')
                attr = pline[:i]
                value = pline[i+1:].rstrip(';')
                if attr in ['x','y'] :
                    vertex[attr] = float(value)

        return {'thing': things,
                'linedef' : linedefs,
                'vertex' : verteces,
                'filepath': filepath}


doom_map = parse_map('doom-map.udmf')
doom_map
```

```
Parsing map doom-map.udmf
{'filepath': 'doom-map.udmf',
 'linedef': [{'id': 0, 'v1': 4, 'v2': 1},
             {'id': 1, 'v1': 1, 'v2': 2},
             {'id': 2, 'v1': 2, 'v2': 3},
             {'id': 3, 'v1': 3, 'v2': 0},
             {'id': 4, 'v1': 0, 'v2': 4},
             {'id': 5, 'v1': 5, 'v2': 6},
             {'id': 6, 'v1': 6, 'v2': 7},
             {'id': 7, 'v1': 7, 'v2': 8},
             {'id': 8, 'v1': 8, 'v2': 5}],
 'thing': [{'id': 0, 'type': 1, 'x': -352.0, 'y': 0.0},
           {'id': 1, 'type': 71, 'x': -160.0, 'y': 128.0},
           {'id': 2, 'type': 67, 'x': -128.0, 'y': -64.0},
           {'id': 3, 'type': 71, 'x': 70.0, 'y': 200.0}],
 'vertex': [{'id': 0, 'x': -480.0, 'y': 192.0},
            {'id': 1, 'x': 160.0, 'y': 192.0},
            {'id': 2, 'x': -64.0, 'y': -160.0},
            {'id': 3, 'x': -384.0, 'y': -160.0},
            {'id': 4, 'x': -160.0, 'y': 416.0},
            {'id': 5, 'x': -224.0, 'y': 64.0},
            {'id': 6, 'x': -256.0, 'y': 256.0},
```

(continues on next page)

(continued from previous page)

```
            {'id': 7, 'x': -32.0, 'y': 256.0},
            {'id': 8, 'x': -32.0, 'y': 32.0}]}
```

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

doom_map = parse_map('doom-map.udmf')
doom_map
```

```
Parsing map doom-map.udmf
{'filepath': 'doom-map.udmf',
 'linedef': [{'id': 0, 'v1': 4, 'v2': 1},
             {'id': 1, 'v1': 1, 'v2': 2},
             {'id': 2, 'v1': 2, 'v2': 3},
             {'id': 3, 'v1': 3, 'v2': 0},
             {'id': 4, 'v1': 0, 'v2': 4},
             {'id': 5, 'v1': 5, 'v2': 6},
             {'id': 6, 'v1': 6, 'v2': 7},
             {'id': 7, 'v1': 7, 'v2': 8},
             {'id': 8, 'v1': 8, 'v2': 5}],
 'thing': [{'id': 0, 'type': 1, 'x': -352.0, 'y': 0.0},
           {'id': 1, 'type': 71, 'x': -160.0, 'y': 128.0},
           {'id': 2, 'type': 67, 'x': -128.0, 'y': -64.0},
           {'id': 3, 'type': 71, 'x': 70.0, 'y': 200.0}],
 'vertex': [{'id': 0, 'x': -480.0, 'y': 192.0},
            {'id': 1, 'x': 160.0, 'y': 192.0},
            {'id': 2, 'x': -64.0, 'y': -160.0},
            {'id': 3, 'x': -384.0, 'y': -160.0},
            {'id': 4, 'x': -160.0, 'y': 416.0},
            {'id': 5, 'x': -224.0, 'y': 64.0},
            {'id': 6, 'x': -256.0, 'y': 256.0},
            {'id': 7, 'x': -32.0, 'y': 256.0},
            {'id': 8, 'x': -32.0, 'y': 32.0}]}
```

[3]:
```
# TESTING
from pprint import pformat; from expected_map import expected_map
for category in expected_map.keys():
    if category not in doom_map:
        print('\nERROR: MISSING category', category); break
    if category != 'filepath' and len(expected_map[category]) != len(doom_
↪map[category]):
        print('\nERROR: DIFFERENT lengths for category', category); break
    for some_id in range(len(expected_map[category])):
        if category != 'filepath' and expected_map[category][some_id] != doom_
↪map[category][some_id]:
            print('\nERROR at category', category, 'id:',some_id)
            print('  ACTUAL:\n', pformat(doom_map[category][some_id]))
            print('  EXPECTED:\n', pformat(expected_map[category][some_id]))
            break
```

A2 simulate

UAC is particularly interested in the tactical value of Pain Elementals, which can generate an apparent infinite amount of Lost Souls. UAC has estimated some parameters of these creatures, and wants you to devise a simulation of their behaviour.

PRINT OUTPUT as in the example

MODIFY provided doom map like so (for full modified map see `expected_sim.py`):

- The simulation is done in discrete steps of one second each, starting at `t=0`
- Every `spawn_time` seconds, each Pain Elemental generates a new Lost Soul, which must be added to things
- Each second a Lost Soul moves of up to +/- `m` integer units along both x axis and/or y axis. The x and y deltas are uniformly distributed independent random variables (hint: to calculate them use `random.randint(a,b)`)
- For simplicity we assume Lost Souls can pass through walls, but we still impose that they **cannot leave the smallest rectangle enclosing the map**.
- At each step, MODIFY every Lost Soul thing by updating its x and y, and keep track of location past values (x,y) as a list in a new parameter `trace` you will associate to the thing. **NOTE**: `trace` holds **only** past values, never the current one.
- Assume all other entities stay still

<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 random

LOST_SOUL_TYPE = 3006
PAIN_ELEMENTAL_TYPE = 71

def simulate(dmap, duration, m, spawn_time):
    #VERY IMPORTANT: DO *NOT* REMOVE THIS LINE: IT INITIALIZES THE PSEUDO RANDOM
    # NUMBER GENERATOR SO DIFFERENT RUNS ALWAYS GIVE THE SAME 'RANDOM' SEQUENCE
    random.seed(1)



    print("Simulating...")

    step = 1
    things = dmap["thing"]

    import math
    minx, miny, maxx, maxy  = math.inf, math.inf, -math.inf, -math.inf


    for vertex in dmap["vertex"]:
        minx = min(minx, vertex["x"])
        miny = min(miny, vertex["y"])
        maxx = max(maxx, vertex["x"])
        maxy = max(maxy, vertex["y"])

    print('Map boundaries:', "  minx", minx, "  miny", miny, "  maxx", maxx, "  maxy",
↪ maxy)

    pain_els = []
```

(continues on next page)

(continued from previous page)

```
    for thing in things:
        if thing["type"] == PAIN_ELEMENTAL_TYPE:
            pain_els.append(thing)

        thing["trace"] = []

    for t in range(0,duration + 1):

        for thing in things:
            if thing["type"] == LOST_SOUL_TYPE:
                thing["trace"].append((thing["x"], thing["y"]))

        for thing in things:
            if thing["type"] == LOST_SOUL_TYPE:
                ntx = thing["x"] + random.randint(-m,m)
                nty = thing["y"] + random.randint(-m,m)

                if ntx < minx:
                    ntx = minx
                if ntx > maxx:
                    ntx = maxx
                if nty < miny:
                    nty = miny
                if nty > maxy:
                    nty = maxy

                thing["x"] = ntx
                thing["y"] = nty


        if t % spawn_time == 0:
            if t > 0: # no span at beginning...
                for pain_el in pain_els:
                    nid = len(things)
                    print("t =",t, "Pain Elemental id =", pain_el["id"], ": Spawning␣
↪Lost Soul id =", nid)
                    things.append({"id": nid,
                                   "x": pain_el["x"],
                                   "y": pain_el["y"],
                                   "type": LOST_SOUL_TYPE,
                                   "trace" : []})

    print("Elapsed time:", t, "seconds")


doom_map = parse_map('doom-map.udmf')
simulate(doom_map, 31,65,8)  # return *nothing* !

print("\nExample of last generated Lost Soul:\n",doom_map["thing"][-1])
```

```
Parsing map doom-map.udmf
Simulating…
Map boundaries:   minx -480.0   miny -160.0   maxx 160.0   maxy 416.0
t = 8 Pain Elemental id = 1 : Spawning Lost Soul id = 4
t = 8 Pain Elemental id = 3 : Spawning Lost Soul id = 5
t = 16 Pain Elemental id = 1 : Spawning Lost Soul id = 6
t = 16 Pain Elemental id = 3 : Spawning Lost Soul id = 7
```

(continues on next page)

(continued from previous page)

```
t = 24 Pain Elemental id = 1 : Spawning Lost Soul id = 8
t = 24 Pain Elemental id = 3 : Spawning Lost Soul id = 9
Elapsed time: 31 seconds

Example of last generated Lost Soul:
 {'id': 9, 'x': 19.0, 'y': 157.0, 'type': 3006, 'trace': [(70.0, 200.0), (50.0, 181.
↪0), (53.0, 144.0), (104.0, 161.0), (66.0, 160.0), (115.0, 224.0), (82.0, 213.0)]}
```

[4]:
```
import random

LOST_SOUL_TYPE = 3006
PAIN_ELEMENTAL_TYPE = 71

def simulate(dmap, duration, m, spawn_time):
    #VERY IMPORTANT: DO *NOT* REMOVE THIS LINE: IT INITIALIZES THE PSEUDO RANDOM
    # NUMBER GENERATOR SO DIFFERENT RUNS ALWAYS GIVE THE SAME 'RANDOM' SEQUENCE
    random.seed(1)

    raise Exception('TODO IMPLEMENT ME !')

doom_map = parse_map('doom-map.udmf')
simulate(doom_map, 31,65,8)  # return *nothing* !

print("\nExample of last generated Lost Soul:\n",doom_map["thing"][-1])
```

```
Parsing map doom-map.udmf
Simulating…
Map boundaries:   minx -480.0   miny -160.0   maxx 160.0   maxy 416.0
t = 8 Pain Elemental id = 1 : Spawning Lost Soul id = 4
t = 8 Pain Elemental id = 3 : Spawning Lost Soul id = 5
t = 16 Pain Elemental id = 1 : Spawning Lost Soul id = 6
t = 16 Pain Elemental id = 3 : Spawning Lost Soul id = 7
t = 24 Pain Elemental id = 1 : Spawning Lost Soul id = 8
t = 24 Pain Elemental id = 3 : Spawning Lost Soul id = 9
Elapsed time: 31 seconds

Example of last generated Lost Soul:
 {'id': 9, 'x': 19.0, 'y': 157.0, 'type': 3006, 'trace': [(70.0, 200.0), (50.0, 181.
↪0), (53.0, 144.0), (104.0, 161.0), (66.0, 160.0), (115.0, 224.0), (82.0, 213.0)]}
```

A3 plot_map

Draw the map:

- use filename as title
- there's no need to plot verteces dots
- only plot entity names (inserting newlines) - to get them import provided entities_db.py which maps an entity type to its data.
- make it fancy following this example[116]: plot dark background, and Lost Soul traces with dark color for old trace and bright for recent using `alpha` parameter

[116] https://en.softpython.org/visualization/visualization-sol.html#Fancy-plots

EXTRA (was not required during exam): plot images taking file names from `entities_db.py` and files from `img/` folder

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

[5]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from entities_db import entities_db

def plot_map(entities, doom_map):



    plt.rcParams['axes.facecolor'] = 'black'
    fig = plt.figure(figsize=(20,20))
    plt.title(doom_map["filepath"])

    for thing in doom_map["thing"]:
        if "trace" in thing:
            for i in range(len(thing["trace"])-1):
                plt.plot([thing["trace"][i][0], thing["trace"][i+1][0]],
                         [thing["trace"][i][1], thing["trace"][i+1][1]],
                         color="yellow",
                         linewidth=2,
                         alpha=max(0.4,i/len(thing["trace"])))
            if len(thing["trace"]) > 0:
                ptr = thing["trace"][-1]
                plt.plot([ptr[0], thing["x"]] , [ptr[1], thing["y"]], color="yellow")

    for linedef in doom_map["linedef"]:
        #print(linedef)
        v1 = doom_map["vertex"][linedef["v1"]]
        v2 = doom_map["vertex"][linedef["v2"]]
        plt.plot([v1["x"], v2["x"]], [v1["y"], v2["y"]], linewidth=3, color="red")

    # images
    from matplotlib.offsetbox import OffsetImage, AnnotationBbox
    ax=fig.gca()
    for thing in doom_map["thing"]:
        thing_img = plt.imread('img/' + entities[thing["type"]]["class"] + '.gif')
        abb = AnnotationBbox(
            OffsetImage(thing_img, zoom=1.5),
            (thing["x"], thing["y"]),
            frameon=False
        )
        ax.add_artist(abb)

    #text
    #super rough image size estimation
    dim = 0.07 * abs(max([v["y"] for v in doom_map["vertex"]]) - min([v["y"] for v in␣
↪doom_map["vertex"]]))

    for thing in doom_map["thing"]:
        plt.text(thing["x"] - 5,
                 thing["y"] - dim,
                 entities_db[thing["type"]]["name"].replace(' ', '\n'),
                 fontweight='bold',
```

(continues on next page)

(continued from previous page)

```
                 color="white")

    plt.show()

plot_map(entities_db, doom_map)
```

[5]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from entities_db import entities_db
```

(continues on next page)

(continued from previous page)

```
def plot_map(entities, doom_map):
    raise Exception('TODO IMPLEMENT ME !')

plot_map(entities_db, doom_map)
```

Part B

- Open Visual Studio Code and start editing the folder on your desktop
- For running tests: open *Accessories -> Terminal*

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 `my_fun` function, explaining your reasoning. **NOTE**: please notice that it calls the function `my_fun2`

```
def my_fun2(L):
    n = len(L)
    tmp = []
    for i in range(n):
        for j in range(n):
            tmp.append(L[i]/(1+L[j]))
    return tmp

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(my_fun2(L1))
        b = my_fun(L2) + max(my_fun2(L2))
        return a - b
```

B1.2 Hash

What is a hash function? Which python data structures use hash functions?

B2 PyraStack

You are given a PyraStack class which holds a list of lists called `_rows`. Internal lists only contain – character. Implement this method:

```
def drop(self, w):
    """ Drops a block of size w on the pyrastack, trying to place it on
        the top leftmost position without having missing blocks below.
        If top row is not feasible, scans for the first available leftmost
        place which can fully accomodate the block.

        - if w is negative, raise ValueError
        - if w is zero, no change is made

        - MUST run in O(h + w) where h is the stack height
    """
```

Example (rows are **printed bottom-up**):

[7]:
```
from pyrastack_sol import *

s = PyraStack()
s.drop(10)
s.drop(7)
s.drop(5)
s.drop(2)
s.drop(3)
s.drop(6)
s.drop(6)
s.drop(1)
s.drop(7)
from pprint import pprint
print("_rows are:");pprint(s._rows, width=150)
```

```
DEBUG:  Dropped 10, pyrastack is:
        ----------
DEBUG:  Dropped 7, pyrastack is:
        -------
        ----------
DEBUG:  Dropped 5, pyrastack is:
        -----
        -------
        ----------
DEBUG:  Dropped 2, pyrastack is:
        --
        -----
        -------
        ----------
DEBUG:  Dropped 3, pyrastack is:
        -----
        -----
        -------
        ----------
DEBUG:  Dropped 6, pyrastack is:
        -----
        -----
        -------
        ----------------
DEBUG:  Dropped 6, pyrastack is:
        -----
        -----
        -------------
        ----------------
DEBUG:  Dropped 1, pyrastack is:
        -
        -----
        -----
        -------------
        ----------------
DEBUG:  Dropped 7, pyrastack is:
        -
        -----
        ------------
        -------------
        ----------------
```

(continues on next page)

(continued from previous page)

```
_rows are:
[['-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-'],
 ['-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-'],
 ['-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-'],
 ['-', '-', '-', '-', '-'],
 ['-']]
```

B3 union_rec

Open `bin_tree.py` and implement following method:

```
def union_rec(self, other):
    """ Supposing this is a binary tree of integers, takes another binary tree
        and MODIFIES self so it becomes the union of the two.

        Imagine to overlay the two trees, and:
        - whenever two nodes occupy the same position, the self one is updated
            by summing the corresponding node data from other
        - if other has more nodes than self, create corresponding NEW nodes in self

        - a recursive solution is acceptable
        - DO *NOT* share nodes between the trees
        - DO *NOT* throw away existing nodes in self
        - MUST run in O(max(n,m)) where n,m are the number of nodes in self
          and other
    """
```

Testing: `python3 -m unittest bin_tree_test.UnionRecTest`

Example:

[9]:
```
from bin_tree_sol import *
from bin_tree_test import bt
```

[10]:
```
ta = bt(3,
        bt(5,
                bt(7,
                    None,
                    bt(1,
                        bt(17)))),
        bt(6))

tb = bt(8,
        bt(10,
                bt(9,
                    bt(13))),
        bt(12,
                bt(11,
                        None,
                        bt(2)),
                bt(4)))
ta.union_rec(tb)
```

[11]:
```
print(ta)
```

```
11
├15
│├16
│└13
│└1
│ ├17
│ └
└18
 ├11
 │├
 │└2
 └4
```

[ ]:

```

```

2.1.24 Exam - Mon 06, Sep 2021

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

Part A - I CHING Divination

> **NOTICE: this part of the exam was ported to** softpython[117] **website**
>
> There you can find a more curated version (notice it may be longer than here)

Open Jupyter and start editing this notebook `exam-2021-09-06.ipynb`

The I Ching, or Book of Changes, is a chinese divination manual and philosophical text which is believed to be one of the world's oldest books, dating from over 3,000 years ago.

The great mathematician Gottfried Wilhelm Leibniz (1646 - 1716) is considered the first information theorist, and extensively documented the binary numeral system. Leibniz was also interested in Chinese culture, and saw in the I Ching diagrams[118] showing solid and broken lines called yin and yang, which progressed in a sequence: that was unmistakably a binary encoding.

You will parse a dataset of hexagrams and develop a divinator software which will predict the outcome of your exam.

Data source: Wikipedia, July 2021, Bagua page[119]

Yin and yang: Yin and yang are represented by lines:

name	line	bit
yin	`- -`	0
yang	`---`	1

Trigrams: Different constructions of three yin and yang lines lead to 8 trigrams. We can express a trigram as a sequence of bits, reading lines from bottom to top. For example *Fire* is `101`, *Thunder* is `100`.

[117] https://en.softpython.org/projects/iching/iching-sol.html
[118] https://en.wikipedia.org/wiki/Gottfried_Wilhelm_Leibniz#Sinophile
[119] https://en.wikipedia.org/wiki/Bagua

乾 Qián ☰ Heaven	坤 Kūn ☷ Earth	震 Zhèn ☳ Thunder	坎 Kǎn ☵ Water	艮 Gèn ☶ Mountain	巽 Xùn ☴ Air	離 Lí ☲ Fire	兌 Duì ☱ Lake

Hexagrams: Combining a lower trigram with an upper trigram leads to 64 hexagrams. Each hexagram can be represented as a sequence of bits and the outcome of a divination. For example trigrams *Fire* (lower) and *Thunder* (upper) gives outcome hexagram *Abounding*: `101100`

Upper → Lower ↓	乾 Qián ☰ Heaven	坤 Kūn ☷ Earth	震 Zhèn ☳ Thunder	坎 Kǎn ☵ Water	艮 Gèn ☶ Mountain	巽 Xùn ☴ Air	離 Lí ☲ Fire	兌 Duì ☱ Lake
乾 Qián ☰ Heaven	01 Force	11 Pervading	34 Great Invigorating	05 Attending	26 Great Accumulating	09 Small Harvest	14 Great Possessing	43 Displacement
坤 Kūn ☷ Earth	12 Obstruction	02 Field	16 Providing-For	08 Grouping	23 Stripping	20 Viewing	35 Prospering	45 Clustering
震 Zhèn ☳ Thunder	25 Innocence	24 Returning	51 Shake	03 Sprouting	27 Swallowing	42 Augmenting	21 Gnawing Bite	17 Following
坎 Kǎn ☵ Water	06 Arguing	07 Leading	40 Deliverance	29 Gorge	04 Enveloping	59 Dispersing	64 Before Completion	47 Confining
艮 Gèn ☶ Mountain	33 Retiring	15 Humbling	62 Small Exceeding	39 Limping	52 Bound	53 Infiltrating	56 Sojourning	31 Conjoining
巽 Xùn ☴ Wind	44 Coupling	46 Ascending	32 Persevering	48 Welling	18 Correcting	57 Ground	50 Holding	28 Great Exceeding
離 Lí ☲ Fire	13 Concording People	36 Intelligence Hidden	55 Abounding	63 Already Fording	22 Adorning	37 Dwelling People	30 Radiance	49 Skinning
兌 Duì ☱ Lake	10 Treading	19 Nearing	54 Converting the Maiden	60 Articulating	41 Diminishing	61 Inner Truth	38 Polarising	58 Open

A1 load_db

Parse iching.csv and output a dictionary mapping each sequence to a dictionary with all the information you can extract. Use CSV reader.

- in headers and first column you will find a bit sequence like `011`
- in body cells, you will **not** find a bit sequence: you will have to determine it according to the corresponding tri-sequences from the header and first column
- note for hexagrams you must extract **only** `name-en`, ignore the decimal numbers
- complete expected output is in file expected_iching_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>

[2]:
```
import csv

def load_db(filepath):

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

        linear = []

        for s in header[1:]:
            diz = {}
            tokens = s.split('\n')
            tks = tokens[0].split('\xa0')
            diz['name-en'] = tokens[2]
            diz['name-ch'] = tks[0]
            diz['spelling'] = tks[1]
            code = tokens[1]
            ret[code] = diz
            linear.append(code)

        i = 1
        for row in my_reader:
            for j in range(1,len(row)):
                tokens = row[j].replace('\n','').split()
                num = int(tokens[0])
                bottom = linear[i-1]
                upper = linear[j-1]
                ret[bottom + upper] = {
                    'name-en':' '.join(tokens[1:])
                }
            i += 1
        return ret


iching_db = load_db('iching.csv')
iching_db
```

[2]:
```
import csv

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

iching_db = load_db('iching.csv')
iching_db
```

```
EXERPT:

{'111': {'name-en': 'Heaven', 'name-ch': '□', 'spelling': 'Qián'}
 '000': {'name-en': 'Earth', 'name-ch': '□', 'spelling': 'Kūn'}
 '100': {'name-en': 'Thunder', 'name-ch': '□', 'spelling': 'Zhèn'}
 '010': {'name-en': 'Water', 'name-ch': '□', 'spelling': 'Kǎn'}
 '001': {'name-en': 'Mountain', 'name-ch': '□', 'spelling': 'Gèn'}
 '011': {'name-en': 'Air', 'name-ch': '□', 'spelling': 'Xùn'}
 '101': {'name-en': 'Fire', 'name-ch': '□', 'spelling': 'Lí'}
 '110': {'name-en': 'Lake', 'name-ch': '□', 'spelling': 'Duì'}
 '111111': {'name-en': 'Force'}
 '111000': {'name-en': 'Pervading'}
 '111100': {'name-en': 'Great Invigorating'}
 '111010': {'name-en': 'Attending'}
 '111001': {'name-en': 'Great Accumulating'}
 '111011': {'name-en': 'Small Harvest'}
 '111101': {'name-en': 'Great Possessing'}
     .
     .
}
```

[4]:
```
# EXECUTE FOR TESTING
from pprint import pformat; from expected_iching_db import expected_iching_db
for seq in expected_iching_db.keys():
    if seq not in iching_db: print('\nERROR: MISSING sequence', seq); break
    for k in expected_iching_db[seq]:
        if k not in iching_db[seq]:
            print('\nERROR at sequence', seq,'\n\n   MISSING key:', k); break
        if expected_iching_db[seq][k] != iching_db[seq][k]:
            print('\nERROR at sequence', seq, 'key:',k)
            print('  ACTUAL:\n', pformat(iching_db[seq][k]))
            print('  EXPECTED:\n', pformat(expected_iching_db[seq][k]))
            break
```

A2 divine

A divination is done by flipping 3 coins to determine the bottom trigram (**bottom up order**), flipping other three coins for the upper trigram (again **bottom up order**), and then the union gives the resulting hexagram. Write a function that PRINTS the process as in the example and RETURNS a string of bits representing the resulting hexagram

HINT: to flip coins use `random.randint(0,1)`

WARNING: DOUBLE CHECK THE ORDER IN WHICH LINES ARE VISUALIZED!

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

[5]:
```
import random

def divine(iching, question):

    #THE SEED DETERMINES FOLLOWING randint RESULTS
    random.seed(109)       # Abounding
                           # Thunder
                           # Fire

    #IMPORTANT: try also this seed to check lines visualization order
    #random.seed(1)
    #
    # Infiltrating 001011
    # Mountain ---
    #          ---
    #          - -
    # Air      ---
    #          - -
    #          - -


    print()
    print("Dear stranger, welcome to SCIPROG I CHING ⸮⸮ DIVINATOR")
    print()
    print("Tell me your question...")
    print()
    print('       ', question)
    print()

    def get_trigram(part):
        lst = []
        stack = []
        for i in range(3):
            r = random.randint(0,1)
            kind = 'yang' if r else ' yin'
            line = '---' if r else '- -'
            coin = "'heads'" if r else "'tails'"
            print('The coin says', coin, ': we get a', kind, line)
            stack.append(line)
            lst.append(str(r))
        stack.reverse()
        digits = ''.join(lst)
        print()
        print("The sacred", part, "trigram is:",)
        print()

        print(iching[digits]['name-en'])
        print()
        print('   ' + '\n   '.join(stack))

        return (stack, ''.join(lst))

    bottom = get_trigram('bottom')
    print()
    upper = get_trigram('upper')

    print()
```

(continues on next page)

(continued from previous page)

```
    print('The final response hexagram is...')
    print()
    print(iching[bottom[1] + upper[1]]['name-en'])
    print()
    print('   ' + '\n   '.join(upper[0] + bottom[0]))

    return bottom[1] + upper[1]


divination = divine(iching_db, "Will I pass the exam?")
print("\nReturned:", divination)
```

```
Dear stranger, welcome to SCIPROG I CHING ⸮⸮ DIVINATOR

Tell me your question…

        Will I pass the exam?

The coin says 'heads' : we get a yang ---
The coin says 'tails' : we get a  yin - -
The coin says 'heads' : we get a yang ---

The sacred bottom trigram is:

Fire

   ---
   - -
   ---

The coin says 'heads' : we get a yang ---
The coin says 'tails' : we get a  yin - -
The coin says 'tails' : we get a  yin - -

The sacred upper trigram is:

Thunder

   - -
   - -
   ---

The final response hexagram is…

Abounding

   - -
   - -
   ---
   ---
   - -
   ---

Returned: 101100
```

[5]:
```
import random

def divine(iching, question):

    #THE SEED DETERMINES FOLLOWING randint RESULTS
    random.seed(109)       # Abounding
                           # Thunder
                           # Fire

    #IMPORTANT: try also this seed to check lines visualization order
    #random.seed(1)
    #
    # Infiltrating 001011
    # Mountain ---
    #          ---
    #          - -
    # Air      ---
    #          - -
    #          - -

    raise Exception('TODO IMPLEMENT ME !')

divination = divine(iching_db, "Will I pass the exam?")
print("\nReturned:", divination)
```

```
Dear stranger, welcome to SCIPROG I CHING ⍰⍰ DIVINATOR

Tell me your question…

        Will I pass the exam?

The coin says 'heads' : we get a yang ---
The coin says 'tails' : we get a  yin - -
The coin says 'heads' : we get a yang ---

The sacred bottom trigram is:

Fire

   ---
   - -
   ---

The coin says 'heads' : we get a yang ---
The coin says 'tails' : we get a  yin - -
The coin says 'tails' : we get a  yin - -

The sacred upper trigram is:

Thunder

   - -
   - -
   ---

The final response hexagram is…
```

(continues on next page)

(continued from previous page)

```
Abounding

   - -
   - -
   ---
   ---
   - -
   ---

Returned: 101100
```

A3 plot_divination

Given a divination as a string of bits, plot the divination.

- first draw the lines, then the rest if you have time.
- make it fancy with these examples[120]
- to center text you can use these parameters: `ha='center', va='center'`

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

[6]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_divination(iching, question, divination):


    fig = plt.figure(figsize=(10,5))

    plt.xlim(0,750)
    plt.ylim(0,700)

    xl = 150
    yd = 50
    segw = 100
    midx = 400

    def plot_trigram(seq, yl):
        plt.text(xl-35,
                 yl + yd*2,
                 iching[seq]['name-en'],
                 fontsize=25,
                 fontweight='bold',
                 color="darkgray",
                 ha='center',
                 va='center')

        lw = 15
        for i in range(3):
            h = yl + yd*(i+1)
            if seq[i] == '0':
                plt.plot([xl + segw,xl + segw*2], [h,h],
```

(continues on next page)

[120] https://en.softpython.org/visualization/visualization-sol.html#Fancy-plots

(continued from previous page)

```
                        color='black',
                        linewidth=lw)

                plt.plot([xl + segw*3, xl + segw*4], [h,h],
                        color='black',
                        linewidth=lw)
            else:
                plt.plot([xl + segw,xl + segw*4], [h,h],
                        color='black',
                        linewidth=lw)

    plt.text(midx,
             570,
             question,
             fontsize=26,
             fontweight='bold',
             color="CornflowerBlue",
             ha='center')

    plot_trigram(divination[:3], 0)
    plot_trigram(divination[3:], 170)

    plt.text(midx,
             420,
             iching[divination]['name-en'],
             fontsize=32,
             fontweight='bold',
             color="orange",
             ha='center')

    plt.show()


plot_divination(iching_db, "Will I pass Sciprog exam?", '101100')  # Abounding
#plot_divination(iching_db, "Will I pass Sciprog exam?", '111011') # Small Harvest
#plot_divination(iching_db, "Will I pass Sciprog exam?",'001011')  # Infiltrating
```

[6]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def plot_divination(iching, question, divination):
    raise Exception('TODO IMPLEMENT ME !')

plot_divination(iching_db, "Will I pass Sciprog exam?", '101100')  # Abounding
#plot_divination(iching_db, "Will I pass Sciprog exam?", '111011') # Small Harvest
#plot_divination(iching_db, "Will I pass Sciprog exam?",'001011')  # Infiltrating
```

Part B

- Open Visual Studio Code and start editing the folder on your desktop

B1 Theory

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

B1.1 Complexity

1.A: Given a list `L` of n elements, please compute the asymptotic computational complexity of the `my_fun` function, explaining your reasoning.

```
def my_fun(L):
    tmp = [1 for x in L]
    n = sum(tmp)

    for i in range(1, int(n/2)+1):
        if L[i-1] != L[-i]:
            return False
        else:
            return True
```

1.B: Do you have any idea to improve this code a little bit?

B1.2 nlogn

What do we mean when we say that an algorithm has asymptotic computational complexity $O(nlogn)$? What do we have to do to prove that an algorithm has asymptotic computational complexity $O(nlogn)$?

B2 Train race

An important train race is taking place in Steam Land. Each train has a length and a velocity. A train is represented as a sequence of asterisks. A *train path* is a list which holds all the train asterisks, and, if the train has moved *m* positions so far, the path also holds *m* dashes − before the asterisks.

Open the file `train_race.py` and implement methods from class `TrainRace`, in particular:

```
def step(self):
    """ Steps the simulation by moving each train toward right
        by a number of cells given by its velocity.


        *** MUST run in O(v) where v is the sum of all velocities

        *** Complexity MUST *NOT* depend on train length nor dashes length

        *** For simplicity, ASSUME velocity is always
            less or equal than train length

        ********      HAVE YOU READ THE REQUIREMENTS ABOVE ?   ********
    """
```

WARNING: AVOID EXPENSIVE LIST METHODS / OPERATORS

Passing tests is easy, the hard part is to make it *fast*: will your program run fast with a train of one million asterisks? And with a train which made a million steps?

Testing: `python3 -m unittest train_race_test.VelocityLessOrEqualThanTrainSizeTest`

Extra (not required during the exam): make it work also when velocity can be greater than train size, then test with:

`python3 -m unittest train_race_test.VelocityGreaterThanTrainSizeTest`

Example:

[8]:
```
from train_race_sol import *

       #train  lengths     velocities
tr = TrainRace([5,3,6,3],  [2,1,3,2])
```

[9]:
```
tr.get_paths()
```

[9]:
```
[['*', '*', '*', '*', '*'],
 ['*', '*', '*'],
 ['*', '*', '*', '*', '*', '*'],
 ['*', '*', '*']]
```

[10]:
```
tr.step()   # returns NOTHING!
```

[11]:
```
tr.get_paths()
```

[11]:
```
[['-', '-', '*', '*', '*', '*', '*'],
 ['-', '*', '*', '*'],
 ['-', '-', '-', '*', '*', '*', '*', '*', '*'],
 ['-', '-', '*', '*', '*']]
```

[12]:
```
tr.step()
```

[13]:
```
tr.get_paths()
```

[13]:
```
[['-', '-', '-', '-', '*', '*', '*', '*', '*'],
 ['-', '-', '*', '*', '*'],
 ['-', '-', '-', '-', '-', '-', '*', '*', '*', '*', '*', '*'],
 ['-', '-', '-', '-', '*', '*', '*']]
```

B3 linked algebra

Open `linked_list.py` and edit this method:

```
def linalg(self):
    """ Assume nodes hold data as a string "kc" where k is a single digit
        and c any character.

        MODIFY the linked list by stripping the k from original nodes,
        and inserting k-1 new nodes next to each node.

        - ASSUME every k is >= 1
        - MUST execute in O(s) where s is the sum of all k found.
    """
```

Testing: `python3 -m unittest linked_list_test`

Example:

[15]:
```
from linked_list_sol import *

ll = LinkedList()

ll.add('2c')
ll.add('5b')
ll.add('3a')

print(ll)
```

```
LinkedList: 3a,5b,2c
```

[16]:
```
ll.linalg()   # returns NOTHING!
```

[17]:
```
print(ll)
```

```
LinkedList: a,a,a,b,b,b,b,b,c,c
```

with 3 nodes modified and 7 new nodes inserted

2.1.25 Midterm sim - Fri 05, Nov 2021

Scientific Programming - Data Science Master @ University of Trento

THIS IS ONLY A SIMULATION: YOU EARN NOTHING, YOU LOSE NOTHING

Download exercises and solutions

Part A - Terence Hill and Bud Spencer movies

> **NOTICE: this part of the exam was ported to** softpython[121] **website**
>
> There you can find a more curated version (notice it may be longer than here)

Among the greatest gifts of Italy to the world we can certainly count Terence Hill and Bud Spencer movies.

Their film career can be found in Wikidata[122], a project by the Wikimedia foundation which aims to store only machine-readable data, like numbers, strings, and so on interlinked with many references. Each entity in Wikidata has an identifier, for example Terence Hill is the entity Q243430[123] and Bud Spencer is Q221074[124].

Wikidata can be queried using the SPARQL language: we performed this query[125] repeated for several languages, and

[121] https://en.softpython.org/projects/bud-spencer-terence-hill-movies/bud-spencer-terence-hill-movies-sol.html
[122] https://wikidata.org/
[123] http://www.wikidata.org/entity/Q243430
[124] http://www.wikidata.org/entity/Q221074
[125] https://query.wikidata.org/#SELECT%20%3Fstar%20%3FstarLabel%20%3Fitem%20%3FitemLabel%20%28MIN%28%3Fdate%29%20AS%20%3FfirstReleased%29%0AWHERE%20%7B%0A%20%20%3Fitem%20wdt%3AP161%20%3Fstar%3B%0A%20%20%20%20%20%20%20%20wdt%3AP577%20%3Fdate.%0A%20%20%20%20%0A%20%20FILTER%20%28%3Fstar%20%3D%20wd%3AQ221074%20%7C%7C%20%3Fstar%20%3D%20wd%3AQ243430%29%20%20%0A%20%20%20%20%20%20%20%20%20%20%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wikibase%3Alanguage%20%22en%22.%20%7D%0A%20%20OPTIONAL%20%7B%20%3Fitem%20wdt%3AP18%20%3F_image.%20%7D%0A%7D%20GROUP%20BY%20%3Fstar%20%3FstarLabel%20%3Fitem%20%3FitemLabel%20%3F_image%0AORDER%20BY%20%28%3Fdate%29

downloaded CSV files (among the many formats which can be chosen). Even if not necessary for the purposes of the exercise, you are invited to play a bit with the interface, like trying different visualizations (i.e. try clicking the eye in the middle-left corner and then select Graph) - or see other examples[126].

The files

You are given some CSVs of movies, all having names ending in `-xy.csv`, where `xy` can be a language tag like `it`, `en`, `de`, `es`... They mostly contain the same data except for the movie labels which are in the corresponding language. The final goal will be displaying the network of movies and put in evidence the ones co-starring the famous duo.

Each file row contains info about a single actor starring in a movie. Multiple lines with same movie id will mean multiple actors are co-starring. We can see an excerpt of **first four** lines of english version: notice second movie has id Q180638[127] and is co-starred by both Bud Spencer and Terence Hill

```
star,starLabel,movie,movieLabel,firstReleased

http://www.wikidata.org/entity/Q221074,Bud Spencer,http://www.wikidata.org/entity/
↪Q116187,Thieves and Robbers,1983-02-11T00:00:00Z

http://www.wikidata.org/entity/Q221074,Bud Spencer,http://www.wikidata.org/entity/
↪Q180638,Odds and Evens,1978-10-28T00:00:00Z

http://www.wikidata.org/entity/Q243430,Terence Hill,http://www.wikidata.org/entity/
↪Q180638,Odds and Evens,1978-10-28T00:00:00Z
```

Now open Jupyter and start editing this notebook `exam-2021-11-05.ipynb`

load

Write a function that given a `filename_prefix` and list of `languages`, parses the corresponding files and RETURNS a **dictionary of dictionaries**, which maps movies id to movies data, in the format as in the exerpt.

- When a label is missing, you will find instead an id like `Q3778078`: substitute it with empty string (HINT: to recognize ids you might use `is_digit()` method)
- convert date numbers to proper integers
- **DO NOT** put constant ids nor language tags in the code (so no `'Q221074'` nor `'it'` ...)

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

[3]:
```
import csv

def load(filename_prefix, languages):


    first_lang = True
    ret = {}
    for lang in languages:
        fn = '%s-%s.csv' % (filename_prefix, lang)
        #print("Reading", fn)
        with open(fn, encoding='utf-8', newline='') as f:
```

(continues on next page)

[126] https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples/human
[127] http://www.wikidata.org/entity/Q180638

(continued from previous page)

```
            my_reader = csv.DictReader(f, delimiter=',')
            for d in my_reader:
                movie_id = d['movie'][len('http://www.wikidata.org/entity/'):]
                actor = (d['star'][len('http://www.wikidata.org/entity/'):], d[
↪'starLabel'])


                if d['movieLabel'][0] == 'Q' and d['movieLabel'][1].isdigit():
                    #print('FOUND MISSING LABEL', d['movieLabel'], 'FOR', lang)
                    movie_label_fixed = ''
                else:
                    movie_label_fixed = d['movieLabel']

                if first_lang:
                    if movie_id in ret:
                        ret[movie_id]['actors'].append(actor)
                    else:
                        ret[movie_id] = {'actors': [actor],
                                         'names' : {lang: movie_label_fixed},
                                         'first_release' : tuple([int(s) for s in d[
↪'firstReleased'][:10].split('-')])
                        }
                else:
                    ret[movie_id]['names'][lang] = movie_label_fixed

            #print("Found", len(ret), "movies")
        first_lang = False

    return ret


movies_db = load('bud-spencer-terence-hill-movies', ['en', 'it', 'de'])
#movies_db = load('bud-spencer-terence-hill-movies', ['es', 'en', 'de','it'])
```

[3]:
```
import csv

def load(filename_prefix, languages):
    raise Exception('TODO IMPLEMENT ME !')

movies_db = load('bud-spencer-terence-hill-movies', ['en', 'it', 'de'])
#movies_db = load('bud-spencer-terence-hill-movies', ['es', 'en', 'de','it'])
```

Complete expected output can be found in expected_db.py

[4]:
```
EXERPT:

{
  'Q116187': {
              'actors': [('Q221074', 'Bud Spencer')],
              'first_release': (1983, 2, 11),
              'names': {'de': 'Bud, der Ganovenschreck',
```

(continues on next page)

(continued from previous page)

```
                        'en': 'Thieves and Robbers',
                        'it': 'Cane e gatto'}
             }
  'Q180638': {
              'actors': [('Q221074', 'Bud Spencer'), ('Q243430', 'Terence Hill')],
              'first_release': (1978, 10, 28),
              'names': {'de': 'Zwei sind nicht zu bremsen',
                        'en': 'Odds and Evens',
                        'it': 'Pari e dispari'}
             }
  'Q231967': {
              'actors': [('Q221074', 'Bud Spencer'), ('Q243430', 'Terence Hill')],
              'first_release': (1981, 1, 1),
              'names': {'de': 'Zwei Asse trumpfen auf',
                        'en': 'A Friend Is a Treasure',
                        'it': 'Chi trova un amico, trova un tesoro'}
             }
  .
  .
  .
}
```

[5]:
```
# TESTING
from pprint import pformat; from expected_movies_db import expected_movies_db
for sid in expected_movies_db.keys():
    if sid not in movies_db: print('\nERROR: MISSING movie', sid); break
    for k in expected_movies_db[sid]:
        if k not in movies_db[sid]:
            print('\nERROR at movie', sid,'\n\n   MISSING key:', k); break
        if expected_movies_db[sid][k] != movies_db[sid][k]:
            print('\nERROR at movie', sid, 'key:',k)
            print('  ACTUAL:\n', pformat(movies_db[sid][k]))
            print('  EXPECTED:\n', pformat(expected_movies_db[sid][k]))
            break
if len(movies_db) > len(expected_movies_db):
    print('ERROR! There are more movies than expected!')
    print('  ACTUAL:\n', len(movies_db))
    print('  EXPECTED:\n', len(expected_movies_db))
```

save_table

Write a function that given a movies db and a list of `languages`, writes a new file `merged.csv`

- separate actor names with `and`
- use only the year as date
- file must be formatted like this:

```
movie_id,name en,name it,first_release,actors
Q116187,Thieves and Robbers,Cane e gatto,1983,Bud Spencer
Q180638,Odds and Evens,Pari e dispari,1978,Bud Spencer and Terence Hill
```

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

[7]:
```
import csv

def save_table(movies, languages):


    with open('merged.csv', 'w', encoding='utf-8', newline='') as csv_out:
        my_writer = csv.writer(csv_out, delimiter=',')

        header = ['movie_id'] + ['name ' + lan for lan in languages] + ['first_release
↪', 'actors']
        my_writer.writerow(header)

        for movie_id in movies:
            movie = movies[movie_id]
            actors_names = ' and '.join([actor[1] for actor in movie['actors']])
            row = [movie_id]
            row.extend([movie['names'][language] for language in languages])
            row.extend([movie['first_release'][0], actors_names])

            my_writer.writerow(row)
        print('saved file to merged.csv')


save_table(movies_db, ['en','it'])
#save_table(movies_db, ['de'])
```

```
saved file to merged.csv
```

[7]:
```
import csv

def save_table(movies, languages):
    raise Exception('TODO IMPLEMENT ME !')

save_table(movies_db, ['en','it'])
#save_table(movies_db, ['de'])
```

```
saved file to merged.csv
```

Complete expected file is in expected-merged.csv

[8]:
```
# TESTING
with open('expected-merged.csv',encoding='utf-8', newline='') as expected_f:
    with open('merged.csv',encoding='utf-8', newline='') as f:
        expected_reader = csv.reader(expected_f, delimiter=',')
        reader = csv.reader(f, delimiter=',')
        i = 0
        for expected_row in expected_reader:
            try:
                row = next(reader)
            except:
                print('ERROR at row', i, ': ACTUAL rows are less than EXPECTED!')
                break
```

(continues on next page)

(continued from previous page)

```
            for j in range(len(expected_row)):
                if expected_row[j] != row[j]:
                    print('ERROR at row', i, '  cell index', j)
                    print(row)
                    print('\nACTUAL  :', row[j])
                    print('\nEXPECTED:', expected_row[j])
                    break
            i += 1
```

show_graph

Display a NetworkX graph of movies (see examples[128]) from `since_year` (included) to `until_year` (included), in the given `language`

- display actor names as capitalized
- display co-starred movies, non co-starred movies and actors with different colors by setting node attributes `style='filled'` and i.e. `fillcolor='green'` (see some color names[129])

DON'T use labels as node ids

DON'T write constants in your code, so no `'Terence'` nor `'TERENCE'`…

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

[9]:
```
import networkx as nx
from sciprog import draw_nx

def show_graph(movies, since_year, until_year, language):

    G = nx.DiGraph()
    G.graph['graph']= { 'layout':'neato'}  # don't delete these!


    for movie_id in movies:
        movie = movies[movie_id]
        if movie['first_release'][0] >= since_year and movie['first_release'][0] <=␣
↪until_year:
            if len(movie['actors']) > 1:
                fillcolor = 'cornsilk'
            else:
                fillcolor = 'moccasin'
            G.add_node(movie_id, label=movie['names'][language], fillcolor=fillcolor,␣
↪color='black', style='filled', fontcolor='black')
            for actor in movie['actors']:
                G.add_node(actor[0], fillcolor='#defac3', fontcolor='black', color=
↪'darkgreen', style='filled', label=actor[1].upper())
                # IMPORTANT: edges connect movie_ids ,  NOT labels !!!!
                G.add_edge(actor[0], movie_id, color='gold')

    draw_nx(G)
```

(continues on next page)

[128] https://en.softpython.org/relational/relational1-intro-sol.html#Fancy-networkx-graphs
[129] https://www.w3.org/wiki/CSS/Properties/color/keywords

(continued from previous page)

```
show_graph(movies_db, 1970, 1975, 'en')
```

[9]:
```
import networkx as nx
from sciprog import draw_nx

def show_graph(movies, since_year, until_year, language):

    G = nx.DiGraph()
    G.graph['graph']= { 'layout':'neato'}  # don't delete these!

    raise Exception('TODO IMPLEMENT ME !')

show_graph(movies_db, 1970, 1975, 'en')
```

[10]:
```
show_graph(movies_db, 1970, 1974, 'it')
```

2.1.26 Exam - Fri 12, Nov 2021

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

Part A - Mexican Drug Wars

> **NOTICE: this part of the exam was ported to** softpython[130] **website**
>
> There you can find a more curated version (notice it may be longer than here)

Drug cartels carried out a shocking wave of lethal attacks against hundreds of local elected officials and party candidates in Mexico during years 2007-2012, attempting to establish criminal governance regimes and conquer local governments, populations, and territories. This quickly forced Mexican authorities to deploy armored vehicles with heavy weapons to perform military operations within their own borders. You will analyze cartels attacks frequency and where they occurred.

Data sources:

- Trejo, Guillermo; Ley, Sandra, 2019, "Replication Data for: High-Profile Criminal Violence. Why Drug Cartels Murder Government Officials and Party Candidates in Mexico", https://doi.org/10.7910/DVN/VIXNNE, Harvard Dataverse, V1, UNF:6:BcqInKD9NBX3NkI48CdqpQ== [fileUNF] License: CC0 - "Public Domain Dedication"
- Coscia, Michele and Viridiana Rios (2012). Knowing Where and How Criminal Organizations Operate Using Web Content. CIKM, 12 (October – November). https://www.michelecoscia.com/?page_id=1032

Open Jupyter and start editing this notebook `exam-2021-11-12.ipynb`

Attacks during elections

In the file Dataset_HighProfileCriminalViolence.tab are listed the number of attacks occurred to elected officials in Mexico from years 2007 to 2012. Focus only on columns `cve_inegi`, `state`, `year`, `aggr_sum`, `elect_local`:

	cve_inegi	state	year	aggr_sum	elect_local
2278	12031	Guerrero	2012	2	1.0
2279	12032	Guerrero	2007	0	0.0
2280	12032	Guerrero	2008	0	1.0
2281	12032	Guerrero	2009	1	0.0

- Municipalities where the attack occurred are identified by a 5 digits `cve_inegi` code: first two digits indicate the state, 3 last ones the town. **NOTE**: first file entries only have 4 digits as the leading zero is implied, take care of this case

[130] https://en.softpython.org/projects/mexican-drug-wars/mexican-drug-wars-sol.html

- `aggr_sum`: number of attacks occurred in a particular municipality / year.
- `elect_local`: `1.0` if a **local** election occurred in the year of the attack (ignore other `elect_*`)

load_mexico

Extract Mexican state codes, names, the counts of attacks, and the years when **local** elections occurred, and RETURN a dictionary of dictionaries mapping **two digit** state codes **as strings** to the extracted info.

- use `csv.DictReader` with `delimiter='\t'` and `utf8` encoding (municipalities will look weird but we don't use them)
- use exactly 6 cells for `attacks` lists: assume all were carried out between `2007`and `2012` included
- **DO NOT** assume the years in rows repeat with a pattern, for example municipality `21132` has two successive 2012 years!

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

[3]:
```
import csv

def load(filename):


    with open(filename, encoding='utf8', newline='') as csvfile_in:
        my_reader = csv.DictReader(csvfile_in, delimiter='\t')

        ret = {}

        for d in my_reader:
            if len(d['cve_inegi']) == 4:
                inegi = '0' + d['cve_inegi']
            else:
                inegi = d['cve_inegi']

            state_code = inegi[:2]

            if not state_code in ret:
                ret[state_code] = {'attacks':[0]*6,
                                   'local_election_years' : [],
                                   'state_name': '',
                                   'state_code': state_code}
            rd = ret[state_code]
            year = int(d['year'])
            rd['state_code'] = state_code
            rd['state_name'] = d['state']
            rd['attacks'][year - 2007] += int(d['aggr_sum'])

            if d['elect_local'] == '1.0':
                if year not in rd['local_election_years']:
                    rd['local_election_years'].append(year)

    for rd in ret:
        ret[rd]['local_election_years'].sort()

    return ret
```

(continues on next page)

(continued from previous page)

```
mexico_db = load('Dataset_HighProfileCriminalViolence.tab')
mexico_db
```

[3]:
```
import csv

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

mexico_db = load('Dataset_HighProfileCriminalViolence.tab')
mexico_db
```

Complete expected output can be found in expected_mexico_db.py

```
EXERPT (note keys order doesn't matter):

{
  '08': {
              'attacks': [0, 5, 7, 12, 7, 2],
              'local_election_years': [2007, 2010],
              'state_code': '08',
              'state_name': 'Chihuahua'
             }
  '12': {
              'attacks': [4, 11, 11, 9, 3, 10],
              'local_election_years': [2008, 2011, 2012],
              'state_code': '12',
              'state_name': 'Guerrero'
             }
  .
  .
  .
}
```

[5]:
```
# TESTING
from pprint import pformat; from expected_mexico_db import expected_mexico_db
for sid in expected_mexico_db.keys():
    if sid not in mexico_db: print('\nERROR: MISSING state', sid); break
    for k in expected_mexico_db[sid]:
        if k not in mexico_db[sid]:
            print('\nERROR at state', sid,'\n\n   MISSING key:', k); break
        if expected_mexico_db[sid][k] != mexico_db[sid][k]:
            print('\nERROR at state', sid, 'key:',k)
            print('  ACTUAL:\n', pformat(mexico_db[sid][k]))
            print('  EXPECTED:\n', pformat(expected_mexico_db[sid][k]))
            break
if len(mexico_db) > len(expected_mexico_db):
    print('ERROR! There are more states than expected!')
    print('  ACTUAL:\n', len(mexico_db))
    print('  EXPECTED:\n', len(expected_mexico_db))
```

show_attacks

Given a `state_code` and , display a chart of the attack counts over the years.

- normalize the height so to have all charts as high as the maximum possible attack count in the db
- show vertical dashed lines in proximity of election years (use `linestyle='dashed'`), using the same color
- you are allowed to use constants for years
- **make sure vertical lines on borders are clearly visible** and separated from borders by setting proper limits
- **remember** to also print the maximum possible attack count in the db

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

[6]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def show_attacks(state_code, mexdb):


    attacks = mexdb[state_code]['attacks']

    plt.figure(figsize=(3.5,2))
    maxy = 0
    for s in mexdb:
        maxy = max(maxy, max(mexdb[s]['attacks']))

    print("max attacks happened in any state:", maxy)

    xs = range(2007, 2013)
    ys = mexdb[state_code]['attacks']

    plt.plot(xs, ys, color='orange')

    plt.title("Attacks per year in %s state" % mexdb[state_code]['state_name'])
    plt.xlabel('Year')
    plt.ylabel('Number of attacks')

    for year in mexdb[state_code]['local_election_years']:
        plt.plot([year, year], [0, maxy], color='lightgreen', linestyle='dashed')

    plt.show()


show_attacks('12', mexico_db)  # Guerrero
```

```
max attacks happened in any state: 39
```

[6]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def show_attacks(state_code, mexdb):
    raise Exception('TODO IMPLEMENT ME !')

show_attacks('12', mexico_db)  # Guerrero
```

```
max attacks happened in any state: 39
```

[7]:
```
show_attacks('16', mexico_db)   # Michoacan
```

```
max attacks happened in any state: 39
```

cartels

In the file CosciaRios2012_DataBase.csv are listed attacks performed by criminal organizations (cartels) in various years. For each row, the columns from 3-12 have a `1` if the corresponding cartel named in the header was involved in the attack, and `0` otherwise. Example:

	Code	State	Year	BeltranLeyva	Beltran_Leyva_F	Familia	Golfo	Juarez	Sinaloa	Sinaloa_Fam	Tijuana	Zetas	Otros
17	1001	1	2007	0	0	1	0	0	0	0	0	1	0
18	1001	1	2008	0	0	1	0	1	0	0	0	0	0
19	1001	1	2009	0	0	1	1	0	1	0	0	1	0

Write a function which given a `filename` and a `year`, processes the dataset and RETURN a dictionary mapping cartel names to a list of **sorted** states (no duplicates) where the cartel performed attacks in the given `year`.

- use a `csv.reader` with `utf8` encoding
- pick state code from `State` column and state names from previous `mexico_db` (you only need names) - if missing put state code (i.e. `09`)
- **NOTE**: `Sinaloa` is a special case, since it is both a state and a cartel.

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

[9]:
```
import csv
def cartels(filename, mexdb, year):



    work = {}

    with open(filename, encoding='utf8', newline='') as csvfile_in:
        my_reader = csv.reader(csvfile_in, delimiter=',')

        header = next(my_reader)

        for cartel_name in header[3:]:
            work[cartel_name] = set()

        for row in my_reader:
            row_year = int(row[2])
            if row_year == year:
                inegi = row[0]
                if len(row[1]) == 1:
                    state_code = '0' + row[1]
                else:
                    state_code = row[1]
                if state_code in mexdb:
                    state_name = mexdb[state_code]['state_name']
                else:
                    state_name = state_code

                for j in range(3,len(row)):
                    if row[j] == '1':
                        org_name = header[j]
                        work[org_name].add(state_name)
```

(continues on next page)

(continued from previous page)

```
        ret = {}
        for k,v in work.items():
            ret[k] = sorted(v)
        return ret


cartels2003 = cartels('CosciaRios2012_DataBase.csv', mexico_db, 2003)
from pprint import pprint
pprint(cartels2003, width=190)

assert cartels2003['Beltran_Leyva'] == ['Colima', 'Morelos', 'Sinaloa']
assert cartels2003['Otros'] ==  ['Veracruz']
assert cartels2003['Zetas'] ==  ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico',
↪'Nuevo Leon', 'Sinaloa', 'Tamaulipas', 'Veracruz', 'Yucatan']
```

```
{'Beltran_Leyva': ['Colima', 'Morelos', 'Sinaloa'],
 'Beltran_Leyva_Family': [],
 'Familia': [],
 'Golfo': ['Campeche', 'Chihuahua', 'Coahuila', 'Durango', 'Mexico', 'Nuevo Leon',
↪'San Luis Potosi', 'Tamaulipas', 'Veracruz', 'Yucatan'],
 'Juarez': ['Baja California', 'Chihuahua', 'Coahuila', 'Colima', 'Durango', 'Guerrero
↪', 'Jalisco', 'Sinaloa', 'Tamaulipas'],
 'Otros': ['Veracruz'],
 'Sinaloa': ['Chiapas', 'Colima', 'Jalisco', 'Mexico', 'Nayarit', 'Nuevo Leon',
↪'Sinaloa', 'Sonora', 'Tamaulipas'],
 'Sinaloa_Family': ['Guerrero'],
 'Tijuana': ['Aguascalientes', 'Baja California', 'Chiapas', 'Chihuahua', 'Coahuila',
↪'Guerrero', 'Jalisco', 'Mexico', 'Michoacan', 'Nuevo Leon', 'Puebla', 'Sinaloa',
↪'Sonora'],
 'Zetas': ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico', 'Nuevo Leon', 'Sinaloa',
↪'Tamaulipas', 'Veracruz', 'Yucatan']}
```

[9]:
```
import csv
def cartels(filename, mexdb, year):
    raise Exception('TODO IMPLEMENT ME !')

cartels2003 = cartels('CosciaRios2012_DataBase.csv', mexico_db, 2003)
from pprint import pprint
pprint(cartels2003, width=190)

assert cartels2003['Beltran_Leyva'] == ['Colima', 'Morelos', 'Sinaloa']
assert cartels2003['Otros'] ==  ['Veracruz']
assert cartels2003['Zetas'] ==  ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico',
↪'Nuevo Leon', 'Sinaloa', 'Tamaulipas', 'Veracruz', 'Yucatan']
```

```
{'Beltran_Leyva': ['Colima', 'Morelos', 'Sinaloa'],
 'Beltran_Leyva_Family': [],
 'Familia': [],
 'Golfo': ['Campeche', 'Chihuahua', 'Coahuila', 'Durango', 'Mexico', 'Nuevo Leon',
↪'San Luis Potosi', 'Tamaulipas', 'Veracruz', 'Yucatan'],
 'Juarez': ['Baja California', 'Chihuahua', 'Coahuila', 'Colima', 'Durango', 'Guerrero
↪', 'Jalisco', 'Sinaloa', 'Tamaulipas'],
 'Otros': ['Veracruz'],
 'Sinaloa': ['Chiapas', 'Colima', 'Jalisco', 'Mexico', 'Nayarit', 'Nuevo Leon',
↪'Sinaloa', 'Sonora', 'Tamaulipas'],
```

(continues on next page)

(continued from previous page)

```
 'Sinaloa_Family': ['Guerrero'],
 'Tijuana': ['Aguascalientes', 'Baja California', 'Chiapas', 'Chihuahua', 'Coahuila',
↪'Guerrero', 'Jalisco', 'Mexico', 'Michoacan', 'Nuevo Leon', 'Puebla', 'Sinaloa',
↪'Sonora'],
 'Zetas': ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico', 'Nuevo Leon', 'Sinaloa',
↪'Tamaulipas', 'Veracruz', 'Yucatan']}
```

[10]:
```
# further tests
expected2003 = {
 'Beltran_Leyva': ['Colima', 'Morelos', 'Sinaloa'],
 'Beltran_Leyva_Family':[],
 'Familia': [],
 'Golfo':   ['Campeche', 'Chihuahua', 'Coahuila', 'Durango', 'Mexico', 'Nuevo Leon',
↪'San Luis Potosi', 'Tamaulipas', 'Veracruz', 'Yucatan'],
 'Juarez':  ['Baja California', 'Chihuahua', 'Coahuila', 'Colima', 'Durango',
↪'Guerrero', 'Jalisco', 'Sinaloa', 'Tamaulipas'],
 'Otros':   ['Veracruz'],
 'Sinaloa': ['Chiapas', 'Colima', 'Jalisco', 'Mexico', 'Nayarit', 'Nuevo Leon',
↪'Sinaloa', 'Sonora', 'Tamaulipas'],
 'Sinaloa_Family': ['Guerrero'],
 'Tijuana': ['Aguascalientes', 'Baja California', 'Chiapas', 'Chihuahua', 'Coahuila',
↪'Guerrero', 'Jalisco', 'Mexico', 'Michoacan', 'Nuevo Leon', 'Puebla', 'Sinaloa',
↪'Sonora'],
 'Zetas':   ['Campeche', 'Guanajuato', 'Jalisco', 'Mexico', 'Nuevo Leon', 'Sinaloa',
↪'Tamaulipas', 'Veracruz', 'Yucatan']
}

assert cartels2003 == expected2003

expected1999 = {
    'Beltran_Leyva': [],
    'Beltran_Leyva_Family': [],
    'Familia': [],
    'Golfo':   ['Baja California', 'Guanajuato', 'Nuevo Leon', 'Puebla'],
    'Juarez':  ['Baja California', 'Chihuahua', 'Durango', 'Tamaulipas', 'Veracruz'],
    'Sinaloa': ['Jalisco', 'Veracruz'],
    'Sinaloa_Family': [],
    'Tijuana': ['Baja California', 'Campeche', 'Coahuila', 'Nuevo Leon', 'Sonora',
↪'Tamaulipas', 'Yucatan'],
    'Zetas':   ['Baja California', 'Mexico', 'Morelos', 'Sinaloa', 'Sonora'],
    'Otros':   []
}

cartels1999 = cartels('CosciaRios2012_DataBase.csv', mexico_db, 1999)
assert cartels1999 == expected1999
```

2.1.27 Midterm B- Thu 16, Dec 2021

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

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 `my_fun` function, explaining your reasoning.

[2]:
```
def my_fun2(L):
    n = len(L)
    tmp = []
    for i in range(n):
        for j in range(n):
            tmp.append(L[i]/(L[j]))
    return tmp

def my_fun(L):
    n = len(L)
    if n <= 1:
        return 1
    else:
        L1 = L[0:n//3]
        L2 = L[2*(n//3):]
        a = my_fun(L1) + max(my_fun2(L1))
        b = my_fun(L2) + max(my_fun2(L2))
        return a - b
```

B1.2 postfix

Postfix notation is used by compilers to process expressions efficiently, respecting operator precedence without needing to scan the whole expression multiple times (programming languages let us write expressions using the infix notation for simplicity). Think of a function implementing the evaluation of expressions in postfix notation. Which data structure would you use, and why? Explain your reasoning.

```
Infix notation: a op1 b (e.g. 7 * 4)
Postfix notation:   a b op1     (e.g. 7 4 *)

Infix notation:     a op1 b op2 c op3 d (e.g. 7 + 4 * 5 + 9)
Postfix notation:   a b c op1 op2 d op3     (e.g. 7 4 5 * + 9 +)
```

B2 norep

- Open Visual Studio Code and start editing the folder on your desktop
- For running tests: open *Accessories -> Terminal*

Open file `linked_list.py` and 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.
    """
```

Testing: `python3 -m unittest more_test.NorepTest`

Example:

[3]:
```
from linked_list_sol import *
```

[4]:
```
ll = LinkedList()
ll.add('e')
ll.add('c')
ll.add('c')
ll.add('c')
ll.add('d')
ll.add('d')
ll.add('b')
ll.add('a')
ll.add('a')
ll.add('c')
ll.add('c')
ll.add('a')
ll.add('a')
ll.add('a')
```

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

```
LinkedList: a,a,a,c,c,a,a,b,d,d,c,c,c,e
```

[6]:
```
ll.norep()
```

[7]:
```
print(ll)
```

```
LinkedList: a,c,a,b,d,c,e
```

B3 family_sum_rec

Open `bin_tree.py` and implement this method:

```
def family_sum_rec(self):
    """ MODIFIES the tree by adding to each node data its *original* parent and␣
↪children data

        - MUST execute in O(n) where n is the size of the tree
```

(continues on next page)

(continued from previous page)

```
        - a recursive implementation is acceptable
        - HINT: you will probably want to define a helper function
    """
```

Testing: `python3 -m unittest bin_tree_test.FamilySumRec`

Example:

[8]:
```
from bin_tree_test import bt
```

[9]:
```
t = bt(5,
          bt(1,
               bt(4,
                    None,
                    bt(3)),
               bt(2)),
          bt(9,
               bt(11)))
```

[10]:
```
from sciprog import draw_bt
draw_bt(t)
```

[11]:
```
t.family_sum_rec()
```

[12]:
```
draw_bt(t)
```

You need to sum to each node data its original parent data + original left child data + original right child data , for example:

- Root: 15 = 5 + 0 + 1 + 9
- left child of root: 12 = 1 + 5 + 4 + 2
- right child of root: 25 = 9 + 5 + 11 + 0
- leftmost grandchild of root: 8 = 4 + 1 + 0 + 3

2.1.28 Exam - Wed 12, Jan 2022

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

Part A - Prezzario

> **NOTICE: this part of the exam was ported to** softpython[131] **website**
>
> There you can find a more curated version (notice it may be longer than here)

Open Jupyter and start editing this notebook `exam-2022-01-22.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[132]

[131] https://en.softpython.org/projects/price-catalog/price-catalog-sol.html
[132] https://dati.trentino.it/dataset/prezziario-dei-lavori-pubblici-della-provincia-autonoma-di-trento

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.

[2]:
```
import pandas as pd
import numpy as np

pd.set_option('display.max_colwidth', None)
df = pd.read_csv('EPPAT-2018-new-compact.csv', encoding='latin-1')
```

The dataset contains several columns, but we will consider the following ones:

[3]:
```
df=df[['Codice Prodotto','Descrizione Breve Prodotto','Categoria','Prezzo']]
df[:22]
```

	Codice Prodotto	Descrizione Breve Prodotto	Categoria	Prezzo
0	A.02.35.0050	ATTREZZATURA PER INFISSIONE PALI PILOTI	nan	nan
1	A.02.35.0050.010	Attrezzatura per infissione pali piloti.	Noli e trasporti	109.09
2	A.02.40	ATTREZZATURE SPECIALI	nan	nan
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
12	A.02.40.0020.030	oltre 13 fino a 20 KW	Noli e trasporti	14.66
13	A.02.40.0020.040	oltre 20 fino a 28 KW	Noli e trasporti	15.62
14	A.02.40.0020.050	oltre 28 fino a 36 KW	Noli e trasporti	16.40
15	A.02.40.0020.060	oltre 36 fino a 56 KW	Noli e trasporti	28.53
16	A.02.40.0020.070	oltre 56 fino a 80 KW	Noli e trasporti	44.06
17	A.02.40.0020.080	oltre 80 fino a 100 KW	Noli e trasporti	50.86
18	A.02.40.0020.090	oltre 100 fino a 120 KW	Noli e trasporti	55.88
19	A.02.40.0020.100	oltre 120 fino a 156 KW	Noli e trasporti	80.47
20	A.02.40.0020.110	oltre 156 fino a 184 KW	Noli e trasporti	94.00
21	A.02.40.0030	NASTRO TRASPORTATORE CON MOTORE AD ARIA COMPRESSA	nan	nan

Pompa completa a motore Example

If we look at the dataset, in some cases we can spot a pattern (rows 3 to 8 included):

[4]:
```
df[3:12]
```

[4]:
```
     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>

[5]:
```
import math

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

def extract_bounds(text):



    fixed_text = text
```

(continues on next page)

(continued from previous page)

```
    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)
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)
```

[5]:
```
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)
```

(continues on next page)

(continued from previous page)

```
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[133]
- **as encoding, use** `latin-1`

[6]:
```
# Suppose we want to get all info about A.02.40.0010 prefix:
df[3:12]
```

[6]:
```
     Codice Prodotto   Descrizione Breve Prodotto         Categoria Prezzo
3       A.02.40.0010     POMPA COMPLETA DI MOTORE               NaN    NaN
4   A.02.40.0010.010                fino a mm 50.  Noli e trasporti   2.21
5   A.02.40.0010.020   oltre mm 50 fino a mm 100.  Noli e trasporti   3.36
6   A.02.40.0010.030  oltre mm 100 fino a mm 150.  Noli e trasporti   4.42
7   A.02.40.0010.040  oltre mm 150 fino a mm 200.  Noli e trasporti   5.63
8   A.02.40.0010.050                oltre mm 200.  Noli e trasporti   6.84
9       A.02.40.0020           GRUPPO ELETTROGENO               NaN    NaN
10  A.02.40.0020.010                 fino a 10 KW  Noli e trasporti   8.77
11  A.02.40.0020.020        oltre 10 fino a 13 KW  Noli e trasporti   9.94
```

A call to

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

Must produce:

```
{'category': 'Noli e trasporti',
 'code': 'A.02.40.0010',
 'description': 'POMPA COMPLETA DI MOTORE',
 'measure_unit': 'mm',
 'models': [{'bounds': (0, 50),        'price': 2.21, 'subcode': '010'},
            {'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>

[133] https://en.softpython.org/formats/formats2-csv-sol.html#Reading-as-dictionaries

[7]:

```
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') == \
    {'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'}]}
```

[7]:
```
import csv
from pprint import pprint

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

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

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

A3 plot_product

Implement following function that takes a dictionary as output by previous `extract_product` and shows its price ranges.

- pay attention to display title and axis labels as shown, using input data and **not** constants.
- in case last range holds a `math.inf`, show a > sign
- **if you don't have a working** `extract_product`, **just copy paste data from previous asserts**.

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

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

[8]:

```
%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])
```

(continues on next page)

(continued from previous page)

```
        else:
            ticks.append('%s - %s' % (bounds[0], bounds[1]))

    plt.xticks(xs, ticks)
    plt.gcf().set_size_inches(11,8)
    plt.xlabel(product['measure_unit'])
    plt.ylabel('Price (€)')

    plt.show()


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

plot_product(product)
```

[8]:

```
%matplotlib inline

import numpy as np
```

(continues on next page)

(continued from previous page)

```
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)
```

Part B

- Open Visual Studio Code and start editing the folder on your desktop
- For running tests: open *Accessories* -> *Terminal*

B1 Theory

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

B1.1

Given a list `L` of n elements, please compute the asymptotic computational complexity of the `myfun` function, explaining your reasoning.

[9]:
```
def myfun(L):
    n = len(L)
    sums = {}
    for i in range(n):
        sums[i] = 0
        for j in range(i):
            sums[i] += j
    for k in range(n):
        sums[k] += k
    return sums
```

B1.2

Describe the differences between the Depth-First and the Breadth-First Search algorithms for visiting graphs.

Then, apply BFS to the graph below (write down the visit order only).

B2 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 linked_list_test.FindCoupleTest`

B3 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 linked_list_test.SwapTest`

[ ]:

```

```

2.1.29 Exam - Wed 09, Feb 2022

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

Part A - Zoom surveillance

> **NOTICE: this part of the exam was** ported to softpython[134] **website**
>
> There you can find a more curated version (notice it may be longer than here)

Open Jupyter and start editing this notebook `exam-2022-02-09.ipynb`

A training center holds online courses with Zoom software. 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.

CSV format

You are provided with the file `UserQos_12345678901.csv`. Unfortunately, it is a weird CSV which actually looks like two completely different CSVs were merged together, one after the other. It contains the following:

- 1st line: general meeting header
- 2nd line: general meeting data
- 3rd line: empty
- 4th line completely different header for participant sessions for that meeting. Each session contains a join time and a leave time, and each participant can have multiple sessions in a meeting.
- 5th line and following: sessions data

The file has lots of useless fields, try to explore it and understand the format (use LibreOffice Calc to help yourself)

Here we only show the few fields we are actually interested in, and examples of 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`

[134] https://en.softpython.org/projects/zoom-surveillance/zoom-sol.html

- `(Bowser left the meeting.Reason: Host closed the meeting. )` should become `Host closed the meeting`
- `(Princess Toadstool left the meeting.Reason: left the meeting.)` should become `left the meeting`

Your first goal will be to load the dataset and restructure the data so it looks like this:

```
[['meeting_id', 'topic', 'date', 'participant', 'join_time', 'leave_time', 'reason'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Luigi','13:54','15:10','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Luigi','15:12','15:54','left the␣
↪meeting'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','14:02','14:16','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','14:19','15:02','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','15:04','15:50','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','15:52','15:55','Network␣
↪connection error'],
 ['123 4567 8901','Hydraulics Exam','Apr 17, 2020','Mario','15:56','16:00','Host␣
↪closed the meeting'],
 ...
]
```

To fix the times, you will first need to implement the following function.

A1 time24

<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 time24(t):
    """ Takes a time string like '06:27 PM' and outputs a string like 18:27
    """

    if t.endswith('AM'):
        if t.startswith('12:00'):
            return '00:00'
        else:
            return t.replace(' AM', '')
    else:
        if t.startswith('12:00'):
            return '12:00'

        h = '%0.d' % (int(t.split(':')[0]) + 12)

        return h + ':' + t.split(':')[1].replace(' PM','')


assert time24('12:00 AM') == '00:00'  # midnight
assert time24('01:06 AM') == '01:06'
assert time24('09:45 AM') == '09:45'
assert time24('12:00 PM') == '12:00'  # special case, it's actually midday
assert time24('01:27 PM') == '13:27'
```

(continues on next page)

(continued from previous page)

```
assert time24('06:27 PM') == '18:27'
assert time24('10:03 PM') == '22:03'
```

[2]:
```
def time24(t):
    """ Takes a time string like '06:27 PM' and outputs a string like 18:27
    """
    raise Exception('TODO IMPLEMENT ME !')

assert time24('12:00 AM') == '00:00'  # midnight
assert time24('01:06 AM') == '01:06'
assert time24('09:45 AM') == '09:45'
assert time24('12:00 PM') == '12:00'  # special case, it's actually midday
assert time24('01:27 PM') == '13:27'
assert time24('06:27 PM') == '18:27'
assert time24('10:03 PM') == '22:03'
```

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 reader[135] 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>

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

(continues on next page)

[135] https://en.softpython.org/formats/formats2-csv-sol.html

(continued from previous page)

```
    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'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Mario', '15:56', '16:00',
↪'Host closed the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '14:15', '14:30',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '14:54', '15:03',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '15:12', '15:40',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Bowser', '15:45', '16:00',
↪'Host closed the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Princess Toadstool', '13:56',
↪'15:33', 'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:05', '14:10',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:15', '14:29',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '14:33', '15:10',
↪'left the meeting'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '15:25', '15:54',
↪'Network connection error'],
 ['123 4567 8901', 'Hydraulics Exam', 'Apr 17, 2020', 'Wario', '15:55', '16:00',
↪'Host closed the meeting']]
```

[3]:
```
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'],
```

(continues on next page)

(continued from previous page)

```
 ['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']]
```

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

(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']]

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>

[5]:
```
def duration(a, b):

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


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

[5]:
```
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>

[6]:
```
def calc_stats(log):

    ret = {}

    last_sessions = {}

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

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

        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
```

(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
```

[6]:
```
{'Luigi': {'max_gap': 2, 'gaps': 1, 'time_away': 2},
 'Mario': {'max_gap': 3, 'gaps': 4, 'time_away': 8},
 'Bowser': {'max_gap': 24, 'gaps': 3, 'time_away': 38},
 'Princess Toadstool': {'max_gap': 0, 'gaps': 0, 'time_away': 0},
 'Wario': {'max_gap': 15, 'gaps': 4, 'time_away': 25}}
```

[6]:
```
def calc_stats(log):
    raise Exception('TODO IMPLEMENT ME !')


stats = calc_stats(meeting_log)

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

stats
```

[6]:
```
{'Luigi': {'max_gap': 2, 'gaps': 1, 'time_away': 2},
 'Mario': {'max_gap': 3, 'gaps': 4, 'time_away': 8},
 'Bowser': {'max_gap': 24, 'gaps': 3, 'time_away': 38},
 'Princess Toadstool': {'max_gap': 0, 'gaps': 0, 'time_away': 0},
 'Wario': {'max_gap': 15, 'gaps': 4, 'time_away': 25}}
```

[7]:
```
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
```

Part B

B1 Theory

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

B1.1 myfun

Given a list `L` of n elements, please compute the asymptotic computational complexity of the `myfun` function, explaining your reasoning.

[8]:
```
def myfun(L):
    n = len(L)
    i = 1
    s = 0
    while i < n:
        j = n
        while j > 0:
            for k in range(j, n, 2):
                s += (i + j * k)
            j = j//2
        i = i*2
```

B 1.2 nlogn

What do we mean when we say that an algorithm has asymptotic computational complexity $O(nlogn)$? What do we have to do to prove that an algorithm has asymptotic computational complexity $O(nlogn)$?

B2 flatv

- Open Visual Studio Code and start editing the folder on your desktop

Open `linked_lists.py` and implement method `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 lists_test.FlatvTest`

B3 univalued_rec

Open `bin_trees.py` and implement this method:

```
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:

[9]:
```
from bin_tree_test import bt
```

[10]:
```
t = bt(3, bt(3), bt(3, bt(3, bt(3, None, bt(3)))))
print(t)
```

```
3
├3
└3
 ├3
 │├3
 │├
 │└3
 │└
 └
```

[11]:
```
t.univalued_rec()
```

[11]:
```
True
```

[12]:
```
t = bt(2, bt(3), bt(6, bt(3, bt(3, None, bt(3)))))
print(t)
```

```
2
├3
└6
 ├3
 │├3
 │├
 │└3
 │└
 └
```

[13]:
```
t.univalued_rec()
```

[13]:
```
False
```

[ ]:

2.1.30 Exam - Wed 08, Jun 2022

Scientific Programming - Data Science Master @ University of Trento

Download exercises and solutions

2.1.31 A. Trans-Atlantic Slave Trade

Open Jupyter and start editing this notebook `exam-2022-06-08.ipynb`. You will analyze voyages of slaves from Africa to the America of two centuries ago.

> **NOTICE: this part of the exam was** ported to softpython[136] **website**
>
> There you can find a more curated version (notice it may be longer than here)

Data provider: The Trans-Atlantic Slave Trade Database. 2020. SlaveVoyages. https://www.slavevoyages.org. You are enouraged to explore the dataset with the very interesting online tool[137] they built, in particular check out the Maps[138] and Timelapse[139] tabs.

Data license:

- Historical data: The Trans-Atlantic Slave Trade Database. 2020. SlaveVoyages. https://www.slavevoyages.org (accessed June 9, 2020). License: Public domain (use restrictions do not apply).
- Imputed data: Estimates. 2020. SlaveVoyages. https://slavevoyages.org/assessment/estimates (accessed June 9, 2020). License: Creative Commons Attribution-Noncommercial 3.0 United States License.

A1 read_trade

Each line in slave-trade.csv represents a ship voyage from a purchase place to a landing place. Parse it with a csv reader and output a list of dictionaries, one per voyage according to the output excerpt.

- Each ship has a nation flag `NATINIMP`
- Each voyage has purchase place code `MJBYPTIMP` and a landing place code `MJSLPTIMP` with five digits format `xyzvt` that indicate a specific town: you **MUST** save more generic codes of the form `xyz00` which indicate broader regions.
- **WARNING 1**: convert to int **only** `VOYAGEID` and `YEARAM`, leave `MJBYPTIMP` and `MJSLPTIMP` as strings
- **WARNING 2:** some codes in `slave-trade.csv` have a space instead of a number, in those cases save code `00000`

[2]:
```
import pandas as pd
import numpy as np
df = pd.read_csv('slave-trade.csv', encoding='UTF-8')
df[df.VOYAGEID.isin([1, 2024, 2393, 4190])]
```

[136] https://en.softpython.org/projects/slave-trade/slave-trade-sol.html
[137] https://www.slavevoyages.org/voyage/database
[138] https://www.slavevoyages.org/voyage/database#maps
[139] https://www.slavevoyages.org/voyage/database#timelapse

[2]:
```
      VOYAGEID  YEARAM         NATINIMP MJBYPTIMP MJSLPTIMP
0            1    1817  Portugal/Brazil     60820     50299
2000      2024    1840           U.S.A.     60615     31399
2361      2393    1829    Spain/Uruguay     60212
4000      4190    1854           U.S.A.     60515     31301
```

Region labels: For each location you need to also save its label, which you can find in separate file region-codes.csv (load the file with a csv reader)

- **WARNING 1:** in `region-codes.csv` there are **only** codes in format `xyz00`
- **WARNING 2:** some region codes are missing, in those cases place label `'unknown'`

[3]:
```
import pandas as pd
dfr = pd.read_csv('region-codes.csv', encoding='UTF-8', dtype=str)
dfr[dfr.Value.isin(['60800','60600','31300','50200','60500'])]
```

[3]:
```
    Value                                      Region
47  31300                                        Cuba
84  50200                                       Bahia
92  60500                               Bight of Benin
93  60600  Bight of Biafra and Gulf of Guinea islands
95  60800   Southeast Africa and Indian Ocean islands
```

Output excerpt: (for full output see expected_db.py)

```
>>> read_voyages('slave-trade.csv', 'region-codes.csv')
[
{'flag': 'Portugal/Brazil',
 'id': 1,
 'landing_id': '50200',
 'landing_label': 'Bahia',
 'purchase_id': '60800',
 'purchase_label': 'Southeast Africa and Indian Ocean islands',
 'year': 1817},
{'flag': 'U.S.A.',
 'id': 2024,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60600',
 'purchase_label': 'Bight of Biafra and Gulf of Guinea islands',
 'year': 1840},
{'flag': 'Spain/Uruguay',
 'id': 2393,
 'landing_id': '00000',
 'landing_label': 'unknown',
 'purchase_id': '60200',
 'purchase_label': 'Sierra Leone',
 'year': 1829},
{'flag': 'U.S.A.',
 'id': 4190,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60500',
 'purchase_label': 'Bight of Benin',
 'year': 1854},
  .
```

(continues on next page)

(continued from previous page)

```
  .
]
```

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

[4]:

```
import csv

def read_voyages(slave_trade_csv, region_codes_csv):


    with open(region_codes_csv, encoding='utf-8', newline='') as fregions:
        my_reader = csv.DictReader(fregions, delimiter=',')

        regions_db = {}
        for d in my_reader:
            regions_db[d['Value']] = d['Region']

    with open(slave_trade_csv, encoding='utf-8', newline='') as f:
        my_reader = csv.DictReader(f, delimiter=',')

        ret = []

        for d in my_reader:

            voyage = {}
            voyage['id'] = int(d['VOYAGEID'])
            voyage['year'] = int(d['YEARAM'])
            voyage['flag'] = d['NATINIMP']

            if  d['MJBYPTIMP'].strip():
                pur_reg = d['MJBYPTIMP'][:3] + '00'
                voyage['purchase_id'] = pur_reg
                if pur_reg in regions_db:
                    voyage['purchase_label'] = regions_db[pur_reg]
                else:
                    voyage['purchase_label'] = 'unknown'
            else:
                voyage['purchase_id'] = '00000'
                voyage['purchase_label'] = 'unknown'

            if d['MJSLPTIMP'].strip():
                lan_reg = d['MJSLPTIMP'][:3] + '00'
                voyage['landing_id'] = lan_reg
                if lan_reg in regions_db:
                    voyage['landing_label'] = regions_db[lan_reg]
                else:
                    voyage['landing_label'] = 'unknown'
            else:
                voyage['landing_id'] = '00000'
                voyage['landing_label'] = 'unknown'

            ret.append(voyage)

        return ret
```

(continues on next page)

(continued from previous page)

```
voyages_db = read_voyages('slave-trade.csv', 'region-codes.csv')

print('OUTPUT EXCERPT:')
from pprint import pformat
print('[\n' +',\n'.join([pformat(voyages_db[vid]) for vid in [0,2000,2361, 4000]]) +
↪',\n  .\n  .\n]')
```

```
OUTPUT EXCERPT:
[
{'flag': 'Portugal/Brazil',
 'id': 1,
 'landing_id': '50200',
 'landing_label': 'Bahia',
 'purchase_id': '60800',
 'purchase_label': 'Southeast Africa and Indian Ocean islands',
 'year': 1817},
{'flag': 'U.S.A.',
 'id': 2024,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60600',
 'purchase_label': 'Bight of Biafra and Gulf of Guinea islands',
 'year': 1840},
{'flag': 'Spain/Uruguay',
 'id': 2393,
 'landing_id': '00000',
 'landing_label': 'unknown',
 'purchase_id': '60200',
 'purchase_label': 'Sierra Leone',
 'year': 1829},
{'flag': 'U.S.A.',
 'id': 4190,
 'landing_id': '31300',
 'landing_label': 'Cuba',
 'purchase_id': '60500',
 'purchase_label': 'Bight of Benin',
 'year': 1854},
  .
  .
]
```

[4]:

```
import csv

def read_voyages(slave_trade_csv, region_codes_csv):
    raise Exception('TODO IMPLEMENT ME !')

voyages_db = read_voyages('slave-trade.csv', 'region-codes.csv')

print('OUTPUT EXCERPT:')
from pprint import pformat
print('[\n' +',\n'.join([pformat(voyages_db[vid]) for vid in [0,2000,2361, 4000]]) +
↪',\n  .\n  .\n]')
```

[5]:
```
# TESTING
from pprint import pformat; from expected_db import expected_db
for i in range(0, len(expected_db)):
    if expected_db[i] != voyages_db[i]:
        print('\nERROR at index', i, ':')
        print('  ACTUAL:\n', pformat(voyages_db[i]))
        print('  EXPECTED:\n', pformat(expected_db[i]))
        break
if len(voyages_db) != len(expected_db):
    print("ERROR: Different lengths!  voyages_db:", len(voyages_db), "   expected_db:
↪", len(expected_db))
```

A2 Deportation

For each link purchase -> landing place, count in how many voyages it was present, then draw result in networkx.

- as edge `weight` use a normalized value from 0.0 to 1.0 (maximal count found in the graph)
- show only edges with weight greater or equal to `min_weight`
- to display the graph from right to left, set `G.graph['graph']= {'rankdir':'RL'}`
- for networkx attributes see this example[140], **make sure to display edges proportionally to the weight**

```
>>> show_deportation(voyages_db, 0.09)
COUNTS EXCERPT SOLUTION:
{
  ('60800', '50200') : 48,
  ('60700', '50200') : 1301,
  ('60700', '50400') : 2770,
  ('60800', '50400') : 443,
  ('60900', '50400') : 196,
    .
}
```

[140] https://en.softpythonorg/relational/relational1-intro-sol.html#Fancy-networkx-graphs

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

[6]:
```
import networkx as nx
import sys
sys.path.append('../../')
from sciprog import draw_nx

def show_deportation(voyages, min_weight):


    edges = {}

    for v in voyages_db:
        t = (v['purchase_id'], v['landing_id'])
```

(continues on next page)

(continued from previous page)

```
        if t in edges:
            edges[t] += 1
        else:
            edges[t] = 1



    G = nx.DiGraph()
    G.graph['graph']= {'rankdir':'RL'}     # force horiz right to left layout

    mx = max(edges.values())
    for v in voyages_db:
        s = edges[(v['purchase_id'], v['landing_id'])]
        r = s / mx
        if r >= min_weight:
            G.add_node(v['purchase_id'], fontcolor='black', color='brown', label=v[
↪'purchase_label'])
            G.add_node(v['landing_id'], fontcolor='darkorange', color='orange',␣
↪label=v['landing_label'])
            G.add_edge(v['purchase_id'],v['landing_id'],
                       color="blue",
                       penwidth= 7 * r,
                       weight=r)

    draw_nx(G,

           )

show_deportation(voyages_db, 0.09)
#show_deportation(voyages_db, 0.06)
```

```
COUNTS EXCERPT SOLUTION:
{
  ('60800', '50200') : 48,
  ('60700', '50200') : 1301,
  ('60700', '50400') : 2770,
  ('60800', '50400') : 443,
  ('60900', '50400') : 196,
    .
    .
}
Image saved to file:  expected-deportation-plot.png
```

[6]:

```
import networkx as nx
import sys
sys.path.append('../../')
from sciprog import draw_nx
```

(continues on next page)

(continued from previous page)

```
def show_deportation(voyages, min_weight):
    raise Exception('TODO IMPLEMENT ME !')
show_deportation(voyages_db, 0.09)
#show_deportation(voyages_db, 0.06)
```

A3 The time to stop

Given a nation flag, plot inside `draw_time` the number of voyages per year done by ships belonging to that flag. **DO NOT** call `plt.show` nor `plt.figure`. We show some counts example but to calculate the data feel free to use any method you want. **To associate a plot to a label, use i.e.** `plt.plot(xs, ys, label='France')`

```
>>> fig = plt.figure(figsize=(15,6))
>>> draw_time(voyages_db, 'France')
>>> draw_time(voyages_db, 'U.S.A.')
>>> draw_time(voyages_db, 'Great Britain')
>>> plt.legend()
>>> plt.show()
France COUNTS EXCERPT SOLUTION:
{
        1816:7,
       1819:30,
       1821:59,
       .
}
U.S.A. COUNTS EXCERPT SOLUTION:
{
        1821:1,
        1827:1,
        1837:2,
       .
}
Great Britain COUNTS EXCERPT SOLUTION:
{       1810:1,
        1809:1,
        1811:1,
       .
}
```

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

[7]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def draw_time(voyages, flag):


    plt.title("Slave trade voyages SOLUTION")
    plt.xlabel('years')
    plt.ylabel('voyages')

    counts = {}
    for v in voyages:
        if v['flag'] == flag:
            y = v['year']
            if y in counts:
                counts[y] += 1
            else:
                counts[y] = 1



    xs = sorted(counts.keys())
    ys = [counts[x] for x in xs]
    plt.plot(xs, ys, label=flag)


fig = plt.figure(figsize=(15,6))
draw_time(voyages_db, 'France')
draw_time(voyages_db, 'U.S.A.')
draw_time(voyages_db, 'Great Britain')
plt.legend()

plt.show()
```

```
France COUNTS EXCERPT SOLUTION:
{
        1816:7,
       1819:30,
       1821:59,
       .
       .
              }
U.S.A. COUNTS EXCERPT SOLUTION:
{
        1821:1,
        1827:1,
        1837:2,
       .
       .
              }
Great Britain COUNTS EXCERPT SOLUTION:
{
        1810:1,
        1809:1,
```

(continues on next page)

(continued from previous page)

```
        1811:1,
       .
       .
              }
```

[7]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def draw_time(voyages, flag):
    raise Exception('TODO IMPLEMENT ME !')

fig = plt.figure(figsize=(15,6))
draw_time(voyages_db, 'France')
draw_time(voyages_db, 'U.S.A.')
draw_time(voyages_db, 'Great Britain')
plt.legend()

plt.show()
```

B1.1 Theory

Write the solution in separate ``theory.txt`` file. Given a list L of n elements, please compute the asymptotic computational complexity of the $myFun$ function, explaining your reasoning.

[8]:
```
def myFun(L):
    T = []
    N = len(L)
    for i in range(N//4):
        k = 0
        tmp = 0
        while k < 4:
            tmp += L[i]*k
            k = k + 1
```

(continues on next page)

(continued from previous page)

```
        T.insert(0,tmp)

    return(T)
```

B1.2 BFS

Describe what a BFS visit of a graph is. Please provide the order of visited nodes of a possible BFS visit on the following graph (starting from node 'a'):

B2 BinaryTree is_heap_stack

- Open Visual Studio Code and start editing the folder on your desktop. Implement this:

```
def is_heap_stack(self):
    """ A tree is a min heap if each node data is less or equal than its children␣
↪data.
        RETURN True if the tree is a min heap, False otherwise

        - DO *NOT* use recursion
        - implement it with a while and a stack (as a python list)
        - MUST run in O(n), where n is the tree size
    """
```

Testing: `python3 -m unittest bin_tree_test`

[10]:
```
from bin_tree_test import bt
from bin_tree_sol import *
t = bt(7,
        bt(9,
                bt(10,
                        bt(40)),
                bt(14)),
        bt(20,
                None,
                bt(30,
                        bt(50),
                        bt(90))))

t.is_heap_stack()
```

[10]:
```
True
```

[11]:
```
t = bt(7,
        bt(9,
                bt(10,
                        bt(40)),
                bt(14)),
        bt(20,
                None,
                bt(30,
                        bt(11),
                        bt(90))))
t.is_heap_stack()
```

[11]:
```
False
```

B3 GenericTree 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.
```

Testing: `python3 -m unittest gen_tree_test.RightmostTest`

[12]:
```
from gen_tree_test import gt
t = gt('a',
           gt('b'),
           gt('c',
                  gt('e')),
           gt('d',
                  gt('f'),
                  gt('g',
                         gt('h'),
```

(continues on next page)

(continued from previous page)

```
                                gt('i'))))
t.rightmost()
```

[12]:
```
['a', 'd', 'g', 'i']
```

[ ]:
```

```

2.2 2017-18 (QCB)

See QCB master past exams on sciprolab2 Github[141]

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[142]

WARNING: keep in mind that 2016-17 exams are for Python 2 - in this course we use Python 3

[ ]:
```

```

[141] https://github.com/DavidLeoni/sciprolab2/tree/master/overlay/_static
[142] https://davidleoni.github.io/algolab/past-exams.html

CHAPTER

THREE

SLIDES 2021/22

old 2020/21 slides

3.1 Part A

3.2 Lab A.1

Thursday 23 Sep 2021

3.2.1 Links

lab site: sciprog.davidleoni.it[143]

- lab presentation[144] (slides)
- *Installation* from sciprog, with links to relevant SoftPython stuff
  - in particular, try installing the toc2 extension[145] for Jupyter for easy navigation
- Tools and Scripts[146] on softpython: Jupyter usage and other things are described more in detail, try finishing it at home.

IMPORTANT: Friday 24 September lab is cancelled. Regular meetings will start again from Thursday 30 September.

3.3 Lab A.2

Thursday, Sep 30th, 2021

Tools recap

- Python Tutor[147]
- Visual Studio Code mention
- **lab repl:** https://replit.com/@DavidLeoni2/sciprog-ds-lab-2021-22

Exercises how to:

[143] https://sciprog.davidleoni.it
[144] https://sciprog.davidleoni.it/sciprog-dslab-slides-1.pdf
[145] https://en.softpython.org/installation.html#Navigating-notebooks
[146] https://en.softpython.org/tools/tools-sol.html
[147] https://pythontutor.com/visualize.html#mode=edit

- I will update softpython / sciprog often, so please download stuff on lab day
- **NOTE 1**: when I ask you think what a certain code does, **write down** the answer, don't just think about it - to pass the exam you must have **zero** uncertainties about syntax & expressions
- **NOTE 2**: many times I will ask you to **not** use `if` statements, even if that solution would be more elegant & short: the reason is I want you to get familiar with boolean expressions, even if they may look ugly.

Basics

In class: Basics 4 challenge[148]

The rest is left as homework:

- Basics 1 ints[149]
- Basics 2 bools[150]
- Basics 3 floats[151]

Strings

In class: Strings 5 - Challenge[152]

The rest is left as homework:

- Strings 1 - Introduction[153]
- Strings 2 - Operators[154]
- Strings 3 - Basic methods[155]
- Strings 4 - Search methods[156]

References

- Andrea Passerini - Introduction to Python slides[157]
- Andrea Passerini - data structures slides[158]

3.4 Lab A.3

Friday, Oct 1st, 2021

Lists:

In class: Lists 5 - Challenges[159]

The rest is left as homework:

- Lists 1 - Introduction[160]

[148] https://en.softpython.org/basics/basics4-chal.html
[149] https://en.softpython.org/basics/basics1-ints-sol.html
[150] https://en.softpython.org/basics/basics2-bools-sol.html
[151] https://en.softpython.org/basics/basics3-floats-sol.html
[152] https://en.softpython.org/strings/strings5-chal.html
[153] https://en.softpython.org/strings/strings1-sol.html
[154] https://en.softpython.org/strings/strings2-sol.html
[155] https://en.softpython.org/strings/strings3-sol.html
[156] https://en.softpython.org/strings/strings4-sol.html
[157] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A01-introduction.pdf
[158] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A02-datastructures.pdf
[159] https://en.softpython.org/lists/lists5-chal.html
[160] https://en.softpython.org/lists/lists1-sol.html

- Lists 2 - Operators[161]
- Lists 3 - Basic methods[162]
- Lists 4 - Search methods[163]

Tuples:

In class: Tuples 2 - Challenges[164]

The rest is left as homework:

- Tuples 1 - Introduction[165]

References

- Andrea Passerini - Introduction to Python slides[166]
- Andrea Passerini - data structures slides[167]

EXTRA challenges

Since for now I want you to focus on expressions, first challenges on softpython don't require loops. Still, if you want, you can generalize them with following extra tasks:

- **Lists - Super DUPER sorted:** 2) Generalize previous approach so it works with and number of words. For this you will need cycles and/or if statements
- **Lists - Toys in the Attic**: 2) Generalize the previous approach. Imagine you have an `attic` which has an unspecified number of categories (here it's 5 but it could 1000):

```
attic = [3, 'doll',     'lego',      'minicar',
         2, 'frame',    'brushes',
         4, 'bike',     'pump',      'racket',  'ball',
         2, 'vase',     'fertilizer',
         3, 'old chair','lamp deco', 'stool']
```

and you want to obtain a list of lists like this:

```
[
    ['doll','lego','minicar'],
    ['frame', 'brushes'],
    ['bike', 'pump', 'racket', 'ball'],
    ['vase','fertilizer'],
    ['old chair','lamp deco','stool']
]
```

(without caring about the variables `toys`, `painting` and `sports`)

- In this case, **you can and should** use cycles and/or if statements when needed.

[161] https://en.softpython.org/lists/lists2-sol.html
[162] https://en.softpython.org/lists/lists3-sol.html
[163] https://en.softpython.org/lists/lists4-sol.html
[164] https://en.softpython.org/tuples/tuples2-chal.html
[165] https://en.softpython.org/tuples/tuples1-sol.html
[166] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A01-introduction.pdf
[167] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A02-datastructures.pdf

3.5 Lab A.4

Thursday, Oct 7th, 2021

References

- Andrea Passerini - Introduction to Python slides[168]
- Andrea Passerini - data structures slides[169]

3.5.1 (sets)

Passerini didn't cover them, *if you want* have a look at

- Sets 1 - Intro[170]
- Sets 2 - Challenge **NOTE**: I moved the challenge to for 5 - sets[171]

They may be useful to start, since keys in dictionaries behave much like sets.

3.5.2 Dictionaries

In class: Dictionaries 5 - Challenges[172]

- they always have a fixed number of inputs, but you may wish to generalize them with loops

The rest is left as homework:

- Dictionaries 1 - Introduction[173]
- Dictionaries 2 - Operators[174]
- Dictionaries 3 - methods[175]
- Dictionaries 4 - special dictionaries[176]

3.5.3 Conditionals

In class: if 2: challenge[177]

The rest is left as homework:

- if 1: intro[178]

References: Andrea Passerini - complex statements[179]

[168] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A01-introduction.pdf
[169] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A02-datastructures.pdf
[170] https://en.softpython.org/sets/sets1-sol.html
[171] https://en.softpython.org/for/for5-sets-sol.html
[172] https://en.softpython.org/dictionaries/dictionaries5-chal.html
[173] https://en.softpython.org/dictionaries/dictionaries1-sol.html
[174] https://en.softpython.org/dictionaries/dictionaries2-sol.html
[175] https://en.softpython.org/dictionaries/dictionaries3-sol.html
[176] https://en.softpython.org/dictionaries/dictionaries4-sol.html
[177] https://en.softpython.org/if/if2-chal.html
[178] https://en.softpython.org/if/if1-sol.html
[179] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A03-controlflow.pdf

3.5.4 only for the pros

on repl: fill_sides[180]

3.6 Lab A.5

Friday, Oct 8th, 2021

3.6.1 In class:

- for challenge[181]
- while challenge[182]
- sequences challenge[183]

References: Andrea Passerini - complex statements[184]

3.6.2 At home:

for loops

- 1. intro[185]
- 2. strings iteration[186]
- 3. lists iteration[187]
- 4. tuples iteration[188]
- 5. sets iteration[189]
- 6. dictionaries iteration[190]

while loops: while 1. intro[191]

sequences: sequences 1. intro[192]

[180] https://replit.com/@DavidLeoni2/sciprog-ds-lab-2021-22#a4_4.py
[181] https://en.softpython.org/for/for8-chal.html
[182] https://en.softpython.org/while/while2-chal.html
[183] https://en.softpython.org/sequences/sequences2-chal.html
[184] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A03-controlflow.pdf
[185] https://en.softpython.org/for/for1-intro-sol.html
[186] https://en.softpython.org/for/for2-strings-sol.html
[187] https://en.softpython.org/for/for3-lists-sol.html
[188] https://en.softpython.org/for/for4-tuples-sol.html
[189] https://en.softpython.org/for/for5-sets-sol.html
[190] https://en.softpython.org/for/for6-dictionaries-sol.html
[191] https://en.softpython.org/while/while1-sol.html
[192] https://en.softpython.org/sequences/sequences1-sol.html

3.7 Lab A.6

Thursday, Oct 14th, 2021

3.7.1 In class:

- functions intro[193]: some discussion on function categories
- functions - error handling and testing[194]: when to use them
- matrices as lists challenge[195]
- mixed structures challenge[196]

References: Andrea Passerini slides - functions[197]

3.7.2 At home:

Functions:

- 1. intro[198]
- 2. error handling and testing[199]
- 3. on strings[200]
- 4. on lists[201]
- 5. on tuples[202]
- 6. on sets[203]

Matrices as lists:

- matrices 1 - intro[204]
- matrices 2 - more exercises[205]

Mixed structures:

- mixed-structures 1 - intro[206]

[193] https://en.softpython.org/functions/fun1-intro-sol.html
[194] https://en.softpython.org/functions/fun2-errors-and-testing-sol.html
[195] https://en.softpython.org/matrices-lists/matrices-lists3-chal.html
[196] https://en.softpython.org/mixed-structures/mixed-structures2-chal.html
[197] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A04-functions.pdf
[198] https://en.softpython.org/functions/fun1-intro-sol.html
[199] https://en.softpython.org/functions/fun2-errors-and-testing-sol.html
[200] https://en.softpython.org/functions/fun3-strings-sol.html
[201] https://en.softpython.org/functions/fun4-lists-sol.html
[202] https://en.softpython.org/functions/fun5-tuples-sol.html
[203] https://en.softpython.org/functions/fun6-sets-sol.html
[204] https://en.softpython.org/matrices-lists/matrices-lists1-sol.html
[205] https://en.softpython.org/matrices-lists/matrices-lists2-sol.html
[206] https://en.softpython.org/mixed-structures/mixed-structures1-sol.html

3.8 Lab A.7

Friday, Oct 15th, 2021

3.8.1 In class:

- matrices as lists challenge[207]

3.9 Lab A.8

Thursday, Oct 21st, 2021

IMPORTANT NOTE:

- today's lab (Thursday) will start at **16:30** in b106
- on Friday 22nd there will be no lab
- next week labs restart regularly

3.9.1 In class:

- Line formats challenge[208] (Spam killer)
- JSON mention[209]
- CSV challenge[210] (Over the top)

References: Andrea Passerini slides - modules and programs[211]

3.9.2 At home:

- Formats 1: line files[212]
- Formats 2: CSV files[213]
- Formats 3: JSON files[214]
- Language parser challenges[215]

[207] https://en.softpython.org/matrices-lists/matrices-lists3-chal.html
[208] https://en.softpython.org/formats/formats4-chal.html
[209] https://en.softpython.org/formats/formats3-json-sol.html
[210] https://en.softpython.org/formats/formats4-chal.html#CSV-Challenge---Over-the-top
[211] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A05-programs.pdf
[212] https://en.softpython.org/formats/formats1-lines-sol.html
[213] https://en.softpython.org/formats/formats2-csv-sol.html
[214] https://en.softpython.org/formats/formats3-json-sol.html
[215] https://en.softpython.org/formats/formats4-chal.html#Parsing-challenge---Markdown

3.10 Lab A.9

Thursday, Oct 28th, 2021

3.10.1 Looking at the midterm

1. After this lab, consider trying the old Part A exams[216] which don't display networks.
2. **IMPORTANT**: there is a References part[217] on SoftPython, where you can find extra resource for Part A of this course. In particular, I recommend doing:
   - the explicitly named exercises from Edabit and LeetCode, plus you can find more on HackerRank and Geeks for Geeks
   - have a look at Foundations of Python programming book[218] in particular the worked projects[219]
   - If you struggle with programming, the stuff above is *not* optional!

3.10.2 In class:

- Numpy images[220]
- Algorythm visualization challenge[221]

References:

Andrea Passerini - Numpy and matplotlib[222]

3.10.3 At home:

- Numpy 1 - intro[223]
- Numpy 2 - exercises[224]
- Visualization 1 - intro[225]

[216] https://sciprog.davidleoni.it/past-exams.html
[217] https://en.softpython.org/references.html
[218] https://en.softpython.org/references.html#Foundations-of-Python-Programming
[219] https://runestone.academy/runestone/books/published/fopp/index.html#project-list
[220] https://en.softpython.org/visualization/visualization-images-sol.html
[221] https://en.softpython.org/visualization/visualization2-chal.html
[222] https://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A08-numpy.pdf
[223] https://en.softpython.org/matrices-numpy/matrices-numpy1-sol.html
[224] https://en.softpython.org/matrices-numpy/matrices-numpy2-sol.html
[225] https://en.softpython.org/visualization/visualization1-sol.html

3.11 Lab A.10

Friday, Oct 29th, 2021

3.11.1 In class:

- Pandas poets challenge[226]

References:

Andrea Passerini - Pandas[227]

3.11.2 At home:

- Pandas 1: intro[228]
- Pandas 2: EURES case study[229]

3.12 Lab A.11

Thursday, Nov 4th, 2021

Today we are going to see applications, particular how to display relational data.

Note focus in Part A is just displaying, **not** exploration algorithms (which we will see in Part B)

First install required libraries[230] (graphviz, pydot, networkx)

If for some reason GraphViz doesn't work on your system: ask/mail mail me, and in the meanwhile you could use an unholy hack on repl[231] which also uses the online service Gravizo and matplotlib (you can also run it from your laptop). Just run it, wait for the thing to load all the modules, and you should see some graph - notice you might need to properly center it.

3.12.1 Looking at the midterm

After this lab, consider trying the remaining old Part A exams[232] which display networks.

[226] https://en.softpython.org/pandas/pandas3-chal.html
[227] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A07-pandas.pdf
[228] https://en.softpython.org/pandas/pandas1-sol.html
[229] https://en.softpython.org/pandas/pandas2-sol.html
[230] https://en.softpython.org/relational/relational1-intro-sol.html#Required-libraries
[231] https://replit.com/@DavidLeoni2/softgraphviz2#main.py
[232] https://sciprog.davidleoni.it/past-exams.html

3.12.2 In class:

- Relational data challenge[233]

In class try doing one exercise per type, es:

- matrices: Trichain, Bipartite (from soft import draw_mat)
- adjacency lists: Friends (from soft import draw_adj)
- networkx: Offshore (from soft import draw_nx)

3.12.3 At home:

- Relational data 1: intro[234]
- Relational data 2: Binary relations[235]
- Relational data 3: Simple statistics[236] (contains also some matplotlib charts)

3.13 Lab B.1

Thursday Nov 11th, 2021

New tutoring service

by Gabriele Masina (gabriele.masina (guess what) studenti.unitn.it) has been set up on the following timetable starting from Monday 15 november until Wednesday 15 december (included)

- Mondays: 9:30-11:30 A202
- Wednesdays: 9:30-11:30 A202

Part B Focus

- complex data structures
- perfomance

How:

- big .py files
- debugging
- testing

VSCode:

From now on we will:

- only use Visual Studio Code
- only edit .py files

Read:

- *Installation on sciprog*

[233] https://en.softpython.org/relational/relational4-chal.html
[234] https://en.softpython.org/relational/relational1-intro-sol.html
[235] https://en.softpython.org/relational/relational2-binrel-sol.html
[236] https://en.softpython.org/relational/relational3-simple-stats-sol.html

- *Debugging with VSCode*
- Collaborative editing with VSCode: for this year there's hopefully no need to use it, but sometimes you might find it handy, in case see how to *install sharing extension* and *sharing test runs*

Btw, if you want you can even load jupyter notebooks (.ipynb) inside VSCode

New testing: In part B we use unittest: *see errors and testing on sciprog*

- in particular, DO READ *TROUBLESHOOTING* - you are very likely to encounter problems

Let's go object oriented: *OOP* (only first part, next time we will do challenges together)

3.14 Lab B.2

Thursday Nov 18th, 2021

3.14.1 OOP

- Finish *OOP ComplexNumber*
- OOP Matrix Challenge[237]

3.14.2 Indexing

Have a look at *Indexing chapter*: probably you will have a hard time understanding the exercises and your solution will look different from proposed ones: try asking yourself why indexing with dictionaries might save computation time

From now on, in general try avoiding:

- `in` operator on sequences like strings, lists, tuples: it scans the whole sequence!
- search methods (`.find`, `.index`, `.remove`...)
  - in particular methods from strings (`.replace`,etc): PartB is about algorithms, not Python tricks!
- `pop(k)` (but `pop(-1)` is fine)
- needlessly using slices, they create new data structures!

3.15 Lab B.3

Friday Nov 19th, 2021

worksheets on sciprog:

- *Recursion 1 - simplefp*
- *Recursion 2 - accumulators and indexes*

NOTE: the worksheet is in progress, I will surely add material on multiple recursion

In general, when you don't know how to tackle a recursive problem, first try the SimpleFP way and then ModAcc

Simple functional programming (SimpleFP)

[237] https://sciprog.davidleoni.it/oop/oop2-matrix-chal.html

- immutability
- very limited instructions set
- never re-assign variables
- always create NEW memory regions
- don't care about performance

Recursion with modifiable accumulators (ModAcc)

- indexes instead of slicing
- extra accumulators variables
- helper functions
- you can mutate accumulators
- care more about performance

References:

- Andrea Passerini slides on programming paradigms[238]
- Andrea Passerini slides on recursion theory[239]

3.16 Lab B.4

Thursday Nov 25th, 2021

I split recursion into many pages and added stuff, see *Recursion 1 - simplefp*, *Recursion 2 - accumulators and indexes* and Recursion 4 - challenges[240]

Today we see:

- Recursion 3 - divide and conquer[241]
- Algorithm analysis[242]

3.17 Lab B.5

Friday Nov 26th, 2021

In lab:

- More Algorithm analysis[243]
- Sorting[244]
  - Selection sort
  - quicksort

[238] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A09-oop.pdf
[239] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A06-recursion.pdf
[240] https://sciprog.davidleoni.it/recursion/recursion4-chal.html
[241] https://sciprog.davidleoni.it/recursion/recursion3-divide.html
[242] https://sciprog.davidleoni.it/algo-analysis/algo-analysis-sol.html
[243] https://sciprog.davidleoni.it/algo-analysis/algo-analysis-sol.html
[244] https://sciprog.davidleoni.it/sorting/sorting.html

At home:

- Other sorting (insertion sort, merge sort, …)
- Sorting challenge[245]

3.18 Lab B.6

Thursday Dec 2nd, 2021

In lab:

LinkedLists 1[246]: the art of

- scanning with `while`
- pointers
- cutting

Please start doing exercises in Part B references[247]

At home:

- Stacks[248]: **in particular, make sure to understand the Train race exercise**
- Linked list2 challenges[249] (will add more, also during tutoring)

3.19 Lab B.7

Friday Dec 3thrd, 2021

In lab:

- *Queues 1 intro*: ItalianQueue (it's actually a LinkedList)
- *Queues 2 CircularQueue*

Again: Please start doing exercises in Part B references[250]

At home: try finishing whole Queues worksheets

3.20 Lab B.8

Thursday Dec 9th, 2021

XI COMMANDMENT: Never ever search keys in a dictionary with looping!

DO NOT write code like this, it's $O(n)$:

[245] https://sciprog.davidleoni.it/sorting/sorting-chal.html
[246] https://sciprog.davidleoni.it/linked-lists/linked-lists1.html
[247] https://sciprog.davidleoni.it/references.html
[248] https://sciprog.davidleoni.it/stacks/stacks.html
[249] https://sciprog.davidleoni.it/linked-lists/linked-lists2-chal.html
[250] https://sciprog.davidleoni.it/references.html

```
for key in d:
    if key == 'searched key':
        do something ...
```

USE instead `in` operator which only takes $O(1)$

```
if 'searched key' in d:
    do something
```

Looping on dictionaries kills performance and in Part B is often a fatal mistake, many people still have this bad habit. Please (re) read softpython - iteration in dictionaries[251] and complexity - dictionaries[252]

Today:

- *Binary trees*

At home: try to finish whole Binary trees worksheet

Yet again: Please start doing exercises in Part B references[253]

3.21 Lab B.9

Friday Dec 10th, 2021

Generic trees, at home: try to finish whole Generic trees worksheet

- Further exercise: try doing bintree exercses also on generic trees.

[251] https://en.softpython.org/for/for6-dictionaries-sol.html#Are-they-equivalent?---Found
[252] https://sciprog.davidleoni.it/algo-analysis/algo-analysis-sol.html#Dictionaries-performance
[253] https://sciprog.davidleoni.it/references.html

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[254] (command line)
- Visual Studio Code[255] (editor, good debugger)
- Jupyter[256] (notebook, good for experimentation and writing reports)
- Google Colab[257] (online, sort-of collaborative)
- repl.it[258] (online, collaborative)
- Python Tutor[259] (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[260] (anaconda installer may ask you to install also visual studio code, in case 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[261]. Downloads for all platforms can be found here[262]
3. Install Python extension for Visual Studio Code[263] by Microsoft

[254] https://www.python.org/
[255] https://code.visualstudio.com/
[256] https://jupyter.org/
[257] https://colab.research.google.com/
[258] https://repl.it/languages/python3
[259] http://www.pythontutor.com/visualize.html#mode=edit
[260] https://www.anaconda.com/products/individual#Downloads
[261] https://code.visualstudio.com/
[262] https://code.visualstudio.com/Download
[263] https://marketplace.visualstudio.com/items?itemName=ms-python.python

4. Now you can read all of Installation on SoftPython[264] **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!)
5. **Jupyter and course material format** is described in detail in Tools and Scripts on SoftPython[265], read it!
6. 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[266] 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:

[264] https://en.softpython.org/installation.html
[265] https://en.softpython.org/tools/tools-sol.html
[266] https://code.visualstudio.com/

You can find useful information on this tool here[267]. 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.

[267] https://code.visualstudio.com/docs#vscode

Finally, select the Python version you want to use i.e. Python3 (if you have anaconda, it should automatically use your Anaconda environment)

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)
```

```
a: 10  b: 33  a/b= 0.30303030303030304
```

A couple of things worth nothing. The first three lines opened and closed by `"""` are some text describing the content of the script. Moreover, comments are proceeded by the hash key (#) and they are just ignored by the python interpreter. **Please remember to comment your code, as it helps readability and will make your life easier when you have to modify or just understand the code you wrote some time in the past.**

Please notice that Visual Studio Code will help you writing your Python scripts. For example, when you start writing the **print** line it will complete the code for you (**if the Pylint extension mentioned above is installed**), suggesting the functions that match the letters written. This useful feature is called *code completion* and, alongside suggesting possible matches, it also visualizes a description of the function and parameters it needs. Here is an example:

Save the file (`Ctrl+S` as shortcut). It is convenient to ask the IDE to highlight potential *syntactic* problems found in the code. You can toggle this function on/off by clicking on **View –> Problems**. The *Problems* panel should look like this

Visual Studio Code is warning us that the variable names `a`,`b`,`c` at lines 4,5,6 do not follow Python naming conventions for constants. This is because they have been defined at the top level (there is no structure to our script yet) and therefore are interpreted as constants. The naming convention for constants states that they should be in capital letters. To amend the code, you can just replace all the names with the corresponding capitalized name (i.e. A,B,C). If you do that, and you save the file again (Ctrl+S), you will see all these problems disappearing as well as the green underlining of the variable names. If your code does not have an empty line before the end, you might get another warning "*Final new line missing*". Note that these were just warnings and the interpreter **in this case** will happily and correctly execute the code anyway,

but it is always good practice to understand what the warnings are telling us before deciding to ignore them!

Had we by mistake mispelled the *print* function name (something that should not happen with the code completion tool that suggests functions names!) writing *printt* (note the double t), upon saving the file, the IDE would have underlined in red the function name and flagged it up as a problem.

This is because the builtin function *printt* does not exist and the python interpreter does not know what to do when it reads it. Note that *printt* is actually underlined in red, meaning that there is an error which will cause the interpreter to stop the execution with a failure. **Please remember that before running any piece of code all errors must be fixed**.

Now it is time to execute the code. By right-clicking in the code panel and selecting **Run Python File in Terminal** (see picture below) you can execute the code you have just written.

Upon clicking on *Run Python File in Terminal* a terminal panel should pop up in the lower section of the coding panel and the result shown above should be reported.

Saving script files like the *example1.py* above is also handy because they can be invoked several times (later on we will learn how to get inputs from the command line to make them more useful…). To do so, you just need to call the python intepreter passing the script file as parameter. From the folder containing the *example1.py* script:

```
python3 example1.py
```

will in fact return:

a: 10 b: 33 a/b= 0.30303030303030304

Before ending this section, let me add another note on errors. The IDE will diligently point you out *syntactic* warnings and errors (i.e. errors/warnings concerning the structure of the written code like name of functions, number and type of parameters, etc.) but it will not detect *semantic* or *runtime* errors (i.e. connected to the meaning of your code or to the value of your variables). These sort of errors will most probably make your code crash or may result in unexpected results/behaviours. In the next section we will introduce the debugger, which is a useful tool to help detecting these errors.

Before getting into that, consider the following lines of code (do not focus on the *import* line, this is only to load the mathematics module and use its method *sqrt*):

[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[268] installed. It should already be included in VS Code 2019, if not got to extensions and install it:

[268] 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[269]

NOTE: in order to share your folder, you will need a Github account[270], so please if you don't have it already create one now

4. Share the folder: See the official How To[271]

6.1.4 VS Code - Sharing test runs

In Part B we will also heavily use unit tests (see Unittest tutorial[272]), so it's a good idea to enabling sharing of test runs by installing Test Explorer Live Share[273] 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.

[269] https://docs.microsoft.com/en-us/visualstudio/liveshare/use/vscode#sign-in
[270] https://github.com
[271] https://docs.microsoft.com/en-us/visualstudio/liveshare/use/vscode#share-a-project
[272] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-Unittest
[273] 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[274]

```
[ ]:
```

[274] https://en.softpython.org/basics/basics-sol.html

6.3 Strings solutions

6.3.1 MOVED TO https://en.softpython.org/#strings

6.4 Lists solutions

6.4.1 MOVED TO https://en.softpython.org/#lists

6.5 Tuples solutions

6.5.1 MOVED TO https://en.softpython.org/tuples/tuples-sol.html

[ ]:

```

```

6.6 Sets solutions

6.6.1 MOVED TO https://en.softpython.org/sets/sets1-sol.html

6.7 Dictionaries solutions

6.7.1 MOVED TO https://en.softpython.org/#dictionaries

[ ]:

```

```

6.8 Control flow solutions

6.8.1 MOVED TO https://en.softpython.org/#control-flow

6.9 Functions - solutions

6.9.1 Download exercises zip

Browse files online[275]

[275] https://github.com/DavidLeoni/sciprog-ds/tree/master/functions

6.9.2 Introduction

References:

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

In this notebook we will see how to define functions to reuse code, and talk about the scope of variables

References

- Andrea Passerini slides A04[276]
- Thinking in Python, Chapter 3, Functions[277]
- Thinking in Python, Chapter 6, Fruitful functions[278] **NOTE**: in the book they use the weird term 'fruitful functions' for those functions which RETURN a value (mind you, RETURN a value, which is different from PRINTing it), and use also the term 'void functions' for functions which do not return anything but have some effect like PRINTing to screen. Please ignore these terms.

What to do

- unzip exercises in a folder, you should get something like this:

```
-jupman.py
-exercises
     |- functions
         |- functions.ipynb
         |- functions-sol.ipynb
```

WARNING: to correctly visualize the notebook, it MUST be in an unzipped folder !

- open Jupyter Notebook from that folder. Two things should open, first a console and then browser. The browser should show a file list: navigate the list and open the notebook `functions/functions.ipynb`
- Go on reading that notebook, and follow instuctions inside.

Shortcut keys:

- to execute Python code inside a Jupyter cell, press `Control + Enter`
- to execute Python code inside a Jupyter cell AND select next cell, press `Shift + Enter`
- to execute Python code inside a Jupyter cell AND a create a new cell aftwerwards, press `Alt + Enter`
- If the notebooks look stuck, try to select `Kernel -> Restart`

[276] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A04-functions.pdf
[277] http://greenteapress.com/thinkpython2/html/thinkpython2004.html
[278] http://greenteapress.com/thinkpython2/html/thinkpython2007.html

What is a function ?

A function is a block of code that has a name and that performs a task. A function can be thought of as a box that gets an input and returns an output.

Why should we use functions? For a lot of reasons including:

1. *Reduce code duplication*: put in functions parts of code that are needed several times in the whole program so that you don't need to repeat the same code over and over again;
2. *Decompose a complex task*: make the code easier to write and understand by splitting the whole program in several easier functions;

both things improve code readability and make your code easier to understand.

The basic definition of a function is:

```
def function_name(input) :
    #code implementing the function
    ...
    ...
    return return_value
```

Functions are defined with the **def** keyword that proceeds the *function_name* and then a list of parameters is passed in the brackets. A colon **:** is used to end the line holding the definition of the function. The code implementing the function is specified by using indentation. A function **might** or **might not** return a value. In the first case a **return** statement is used.

Example:

Define a function that implements the sum of two integer lists (note that there is no check that the two lists actually contain integers and that they have the same size).

[2]:
```
def int_list_sum(la,lb):
    """implements the sum of two lists of integers having the same size
    """
    ret =[]
    for i in range(len(la)):
        ret.append(la[i] + lb[i])
    return ret

La = list(range(1,10))
print("La:", La)
```

```
La: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[3]:
```
Lb = list(range(20,30))
print("Lb:", Lb)
```

```
Lb: [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
```

[4]:
```
res = int_list_sum(La,Lb)
```

[5]:
```
print("La+Lb:", res)
```

```
La+Lb: [21, 23, 25, 27, 29, 31, 33, 35, 37]
```

[6]:
```
res = int_list_sum(La,La)
```

[7]:
```
print("La+La", res)
```

```
La+La [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Note that once the function has been defined, it can be called as many times as wanted with different input parameters. Moreover, **a function does not do anything until it is actually called.** A function can return **0** (in this case the return value would be "None"), **1** or **more** results. Notice also that collecting the results of a function is **not mandatory**.

Example: Let's write a function that, given a list of elements, prints only the even-placed ones without returning anything.

[8]:
```
def get_even_placed(myList):
    """returns the even placed elements of myList"""
    ret = [myList[i] for i in range(len(myList)) if i % 2 == 0]
    print(ret)
```

[9]:
```
L1 = ["hi", "there", "from","python","!"]
```

[10]:
```
L2 = list(range(13))
```

[11]:
```
print("L1:", L1)
```

```
L1: ['hi', 'there', 'from', 'python', '!']
```

[12]:
```
print("L2:", L2)
```

```
L2: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

[13]:
```
print("even L1:")
get_even_placed(L1)
```

```
even L1:
['hi', 'from', '!']
```

[14]:
```
print("even L2:")
get_even_placed(L2)
```

```
even L2:
[0, 2, 4, 6, 8, 10, 12]
```

Note that the function above is polymorphic (i.e. it works on several data types, provided that we can iterate through them).

Example: Let's write a function that, given a list of integers, returns the number of elements, the maximum and minimum.

[15]:
```
def get_info(myList):
    """returns len of myList, min and max value (assumes elements are integers)"""
    tmp = myList[:] #copy the input list
    tmp.sort()
    return len(tmp), tmp[0], tmp[-1] #return type is a tuple

A = [7, 1, 125, 4, -1, 0]

print("Original A:", A, "\n")
```

```
Original A: [7, 1, 125, 4, -1, 0]
```

[16]:
```
result = get_info(A)
```

[17]:
```
print("Len:", result[0], "Min:", result[1], "Max:",result[2], "\n" )
```

```
Len: 6 Min: -1 Max: 125
```

[18]:
```
print("A now:", A)
```

```
A now: [7, 1, 125, 4, -1, 0]
```

[19]:
```
def my_sum(myList):
    ret = 0
    for el in myList:
        ret += el # == ret = ret + el
    return ret

A = [1,2,3,4,5,6]
B = [7, 9, 4]
```

[20]:
```
s = my_sum(A)
```

[21]:
```
print("List A:", A)
print("Sum:", s)
```

```
List A: [1, 2, 3, 4, 5, 6]
Sum: 21
```

[22]:
```
s = my_sum(B)
```

[23]:
```
print("List B:", B)
print("Sum:", s)
```

```
List B: [7, 9, 4]
Sum: 20
```

Please note that the return value above is actually a tuple. Importantly enough, a function needs to be defined (i.e. its code has to be written) before it can actually be used.

[24]:
```
A = [1,2,3]
my_sum(A)

def my_sum(myList):
    ret = 0
    for el in myList:
        ret += el
    return ret
```

6.9.3 Namespace and variable scope

Namespaces are mappings from *names* to objects, or in other words places where names are associated to objects. Namespaces can be considered as the context. According to Python's reference a **scope** is a *textual region of a Python program, where a namespace is directly accessible*, which means that Python will look into that *namespace* to find the object associated to a name. Four **namespaces** are made available by Python:

1. **Local**: the innermost that contains local names (inside a function or a class);
2. **Enclosing**: the scope of the enclosing function, it does not contain local nor global names (nested functions) ;
3. **Global**: contains the global names;
4. **Built-in**: contains all built in names (e.g. print, if, while, for,…)

When one refers to a name, Python tries to find it in the current namespace, if it is not found it continues looking in the namespace that contains it until the built-in namespace is reached. If the name is not found there either, the Python interpreter will throw a **NameError** exception, meaning it cannot find the name. The order in which namespaces are considered is: Local, Enclosing, Global and Built-in (LEGB).

Consider the following example:

[25]:
```
def my_function():
    var = 1  #local variable
    print("Local:", var)
    b = "my string"
    print("Local:", b)
```

```
var = 7 #global variable
my_function()
print("Global:", var)
print(b)
```

```
Local: 1
Local: my string
Global: 7

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-56-7dd8330a24f0> in <module>
      8 my_function()
      9 print("Global:", var)
---> 10 print(b)

NameError: name 'b' is not defined
```

Variables defined within a function can only be seen within the function. That is why variable b is defined only within the function. Variables defined outside all functions are **global** to the whole program. The namespace of the local variable is within the function my_function, while outside it the variable will have its global value.

And the following:

[26]:
```
def outer_function():
    var = 1 #outer

    def inner_function():
        var = 2 #inner
        print("Inner:", var)
```

(continues on next page)

(continued from previous page)

```
        print("Inner:", B)

    inner_function()
    print("Outer:", var)


var = 3 #global
B = "This is B"
outer_function()
print("Global:", var)
print("Global:", B)
```

```
Inner: 2
Inner: This is B
Outer: 1
Global: 3
Global: This is B
```

Note in particular that the variable B is global, therefore it is accessible everywhere and also inside the inner_function. On the contrary, the value of var defined within the inner_function is accessible only in the namespace defined by it, outside it will assume different values as shown in the example.

In a nutshell, remember the three simple rules seen in the lecture. Within a **def**:

```
1. Name assignments create local names by default;
2. Name references search the following four scopes in the order:
local, enclosing functions (if any), then global and finally built-in (LEGB)
3. Names declared in global and nonlocal statements map assigned names to
enclosing module and function scopes.
```

6.9.4 Argument passing

Arguments are the parameters and data we pass to functions. When passing arguments, there are three important things to bear in mind are:

1. Passing an argument is actually assigning an object to a local variable name;
2. Assigning an object to a variable name within a function **does not affect the caller**;
3. Changing a **mutable** object variable name within a function **affects the caller**

Consider the following examples:

[27]:
```
"""Assigning the argument does not affect the caller"""

def my_f(x):
    x = "local value" #local
    print("Local: ", x)

x = "global value" #global
my_f(x)
print("Global:", x)
my_f(x)
```

```
Local:  local value
Global: global value
Local:  local value
```

[28]:
```
"""Changing a mutable affects the caller"""

def my_f(myList):
    myList[1] = "new value1"
    myList[3] = "new value2"
    print("Local: ", myList)

myList = ["old value"]*4
print("Global:", myList)
my_f(myList)
print("Global now: ", myList)
```

```
Global: ['old value', 'old value', 'old value', 'old value']
Local:  ['old value', 'new value1', 'old value', 'new value2']
Global now:  ['old value', 'new value1', 'old value', 'new value2']
```

Recall what seen in the lecture:

The behaviour above is because **immutable objects** are passed **by value** (therefore it is like making a copy), while **mutable objects** are passed **by reference** (therefore changing them effectively changes the original object).

To avoid making changes to a **mutable object** passed as parameter one needs to **explicitely make a copy** of it.

Consider the example seen before. **Example:** Let's write a function that, given a list of integers, returns the number of elements, the maximum and minimum.

[29]:
```
def get_info(myList):
    """returns len of myList, min and max value (assumes elements are integers)"""
    myList.sort()
    return len(myList), myList[0], myList[-1] #return type is a tuple


def get_info_copy(myList):
    """returns len of myList, min and max value (assumes elements are integers)"""
```

(continues on next page)

(continued from previous page)

```
    tmp = myList[:] #copy the input list!!!!
    tmp.sort()
    return len(tmp), tmp[0], tmp[-1] #return type is a tuple

A = [7, 1, 125, 4, -1, 0]
B = [70, 10, 1250, 40, -10, 0, 10]

print("A:", A)
result = get_info(A)
```

```
A: [7, 1, 125, 4, -1, 0]
```

[30]:
```
print("Len:", result[0], "Min:", result[1], "Max:",result[2] )
```

```
Len: 6 Min: -1 Max: 125
```

[31]:
```
print("A now:", A) #whoops A is changed!!!
```

```
A now: [-1, 0, 1, 4, 7, 125]
```

[32]:
```
print("\nB:", B)
```

```

B: [70, 10, 1250, 40, -10, 0, 10]
```

[33]:
```
result = get_info_copy(B)
```

[34]:
```
print("Len:", result[0], "Min:", result[1], "Max:",result[2] )
```

```
Len: 7 Min: -10 Max: 1250
```

[35]:
```
print("B now:", B) #B is not changed!!!
```

```
B now: [70, 10, 1250, 40, -10, 0, 10]
```

Positional arguments

Arguments can be passed to functions following the order in which they appear in the function definition.

Consider the following example:

[36]:
```
def print_parameters(a,b,c,d):
    print("1st param:", a)
    print("2nd param:", b)
    print("3rd param:", c)
    print("4th param:", d)

print_parameters("A", "B", "C", "D")
```

```
1st param: A
2nd param: B
3rd param: C
4th param: D
```

Passing arguments by keyword

Given the name of an argument as specified in the definition of the function, parameters can be passed using the **name = value** syntax.

For example:

[37]:
```
def print_parameters(a,b,c,d):
    print("1st param:", a)
    print("2nd param:", b)
    print("3rd param:", c)
    print("4th param:", d)

print_parameters(a = 1, c=3, d=4, b=2)
```

```
1st param: 1
2nd param: 2
3rd param: 3
4th param: 4
```

[38]:
```
print_parameters("first","second",d="fourth",c="third")
```

```
1st param: first
2nd param: second
3rd param: third
4th param: fourth
```

Arguments passed positionally and by name can be used at the same time, but parameters passed by name must always be to the left of those passed by name. The following code in fact is not accepted by the Python interpreter:

```
def print_parameters(a,b,c,d):
    print("1st param:", a)
    print("2nd param:", b)
    print("3rd param:", c)
    print("4th param:", d)

print_parameters(d="fourth",c="third", "first","second")
```

```
File "<ipython-input-60-4991b2c31842>", line 7
    print_parameters(d="fourth",c="third", "first","second")
                                           ^
SyntaxError: positional argument follows keyword argument
```

Specifying default values

During the definition of a function it is possible to specify default values. The syntax is the following:

```
def my_function(par1 = val1, par2 = val2, par3 = val3):
```

Consider the following example:

[39]:
```
def print_parameters(a="defaultA", b="defaultB",c="defaultC"):
    print("a:",a)
    print("b:",b)
    print("c:",c)

print_parameters("param_A")
```

```
a: param_A
b: defaultB
c: defaultC
```

[40]:
```
print_parameters(b="PARAMETER_B")
```

```
a: defaultA
b: PARAMETER_B
c: defaultC
```

[41]:
```
print_parameters()
```

```
a: defaultA
b: defaultB
c: defaultC
```

[42]:
```
print_parameters(c="PARAMETER_C", b="PAR_B")
```

```
a: defaultA
b: PAR_B
c: PARAMETER_C
```

6.9.5 Simple exercises

sum2

⊛ Write function `sum2` which given two numbers `x` and `y` RETURN their sum

QUESTION: Why do we call it `sum2` instead of just `sum` ??

[43]:
```
sum([2,51])
```

[43]:
```
53
```

<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):
```

(continues on next page)

(continued from previous page)

```
    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[279]

Tests with asserts[280]: Following exercises contain automated tests to help you spot errors. To understand how to do them, read before Error handling and testing[281]

[279] http://disi.unitn.it/~passerini/teaching/2019-2020/sci-pro/slides/A03-controlflow.pdf
[280] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html#Testing-with-asserts
[281] https://sciprog.davidleoni.it/errors-and-testing/errors-and-testing-sol.html

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
```

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
```

(continues on next page)

(continued from previous page)

```
    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)):
```

(continues on next page)

(continued from previous page)

```
        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[282]

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[283]
- **Part B testing with unittest**: read this notebook

6.10.1 Testing

- If it seems to work, then it actually works? *Probably not*.
- The devil is in the details, especially for complex algorithms.
- We will do a crash course on testing in Python

> **WARNING**: Bad software can cause losses of million $/€ or even harm people. Suggested reading: Software Horror Stories[284]

[282] https://en.softpython.org/functions/fun2-errors-and-testing-sol.html
[283] https://en.softpython.org/functions/fun2-errors-and-testing-sol.html#Testing-with-asserts
[284] https://www.cs.tau.ac.il/~nachumd/horror.html

Where Is Your Software?

As a data scientist, you might likely end up with code which is moderately complex from an algorithmic point of view, but maybe not too big in size. Either way, when red line is crossed you should start testing properly:

In a typical scenario, you are a junior programmer and your senior colleague ask you to write a function to perform some task, giving only an informal description:

```
def my_sum(x,y):
    """ RETURN the sum of x and y
    """
    raise Exception("TODO IMPLEMENT ME!")
```

Even better, your colleague might provide you with some automated tests you might run to check your function meets his/her expectations. If you are smart, you will even write tests for your own functions to make sure every little piece you add to your software is a solid block you can build upon.

even_numbers example

Let's see a slightly more complex function:

```
[2]: def even_numbers(n):
         """
         Return a list of the first n even numbers

         Zero is considered to be the first even number.

         >>> even_numbers(5)
         [0,2,4,6,8]
         """
         raise Exception("TODO IMPLEMENT ME!")
```

In this case, if you run the function as it is, you are reminded to implement it:

```
>>> even_numbers(5)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-2-d2cbc915c576> in <module>()
----> 1 even_numbers(5)

<ipython-input-1-a20a4ea4b42a> in even_numbers(n)
      8     [0,2,4,6,8]
      9     """
---> 10     raise Exception("TODO IMPLEMENT ME!")

Exception: TODO IMPLEMENT ME!
```

Why? The instruction

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

tells Python to immediatly stop execution, and signal an error to the caller of the function `even_number`. If there were commands right after `raise Exception("TODO IMPLEMENT ME")`, they would not be executed. Here, we are directly calling the function from the prompt, and we didn't tell Python how to handle the `Exception`, so Python just stopped and showed the error message given as parameter to the `Exception`

Spend time reading the function text!

Always carefully read the function text and ask yourself questions! What is the supposed input? What should be the output? Is there any output to return at all, or should you instead modify *in-place* a passed parameter (i.e. for example, when you sort a list)? Are there any edge cases, es what happens for `n=0`)? What about `n < 0` ?

Let's code a possible solution. As it often happens, first version may be buggy, in this case for example purposes we intentionally introduce a bug:

[3]:
```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    r = [2 * x for x in range(n)]
    r[n // 2] = 3   # <-- evil bug, puts number '3' in the middle, and 3 is not even .
↪.
    return r
```

Typically the first test we do is printing the output and do some 'visual inspection' of the result, in this case we find many numbers are correct but we might miss errors such as the wrong `3` in the middle:

[4]:
```
print(even_numbers(5))
```

```
[0, 2, 3, 6, 8]
```

Furthermore, if we enter commands a the prompt, each time we fix something in the code, we need to enter commands again to check everything is ok. This is inefficient, boring, and prone to errors.

Let's add assertions

To go beyond the dumb "visual inspection" testing, it's better to write some extra code to allow Python checking for us if the function actually returns what we expect, and throws an error otherwise. We can do so with `assert` command, which verifies if its argument is True. If it is not, it raises an `AssertionError` immediately stopping execution.

Here we check the result of `even_numbers(5)` is actually the list of even numbers `[0,2,4,6,8]` we expect:

```
assert even_numbers(5) == [0,2,4,6,8]
```

Since our code is faulty, `even_numbers` returns the wrong list `[0,2,3,6,8]` which is different from `[0,2,4,6,8]` so assertion fails showing `AssertionError`:

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-21-d4198f229404> in <module>()
----> 1 assert even_numbers(5) != [0,2,4,6,8]

AssertionError:
```

We got some output, but we would like to have it more informative. To do so, we may add a message, separated by a comma:

```
assert even_numbers(5) == [0,2,4,6,8], "even_numbers is not working !!"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-18-8544fcd1b7c8> in <module>()
----> 1 assert even_numbers(5) == [0,2,4,6,8], "even_numbers is not working !!"

AssertionError: even_numbers is not working !!
```

So if we modify code to fix bugs we can just launch the assert commands and have a quick feedback about possible errors.

Error kinds

As a fact of life, errors happen. Sometimes, your program may have inconsistent data, like wrong parameter type passed to a function (i.e. string instead of integer). A good principle to follow in these cases is to try have the program detect weird situations, and stop as early as such a situation is found (i.e. in the Therac 25 case, if you detect excessive radiation, showing a warning sign is not enough, it's better to stop). Note stopping might not always be the desirable solution (if one pidgeon enters one airplane engine, you don't want to stop all the other engines). If you want to check function parameters are correct, you do the so called *precondition checking*.

There are roughly two cases for errors, external user misusing you program, and just plain wrong code. Let's analyize both:

Error kind a) An external user misuses you program.

You can assume whover uses your software, final users or other programmers , they will try their very best to wreck your precious code by passing all sort of non-sense to functions. Everything can come in, strings instead of numbers, empty arrays, `None` objects … In this case you should signal the user he made some mistake. The most crude signal you can have is raising an `Exception` with `raise Exception("Some error occurred")`, which will stop the program and print the stacktrace in the console. Maybe final users won't understand a stacktrace, but at least programmers hopefully will get a clue about what is happening.

In these case you can raise an appropriate Exception, like TypeError[285] for wrong types and ValueError[286] for more generic errors. Other basic exceptions can be found in Python documentation[287]. Notice you can also define your own, if needed (we won't consider custom exceptions in this course).

NOTE: Many times, you can consider yourself the 'careless external user' to guard against.

Let's enrich the function with some appropriate type checking:

Note that for checking input types, you can use the function `type()` :

[5]:
```
type(3)
```

[5]:
```
int
```

[6]:
```
type("ciao")
```

[6]:
```
str
```

Let's add the code for checking the *even_numbers example*:

[7]:
```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    if type(n) is not int:
        raise TypeError("Passed a non integer number: " + str(n))

    if n < 0:
        raise ValueError("Passed a negative number: " + str(n))

    r = [2 * x for x in range(n)]
    return r
```

Let's pass a wrong type and see what happens:

```
>>> even_numbers("ciao")

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-a908b20f00c4> in <module>()
```

(continues on next page)

[285] https://docs.python.org/3/library/exceptions.html#TypeError
[286] https://docs.python.org/3/library/exceptions.html#ValueError
[287] https://docs.python.org/3/library/exceptions.html#built-in-exceptions

(continued from previous page)

```
----> 1 even_numbers("ciao")

<ipython-input-13-b0b3a85f2b2a> in even_numbers(n)
      9     """
     10     if type(n) is not int:
---> 11         raise TypeError("Passed a non integer number: " + str(n))
     12
     13     if n < 0:

TypeError: Passed a non integer number: ciao
```

Now let's try to pass a negative number - it should suddenly stop with a meaningful message:

```
>>> even_numbers(-5)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-3f648fdf6de7> in <module>()
----> 1 even_numbers(-5)

<ipython-input-13-b0b3a85f2b2a> in even_numbers(n)
     12
     13     if n < 0:
---> 14         raise ValueError("Passed a negative number: " + str(n))
     15
     16     r = [2 * x for x in range(n)]

ValueError: Passed a negative number: -5
```

Now, even if you ship your code to careless users, and as soon as they commit a mistrake, they will get properly notified.

Error kind b): Your code is just plain wrong

In this case, it's 100% your fault, and these sort of bugs should never pop up in production. For example your code passes internally wrong stuff, like strings instead of integers, or wrong ranges (typically integer outside array bounds). So if you have an internal function nobody else should directly call, and you suspect it is being passed wrong parameters or at some point it has inconsistent data, to quickly spot the error you could add an assertion:

[8]:
```
def even_numbers(n):
    """
    Return a list of the first n even numbers

    Zero is considered to be the first even number.

    >>> even_numbers(5)
    [0,2,4,6,8]
    """
    assert type(n) is int, "type of n is not correct: " + str(type(n))
    assert n >= 0, "Found negative n: " + str(n)

    r = [2 * x for x in range(n)]

    return r
```

As before, the function will stop as soon we call it we wrong parameters. The big difference is, this time we are assuming `even_numbers` is just for personal use and nobody else except us should directly call it.

Since assertion consume CPU time, IF we care about performances AND once we are confident our program behaves correctly, we can even remove them from compiled code by using the `-O` compiler flag. For more info, see Python wiki[288]

EXERCISE: try to call latest definition of `even_numbers` with wrong parameters, and see what happens.

NOTE: here we are using the correct definition of `even_numbers`, not the buggy one with the `3` in the middle of returned list !

6.10.2 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[289]. 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[290] (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:

[9]:
```
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[291]), 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:

[288] https://wiki.python.org/moin/UsingAssertionsEffectively
[289] https://docs.python.org/3/library/unittest.html
[290] https://docs.pytest.org/
[291] http://interactivepython.org/runestone/static/pythonds/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html

- method is declared like a function, and begins with `'test_'` word
- method takes `self` as parameter
- `self.assertEqual(even_numbers(5),[0,2,4,6,8])` executes the assertion. Other assertions could be `self.assertTrue(some_condition)` or `self.assertFalse(some_condition)`

Running tests

To run the tests, enter the following command in the terminal:

```
python -m unittest file_test
```

!!!!! WARNING: In the call above, DON'T append the extension `.py` to `file_test` !!!!!!

!!!!! WARNING: Still, on the hard-disk the file MUST be named with a `.py` at the end, like `file_test.py`!!!!!!

You should see an output like the following:

[10]:
```
jupman.show_run(MyTest)
```

```
F
======================================================================
FAIL: test_long_list (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/ipykernel_25001/3269760140.py", 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 0.002s

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:

[11]:
```
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
```

[12]:
```
jupman.show_run(MyTest)
```

```
FF
======================================================================
FAIL: test_long_list (__main__.MyTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/ipykernel_25001/1413161586.py", 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)
```

(continues on next page)

(continued from previous page)

```
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/tmp/ipykernel_25001/1413161586.py", 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:

[13]:
```
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
```

[14]:
```
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[292] (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[293]:

- 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])
```

[292] https://docs.python.org/3/library/exceptions.html#TypeError
[293] https://docs.python.org/3/library/exceptions.html#TypeError

Running unittests in Visual Studio Code

You can run and debug tests in Visual Studio Code, which is very handy. First, you need to set it up.

1. Hit `Control-Shift-P` (on Mac: `Command-Shift-P`) and type `Python: Configure Tests`

2. Select unittest:

3. Select `. root directory` (we assume tests are in the folder that you've opened):

4. Select `*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* like 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

6.10.3 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[294] 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[295] (will talk more about it in class)

[294] https://spark.apache.org
[295] 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-lists1-sol.html

6.12 Matrices: Numpy solutions

6.12.1 Moved to https://en.softpython.org/matrices-numpy/matrices-numpy-sol.html

6.13 Data formats solutions

6.13.1 MOVED TO https://en.softpython.org/formats/formats-sol.html

6.14 Graph formats solutions

6.14.1 Moved to https://en.softpython.org/formats/formats4-graph-sol.html

[ ]:

```

```

6.15 Visualization solutions

6.15.1 Moved to https://en.softpython.org/visualization/visualization-sol.html

6.16 Pandas solutions

6.16.1 Moved to https://en.softpython.org/pandas/pandas1-sol.html

[ ]:

```

```

6.17 Binary relations solutions

6.17.1 MOVED TO https://en.softpython.org/binary-relations/binary-relations-sol.html

CHAPTER

SEVEN

PART B

7.1 OOP

7.1.1 OOP

Download exercises zip

Browse files online[296]

What to do

- unzip exercises in a folder, you should get something like this:

```
oop
   complex_number.py
   complex_number_test.py
   complex_number_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.

1. Abstract Data Types (ADT) Theory

1.1. Intro

- Theory from the slides:
  - Andrea Passerini - Programming paradagims, Object-Oriented Python[297]

[296] https://github.com/DavidLeoni/sciprog-ds/tree/master/oop
[297] http://disi.unitn.it/~passerini/teaching/2020-2021/sci-pro/handouts/A09-oop.pdf

- Object Oriented programming on the the book[298] (In particular, Fraction class[299], in this course we won't focus on inheritance)

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)
```

```
{'real': 3.0, 'imaginary': 5.0}
```

[5]:
```
print(c['real'])
```

```
3.0
```

[6]:
```
print(c['imaginary'])
```

[298] https://runestone.academy/runestone/books/published/pythonds/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html
[299] https://runestone.academy/runestone/books/published/pythonds/Introduction/ObjectOrientedProgramminginPythonDefiningClasses.html#a-fraction-class

```
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
    d['imaginary'] = imaginary
    return d
```

[8]:
```
c = complex_number(3.0, 5.0)
```

[9]:
```
print(c)
```

```
{'real': 3.0, 'imaginary': 5.0}
```

To do something with our dictionary, we would then define functions, like for example `complex_str` to show them nicely:

[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))
```

```
{'real': 1.5849625007211563, 'imaginary': 1.4865195378735334}
```

You see we got our dictionary representing a complex number. If we want a nicer display we can call on it the `complex_str` we defined:

[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*.

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` could be any name!

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 0x7f1fd825ded0>
```

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: you need round parenthesis 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: You don't need to pass a dictionary to instantiate a class!

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: self is special!

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[300] : 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}
```

[300] https://en.softpython.org/commandments.html#VIII-COMMANDMENT

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`

> **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 0x7f1fd825ded0>
```

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: instances are different from a class

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

    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

    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 0x7f1fd823dbd0>
```

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 0x7f1fd82654d0>
```

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
```

(continues on next page)

(continued from previous page)

```
            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: follow the specs!

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__[301] 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[302] 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[303]

[301] https://docs.python.org/3/reference/datamodel.html#object.__str__
[302] https://docs.python.org/3/reference/datamodel.html#object.__repr__
[303] https://docs.python.org/3/reference/datamodel.html#object.__str__

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 complex_number_test.ComplexNumberTest
```

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 complex_number_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 by implictly using the operator `is`. Since here the addresses point to different memory regions 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[304] 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)
```

(continues on next page)

[304] https://docs.python.org/3/reference/datamodel.html#object.__eq__

(continued from previous page)

```
    c.real = 1
    c.imaginary = 2
    self.assertEquals(c, ComplexNumber(1,2))
```

To run the test, in the console type:

```
python3 -m unittest complex_number_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[305]
  - Messing with hashing[306]

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:

[305] http://jcalderone.livejournal.com/32837.html
[306] http://www.asmeurer.com/blog/posts/what-happens-when-you-mess-with-hashing-in-python/

```
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:

```
python3 -m unittest complex_number_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, type in the console:

```
python3 -m unittest complex_number_test.AddTest
```

NOTE: The other `RAddTest` (note the `R`) will **not** pass, to deal with it see next paragraph.

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));
```

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 *
```

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
```

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[307]

[ ]:

```

```

7.1.2 OOP Matrix Challenge

Download exercises zip

Browse files online[308]

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.

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.

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
```

(continues on next page)

[307] http://sciprog.davidleoni.it/oop/oop-matrix-chal.html
[308] https://github.com/DavidLeoni/sciprog-ds/tree/master/oop

(continued from previous page)

```
        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)
    """
```

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] ]
```

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)
```

Brackets operator

Override the square bracket operator by reimplementing the method `__getitem__`

IMPORTANT: Read carefully Python documentation[309]!!

```
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:

          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
    """
```

[309] https://docs.python.org/3/reference/datamodel.html#object.__getitem__

[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

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)]

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
```

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
```

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] ]
```

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
        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'
```

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[310]), in this challenge we will adopt the simplest which is the so-called Dictionary of Keys[311] (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.

[310] https://en.wikipedia.org/wiki/Sparse_matrix
[311] https://en.wikipedia.org/wiki/Sparse_matrix#Dictionary_of_keys_(DOK)

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)
```

[44]:
```
'SparseMatrix(30,20, [(24, 11, 5), (26, 17, 3)])'
```

[45]:
```
sma
```

[45]:
```
SparseMatrix(30,20, [(24, 11, 5), (26, 17, 3)])
```

Sparse shape

Implement `shape()`:

[46]:
```
sma.shape()    # note the round brackets
```

[46]:
```
(30, 20)
```

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[312]!!
>
> 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
```

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)]
```

[312] https://docs.python.org/3/reference/datamodel.html#object.__getitem__

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
```

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
```

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] ]
```

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],
                      [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] ]
```

[ ]:
```

```

7.2 Indexing

7.2.1 Download exercises zip

Browse files online[313]

In this worksheet we will try to apply some indexing technique to improve our algorithms performance. Typically, if you want to avoid scanning an entire sequence repeatedly, a good idea might be to first preprocess it by indexing its contents in a dictionary which maps subsequence of elements to their positions.

[313] https://github.com/DavidLeoni/sciprog-ds/tree/master/indexing

THESE EXERCISES ARE ABOUT CUSTOM INDEXING, NOT STRINGS!

- **DO NOT** convert everything to a string, use regexes, nor use string methods like `.index()`, `.replace()`, etc

7.2.2 1. Exercise - chains

You will be doing exercises about chainable lists, using plain old Python lists: we just want to detect duplicates and chain sequences fast.

Start editing the file `chains.py` and read the following.

1.1 has_duplicates

Implement this function:

```
def has_duplicates(external_list):
    """ Returns True if internal lists inside external_list contain duplicates,
        False otherwise. See tests for more examples.

        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`

1.2 chain

Implement this function:

```
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. See tests for more examples.

        INPUT: a list of list of strings , like:

                [
                    ['ab', 'c', 'de'],
                    ['gh', 'i'],
                    ['de', 'f', 'gh']
```

(continues on next page)

(continued from previous page)

```
                ]

        OUTPUT: a list of strings, like   ['ab', 'c', 'de', 'f', 'gh', 'i']
    """
```

It is assumed that

- `external_list` always contains at least one internal list
- internal lists always contain at least two strings
- no string is duplicated among all internal lists

Output sequence is constructed as follows:

- it starts will all the items from the first internal list
- successive items are taken from an internal list which starts with a string equal to the previous taken internal list last string
- sequence must not contain repetitions (so joint strings are taken only once).
- *all* internal lists must be used. If this is not possible (because there are no joint strings), raise `ValueError`

Be careful that:

- **MUST BE WRITTEN WITH STANDARD PYTHON FUNCTIONS**
- **MUST RUN IN** $O(m * n)$, where m is the number of internal lists and n is the length of the longest internal list (just to calculate complexity think about the scenario where all lists have equal size)
- **HINT**: Given the above constraint, whenever you find a string, you cannot start another for loop to check if the string exists elsewhere (that would likely introduce a quadratic m^2 factor) Instead, you need to first keep track of both starting strings and the list they are contained within using some other data structure of your choice, which must allow fast read and writes.
- if possible avoid slicing (which doubles memory usage) and use `itertools.islice` instead

Testing: `python3 -m unittest chains_test.TestChain`

7.2.3 2. Exercise - Bank

A bank stores all its transactions in a log: possible types of transactions are opening an account, transfering some amount of money, withdrawal, etc. For simplicity, we represent each transaction type with a character, and in the log which is a Python list we put only such characters (we don't care about their meaning).

The bank knows some sequences of events are extremely rare, and probably denote suspicious activity. For simplicitly, we assume such sequences only contain three transactions and represent them in tuples called *triseqs*.

Note this was given at an exam, we changed biseqs to triseqs to make it more realistic.

A triseq can occur multiple times, but you can consider the number of repetitions negligible with respect to bank log size.

Start editing `bank.py` and read the following

2.1 bank constructor, log and pos

Implement following methods, adding some appropriate indexing data structure.

BEFORE IMPLEMENTING, DOUBLE CHECK COMPLEXITY REQUIREMENTS!

The idea is that given *an entire triseq* (not just the first character !), we want to find all of its (rare) occurrences *very fast* in $O(1)$

Since performance is important here, passing tests does *not* imply your work is correct!

```
class Bank:

    def __init__(self):
        """ Initializes an empty bank

            Add a data structure of your choice for indexing the log
        """
        self._trans = []
        raise Exception("TODO IMPLEMENT ME!")

    def log(self, transaction):
        """ Appends transaction to the log

            - REMEMBER to also update the index
            - MUST EXECUTE IN O(1)
        """
```

```
def pos(self, triseq):
    """ RETURN a NEW list with all the indeces where the sequence of
        three transactions can be found

        - MUST execute in O(1)
    """
```

Testing: `python3 -m unittest bank_test.LogTest`

Example:

[4]:
```
from bank_sol import *

bank = Bank()
print(bank)
```

```
Bank:
```

[5]:
```
bank.log('a')
```

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

```
Bank: a
```

[7]:
```
print(bank.pos( ('a','b','a') ))  # <-- a triseq is a tuple
```

```
[]
```

[8]:
```
bank.log('b')
```

[9]:
```
print(bank.pos( ('a','b','a') ))
```

```
[]
```

[10]:
```
bank.log('a')
```

[11]:
```
print(bank.pos( ('a','b','a') ))
```

```
[0]
```

[12]:
```
bank.log('a')
bank.log('a')
bank.log('b')
bank.log('a')
bank.log('a')
bank.log('a')
bank.log('b')
bank.log('a')
print(bank)
```

```
Bank: a,b,a,a,a,b,a,a,a,b,a
```

[13]:
```
print(bank.pos(('a','b','a')))
```

```
[0, 4, 8]
```

[14]:
```
print(bank.pos(('a','a','a')))
```

```
[2, 6]
```

[15]:
```
print(bank.pos(('b','a','a')))
```

```
[1, 5]
```

[16]:
```
print(bank.pos(('b','a','c')))
```

```
[]
```

2.2 bank revert

```
def revert(self):
    """ Completely eliminates last transaction and RETURN it

        - if bank is empty, raises IndexError

        - REMEMBER to update any index referring to it
        - *MUST* EXECUTE IN O(1)
    """
```

Testing: `python3 -m unittest RevertTest`

Example:

[18]:
```
bank = Bank()
bank.log('a')
bank.log('b')
bank.log('a')
bank.log('a')
bank.log('a')
bank.log('b')
bank.log('a')
bank.log('a')
print(bank)
```

```
Bank: a,b,a,a,a,b,a,a
```

[19]:
```
print(bank.pos(('b','a','a')))
```

```
[1, 5]
```

[20]:
```
bank.revert()
```

[20]:
```
'a'
```

[21]:
```
print(bank)
```

```
Bank: a,b,a,a,a,b,a
```

[22]:
```
print(bank.pos(('b','a','a')))
```

```
[1]
```

[23]:
```
bank.revert()
```

[23]:
```
'a'
```

[24]:
```
print(bank)
```

```
Bank: a,b,a,a,a,b
```

[25]:
```
print(bank.pos(('b','a','a')))
```

```
[1]
```

[26]:
```
bank.revert()
```

[26]:
```
'b'
```

[27]:
```
print(bank)
```

```
Bank: a,b,a,a,a
```

[28]:
```
print(bank.pos(('b','a','a')))
```

```
[1]
```

[29]:
```
bank.revert()
```

[29]:
```
'a'
```

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

```
Bank: a,b,a,a
```

[31]:
```
print(bank.pos(('b','a','a')))
```

```
[1]
```

2.3 bank max_interval

Typically, there is a triseq which triggers a period of suspicious activity, and there is another triseq which ends it. So given a start and an end triseq, the bank wants a report of all the transactions that happened in between:

```
def max_interval(self, tri_start, tri_end):
    """ RETURN a list with all the transactions occurred between
        the *largest* interval among tri-sequences tri_start and tri_end

        - tri_start and tri_end are EXCLUDED
        - if tri_start / tri_end are not found, or if tri_end is before/includes tri_
↪start,
          raise LookupError


        - DO *NOT* MODIFY the data structure
        - MUST EXECUTE IN O(k) where k is the length of the *largest* interval you␣
↪can return
        - consider number of repetitions a negligible size
    """
```

Testing: `python -m unittest bank_test.MaxIntervalTest`

Example:

[33]:
```
bank = Bank()
bank.log('c')
bank.log('d')
bank.log('c')
bank.log('a') #      a
bank.log('b') # <--- b
bank.log('e') #      e
bank.log('f') #      --- f     |
bank.log('a') #          a     |
bank.log('f') #          f     | k
bank.log('b') #          b     |
bank.log('c') #          c     |
bank.log('b') #      --- b     |
bank.log('a') # <--- a
bank.log('f') #      f
bank.log('b') #      b
bank.log('e')
bank.log('a')
bank.log('b')
bank.log('e')
bank.log('l')
bank.max_interval( ('a','b','e'), ('a','f','b') )
```

[33]:
```
['f', 'a', 'f', 'b', 'c', 'b']
```

7.3 Recursion

7.3.1 Recursion 1 - Simple functional programming

In the past, you looped with `for` and `while` cycles: everything was so easy back then.

Today you're gonna try a new thing called *recursion*.

References:

- Andrea Passerini slides on programming paradigms[314]
- Andrea Passerini slides on recursion theory[315]

Functional programming paradigm[316] heavily relies upon recursion and is becoming quite popular (you will probably need using it during the master), so let's gain some basic understanding of its principles.

In a purely functional programming paradigm:

- functions behave as mathematical ones: they always take some parameter and always return NEW data or pointers to (parts of) original input, without ever changing it
- variables can be assigned only once
- all looping is done via recursion
- side effects are not allowed (will talk more about them later)

Python was not designed to fully support functional programming, so if we adhere to its strictest principles we will end up with inefficient code. Why whould we bother, then? The bright side is that by limiting what we can do we can focus on the *logic* of recursion, without caring about performance issues (we will deal with them later).

So first you will try to develop some functions in *SimpleFP*, a purely functional programming style we just invented for these worksheets.

SimpleFP - how-to

In this section we ask you to devise algorithms in which you:

- never mutate data
- always RETURN NEW data or pointers to (part of) input
- assign variables only once
- always return from each branch of functions
- never use side-effects inside functions: a side-effect is something that is independent from function parameters and/or modifies the environment without storing reusable values in memory. Examples:
  - getting the current time or interactively asking `input` from the user implies a different result at each execution
  - calling `print` only illuminates pixels on the screen without storing resuable values in memory.

Your code is allowed to:

- be inefficient, both in memory and execution time
- hit recursion stack limits imposed by Python: since we will only test with small data in practice this shouldn't happen anyway

[314] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A09-oop.pdf
[315] http://disi.unitn.it/~passerini/teaching/2021-2022/sci-pro/slides/A06-recursion.pdf
[316] https://en.wikipedia.org/wiki/Functional_programming

SimpleFP - variable assignment

Assign variables **only once**.

Avoid:

[2]:
```
x = 3
x = x + 1   # don't
```

Instead, feel free to create new variables at will:

[3]:
```
x = 3
y = x + 1    # ok
```

SimpleFP - list creation

To create new list, use boxing, concatenation or slicing.

SimpleFP - boxing

- wrap one or more objects with `[, ]`

[4]:
```
la = ['a']           # list of one element
lb = ['a','b','c']   # list of n elements
```

SimpleFP - concatenation

To join two lists only use + operator, which creates a NEW list.

Avoid methods which modify lists:

[5]:
```
lst = ['a','b']
lst.append('c')        # don't, it's mutating lst
lst.append('d')
lst.extend(['c','d']) # don't, it's mutating lst
```

Instead, do:

[6]:
```
la = ['a','b']
lb = la + ['c','d']   # '+' creates a NEW list without modifying the operands
                      # here also reassigns to a NEW variable
lc = la + ['e']      # if you need to add only one element just wrap it in square␣
↪brackets
```

SimpleFP - accessing list elements

You can take an element at any position

[7]:
```
lst = [5,2,8,7,1,3]
```

[8]:
```
lst[0]
```

[8]:
```
5
```

[9]:
```
lst[4]
```

[9]:
```
1
```

SimpleFP - slicing

To take sublists in this purely functional model we allow slicing, even if on Python lists it's quite inefficient as it creates entirely NEW lists.

DO NOT use stepping

[10]:
```
lst = ['a','b','c','d','e','f','g','h','i']
lst[1:]
```

[10]:
```
['b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
```

[11]:
```
lst[len(lst)//2:]
```

[11]:
```
['e', 'f', 'g', 'h', 'i']
```

[12]:
```
lst[:len(lst)//2]
```

[12]:
```
['a', 'b', 'c', 'd']
```

[13]:
```
lst[0:5:2]  # don't use stepping
```

[13]:
```
['a', 'c', 'e']
```

QUESTION: If we used numpy arrays[317], would we waste memory when slicing?

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

ANSWER: no, numpy slicing (and only numpy slicing) provides a *view* on the original data without copying, thus taking slices is a fast operation. Since in purely functional model we force ourselves to never modify data, we could consider access to original data as safe.

[317] https://en.softpython.org/matrices-numpy/matrices-numpy1-sol.html#Slices

SimpleFP - minimal instruction set

To force you using recursion, we require these restrictions:

ONLY USE:

- list concatenation with +
- getting length with `len(lst)`
- `in` operator on dictionaries or sets

DO NOT use `in` operator in sequences like strings, lists, tuples

DO NOT use `*` replication operator on lists

DO NOT use predefined functions (`sum`, `max`, `min`, `range`, ….)

DO NOT use methods!

[14]:
```
5 in [3,5,2,6]        # don't
[7,5,6,9] * 3         # don't
sum([5,2,7,4])        # don't
[3,5,2,5,4].count(5)  # don't

[3,4] + [9,2,3]   # ok
len([3,5,1,6])    # ok
'a' in {'a':4,    # ok
        'b':9}
```

SimpleFP - functions and conditionals

In purely *functional* programming we want of course… functions: we require they always get some input and always RETURN NEW data taken from (parts of) the input

Avoid:

[15]:
```
x = 4

def inc():             # bad, takes  no explicit input ....
    return x + 1

def app(lst):
    lst.append('a')    # bad, modifies input

def srt(lst):
    lst.sort()         # bad, modifies input
```

Instead, do:

[16]:
```
def inc(x):    # ok, uses only input
    return x + 3

def app(lst):
    return lst + ['a']    # ok, creates a NEW list

def srt(lst):
    return sorted(lst)    # ok, creates a NEW list
                          # (note sorted must be explicitly allowed by text)
```

SimpleFP - recursion

In a purely functional model we don't use loops at all! Instead, we prefer recursive calls.

Always remember:

- at least one base case
- always return in each branch
- at least one recursive call on smaller parts of the input
- always return the proper type (you might need boxing)

> **DO NOT** use `for`, `while` nor list comprehensions

Simple FP - Example - `scount`

Suppose given a number `n`, you want to build a list with first numbers from `1` to `n` **without** using `range`

[17]:
```
def scount(n):
    if n <= 0:
        return []
    else:
        return scount(n - 1) + [n]

scount(3)
jupman.pytut()
```

[17]:
```
<IPython.core.display.HTML object>
```

Simple FP - debugging

To debug you can and should of course use Python tutor[318].

Another trick could be to pass an additional optional parameter called `level` and do some indented `print` like so:

> **IMPORTANT:** `print` **is ONLY for debugging!**
>
> `print` by definition illuminates the pixels on your screen and doesn't produce any reusable value in memory. So your functions are still required to always RETURN proper values!

[19]:
```
def scount(n, level=0):
    print(' ' * level, 'n=', n)
    if n <= 0:
        return []
    else:
        return scount(n - 1, level + 1) + [n]
scount(3)
```

[318] https://pythontutor.com/

```
 n= 3
  n= 2
   n= 1
    n= 0
```

[19]:
```
[1, 2, 3]
```

Decomposing computation: Furthermore, you can try separate the steps in smaller parts and print intermediate results. Here we added the extra variable `res` and printed it after the recursive call:

[21]:
```
def scount(n, level=0):
    print(' ' * level, 'n=', n)
    if n <= 0:
        return []
    else:
        res = scount(n - 1, level + 1)
        print(' ' * level, 'res=', res)
        return res + [n]
scount(5)
```

```
 n= 5
  n= 4
   n= 3
    n= 2
     n= 1
      n= 0
     res= []
    res= [1]
   res= [1, 2]
  res= [1, 2, 3]
 res= [1, 2, 3, 4]
```

[21]:
```
[1, 2, 3, 4, 5]
```

SimpleFP: a recursion scheme

Suppose we now want to get a list as input and return another one.

For this first section, we will follow a simple scheme:

1. check sequence length
2. if base case, immediately return something
3. otherwise, do recursion on the rest of the sequence
4. combine recursion result in some way with first element
5. return the combination result

Example - `sdouble`

Let's see how to write a function which takes a `lst` of numbers and RETURN a NEW list with the numbers doubled. You might write stuff like this:

[23]:
```
def sdouble(lst):
    if len(lst) == 0:
        return []               # base case
    else:
        return [lst[0]*2] + sdouble(lst[1:])     # returning operation containing
                                                 # a recursive call on a smaller input

result = sdouble([7,9,5])
jupman.pytut()
```

[23]:
```
<IPython.core.display.HTML object>
```

Exercise - debug double

Try placing extra debug prints and separating the steps in the code above

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

[25]:
```
def sdouble(lst, level=0):
    print(' '*level, 'lst=', lst)
    if len(lst) == 0:
        return []               # base case
    else:
        res = sdouble(lst[1:], level+1)
        print(' '*level, 'res=', res)
        return [lst[0]*2] + res      # returning operation containing
                                     # a recursive call on a smaller input

result = sdouble([4,9,1,5,7,3])
result
```

```
 lst= [4, 9, 1, 5, 7, 3]
  lst= [9, 1, 5, 7, 3]
   lst= [1, 5, 7, 3]
    lst= [5, 7, 3]
     lst= [7, 3]
      lst= [3]
       lst= []
      res= []
     res= [6]
    res= [14, 6]
   res= [10, 14, 6]
  res= [2, 10, 14, 6]
 res= [18, 2, 10, 14, 6]
```

[25]:
```
[8, 18, 2, 10, 14, 6]
```

[25]:
```
result = sdouble([4,9,1,5,7,3])
result
```

```
 lst= [4, 9, 1, 5, 7, 3]
  lst= [9, 1, 5, 7, 3]
   lst= [1, 5, 7, 3]
    lst= [5, 7, 3]
     lst= [7, 3]
      lst= [3]
       lst= []
      res= []
     res= [6]
    res= [14, 6]
   res= [10, 14, 6]
  res= [2, 10, 14, 6]
 res= [18, 2, 10, 14, 6]
```

[25]:
```
[8, 18, 2, 10, 14, 6]
```

Exercise - sfilter_even

Takes a `lst` of numbers, and RETURN a NEW list with only the even numbers

<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 sfilter_even(lst):

    if len(lst) == 0:
        return []
    else:
        if lst[0] % 2 == 0:
            return [lst[0]] + sfilter_even(lst[1:])
        else:
            return sfilter_even(lst[1:])


result = sfilter_even([3,7,10,5,4,8,6,11])
print(result) # [10, 4, 8, 6]

assert sfilter_even([3]) == []
assert sfilter_even([4]) == [4]
assert sfilter_even([3,6]) == [6]
assert sfilter_even([8,3]) == [8]
assert sfilter_even([8,3,2]) == [8,2]
assert sfilter_even([7,7,8,3,2]) == [8,2]
assert sfilter_even([3,7,10,5,4,8,6,11]) == [10,4,8,6]
```

```
[10, 4, 8, 6]
```

[27]:

(continues on next page)

(continued from previous page)

```
def sfilter_even(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = sfilter_even([3,7,10,5,4,8,6,11])
print(result) # [10, 4, 8, 6]

assert sfilter_even([3]) == []
assert sfilter_even([4]) == [4]
assert sfilter_even([3,6]) == [6]
assert sfilter_even([8,3]) == [8]
assert sfilter_even([8,3,2]) == [8,2]
assert sfilter_even([7,7,8,3,2]) == [8,2]
assert sfilter_even([3,7,10,5,4,8,6,11]) == [10,4,8,6]
```

Exercise - smerry

Given a `lst` with an even number of elements, RETURN a NEW list holding two element lists with consecutive content taken from the original list.

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

[28]:
```
def smerry(lst):

    if len(lst) == 0:
        return []
    else:
        return [[lst[0], lst[1]]] + smerry(lst[2:])


result = smerry(['a','b','c','c','a','d','b','a'])
print(result)  # [['a','b'],['c','c'],['a','d'],['b','a']]

assert smerry([]) == []
assert smerry(['d','b']) == [['d','b']]
assert smerry(['a','a','b','b']) == [['a','a'],['b','b']]
assert smerry(['a','b','c','d']) == [['a','b'],['c','d']]
assert smerry(['a','b','c','c','a','d','b','a']) == [['a','b'],['c','c'],['a','d'],['b
↪','a']]
```

```
[['a', 'b'], ['c', 'c'], ['a', 'd'], ['b', 'a']]
```

[28]:
```
def smerry(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = smerry(['a','b','c','c','a','d','b','a'])
print(result)  # [['a','b'],['c','c'],['a','d'],['b','a']]

assert smerry([]) == []
assert smerry(['d','b']) == [['d','b']]
assert smerry(['a','a','b','b']) == [['a','a'],['b','b']]
```

(continues on next page)

(continued from previous page)

```
assert smerry(['a','b','c','d']) == [['a','b'],['c','d']]
assert smerry(['a','b','c','c','a','d','b','a']) == [['a','b'],['c','c'],['a','d'],['b
↪','a']]
```

Exercise - ssum

Takes a `lst` of numbers and RETURN the sum of all numbers.

DO NOT use `sum` function ;-)

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

[29]:
```
def ssum(lst):

    if len(lst) == 0:
        return 0
    else:
        return lst[0] + ssum(lst[1:])


result = ssum([4,2,6,3])
print(result)  # 15

assert ssum([4,2,6,3]) == 15
assert ssum([]) == 0
assert ssum([3]) == 3
assert ssum([5,1]) == 6
```

```
15
```

[29]:
```
def ssum(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = ssum([4,2,6,3])
print(result)  # 15

assert ssum([4,2,6,3]) == 15
assert ssum([]) == 0
assert ssum([3]) == 3
assert ssum([5,1]) == 6
```

Exercise - smin

Given a `lst` of numbers, RETURN the minimum

- **DON'T** use `min` function..
- **assume the list has at least one element**

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

[30]:
```
import math
def smin(lst):

    if  len(lst) == 1:
        return lst[0]
    else:
        res = smin(lst[1:])
        if lst[0] < res:
            return lst[0]
        else:
            return res


result = smin([3,6,5,8,-3,8,-2,-10,4])
print(result)  # -10

assert smin([4]) == 4
assert smin([4,1]) == 1
assert smin([0,9]) == 0
assert smin([6,8,5]) == 5
assert smin([3,6,5,8,3,8,2,10,4]) == 2
```

```
-10
```

[30]:
```
import math
def smin(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = smin([3,6,5,8,-3,8,-2,-10,4])
print(result)  # -10

assert smin([4]) == 4
assert smin([4,1]) == 1
assert smin([0,9]) == 0
assert smin([6,8,5]) == 5
assert smin([3,6,5,8,3,8,2,10,4]) == 2
```

Exercise - ssearch

Given an `el` to search and a `lst`, RETURN `True` if such element is present or `False` otherwise

- **DO NOT** use `in` operator nor search methods like `.find`, `.index`, ...
- consider shortening your code by using Boolean evaluation order[319]

<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 ssearch(el, lst):

    if len(lst) == 0:
        return False
    else:
        return lst[0] == el or ssearch(el, lst[1:])  # early termination using ␣
↪boolean evaluation order!


print(ssearch(7, [6,4,7,8]))   # True
print(ssearch(5, [6,4,7,8]))   # False

assert ssearch(7, [6,4,7,8]) == True
assert ssearch(7, [8]) == False
assert ssearch(8, [8]) == True
assert ssearch(5, [8,5]) == True
assert ssearch(2, [8,5]) == False
assert ssearch(7, [7,8,5]) == True
assert ssearch(2, [8,2,5]) == True
assert ssearch(9, [7,8,9]) == True
assert ssearch(2, [8,9,5]) == False
```

```
True
False
```

[31]:
```
def ssearch(el, lst):
    raise Exception('TODO IMPLEMENT ME !')

print(ssearch(7, [6,4,7,8]))   # True
print(ssearch(5, [6,4,7,8]))   # False

assert ssearch(7, [6,4,7,8]) == True
assert ssearch(7, [8]) == False
assert ssearch(8, [8]) == True
assert ssearch(5, [8,5]) == True
assert ssearch(2, [8,5]) == False
assert ssearch(7, [7,8,5]) == True
assert ssearch(2, [8,2,5]) == True
assert ssearch(9, [7,8,9]) == True
assert ssearch(2, [8,9,5]) == False
```

[319] https://en.softpython.org/basics/basics2-bools-sol.html#Evaluation-order

Exercise - sbin_search

If the list is already sorted we can write a faster search. Start looking in the middle of the list: if that cell is the searched element, you're done. Otherwise if searched element is lesser do recursion on left sublist, otherwise on right sublist.

- `lst` is a list of n **already sorted** elements

DO NOT use `in` operator, `.find` nor `.index` methods

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

[32]:

```
def sbin_search(el, lst):


    k = len(lst) // 2
    if len(lst) == 0:
        return False
    elif el == lst[k]:
        return True
    elif el < lst[k]:
        return sbin_search(el, lst[:k])
    else: # >
        return sbin_search(el, lst[k+1:])


print(sbin_search(33, [8,9,15,17,20,24,28,29,33,36,39]))  # True
print(sbin_search(10, [8,9,15,17,20,24,28,29,33,36,39]))  # False

assert sbin_search(5, [5]) == True
assert sbin_search(4, []) == False
assert sbin_search(3, [7]) == False
assert sbin_search(3, [3,7]) == True
assert sbin_search(3, [3,6]) == True
assert sbin_search(4, [2,9]) == False
assert sbin_search(5, [2,9]) == False
assert sbin_search(5, [5,7,9]) == True
assert sbin_search(7, [5,7,9]) == True
assert sbin_search(9, [5,7,9]) == True
assert sbin_search(4, [5,7,9]) == False
assert sbin_search(6, [5,7,9]) == False
assert sbin_search(8, [5,7,9]) == False
assert sbin_search(10, [5,7,9]) == False
assert sbin_search(33, [8,9,15,17,20,24,28,29,33,36,39]) == True
assert sbin_search(10, [8,9,15,17,20,24,28,29,33,36,39]) == False
```

```
True
False
```

[32]:

```
def sbin_search(el, lst):
    raise Exception('TODO IMPLEMENT ME !')

print(sbin_search(33, [8,9,15,17,20,24,28,29,33,36,39]))  # True
```

(continues on next page)

(continued from previous page)

```
print(sbin_search(10, [8,9,15,17,20,24,28,29,33,36,39]))  # False

assert sbin_search(5, [5]) == True
assert sbin_search(4, []) == False
assert sbin_search(3, [7]) == False
assert sbin_search(3, [3,7]) == True
assert sbin_search(3, [3,6]) == True
assert sbin_search(4, [2,9]) == False
assert sbin_search(5, [2,9]) == False
assert sbin_search(5, [5,7,9]) == True
assert sbin_search(7, [5,7,9]) == True
assert sbin_search(9, [5,7,9]) == True
assert sbin_search(4, [5,7,9]) == False
assert sbin_search(6, [5,7,9]) == False
assert sbin_search(8, [5,7,9]) == False
assert sbin_search(10, [5,7,9]) == False
assert sbin_search(33, [8,9,15,17,20,24,28,29,33,36,39]) == True
assert sbin_search(10, [8,9,15,17,20,24,28,29,33,36,39]) == False
```

Exercise - szip

Given two lists, RETURN a NEW list having couples as tuples with elements taken pairwise from `lst1` and `lst2`

- **DO NOT** use `zip` function

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

[33]:
```
def szip(lst1, lst2):

    if len(lst1) == 0:
        return []
    else:
        return [(lst1[0], lst2[0])] +  szip(lst1[1:], lst2[1:])


result = szip([3,5,1,6], ['a','d','e','w'])
print(result)  # [(3,'a'), (5,'d'), (1,'e'), (6,'w')]

assert szip([], []) == []
assert szip(['a'], [0]) == [('a',0)]
assert szip(['z','a'], [2,4]) == [('z',2), ('a',4)]
assert szip([3,5,1,6], ['a','d','e','w']) == [(3,'a'), (5,'d'), (1,'e'), (6,'w')]
```

```
[(3, 'a'), (5, 'd'), (1, 'e'), (6, 'w')]
```

[33]:
```
def szip(lst1, lst2):
    raise Exception('TODO IMPLEMENT ME !')

result = szip([3,5,1,6], ['a','d','e','w'])
print(result)  # [(3,'a'), (5,'d'), (1,'e'), (6,'w')]
```

(continues on next page)

(continued from previous page)

```
assert szip([], []) == []
assert szip(['a'], [0]) == [('a',0)]
assert szip(['z','a'], [2,4]) == [('z',2), ('a',4)]
assert szip([3,5,1,6], ['a','d','e','w']) == [(3,'a'), (5,'d'), (1,'e'), (6,'w')]
```

Exercise - sunnest

Given a `lst` of lists, RETURN a NEW list without sublists (also called *shallow unnest*)

- assume sublists cannot have further sublists inside

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

[34]:
```
def sunnest(lst):

    if len(lst) == 0:
        return []
    else:
        return lst[0] + sunnest(lst[1:])


result = sunnest([[3,2,5], [2,4], [0,9,5,1]])
print(result)  # [3,2,5,2,4,0,9,5,1]

assert sunnest([]) == []
assert sunnest([[]]) == []
assert sunnest([[],[]]) == []
assert sunnest([[],[4]]) == [4]
assert sunnest([[3],[]]) == [3]
assert sunnest([[3,2]]) == [3,2]
assert sunnest([[3,2]]) == [3,2]
assert sunnest([[3,2,5], [2,4], [0,9,5,1]]) == [3,2,5,2,4,0,9,5,1]
```

```
[3, 2, 5, 2, 4, 0, 9, 5, 1]
```

[34]:
```
def sunnest(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = sunnest([[3,2,5], [2,4], [0,9,5,1]])
print(result)  # [3,2,5,2,4,0,9,5,1]

assert sunnest([]) == []
assert sunnest([[]]) == []
assert sunnest([[],[]]) == []
assert sunnest([[],[4]]) == [4]
assert sunnest([[3],[]]) == [3]
assert sunnest([[3,2]]) == [3,2]
assert sunnest([[3,2]]) == [3,2]
assert sunnest([[3,2,5], [2,4], [0,9,5,1]]) == [3,2,5,2,4,0,9,5,1]
```

Exercise - sfib

Given a number `n`, RETURN a NEW list with all the numbers of Fibonacci sequence:

n	0	1	2	3	4	5	6	7	8	9	10
x_n	0	1	1	2	3	5	8	13	21	34	…

Base cases:

- $x_0 = 0$
- $x_1 = 1$

The next number is found by adding up the two numbers before it:

- x_2 is 0+1
- x_3 is 1+1
- x_4 is 2+1
- x_5 is 3+2
- and so on…

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

[35]:
```
def sfib(n):

    if n == 0:
        return [0]
    elif n == 1:
        return [0,1]
    else:
        prev = sfib(n-1)
        return prev + [prev[n-1] + prev[n-2]]


result = sfib(7)
print(result) # [0,1,1,2,3,5,8,13]

assert sfib(0) == [0]
assert sfib(1) == [0,1]
assert sfib(2) == [0,1,1]
assert sfib(3) == [0,1,1,2]
assert sfib(4) == [0,1,1,2,3]
assert sfib(5) == [0,1,1,2,3,5]
assert sfib(6) == [0,1,1,2,3,5,8]
assert sfib(7) == [0,1,1,2,3,5,8,13]
```

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

[35]:
```
def sfib(n):
    raise Exception('TODO IMPLEMENT ME !')

result = sfib(7)
print(result) # [0,1,1,2,3,5,8,13]
```

(continues on next page)

(continued from previous page)

```
assert sfib(0) == [0]
assert sfib(1) == [0,1]
assert sfib(2) == [0,1,1]
assert sfib(3) == [0,1,1,2]
assert sfib(4) == [0,1,1,2,3]
assert sfib(5) == [0,1,1,2,3,5]
assert sfib(6) == [0,1,1,2,3,5,8]
assert sfib(7) == [0,1,1,2,3,5,8,13]
```

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

Exercise - sall

Given a `lst` of booleans, RETURN `True` if all elements are `True`, otherwise RETURN `False`

- **DO NOT** use `all` function
- assume `sall` of empty list is `True`
- **HINT**: consider exploiting Boolean evaluation order[320] to shorten your code

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

[36]:
```
def sall(lst):

    if len(lst) == 0:
        return True
    else:
        return lst[0] and sall(lst[1:])  # note if lst[0] is False evaluation stops␣
↪immediately


assert sall([]) == True
assert sall([True]) == True
assert sall([False]) == False
assert sall([False, False]) == False
assert sall([False, True]) == False
assert sall([True, False]) == False
assert sall([True, True]) == True
assert sall([False, False, False]) == False
assert sall([False, False, True]) == False
assert sall([False, True, False]) == False
assert sall([False, True, True]) == False
assert sall([True, False, False]) == False
assert sall([True, False, True]) == False
assert sall([True, True, False]) == False
assert sall([True, True, True]) == True
```

[36]:
```
def sall(lst):
```

(continues on next page)

[320] https://en.softpython.org/basics/basics2-bools-sol.html#Evaluation-order

(continued from previous page)

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

assert sall([]) == True
assert sall([True]) == True
assert sall([False]) == False
assert sall([False, False]) == False
assert sall([False, True]) == False
assert sall([True, False]) == False
assert sall([True, True]) == True
assert sall([False, False, False]) == False
assert sall([False, False, True]) == False
assert sall([False, True, False]) == False
assert sall([False, True, True]) == False
assert sall([True, False, False]) == False
assert sall([True, False, True]) == False
assert sall([True, True, False]) == False
assert sall([True, True, True]) == True
```

Exercise - srev

Given a list, RETURN a NEW list reversed.

- **DO NOT** use `reversed` function
- **DO NOT** use slice stepping

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

[37]:
```
def srev(lst):

    if len(lst) == 0:
        return []
    else:
        return srev(lst[1:]) + [lst[0]]


result = srev([4,9,7,5,6])
print(result)

assert srev([]) == []
assert srev([3]) == [3]
assert srev([5,3]) == [3,5]
assert srev([6,6,2]) == [2,6,6]
assert srev([4,9,7,5,6]) == [6,5,7,9,4]
```

```
[6, 5, 7, 9, 4]
```

[37]:
```
def srev(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = srev([4,9,7,5,6])
```

(continues on next page)

(continued from previous page)

```
print(result)

assert srev([]) == []
assert srev([3]) == [3]
assert srev([5,3]) == [3,5]
assert srev([6,6,2]) == [2,6,6]
assert srev([4,9,7,5,6]) == [6,5,7,9,4]
```

Exercise - spalindrome

A string is palindrome if it reads the same backward as forward.

Given a string `s`, return `True` if the string is palindrome, otherwise return `False`.

- we assume the empty string is palindrome
- here we use strings but same rules for lists apply
- **DO NOT** use string methods (so no `.index()`, `.find()`, ...)

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

[38]:
```
def spalindrome(s):

    if len(s) == 0:
        return True
    elif len(s) == 1:
        return True
    else:
        if s[0] == s[-1]:
            return spalindrome(s[1:-1])
        else:
            return False


assert spalindrome('') == True
assert spalindrome('a') == True
assert spalindrome('ab') == False
assert spalindrome('cc') == True
assert spalindrome('ddv') == False
assert spalindrome('vddv') == True
assert spalindrome('axyb') == False
assert spalindrome('kayak') == True
assert spalindrome('kayag') == False
assert spalindrome('rotator') == True
assert spalindrome('rotating') == False
assert spalindrome('noon') == True
assert spalindrome('soon') == False
assert spalindrome('abcdba') == False
```

[38]:
```
def spalindrome(s):
    raise Exception('TODO IMPLEMENT ME !')
```

(continues on next page)

(continued from previous page)

```
assert spalindrome('') == True
assert spalindrome('a') == True
assert spalindrome('ab') == False
assert spalindrome('cc') == True
assert spalindrome('ddv') == False
assert spalindrome('vddv') == True
assert spalindrome('axyb') == False
assert spalindrome('kayak') == True
assert spalindrome('kayag') == False
assert spalindrome('rotator') == True
assert spalindrome('rotating') == False
assert spalindrome('noon') == True
assert spalindrome('soon') == False
assert spalindrome('abcdba') == False
```

Exercise - snest

Given a `lst`, RETURN a NEW list of deeply right nested lists as in the example

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

[39]:
```
def snest(lst):

    if len(lst) == 0:
        return []
    elif len(lst) == 1:
        return [lst[0]]
    else:
        return [lst[0], snest(lst[1:])]


result = snest(['a','b','c','d','e','g','h'])
print(result)  # ['a',['b',['c',['d',['e',['g',['h']]]]]]]

assert snest([]) == []
assert snest(['a']) == ['a']
assert snest(['a','b']) == ['a',['b']]
assert snest(['a','b','c']) == ['a',['b',['c']]]
assert snest(['x','x','y']) == ['x',['x',['y']]]
assert snest(['a','b','c','d','e','g','h']) == ['a',['b',['c',['d',['e',['g',['h
↪']]]]]]]
```

```
['a', ['b', ['c', ['d', ['e', ['g', ['h']]]]]]]
```

[39]:
```
def snest(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = snest(['a','b','c','d','e','g','h'])
print(result)  # ['a',['b',['c',['d',['e',['g',['h']]]]]]]
```

(continues on next page)

(continued from previous page)

```
assert snest([]) == []
assert snest(['a']) == ['a']
assert snest(['a','b']) == ['a',['b']]
assert snest(['a','b','c']) == ['a',['b',['c']]]
assert snest(['x','x','y']) == ['x',['x',['y']]]
assert snest(['a','b','c','d','e','g','h']) == ['a',['b',['c',['d',['e',['g',['h
↪']]]]]]]
```

Exercise - spalace

Given a `lst` of characters, RETURN a NEW list of deeply nested lists as in the example.

- Pay attention to base cases, there are several

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

[40]:

```
def spalace(lst):

    if len(lst) == 0:
        return []
    elif len(lst) == 1:
        return [lst[0]]
    elif len(lst) == 2:
        return [lst[0],lst[1]]
    else:
        return [lst[0], spalace(lst[1:-1]),lst[-1]]


result = spalace(['a','b','c','d','e','f','g','h','i'])
print(result)   # ['a', ['b', ['c', ['d', ['e'], 'f'], 'g'], 'h'], 'i']

assert spalace([])  == []
assert spalace(['a'])  == ['a']
assert spalace(['a','b'])  == ['a','b']
assert spalace(['b','a'])  == ['b','a']
assert spalace(['c','b','a'])  == ['c',['b'],'a']
assert spalace(['z','w','z'])  == ['z',['w'],'z']
assert spalace(['p','x','y','z','q'])  == ['p',['x',['y'],'z'],'q']
assert spalace(['a','b','c','d','e','f','g','h','i'])  == ['a', ['b', ['c', ['d', ['e
↪'], 'f'], 'g'], 'h'], 'i']
```

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

[40]:

```
def spalace(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = spalace(['a','b','c','d','e','f','g','h','i'])
print(result)   # ['a', ['b', ['c', ['d', ['e'], 'f'], 'g'], 'h'], 'i']
```

(continues on next page)

(continued from previous page)

```
assert spalace([])  == []
assert spalace(['a'])  == ['a']
assert spalace(['a','b'])  == ['a','b']
assert spalace(['b','a'])  == ['b','a']
assert spalace(['c','b','a'])  == ['c',['b'],'a']
assert spalace(['z','w','z'])  == ['z',['w'],'z']
assert spalace(['p','x','y','z','q'])  == ['p',['x',['y'],'z'],'q']
assert spalace(['a','b','c','d','e','f','g','h','i'])  == ['a', ['b', ['c', ['d', ['e
↪'], 'f'], 'g'], 'h'], 'i']
```

Exercise - srep

Given a list `la`, and a list `lb` of integer numbers ≥ 0, RETURN a NEW list where all elements of `la` are repeated the amount of times indicated in the corresponding cell of `lb`

- assume `la` and `lb` have the same size
- **DO NOT** use `*` replication operator

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

[41]:

```
def srep(la, lb):

    if len(la) == 0:
        return []
    elif lb[0] > 0:
        return [la[0]] + srep(la, [lb[0]-1] + lb[1:])
    else:  # == 0
        return srep(la[1:], lb[1:])


result = srep(['a','b','c','d'], [2,5,0,3])
print(result) # ['a','a','b','b','b','b','b','d','d','d']

assert srep([], []) == []
assert srep(['a'], [1]) == ['a']
assert srep(['a'], [0]) == []
assert srep(['b'], [2]) == ['b','b']
assert srep(['a','b'], [1,1]) == ['a','b']
assert srep(['a','b'], [1,2]) == ['a','b','b']
assert srep(['a','b'], [2,1]) == ['a','a','b']
assert srep(['a','b'], [2,0]) == ['a','a']
assert srep(['a','b'], [0,2]) == ['b','b']
assert srep(['a','b','c'], [1,1,1]) == ['a','b','c']
assert srep(['a','b','c'], [0,1,1]) == ['b','c']
assert srep(['a','b','c'], [0,1,0]) == ['b']
assert srep(['a','b','c'], [3,1,2]) == ['a','a','a','b','c','c']
assert srep(['a','b','c','d'], [2,5,0,3]) == ['a','a','b','b','b','b','b','d','d','d']
```

```
['a', 'a', 'b', 'b', 'b', 'b', 'b', 'd', 'd', 'd']
```

[41]:
```
def srep(la, lb):
    raise Exception('TODO IMPLEMENT ME !')

result = srep(['a','b','c','d'], [2,5,0,3])
print(result) # ['a','a','b','b','b','b','b','d','d','d']

assert srep([], []) == []
assert srep(['a'], [1]) == ['a']
assert srep(['a'], [0]) == []
assert srep(['b'], [2]) == ['b','b']
assert srep(['a','b'], [1,1]) == ['a','b']
assert srep(['a','b'], [1,2]) == ['a','b','b']
assert srep(['a','b'], [2,1]) == ['a','a','b']
assert srep(['a','b'], [2,0]) == ['a','a']
assert srep(['a','b'], [0,2]) == ['b','b']
assert srep(['a','b','c'], [1,1,1]) == ['a','b','c']
assert srep(['a','b','c'], [0,1,1]) == ['b','c']
assert srep(['a','b','c'], [0,1,0]) == ['b']
assert srep(['a','b','c'], [3,1,2]) == ['a','a','a','b','c','c']
assert srep(['a','b','c','d'], [2,5,0,3]) == ['a','a','b','b','b','b','b','d','d','d']
```

Exercise - ssortin

Given two **already sorted** lists `la` and `lb` of arbitrary sizes, return `True` if all numbers of `la` are present in `lb`

- assume empty list is always contained in any list
- **DO NOT** use search methods (`find()`, `index()`, ...)
- **HINT 1**: compare list heads one at a time, try thinking when you can return early
- **HINT 2**: you will need to progress lists at different times

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

[42]:
```
def ssortin(la, lb):

    if len(la) == 0:
        return True
    elif len(lb) == 0:
        return False
    elif la[0] < lb[0]:
        return False
    elif la[0] > lb[0]:
        return ssortin(la, lb[1:])
    else:   # l[0] == lb[0]
        return ssortin(la[1:], lb[1:])


assert ssortin([], []) == True
assert ssortin([], [4,5,8,1]) == True
assert ssortin([5], [5]) == True
assert ssortin([5], [6]) == False
assert ssortin([3], [3,8]) == True
```

(continues on next page)

(continued from previous page)

```
assert ssortin([8], [3,8]) == True
assert ssortin([2], [3,8]) == False
assert ssortin([7,9], [7,9]) == True
assert ssortin([7,9], [3,7,9]) == True
assert ssortin([7,9], [7,8,9]) == True
assert ssortin([7,9], [7,9,10]) == True
assert ssortin([5,8], [2,8]) == False
assert ssortin([5,8], [5,6]) == False
assert ssortin([5,8], [2,4,8]) == False
assert ssortin([5,8], [4,5,7]) == False
assert ssortin([3,5,6], [3,5,6]) == True
assert ssortin([3,5,6], [3,4,5,6]) == True
assert ssortin([3,5,6], [3,5,6,7]) == True
assert ssortin([4,6,11], [1,3,4,5,6,7,8,10,11,12]) == True
```

[42]:

```
def ssortin(la, lb):
    raise Exception('TODO IMPLEMENT ME !')

assert ssortin([], []) == True
assert ssortin([], [4,5,8,1]) == True
assert ssortin([5], [5]) == True
assert ssortin([5], [6]) == False
assert ssortin([3], [3,8]) == True
assert ssortin([8], [3,8]) == True
assert ssortin([2], [3,8]) == False
assert ssortin([7,9], [7,9]) == True
assert ssortin([7,9], [3,7,9]) == True
assert ssortin([7,9], [7,8,9]) == True
assert ssortin([7,9], [7,9,10]) == True
assert ssortin([5,8], [2,8]) == False
assert ssortin([5,8], [5,6]) == False
assert ssortin([5,8], [2,4,8]) == False
assert ssortin([5,8], [4,5,7]) == False
assert ssortin([3,5,6], [3,5,6]) == True
assert ssortin([3,5,6], [3,4,5,6]) == True
assert ssortin([3,5,6], [3,5,6,7]) == True
assert ssortin([4,6,11], [1,3,4,5,6,7,8,10,11,12]) == True
```

Continue

Go on with accumulators worksheet[321]

[321] https://sciprog.davidleoni.it/recursion/recursion2-modacc.html

7.3.2 Recursion 2 - Accumulators and indexes

So far in the SimpleFP model we have been creating a lot of new lists by concatenating with + operator and taking slices: this wasted a lot of memory and computation time.

Let's try to improve performance by creating a new model we named **ModAcc** with these requirements:

ALLOWED:

- **add helper functions** to easily pass extra variables
- **add special accumulator parameters** to the helper function: feel free to MODIFY them (and only them!) by using `.append`, `.extend`, etc
- **add indexes parameters** to the helper function to keep track where to look in the original list
- getting length with `len(lst)`
- `in` operator on dictionaries or sets

DO NOT use slices

DO NOT MODIFY input parameters!

DO NOT use `in` operator on sequences like strings, lists, tuples

DO NOT use `*` replication operator on lists

DO NOT use predefined functions (`sum`, `max`, `min`, `range`, ...)

DO NOT use methods!

DO NOT INITIALIZE AN ACCUMULATOR IN THE PARAMETER DECLARATION

- **NEVER** write stuff like:

```
def my_helper(lst, acc=[]):
```

```
def my_helper(lst, acc={}):
```

If you initialize to a mutable data structure such as lists or dictionaries, Python will create exactly **one** memory region for **all** function invocations, which is quite counterintuitive and bug prone!

Example - `adouble`

Let's see an example by improving our `double` function to take a `lst` of numbers and RETURN a NEW list with the numbers doubled:

[2]:
```
def adouble_helper(lst, i, acc):    # we create a helper function
                                    # with index where to start looking
                                    # and accumulator variable
                                    # note acc is *not* initialized in the parameters
    if i == len(lst):    # base case compares index against end of list
        return acc       #    notice it returns the accumulator
    else:
        acc.append(lst[i]*2)     #  we first MODIFY the accumulator
        return adouble_helper(lst, i+1, acc)  # then do recursion by passing the␣
↪modified accumulator
                                                # notice passed list is always the␣
↪same,
```

(continues on next page)

(continued from previous page)

```
                                                    # what changes is the starting index

def adouble(lst):
    return adouble_helper(lst, 0, [])

result = adouble([7,9,5])
result
jupman.pytut()
```

[2]:
```
<IPython.core.display.HTML object>
```

Exercise - debug adouble

Add a `level` parameter and some debugging indented print to the function

<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 adouble_helper(lst, i, acc, level=0):
    print(' '*level, 'i=',i,' acc=', acc)
    if i == len(lst):
        return acc
    else:
        acc.append(lst[i]*2)
        res = adouble_helper(lst, i+1, acc, level=level+1)
        print(' '*level, 'res=', res)
        return res

def adouble(lst):
    return adouble_helper(lst, 0, [])


result = adouble([4,9,1,5,7,3])
result
```

```
 i= 0  acc= []
  i= 1  acc= [8]
   i= 2  acc= [8, 18]
    i= 3  acc= [8, 18, 2]
     i= 4  acc= [8, 18, 2, 10]
      i= 5  acc= [8, 18, 2, 10, 14]
       i= 6  acc= [8, 18, 2, 10, 14, 6]
      res= [8, 18, 2, 10, 14, 6]
     res= [8, 18, 2, 10, 14, 6]
    res= [8, 18, 2, 10, 14, 6]
   res= [8, 18, 2, 10, 14, 6]
  res= [8, 18, 2, 10, 14, 6]
 res= [8, 18, 2, 10, 14, 6]
```

[4]:
```
[8, 18, 2, 10, 14, 6]
```

[4]:
```
result = adouble([4,9,1,5,7,3])
result
```

```
 i= 0  acc= []
  i= 1  acc= [8]
   i= 2  acc= [8, 18]
    i= 3  acc= [8, 18, 2]
     i= 4  acc= [8, 18, 2, 10]
      i= 5  acc= [8, 18, 2, 10, 14]
       i= 6  acc= [8, 18, 2, 10, 14, 6]
      res= [8, 18, 2, 10, 14, 6]
     res= [8, 18, 2, 10, 14, 6]
    res= [8, 18, 2, 10, 14, 6]
   res= [8, 18, 2, 10, 14, 6]
  res= [8, 18, 2, 10, 14, 6]
 res= [8, 18, 2, 10, 14, 6]
```

[4]:
```
[8, 18, 2, 10, 14, 6]
```

Exercise - afilter_even

Takes a `lst` of numbers, and RETURN a NEW list with only the even numbers

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

[5]:
```python
def afilter_even_helper(lst, i, acc):

    if i == len(lst):
        return acc
    else:
        if lst[i] % 2 == 0:
            acc.append(lst[i])
            return afilter_even_helper(lst, i=i+1, acc=acc)
        else:
            return afilter_even_helper(lst, i=i+1, acc=acc)


def afilter_even(lst):

    return afilter_even_helper(lst, 0, [])


result = afilter_even([3,7,10,5,4,8,6,11])
print(result)  # [10,4,8,6]

assert afilter_even([3]) == []
assert afilter_even([4]) == [4]
assert afilter_even([3,6]) == [6]
assert afilter_even([8,3]) == [8]
assert afilter_even([8,3,2]) == [8,2]
assert afilter_even([7,7,8,3,2]) == [8,2]
assert afilter_even([3,7,10,5,4,8,6,11]) == [10,4,8,6]
```

```
[10, 4, 8, 6]
```

[5]:

```
def afilter_even(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = afilter_even([3,7,10,5,4,8,6,11])
print(result)  # [10,4,8,6]

assert afilter_even([3]) == []
assert afilter_even([4]) == [4]
assert afilter_even([3,6]) == [6]
assert afilter_even([8,3]) == [8]
assert afilter_even([8,3,2]) == [8,2]
assert afilter_even([7,7,8,3,2]) == [8,2]
assert afilter_even([3,7,10,5,4,8,6,11]) == [10,4,8,6]
```

Exercise - amerry

Given a `lst` with an even number of elements, RETURN a NEW list holding two element lists with consecutive content taken from the original list.

<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 amerry_helper(lst, i, acc):
    if i == len(lst):
        return acc
    else:
        acc.append([lst[i], lst[i+1]])
        return  amerry_helper(lst, i+2, acc)


def amerry(lst):

    return amerry_helper(lst, 0, [])


result = amerry(['a','b','c','c','a','d','b','a'])
print(result)  # [['a','b'],['c','c'],['a','d'],['b','a']]

assert amerry([]) == []
assert amerry(['d','b']) == [['d','b']]
assert amerry(['a','a','b','b']) == [['a','a'],['b','b']]
assert amerry(['a','b','c','d']) == [['a','b'],['c','d']]
assert amerry(['a','b','c','c','a','d','b','a']) == [['a','b'],['c','c'],['a','d'],['b
↪','a']]
```

```
[['a', 'b'], ['c', 'c'], ['a', 'd'], ['b', 'a']]
```

[6]:
```
def amerry(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = amerry(['a','b','c','c','a','d','b','a'])
print(result)  # [['a','b'],['c','c'],['a','d'],['b','a']]

assert amerry([]) == []
assert amerry(['d','b']) == [['d','b']]
assert amerry(['a','a','b','b']) == [['a','a'],['b','b']]
assert amerry(['a','b','c','d']) == [['a','b'],['c','d']]
assert amerry(['a','b','c','c','a','d','b','a']) == [['a','b'],['c','c'],['a','d'],['b
↪','a']]
```

Exercise - afib

Given a number `n`, RETURN a NEW list with all the numbers of Fibonacci sequence:

n	0	1	2	3	4	5	6	7	8	9	10
x_n	0	1	1	2	3	5	8	13	21	34	…

Base cases:

- $x_0 = 0$
- $x_1 = 1$

The next number is found by adding up the two numbers before it:

- x_2 is 0+1
- x_3 is 1+1
- x_4 is 2+1
- x_5 is 3+2
- and so on…

<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 afib_helper(n, acc):
    if n == 0:
        acc.append(0)
        return acc
    elif n == 1:
        acc.append(0)
        acc.append(1)
        return acc
    else:
        acc = afib_helper(n-1, acc)
        acc.append(acc[n-1] + acc[n-2])
        return acc
```

(continues on next page)

(continued from previous page)

```
def afib(n):

    return afib_helper(n, [])


result = afib(7)
print(result) # [0,1,1,2,3,5,8,13]

assert afib(0) == [0]
assert afib(1) == [0,1]
assert afib(2) == [0,1,1]
assert afib(3) == [0,1,1,2]
assert afib(4) == [0,1,1,2,3]
assert afib(5) == [0,1,1,2,3,5]
assert afib(6) == [0,1,1,2,3,5,8]
assert afib(7) == [0,1,1,2,3,5,8,13]
```

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

[7]:

```
def afib(n):
    raise Exception('TODO IMPLEMENT ME !')

result = afib(7)
print(result) # [0,1,1,2,3,5,8,13]

assert afib(0) == [0]
assert afib(1) == [0,1]
assert afib(2) == [0,1,1]
assert afib(3) == [0,1,1,2]
assert afib(4) == [0,1,1,2,3]
assert afib(5) == [0,1,1,2,3,5]
assert afib(6) == [0,1,1,2,3,5,8]
assert afib(7) == [0,1,1,2,3,5,8,13]
```

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

Exercise - asearch

Given an `el` to search and a `lst`, RETURN `True` if such element is present or `False` otherwise

- **DO NOT** use `in` operator nor search methods like `.find`, `.index`, ….
- you might consider shortening your code by using Boolean evaluation order[322]

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

[8]:

```
```

(continues on next page)

[322] https://en.softpython.org/basics/basics2-bools-sol.html#Evaluation-order

(continued from previous page)

```
def asearch_helper(el, lst, i):

    if i == len(lst):
        return False
    else:
        return lst[i] == el or asearch_helper(el,lst, i+1)  # early termination using␣
↪ boolean evaluation order!


def asearch(el, lst):

    return asearch_helper(el,lst, 0)


print(asearch(7, [6,4,7,8]))   # True
print(asearch(5, [6,4,7,8]))   # False

assert asearch(7, [6,4,7,8]) == True
assert asearch(7, [8]) == False
assert asearch(8, [8]) == True
assert asearch(5, [8,5]) == True
assert asearch(2, [8,5]) == False
assert asearch(7, [7,8,5]) == True
assert asearch(2, [8,2,5]) == True
assert asearch(9, [7,8,9]) == True
assert asearch(2, [8,9,5]) == False
```

```
True
False
```

[8]:

```
def asearch(el, lst):
    raise Exception('TODO IMPLEMENT ME !')

print(asearch(7, [6,4,7,8]))   # True
print(asearch(5, [6,4,7,8]))   # False

assert asearch(7, [6,4,7,8]) == True
assert asearch(7, [8]) == False
assert asearch(8, [8]) == True
assert asearch(5, [8,5]) == True
assert asearch(2, [8,5]) == False
assert asearch(7, [7,8,5]) == True
assert asearch(2, [8,2,5]) == True
assert asearch(9, [7,8,9]) == True
assert asearch(2, [8,9,5]) == False
```

Exercise - abin_search

If the list is already sorted we can write a faster search. Start looking in the middle of the list: if that cell is the searched element, you're done. Otherwise, if searched element is lesser do recursion on left sublist, otherwise on right sublist.

- `lst` is a list of n **already sorted** elements
- **HINT 1**: add indeces parameters `i` and `j` are indeces between `0` included and n **included**
- **HINT 2**: keep splitting the search interval in halves until you have a 1 element interval (or empty interval)

DO NOT use `in` operator, `.find` nor `.index` methods

<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 abin_search_helper(el, lst, i, j):
    #print('i =', i, ' j =', j)
    interval = j - i
    #print('interval =', interval)
    k = (i + j) // 2
    #print('k =', k)

    if interval <= 0:
        return False
    elif el == lst[k]:
        return True
    #elif interval == 1:
    #    return False
    elif el < lst[k]:
        return abin_search_helper(el, lst, 0, k)
    else: # >
        return abin_search_helper(el, lst, k+1, j)


def abin_search(el, lst):

    return abin_search_helper(el, lst, 0, len(lst))



print(abin_search(33, [8,9,15,17,20,24,28,29,33,36,39])) # True
print(abin_search(10, [8,9,15,17,20,24,28,29,33,36,39])) # False

assert abin_search(5, [5]) == True
assert abin_search(4, []) == False
assert abin_search(3, [7]) == False
assert abin_search(3, [3,7]) == True
assert abin_search(3, [3,6]) == True
assert abin_search(4, [2,9]) == False
assert abin_search(5, [2,9]) == False
assert abin_search(5, [5,7,9]) == True
assert abin_search(7, [5,7,9]) == True
assert abin_search(9, [5,7,9]) == True
assert abin_search(4, [5,7,9]) == False
assert abin_search(6, [5,7,9]) == False
assert abin_search(8, [5,7,9]) == False
```

(continues on next page)

(continued from previous page)

```
assert abin_search(10, [5,7,9]) == False
assert abin_search(33, [8,9,15,17,20,24,28,29,33,36,39]) == True
assert abin_search(10, [8,9,15,17,20,24,28,29,33,36,39]) == False
```

```
True
False
```

[9]:

```
def abin_search(el, lst):
    raise Exception('TODO IMPLEMENT ME !')


print(abin_search(33, [8,9,15,17,20,24,28,29,33,36,39])) # True
print(abin_search(10, [8,9,15,17,20,24,28,29,33,36,39])) # False

assert abin_search(5, [5]) == True
assert abin_search(4, []) == False
assert abin_search(3, [7]) == False
assert abin_search(3, [3,7]) == True
assert abin_search(3, [3,6]) == True
assert abin_search(4, [2,9]) == False
assert abin_search(5, [2,9]) == False
assert abin_search(5, [5,7,9]) == True
assert abin_search(7, [5,7,9]) == True
assert abin_search(9, [5,7,9]) == True
assert abin_search(4, [5,7,9]) == False
assert abin_search(6, [5,7,9]) == False
assert abin_search(8, [5,7,9]) == False
assert abin_search(10, [5,7,9]) == False
assert abin_search(33, [8,9,15,17,20,24,28,29,33,36,39]) == True
assert abin_search(10, [8,9,15,17,20,24,28,29,33,36,39]) == False
```

Exercise - azip

Given two lists, RETURN a NEW list having couples as tuples with elements taken pairwise from `lst1` and `lst2`

- **DO NOT** use `zip` 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 azip_helper(lst1, lst2, i, acc):

    if i == len(lst1):
        return acc
    else:
        acc.append((lst1[i], lst2[i]))
        return azip_helper(lst1, lst2, i+1, acc)


def azip(lst1, lst2):
```

(continues on next page)

(continued from previous page)

```
    return azip_helper(lst1, lst2, 0, [])


result = azip([3,5,1,6], ['a','d','e','w'])
print(result)  # [(3,'a'), (5,'d'), (1,'e'), (6,'w')]

assert azip([], []) == []
assert azip(['a'], [0]) == [('a',0)]
assert azip(['z','a'], [2,4]) == [('z',2), ('a',4)]
assert azip([3,5,1,6], ['a','d','e','w']) == [(3,'a'), (5,'d'), (1,'e'), (6,'w')]
```

```
[(3, 'a'), (5, 'd'), (1, 'e'), (6, 'w')]
```

[10]:

```
def azip(lst1, lst2):
    raise Exception('TODO IMPLEMENT ME !')

result = azip([3,5,1,6], ['a','d','e','w'])
print(result)  # [(3,'a'), (5,'d'), (1,'e'), (6,'w')]

assert azip([], []) == []
assert azip(['a'], [0]) == [('a',0)]
assert azip(['z','a'], [2,4]) == [('z',2), ('a',4)]
assert azip([3,5,1,6], ['a','d','e','w']) == [(3,'a'), (5,'d'), (1,'e'), (6,'w')]
```

Exercise - aunnest

Given a `lst` of lists, RETURN a NEW list without sublists (also called *shallow unnest*)

- assume sublists cannot have further sublists inside

<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 aunnest_helper(lst, i, acc):

    if i == len(lst):
        return acc
    else:
        acc.extend(lst[i])
        return aunnest_helper(lst, i+1, acc)


def aunnest(lst):

    return aunnest_helper(lst, 0, [])


result = aunnest([[3,2,5], [2,4], [0,9,5,1]])
```

(continues on next page)

(continued from previous page)

```
print(result)  # [3,2,5,2,4,0,9,5,1]

assert aunnest([]) == []
assert aunnest([[]]) == []
assert aunnest([[],[]]) == []
assert aunnest([[],[4]]) == [4]
assert aunnest([[3],[]]) == [3]
assert aunnest([[3,2]]) == [3,2]
assert aunnest([[3,2]]) == [3,2]
assert aunnest([[3,2,5], [2,4], [0,9,5,1]]) == [3,2,5,2,4,0,9,5,1]
```

```
[3, 2, 5, 2, 4, 0, 9, 5, 1]
```

[11]:

```
def aunnest(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = aunnest([[3,2,5], [2,4], [0,9,5,1]])
print(result)  # [3,2,5,2,4,0,9,5,1]

assert aunnest([]) == []
assert aunnest([[]]) == []
assert aunnest([[],[]]) == []
assert aunnest([[],[4]]) == [4]
assert aunnest([[3],[]]) == [3]
assert aunnest([[3,2]]) == [3,2]
assert aunnest([[3,2]]) == [3,2]
assert aunnest([[3,2,5], [2,4], [0,9,5,1]]) == [3,2,5,2,4,0,9,5,1]
```

Exercise - arev

Given a list, RETURN a NEW list reversed.

- **DO NOT** use `reversed` function
- **DO NOT** use slice stepping

<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 arev_helper(lst, i, acc):
    if i == -1:
        return acc
    else:
        acc.append(lst[i])
        return arev_helper(lst, i-1,acc)


def arev(lst):
```

(continues on next page)

(continued from previous page)

```
    return arev_helper(lst, len(lst)-1, [])


result = arev([4,9,7,5,6])
print(result)

assert arev([]) == []
assert arev([3]) == [3]
assert arev([5,3]) == [3,5]
assert arev([6,6,2]) == [2,6,6]
assert arev([4,9,7,5,6]) == [6,5,7,9,4]
```

```
[6, 5, 7, 9, 4]
```

[12]:

```
def arev(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = arev([4,9,7,5,6])
print(result)

assert arev([]) == []
assert arev([3]) == [3]
assert arev([5,3]) == [3,5]
assert arev([6,6,2]) == [2,6,6]
assert arev([4,9,7,5,6]) == [6,5,7,9,4]
```

Exercise - apalindrome

A string is palindrome if it reads the same backward as forward.

Given a string `s`, return `True` if the string is palindrome, otherwise return `False`.

- we assume the empty string is palindrome
- here we use strings but same rules for lists apply
- **DO NOT** use string methods (so no `.index()`, `.find()`, ...)

<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 apalindrome_helper(s, i):
    if i == len(s)//2:
        return True
    elif i == len(s)//2 + 1:
        return True
    else:
        if s[i] == s[-i-1]:
            return apalindrome_helper(s, i+1)
        else:
            return False
```

(continues on next page)

(continued from previous page)

```
def apalindrome(s):

    return apalindrome_helper(s,0)


assert apalindrome('') == True
assert apalindrome('a') == True
assert apalindrome('ab') == False
assert apalindrome('cc') == True
assert apalindrome('ddv') == False
assert apalindrome('vddv') == True
assert apalindrome('axyb') == False
assert apalindrome('kayak') == True
assert apalindrome('kayag') == False
assert apalindrome('rotator') == True
assert apalindrome('rotating') == False
assert apalindrome('noon') == True
assert apalindrome('soon') == False
assert apalindrome('abcdba') == False
```

[13]:

```
def apalindrome(s):
    raise Exception('TODO IMPLEMENT ME !')

assert apalindrome('') == True
assert apalindrome('a') == True
assert apalindrome('ab') == False
assert apalindrome('cc') == True
assert apalindrome('ddv') == False
assert apalindrome('vddv') == True
assert apalindrome('axyb') == False
assert apalindrome('kayak') == True
assert apalindrome('kayag') == False
assert apalindrome('rotator') == True
assert apalindrome('rotating') == False
assert apalindrome('noon') == True
assert apalindrome('soon') == False
assert apalindrome('abcdba') == False
```

Exercise - anest

Given a `lst`, RETURN a NEW list of deeply right nested lists as in the example

- **HINT**: you need to create a new list anyway at each helper call

<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 anest_helper(lst, i, acc):
    if i == len(lst):
        return acc
```

(continues on next page)

(continued from previous page)

```
    elif i == len(lst) - 1:
        acc.append(lst[i])
        return acc
    else:
        acc.append(lst[i])
        res = anest_helper(lst, i+1, []) # we need to create a NEW list anyway
        acc.append(res)
        return acc


def anest(lst):

    return anest_helper(lst, 0, [])


result = anest(['a','b','c','d','e','g','h'])
print(result)  # ['a',['b',['c',['d',['e',['g',['h']]]]]]]

assert anest([]) == []
assert anest(['a']) == ['a']
assert anest(['a','b']) == ['a',['b']]
assert anest(['a','b','c']) == ['a',['b',['c']]]
assert anest(['x','x','y']) == ['x',['x',['y']]]
assert anest(['a','b','c','d','e','g','h']) == ['a',['b',['c',['d',['e',['g',['h
↪']]]]]]]
```

```
['a', ['b', ['c', ['d', ['e', ['g', ['h']]]]]]]
```

[14]:

```
def anest(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = anest(['a','b','c','d','e','g','h'])
print(result)  # ['a',['b',['c',['d',['e',['g',['h']]]]]]]

assert anest([]) == []
assert anest(['a']) == ['a']
assert anest(['a','b']) == ['a',['b']]
assert anest(['a','b','c']) == ['a',['b',['c']]]
assert anest(['x','x','y']) == ['x',['x',['y']]]
assert anest(['a','b','c','d','e','g','h']) == ['a',['b',['c',['d',['e',['g',['h
↪']]]]]]]
```

Exercise - apalace

Given a `lst` of characters, RETURN a NEW list of deeply nested lists as in the example.

- Pay attention to base cases, there are several
- **HINT**: you need to create a new list anyway at each helper call

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

[15]:

```
def apalace_helper(lst, i, acc):
    if len(lst) == 0:
        return acc
    elif i == len(lst)//2 + 1:
        return acc
    elif i == len(lst)//2:
        acc.append(lst[i])
        return acc
    elif i == len(lst) - 2:
        acc.append(lst[i])
        acc.append(lst[i+1])
        return acc
    else:
        acc.append(lst[i])
        res = apalace_helper(lst, i+1, [])  # we need to create a NEW list anyway
        acc.append(res)
        acc.append(lst[-i-1])
        return acc


def apalace(lst):

    return apalace_helper(lst,0,[])


result = apalace(['a','b','c','d','e','f','g','h','i'])
print(result)   # ['a', ['b', ['c', ['d', ['e'], 'f'], 'g'], 'h'], 'i']

assert apalace([])  == []
assert apalace(['a'])  == ['a']
assert apalace(['a','b'])  == ['a','b']
assert apalace(['b','a'])  == ['b','a']
assert apalace(['c','b','a'])  == ['c',['b'],'a']
assert apalace(['z','w','z'])  == ['z',['w'],'z']
assert apalace(['p','x','y','z','q'])  == ['p',['x',['y'],'z'],'q']
assert apalace(['a','b','c','d','e','f','g','h','i'])  == ['a', ['b', ['c', ['d', ['e
↪'], 'f'], 'g'], 'h'], 'i']
```

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

[15]:

```
def apalace(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = apalace(['a','b','c','d','e','f','g','h','i'])
print(result)   # ['a', ['b', ['c', ['d', ['e'], 'f'], 'g'], 'h'], 'i']

assert apalace([])  == []
assert apalace(['a'])  == ['a']
assert apalace(['a','b'])  == ['a','b']
```

(continues on next page)

(continued from previous page)

```
assert apalace(['b','a'])  == ['b','a']
assert apalace(['c','b','a'])  == ['c',['b'],'a']
assert apalace(['z','w','z'])  == ['z',['w'],'z']
assert apalace(['p','x','y','z','q'])  == ['p',['x',['y'],'z'],'q']
assert apalace(['a','b','c','d','e','f','g','h','i'])  == ['a', ['b', ['c', ['d', ['e
↪'], 'f'], 'g'], 'h'], 'i']
```

Exercise - arep

Given a list `la`, and a list `lb` of integer numbers ≥ 0, RETURN a NEW list where all elements of `la` are repeated the amount of times indicated in the corresponding cell of `lb`

- assume `la` and `lb` have the same size
- **DO NOT** use `*` replication operator
- **HINT**: you will need an additional parameter `k` in the helper to keep track of the amount of repetitions left

<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 arep_helper(la, lb, i, k, acc):
    if i == len(la):
        return acc
    elif k > 0:
        acc.append(la[i])
        return arep_helper(la, lb, i, k-1, acc)
    else:  # k == 0
        if i + 1 < len(la):
            return arep_helper(la, lb, i+1, lb[i+1], acc)
        else:
            return acc


def arep(la, lb):

    if len(la) == 0:
        return []
    else:
        return arep_helper(la,lb,0,lb[0],[])


result = arep(['a','b','c','d'], [2,5,0,3])
print(result) # ['a','a','b','b','b','b','b','d','d','d']

assert arep([], []) == []
assert arep(['a'], [1]) == ['a']
assert arep(['a'], [0]) == []
assert arep(['b'], [2]) == ['b','b']
assert arep(['a','b'], [1,1]) == ['a','b']
assert arep(['a','b'], [1,2]) == ['a','b','b']
assert arep(['a','b'], [2,1]) == ['a','a','b']
assert arep(['a','b'], [2,0]) == ['a','a']
assert arep(['a','b'], [0,2]) == ['b','b']
```

(continues on next page)

(continued from previous page)

```
assert arep(['a','b','c'], [1,1,1]) == ['a','b','c']
assert arep(['a','b','c'], [0,1,1]) == ['b','c']
assert arep(['a','b','c'], [0,1,0]) == ['b']
assert arep(['a','b','c'], [3,1,2]) == ['a','a','a','b','c','c']
assert arep(['a','b','c','d'], [2,5,0,3]) == ['a','a','b','b','b','b','b','d','d','d']
```

```
['a', 'a', 'b', 'b', 'b', 'b', 'b', 'd', 'd', 'd']
```

[16]:

```
def arep(la, lb):
    raise Exception('TODO IMPLEMENT ME !')

result = arep(['a','b','c','d'], [2,5,0,3])
print(result) # ['a','a','b','b','b','b','b','d','d','d']

assert arep([], []) == []
assert arep(['a'], [1]) == ['a']
assert arep(['a'], [0]) == []
assert arep(['b'], [2]) == ['b','b']
assert arep(['a','b'], [1,1]) == ['a','b']
assert arep(['a','b'], [1,2]) == ['a','b','b']
assert arep(['a','b'], [2,1]) == ['a','a','b']
assert arep(['a','b'], [2,0]) == ['a','a']
assert arep(['a','b'], [0,2]) == ['b','b']
assert arep(['a','b','c'], [1,1,1]) == ['a','b','c']
assert arep(['a','b','c'], [0,1,1]) == ['b','c']
assert arep(['a','b','c'], [0,1,0]) == ['b']
assert arep(['a','b','c'], [3,1,2]) == ['a','a','a','b','c','c']
assert arep(['a','b','c','d'], [2,5,0,3]) == ['a','a','b','b','b','b','b','d','d','d']
```

Exercise - asortin

Given two **already sorted** lists `la` and `lb` of arbitrary sizes, return `True` if all numbers of `la` are present in `lb`

- assume empty list is always contained in any list
- **DO NOT** use search methods (`find()`, `index()`, …)
- **HINT 1**: compare list heads one at a time, try thinking when you can return early
- **HINT 2**: you will need a couple of indexes in the helper (you don't need an accumulator)

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

[17]:

```
def asortin_helper(la, lb, i, j):
    if i == len(la):
        return True
    elif j == len(lb):
        return False
    elif la[i] < lb[j]:
```

(continues on next page)

(continued from previous page)

```
        return False
    elif la[i] > lb[j]:
        return asortin_helper(la, lb, i, j+1)
    else:   # l[i] == lb[j]
        return asortin_helper(la, lb, i+1, j)


def asortin(la, lb):


    return asortin_helper(la,lb,0,0)


assert asortin([], []) == True
assert asortin([], [4,5,8,1]) == True
assert asortin([5], [5]) == True
assert asortin([5], [6]) == False
assert asortin([3], [3,8]) == True
assert asortin([8], [3,8]) == True
assert asortin([2], [3,8]) == False
assert asortin([7,9], [7,9]) == True
assert asortin([7,9], [3,7,9]) == True
assert asortin([7,9], [7,8,9]) == True
assert asortin([7,9], [7,9,10]) == True
assert asortin([5,8], [2,8]) == False
assert asortin([5,8], [5,6]) == False
assert asortin([5,8], [2,4,8]) == False
assert asortin([5,8], [4,5,7]) == False
assert asortin([3,5,6], [3,5,6]) == True
assert asortin([3,5,6], [3,4,5,6]) == True
assert asortin([3,5,6], [3,5,6,7]) == True
assert asortin([4,6,11], [1,3,4,5,6,7,8,10,11,12]) == True
```

[17]:

```
def asortin(la, lb):
    raise Exception('TODO IMPLEMENT ME !')

assert asortin([], []) == True
assert asortin([], [4,5,8,1]) == True
assert asortin([5], [5]) == True
assert asortin([5], [6]) == False
assert asortin([3], [3,8]) == True
assert asortin([8], [3,8]) == True
assert asortin([2], [3,8]) == False
assert asortin([7,9], [7,9]) == True
assert asortin([7,9], [3,7,9]) == True
assert asortin([7,9], [7,8,9]) == True
assert asortin([7,9], [7,9,10]) == True
assert asortin([5,8], [2,8]) == False
assert asortin([5,8], [5,6]) == False
assert asortin([5,8], [2,4,8]) == False
assert asortin([5,8], [4,5,7]) == False
assert asortin([3,5,6], [3,5,6]) == True
assert asortin([3,5,6], [3,4,5,6]) == True
assert asortin([3,5,6], [3,5,6,7]) == True
```

(continues on next page)

(continued from previous page)

```
assert asortin([4,6,11], [1,3,4,5,6,7,8,10,11,12]) == True
```

Exercise - ahist

Given a `lst`, RETURN a NEW dictionary with the counts of elements of `lst`

- **DO NOT** use search methods (`.index`, `.count`, `.remove`...)

<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 ahist_helper(lst, i, acc):
    if i == len(lst):
        return acc
    else:
        if lst[i] in acc:
            acc[lst[i]] += 1
        else:
            acc[lst[i]] = 1
        return ahist_helper(lst, i+1, acc)


def ahist(lst):

    return ahist_helper(lst, 0, {})


result = ahist(['a','b','a','c','b','b','c','a','b','c','d'])
print(result)  # {'a':3,
               #  'b':4,
               #  'c':3,
               #  'd':1}

assert ahist([]) == {}
assert ahist(['a']) == {'a':1}
assert ahist(['a','a']) == {'a':2}
assert ahist(['a','b']) == {'a':1,
                            'b':1}
assert ahist(['b','c']) == {'c':1,
                            'b':1}
assert ahist(['a','c','a']) == {'a':2,
                                'c':1}
assert ahist(['c','c','a']) == {'a':1,
                                'c':2}
assert ahist(['a','b','a','c','b','b','c','a','b','c','d']) == {'a':3,
                                                               'b':4,
                                                               'c':3,
                                                               'd':1}
```

```
{'a': 3, 'b': 4, 'c': 3, 'd': 1}
```

[18]:

```
def ahist(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = ahist(['a','b','a','c','b','b','c','a','b','c','d'])
print(result)  # {'a':3,
               #  'b':4,
               #  'c':3,
               #  'd':1}

assert ahist([]) == {}
assert ahist(['a']) == {'a':1}
assert ahist(['a','a']) == {'a':2}
assert ahist(['a','b']) == {'a':1,
                            'b':1}
assert ahist(['b','c']) == {'c':1,
                            'b':1}
assert ahist(['a','c','a']) == {'a':2,
                                'c':1}
assert ahist(['c','c','a']) == {'a':1,
                                'c':2}
assert ahist(['a','b','a','c','b','b','c','a','b','c','d']) == {'a':3,
                                                               'b':4,
                                                               'c':3,
                                                               'd':1}
```

Exercise - agap

In a list L containing $n \geq 2$ integers, a gap is an index i, $0 < i < n$, such that $L[i-1] < L[i]$

If $n \geq 2$ and $L[0] < L[n-1]$, L contains at least one gap.

Design an algorithm that, given a list L containing $n \geq 2$ integers such that $L[0] < L[n-1]$, finds a gap in the list.

We wrote here the `gap` function as pseudocode, try coding it in Python:

gap(int[] L, int i, int j)

if $j == i + 1$ **then**
  return j
$m = \lfloor (i + j)/2 \rfloor$
if $L[m] \leq L[i]$ **then**
  return gap(L, m, j)
else
  return gap(L, i, m)

Use the following skeleton and add some test. To understand what's going on, try copy-pasting in Python tutor[323]

[323] http://pythontutor.com/visualize.html#mode=edit

CAREFUL ABOUT `agap` ASSUMPTIONS!

The specification of `agap` assumes the input is always a list of at least two elements, and that the first element is less or equal than the last one. *If these conditions are not met, function behaviour could be completely erroneus!*

When preconditions are not met, execution could stop because of an error like index out of bounds, or, even worse, we might get back some wrong index as a gap! To prevent misuse of the function, a good idea can be putting a check at the beginning of the `agap` function. Such check should immediately stop the execution and raise an error if the parameters don't satisfy the preconditions. One way to do this could be adding some assertion[324] that will make python interrupt execution and throw an error as soon it detects list `L` is too small or with wrong values * This kind of behaviour is also called *fail fast*, which is better than returning wrong values! * You can put any condition you want after `assert`, but ideally they should be fast to execute. * asserts might be better here than `raise Exception` constructs because asserts can be disabled with a flag passed to the interpreter. So, when you debug you can take advantage of them, and when the code is production quality and supposed to be bug free you can disable all assertions at once to gain in execution speed.

GOOD PRACTICE: Notice we wrote what the helper function is expected to receive as a comment. Writing down specs often helps understanding what the function is supposed to do, and helps users of your code as well!

VII COMMANDMENT[325]: You shall also write on paper!

To get an idea of how `agap` is working, draw histograms on paper like the following, with different heights at index `m`:

Notice how at each recursive call, we end up with a histogram that is similar to the inital one, that is, it respects the same preconditions (a list of size ≥ 2 where first element is smaller or equal than the last one)

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

[19]:
```
def agap_helper(L, i, j):

    if j == i+1:
        return j
    else:
        m = (i+j) // 2
        if L[m] < L[j]:
            return agap_helper(L,m,j)
```

(continues on next page)

[324] https://sciprog.davidleoni.it/testing/testing.html#Assertions
[325] https://en.softpython.org/commandments.html#VII-COMMANDMENT

(continued from previous page)

```
        else:
            return agap_helper(L,i,m)


def agap(L):

    assert len(L) >= 2
    assert L[0] <= L[len(L)-1]
    return agap_helper(L, 0, len(L) - 1)




# try also to write asserts
```

[19]:

```
def agap_helper(L, i, j):
    raise Exception('TODO IMPLEMENT ME !')

def agap(L):
    raise Exception('TODO IMPLEMENT ME !')



# try also to write asserts
```

Continue

Go on with divide and conquer strategy[326]

7.3.3 Recursion 3 - Divide and conquer

When confronted with a problem, we can try to solve it by splitting its dimension in half (or more), look for solutions in each of the halves with two recursive calls, and then decide what to do with the found solutions, if any.

Why should we do this? So far we wrote sequential algorithms that basically exploit only one CPU core on one computer, but modern CPUs have multiple cores and data centers multiple servers. If we could keep busy all the CPU cores on all the servers our algorithms would perform way faster.

By partitioning a problem into completely separate parts with a divide and conquer strategy, we might hope it becomes amenable to parallelization. To understand if a strategy really separates the problem into clear-cut parts, without possibility of i.e. recursive calls overwriting data which should only be read by other calls, we might ask ourselves the following question: if we swapped the recursive calls, would we get a different result? If the answer is no, the program is probably going to be a good candidate for parallelization.

[326] https://sciprog.davidleoni.it/recursion/recursion3-divide.html

Note in these worksheets we only care about the logic and will not actually see how to execute programs in parallel - if interested, search for multithreading programming or big data frameworks such as Spark[327])

Managing solutions

After we split a problem in halves and do recursive calls on each halve, several cases may occur:

1. No solution is found
2. One solution is found
3. Two solutions are found

case 1): we can only give up

case 2): we have only one solution, so we can just return that one

case 3): we have two solutions, so we need to decide what is the purpose of the algorithm

Is the purpose to …

- find *all* possible solutions? Then we return both of them.
- find the *best* solution, according to some measure of 'goodness'? Then we measure each of the solutions and give back the highest scoring one.
- provide a *combination* of existing solutions, according to some mixing method? Then we combine the found solutions and give them back

Divide and conquer - SimpleFP style

Example - `dsdouble`

Suppose we are tasked with reimplementing the `double` like this:

Take a `lst` of numbers and RETURN a NEW list with the numbers doubled as in SimpleFP style but this time following a divide and conquer strategy.

Let's first focus on the logic, without caring about performance: we will divide the input in half, compute the doubled version of the halved lists (that is, find two solutions), and then join them at the end (combine the found solutions).

Notice the two recursive calls here are done sequentially one after each other, but if we wanted nothing would prevent us from executing them in parallel (in other words, if you swapped the two recursive calls, you would obtain exactly the same result)

[2]:
```
def dsdouble(lst):
    if len(lst) == 0:             # base case
        return []
    elif len(lst) == 1:           # base case
        return [lst[0] * 2]
    else:
        left = dsdouble(lst[:len(lst)//2])
        right = dsdouble(lst[len(lst)//2:])
        return left + right

result = dsdouble([4,6,2,7,3])
jupman.pytut()
```

[327] https://towardsdatascience.com/a-neanderthals-guide-to-apache-spark-in-python-9ef1f156d427

[2]:
```
<IPython.core.display.HTML object>
```

Exercise - debug dsdouble

Add a `level` parameter and some debugging indented print to the function

<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 dsdouble(lst, level=0):
    print('  '*level, 'lst=', lst)
    if len(lst) == 0:
        return []
    elif len(lst) == 1:
        return [lst[0] * 2]
    else:
        print('  '*level, 'left call')
        left = dsdouble(lst[:len(lst)//2], level + 1)
        print('  ' * level, 'left=', left)
        print('  '*level, 'right call')
        right = dsdouble(lst[len(lst)//2:], level + 1)
        print('  ' * level, 'right=', right)
        return left + right


result = dsdouble([4,6,2,7,3])
print('final result=', result)
```

```
 lst= [4, 6, 2, 7, 3]
 left call
   lst= [4, 6]
   left call
     lst= [4]
   left= [8]
   right call
     lst= [6]
   right= [12]
 left= [8, 12]
 right call
   lst= [2, 7, 3]
   left call
     lst= [2]
   left= [4]
   right call
     lst= [7, 3]
     left call
       lst= [7]
     left= [14]
     right call
       lst= [3]
     right= [6]
   right= [14, 6]
 right= [4, 14, 6]
final result= [8, 12, 4, 14, 6]
```

[4]:

```
result = dsdouble([4,6,2,7,3])
print('final result=', result)
```

```
 lst= [4, 6, 2, 7, 3]
 left call
   lst= [4, 6]
   left call
     lst= [4]
   left= [8]
   right call
     lst= [6]
   right= [12]
 left= [8, 12]
 right call
   lst= [2, 7, 3]
   left call
     lst= [2]
   left= [4]
   right call
     lst= [7, 3]
     left call
       lst= [7]
     left= [14]
     right call
       lst= [3]
     right= [6]
   right= [14, 6]
 right= [4, 14, 6]
final result= [8, 12, 4, 14, 6]
```

Exercise - dssum

Takes a `lst` of numbers and RETURN the sum of all numbers.

- **USE** SimpleFP style with a divide and conquer strategy.
- **DO NOT** use `sum` function ;-)

<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 dssum(lst):

    if len(lst) == 0:
        return 0
    else:
        return lst[0] + dssum(lst[1:])

    if len(lst) == 0:
        return 0
    else:
        k = len(lst) // 2
        res1 = dssum(lst[:k])
        res2 = dssum(lst[k:])
```

(continues on next page)

(continued from previous page)

```
        return res1 + lst[k] + res2


result = dssum([4,2,6,3])
print(result)  # 15

assert dssum([4,2,6,3]) == 15
assert dssum([]) == 0
assert dssum([3]) == 3
assert dssum([5,1]) == 6
```

```
15
```

[6]:

```
def dssum(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = dssum([4,2,6,3])
print(result)  # 15

assert dssum([4,2,6,3]) == 15
assert dssum([]) == 0
assert dssum([3]) == 3
assert dssum([5,1]) == 6
```

Exercise - dsmin

Given a `lst` of numbers, RETURN the minimum

- **assume the list has at least one element**
- **USE** SimpleFP style with a divide and conquer strategy
- **DON'T** use `min` function..

<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 dsmin(lst):

    if len(lst) == 1:
        return lst[0]
    else:
        k = len(lst) // 2
        res1 = dsmin(lst[:k])
        res2 = dsmin(lst[k:])
        if res1 < res2:
            return res1
        else:
            return res2
```

(continues on next page)

(continued from previous page)

```
result = dsmin([3,6,5,8,-3,8,-2,-10,4])
print(result)  # -10

assert dsmin([4]) == 4
assert dsmin([4,1]) == 1
assert dsmin([0,9]) == 0
assert dsmin([6,8,5]) == 5
assert dsmin([3,6,5,8,3,8,2,10,4]) == 2
```

```
-10
```

[7]:

```
def dsmin(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = dsmin([3,6,5,8,-3,8,-2,-10,4])
print(result)  # -10

assert dsmin([4]) == 4
assert dsmin([4,1]) == 1
assert dsmin([0,9]) == 0
assert dsmin([6,8,5]) == 5
assert dsmin([3,6,5,8,3,8,2,10,4]) == 2
```

Exercise - dszip

Given two lists, RETURN a NEW list having two-element tuples with elements taken pairwise from `lst1` and `lst2`

- **USE** SimpleFP style with a divide and conquer strategy
- **DO NOT** use `zip` 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]:

```
def dszip(lst1, lst2):

    if len(lst1) == 0:
        return []
    else:
        k = len(lst1) // 2
        mt = [(lst1[k], lst2[k])]
        res1 = dszip(lst1[:k], lst2[:k])
        res2 = dszip(lst1[k+1:], lst2[k+1:])
        return res1 + mt + res2


result = dszip([3,5,1,6], ['a','d','e','w'])
print(result)  # [(3,'a'), (5,'d'), (1,'e'), (6,'w')]

assert dszip([], []) == []
assert dszip(['a'], [0]) == [('a',0)]
```

(continues on next page)

(continued from previous page)

```
assert dszip(['z','a'], [2,4]) == [('z',2), ('a',4)]
assert dszip([3,5,1,6], ['a','d','e','w']) == [(3,'a'), (5,'d'), (1,'e'), (6,'w')]
```

```
[(3, 'a'), (5, 'd'), (1, 'e'), (6, 'w')]
```

[8]:

```
def dszip(lst1, lst2):
    raise Exception('TODO IMPLEMENT ME !')

result = dszip([3,5,1,6], ['a','d','e','w'])
print(result)  # [(3,'a'), (5,'d'), (1,'e'), (6,'w')]

assert dszip([], []) == []
assert dszip(['a'], [0]) == [('a',0)]
assert dszip(['z','a'], [2,4]) == [('z',2), ('a',4)]
assert dszip([3,5,1,6], ['a','d','e','w']) == [(3,'a'), (5,'d'), (1,'e'), (6,'w')]
```

Exercise - dsunnest

Given a `lst` of lists, RETURN a NEW list without sublists (also called *shallow unnest*)

- assume sublists cannot have further sublists inside
- **USE** SimpleFP style with a divide and conquer strategy

<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 dsunnest(lst):

    if len(lst) == 0:
        return []
    else:
        k = len(lst) // 2
        res1 = dsunnest(lst[:k])
        res2 = dsunnest(lst[k+1:])
        return res1 + lst[k] + res2

result = dsunnest([[3,2,5], [2,4], [0,9,5,1]])
print(result)  # [3,2,5,2,4,0,9,5,1]

assert dsunnest([]) == []
assert dsunnest([[]]) == []
assert dsunnest([[],[]]) == []
assert dsunnest([[],[4]]) == [4]
assert dsunnest([[3],[]]) == [3]
assert dsunnest([[3,2]]) == [3,2]
assert dsunnest([[3,2]]) == [3,2]
assert dsunnest([[3,2,5], [2,4], [0,9,5,1]]) == [3,2,5,2,4,0,9,5,1]
```

```
[3, 2, 5, 2, 4, 0, 9, 5, 1]
```

[9]:
```
def dsunnest(lst):
    raise Exception('TODO IMPLEMENT ME !')
result = dsunnest([[3,2,5], [2,4], [0,9,5,1]])
print(result)  # [3,2,5,2,4,0,9,5,1]

assert dsunnest([]) == []
assert dsunnest([[]]) == []
assert dsunnest([[],[]]) == []
assert dsunnest([[],[4]]) == [4]
assert dsunnest([[3],[]]) == [3]
assert dsunnest([[3,2]]) == [3,2]
assert dsunnest([[3,2]]) == [3,2]
assert dsunnest([[3,2,5], [2,4], [0,9,5,1]]) == [3,2,5,2,4,0,9,5,1]
```

Exercise - dsall

Given a `lst` of booleans, RETURN `True` if all elements are `True`, otherwise RETURN `False`

- **DO NOT** use `all` function
- assume `dsall` of empty list is `True`
- **USE** SimpleFP style with a divide and conquer strategy
- **HINT**: consider exploiting Boolean evaluation order[328] to shorten your code

<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 dsall(lst):

    if len(lst) == 0:
        return True
    else:
        k = len(lst) // 2
        # note that the first expression to become zero prevents further execution of␣
↪the following ones
        # because of boolean evaluation order
        return lst[k] and dsall(lst[:k]) and dsall(lst[k+1:])


assert dsall([]) == True
assert dsall([True]) == True
assert dsall([False]) == False
assert dsall([False, False]) == False
assert dsall([False, True]) == False
assert dsall([True, False]) == False
assert dsall([True, True]) == True
assert dsall([False, False, False]) == False
assert dsall([False, False, True]) == False
assert dsall([False, True, False]) == False
assert dsall([False, True, True]) == False
assert dsall([True, False, False]) == False
```

(continues on next page)

[328] https://en.softpython.org/basics/basics2-bools-sol.html#Evaluation-order

(continued from previous page)

```
assert dsall([True, False, True]) == False
assert dsall([True, True, False]) == False
assert dsall([True, True, True]) == True
```

[10]:

```
def dsall(lst):
    raise Exception('TODO IMPLEMENT ME !')

assert dsall([]) == True
assert dsall([True]) == True
assert dsall([False]) == False
assert dsall([False, False]) == False
assert dsall([False, True]) == False
assert dsall([True, False]) == False
assert dsall([True, True]) == True
assert dsall([False, False, False]) == False
assert dsall([False, False, True]) == False
assert dsall([False, True, False]) == False
assert dsall([False, True, True]) == False
assert dsall([True, False, False]) == False
assert dsall([True, False, True]) == False
assert dsall([True, True, False]) == False
assert dsall([True, True, True]) == True
```

Exercise - dsrev

Given a list, RETURN a NEW list reversed.

- **USE** SimpleFP style with a divide and conquer strategy
- **DO NOT** use `reversed` function

<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 dsrev(lst):

    if len(lst) == 0:
        return []
    else:
        k = len(lst) // 2
        res1 = dsrev(lst[:k])
        res2 = dsrev(lst[k+1:])
        return res2 + [lst[k]] + res1


result = dsrev([4,9,7,5,6])
print(result)

assert dsrev([]) == []
assert dsrev([3]) == [3]
assert dsrev([5,3]) == [3,5]
```

(continues on next page)

(continued from previous page)

```
assert dsrev([6,6,2]) == [2,6,6]
assert dsrev([4,9,7,5,6]) == [6,5,7,9,4]
```

```
[6, 5, 7, 9, 4]
```

[11]:

```
def dsrev(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = dsrev([4,9,7,5,6])
print(result)

assert dsrev([]) == []
assert dsrev([3]) == [3]
assert dsrev([5,3]) == [3,5]
assert dsrev([6,6,2]) == [2,6,6]
assert dsrev([4,9,7,5,6]) == [6,5,7,9,4]
```

Exercise - dsrep

Given a list `la`, and a list `lb` of integer numbers ≥ 0, RETURN a NEW list where all elements of `la` are repeated the amount of times indicated in the corresponding cell of `lb`. A bit hard, but gives satisfaction.

- assume `la` and `lb` have the same size
- **USE** SimpleFP style with a divide and conquer strategy
- **DO NOT** use `*` replication operator
- **HINT**: since both lists have equal length, you can apply splitting logic to both

<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 dsrep(la, lb):


    if len(la) == 0:
        return []
    else:
        k = len(la) // 2
        if lb[k] == 0:
            res1 = dsrep(la[:k], lb[:k])
            res2 = dsrep(la[k+1:], lb[k+1:])
            return res1 + res2
        else:
            res1 = dsrep(la[:k], lb[:k])
            res2 = dsrep(la[k:], [lb[k]-1] + lb[k+1:])
            return res1 + [la[k]] + res2


result = dsrep(['a','b','c','d'], [2,5,0,3])
print(result) # ['a','a','b','b','b','b','b','d','d','d']
```

(continues on next page)

(continued from previous page)

```
assert dsrep([], []) == []
assert dsrep(['a'], [1]) == ['a']
assert dsrep(['a'], [0]) == []
assert dsrep(['b'], [2]) == ['b','b']
assert dsrep(['a','b'], [1,1]) == ['a','b']
assert dsrep(['a','b'], [1,2]) == ['a','b','b']
assert dsrep(['a','b'], [2,1]) == ['a','a','b']
assert dsrep(['a','b'], [2,0]) == ['a','a']
assert dsrep(['a','b'], [0,2]) == ['b','b']
assert dsrep(['a','b','c'], [1,1,1]) == ['a','b','c']
assert dsrep(['a','b','c'], [0,1,1]) == ['b','c']
assert dsrep(['a','b','c'], [0,1,0]) == ['b']
assert dsrep(['a','b','c'], [3,1,2]) == ['a','a','a','b','c','c']
assert dsrep(['a','b','c','d'], [2,5,0,3]) == ['a','a','b','b','b','b','b','d','d','d
↪']
```

```
['a', 'a', 'b', 'b', 'b', 'b', 'b', 'd', 'd', 'd']
```

[12]:

```
def dsrep(la, lb):
    raise Exception('TODO IMPLEMENT ME !')

result = dsrep(['a','b','c','d'], [2,5,0,3])
print(result) # ['a','a','b','b','b','b','b','d','d','d']

assert dsrep([], []) == []
assert dsrep(['a'], [1]) == ['a']
assert dsrep(['a'], [0]) == []
assert dsrep(['b'], [2]) == ['b','b']
assert dsrep(['a','b'], [1,1]) == ['a','b']
assert dsrep(['a','b'], [1,2]) == ['a','b','b']
assert dsrep(['a','b'], [2,1]) == ['a','a','b']
assert dsrep(['a','b'], [2,0]) == ['a','a']
assert dsrep(['a','b'], [0,2]) == ['b','b']
assert dsrep(['a','b','c'], [1,1,1]) == ['a','b','c']
assert dsrep(['a','b','c'], [0,1,1]) == ['b','c']
assert dsrep(['a','b','c'], [0,1,0]) == ['b']
assert dsrep(['a','b','c'], [3,1,2]) == ['a','a','a','b','c','c']
assert dsrep(['a','b','c','d'], [2,5,0,3]) == ['a','a','b','b','b','b','b','d','d','d
↪']
```

Divide and conquer - Accumulator style

So far we've been creating lists at each call, thus wasting lots of memory and computation time.

Let's first try using indeces with a simple example:

Example - `dacount`

Suppose given a number `n`, you want to build a list with first numbers from `1` to `n` **without** using `range` and using ModAcc style with a divide and conquer strategy

Notie a couple of things:

1. to promote parallelism, ideally it should be possibile to execute left and right calls independently of their order: to allow for this you should first allocate all the needed space in the accumulator. Since for us it's a special variable, you can preallocate space using the `*` operator before calling the helpers.
2. since `acc` is created in the main function and keeps getting modified, we dropped the `return` statements in the helper

[13]:
```
def dacount_helper(i, j, acc):
    if i > j:
        pass
    elif i == j:
        acc[i - 1] = i
    else:
        k = (i + j) // 2
        dacount_helper(i, k, acc)     # no need to return
        dacount_helper(k+1, j, acc)   # no need to return

def dacount(n):
    acc = [0]*n  # let's create space to be filled in later
                 # (acc is special so let's allow us the freedom of using * operator)
    dacount_helper(1,n, acc)
    return acc

result = dacount(5)
print(result)
jupman.pytut()
```

```
[1, 2, 3, 4, 5]
```

[13]:
```
<IPython.core.display.HTML object>
```

Example - `dadouble`

Let's see now how we could use indeces and accumulators with our `double` function:

Take a `lst` of numbers and RETURN a NEW list with the numbers doubled, following the **ModAcc** style with a Divide and Conquer strategy

Again, notice we preallocated the accumulator in the main function and dropped the return instructions in the helper

[15]:
```
def dadouble_helper(lst, i,j,acc):
    if i >= j:    # note the >= in this case
        return acc
    else:
```

(continues on next page)

(continued from previous page)

```
        k = (i + j) // 2
        acc[k] = lst[k] * 2
        dadouble_helper(lst, i, k, acc)
        dadouble_helper(lst, k+1, j, acc)
        return acc

def dadouble(lst):
    acc = [None]*len(lst)  # let's create space to be filled in later
                           # (acc is special so let's allow us the freedom of using *␣
↪operator)
    dadouble_helper(lst, 0, len(lst), acc)
    return acc

result = dadouble([4,6,2])
print(result)

jupman.pytut()
```

```
[8, 12, 4]
```

[15]:
```
<IPython.core.display.HTML object>
```

Exercise - debug dadouble

Add a `level` parameter and some debugging indented print to the `dadouble` function.

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

[17]:
```
def dadouble_helper(lst, i,j,acc, level):
    print('  '*level, "i=", i, "j=",j, "acc=", acc)
    if i >= j:   # note the >= in this case
        return acc
    else:
        k = (i + j) // 2
        acc[k] = lst[k] * 2
        print('  '*level, "left call")
        dadouble_helper(lst, i, k, acc, level + 1)
        print('  '*level, "right call")
        dadouble_helper(lst, k+1, j, acc, level + 1)
        return acc

def dadouble(lst):
    acc = [None]*len(lst)  # let's create space to be filled in later
                           # (acc is special so let's allow us the freedom of using *␣
↪operator)
    dadouble_helper(lst, 0, len(lst), acc, 0)
    return acc


result = dadouble([4,6,2,7,3])
print("Final result=", result)
```

```
 i= 0 j= 5 acc= [None, None, None, None, None]
 left call
```

(continues on next page)

(continued from previous page)

```
   i= 0 j= 2 acc= [None, None, 4, None, None]
   left call
     i= 0 j= 1 acc= [None, 12, 4, None, None]
     left call
       i= 0 j= 0 acc= [8, 12, 4, None, None]
     right call
       i= 1 j= 1 acc= [8, 12, 4, None, None]
   right call
     i= 2 j= 2 acc= [8, 12, 4, None, None]
 right call
   i= 3 j= 5 acc= [8, 12, 4, None, None]
   left call
     i= 3 j= 4 acc= [8, 12, 4, None, 6]
     left call
       i= 3 j= 3 acc= [8, 12, 4, 14, 6]
     right call
       i= 4 j= 4 acc= [8, 12, 4, 14, 6]
   right call
     i= 5 j= 5 acc= [8, 12, 4, 14, 6]
Final result= [8, 12, 4, 14, 6]
```

[17]:

```
result = dadouble([4,6,2,7,3])
print("Final result=", result)
```

```
 i= 0 j= 5 acc= [None, None, None, None, None]
 left call
   i= 0 j= 2 acc= [None, None, 4, None, None]
   left call
     i= 0 j= 1 acc= [None, 12, 4, None, None]
     left call
       i= 0 j= 0 acc= [8, 12, 4, None, None]
     right call
       i= 1 j= 1 acc= [8, 12, 4, None, None]
   right call
     i= 2 j= 2 acc= [8, 12, 4, None, None]
 right call
   i= 3 j= 5 acc= [8, 12, 4, None, None]
   left call
     i= 3 j= 4 acc= [8, 12, 4, None, 6]
     left call
       i= 3 j= 3 acc= [8, 12, 4, 14, 6]
     right call
       i= 4 j= 4 acc= [8, 12, 4, 14, 6]
   right call
     i= 5 j= 5 acc= [8, 12, 4, 14, 6]
Final result= [8, 12, 4, 14, 6]
```

Exercise - dahist

Given a `lst`, RETURN a NEW dictionary with the counts of elements of `lst`

- **DO NOT** use search methods (`.index`, `.count`, `.remove`...)
- NOTE in this case we can start with an empty dictionary

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

[19]:

```
def dahist_helper(lst, i, j, acc):
    if i >= j:
        return acc
    else:
        k = (i + j) // 2
        if lst[k] in acc:
            acc[lst[k]] += 1
        else:
            acc[lst[k]] = 1
        res1 = dahist_helper(lst, i, k, acc)
        res2 = dahist_helper(lst, k+1, j, acc)
        return acc


def dahist(lst):

    return dahist_helper(lst, 0, len(lst), {})


result = dahist(['a','b','a','c','b','b','c','a','b','c','d'])
print(result)  # {'a':3,
               #  'b':4,
               #  'c':3,
               #  'd':1}

assert dahist([]) == {}
assert dahist(['a']) == {'a':1}
assert dahist(['a','a']) == {'a':2}
assert dahist(['a','b']) == {'a':1,
                             'b':1}
assert dahist(['b','c']) == {'c':1,
                             'b':1}
assert dahist(['a','c','a']) == {'a':2,
                                 'c':1}
assert dahist(['c','c','a']) == {'a':1,
                                 'c':2}
assert dahist(['a','b','a','c','b','b','c','a','b','c','d']) == {'a':3,
                                                                 'b':4,
                                                                 'c':3,
                                                                 'd':1}
```

```
{'b': 4, 'a': 3, 'c': 3, 'd': 1}
```

[19]:

```
def dahist(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = dahist(['a','b','a','c','b','b','c','a','b','c','d'])
print(result)  # {'a':3,
               #  'b':4,
               #  'c':3,
               #  'd':1}

assert dahist([]) == {}
assert dahist(['a']) == {'a':1}
assert dahist(['a','a']) == {'a':2}
assert dahist(['a','b']) == {'a':1,
                             'b':1}
assert dahist(['b','c']) == {'c':1,
                             'b':1}
assert dahist(['a','c','a']) == {'a':2,
                                 'c':1}
assert dahist(['c','c','a']) == {'a':1,
                                 'c':2}
assert dahist(['a','b','a','c','b','b','c','a','b','c','d']) == {'a':3,
                                                                 'b':4,
                                                                 'c':3,
                                                                 'd':1}
```

TODO will add more

More exercises

Computer Science Circles[329]: nice chapter about recursion with some exercises

W3 Resources[330]: some recursion exercises

Edabit[331]: Search by setting 'Recursion' as tag - notice many provided solutions are not actually in recursive format.

Geeks for geeks:

- questions set 2[332]
- questions set 3[333]
- questions set 4[334]
- questions set 5[335]
- questions 6[336]
- questions 7[337]

[329] https://cscircles.cemc.uwaterloo.ca/16-recursion/
[330] https://www.w3resource.com/python-exercises/data-structures-and-algorithms/python-recursion.php
[331] https://edabit.com/challenges
[332] https://www.geeksforgeeks.org/practice-questions-for-recursion-set-2/?ref=rp
[333] https://www.geeksforgeeks.org/practice-questions-for-recursion-set-3/?ref=lbp
[334] https://www.geeksforgeeks.org/practice-questions-for-recursion-set-4/?ref=rp
[335] https://www.geeksforgeeks.org/practice-questions-for-recursion-set-5/?ref=lbp
[336] https://www.geeksforgeeks.org/practice-questions-for-recursion-set-6/?ref=lbp
[337] https://www.geeksforgeeks.org/practice-questions-for-recursion-set-7/?ref=lbp

- Length of a string[338]
- Recursive programs to find min and max[339]
- Print a pattern without using any loop[340]
- Sum triangle from array[341]
- Print all n-digit strictly increasing numbers[342]
- Recursive insertion sort[343]

Continue

Go on with the challenges[344]

7.3.4 Recursion challenges

While exporing the inner mysteries of recursion, your quest for knowledge has lead you to unspeakable torments few souls ever dared to challenge.

Start solving them in SimpleFP style, then consider using ModAcc style with possibly Divide and Conquer strategy (where appropriate / feasible).

Challenge - Cthulhu

Given an `lst` with an **even** nuber of elements, RETURN a NEW list with the elements from both sides alternated

- **DO NOT** use slice stepping

[1]:

```
def cthu(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = cthu(['a','b','c','d','e','f','g','h'])
print(result)  # ['a', 'e', 'b', 'f', 'c', 'g', 'd', 'h']

assert cthu([]) == []
assert cthu(['a','b']) == ['a','b']
assert cthu(['x','y','z','w']) == ['x','z','y','w']
assert cthu(['x','y','t','z','w','g']) == ['x','z','y','w','t','g']
assert cthu(['a','b','c','d','e','f','g','h']) == ['a', 'e', 'b', 'f', 'c', 'g', 'd',
↪'h']
```

[338] https://www.geeksforgeeks.org/program-for-length-of-a-string-using-recursion/
[339] https://www.geeksforgeeks.org/recursive-programs-to-find-minimum-and-maximum-elements-of-array/?ref=rp
[340] https://www.geeksforgeeks.org/print-a-pattern-without-using-any-loop/
[341] https://www.geeksforgeeks.org/sum-triangle-from-array/
[342] https://www.geeksforgeeks.org/print-all-n-digit-strictly-increasing-numbers/
[343] https://www.geeksforgeeks.org/recursive-insertion-sort/?ref=rp
[344] https://sciprog.davidleoni.it/recursion/recursion4-chal.html

Challenge - Ghatanothoa

Given a list of characters, RETURN `True` if all `a` characters are both preceded and followed by `b` character(s).

[2]:

```
def ghat(lst):
    raise Exception('TODO IMPLEMENT ME !')


result1 = ghat(['a','b','a','b','e','b','a','b'])
print(result1) # True
result2 = ghat(['c','a','b','a','b','a','b'])
print(result2) # False

assert ghat(['b','a','b']) == True
assert ghat(['b','a','b','b']) == True
assert ghat(['b','b','a','b','b']) == True
assert ghat(['b','a','b','a']) == False
assert ghat(['c','a','b','a']) == False
assert ghat(['b','a','b','d']) == True
assert ghat(['c','b','a','b']) == True
assert ghat(['a','b','a','b']) == False
assert ghat(['b','a','b','a','b']) == True
assert ghat(['b','a','b','b','a','b']) == True
assert ghat(['b','a','b','b','c','b','a','b','d']) == True
assert ghat(['c','a','b','a','b','a','b']) == False
```

Challenge - Hastur

Given an `lst` alternating characters and sequences of dashes `-`, RETURN a NEW list of characters where dashes are replaced by the characters preceding them.

- assume `lst` never begins with a character which is a dash

[3]:

```
def hastur(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = hastur(['a','-','-','-','b','-','-','c','-','-','-','-'])
print(result)  #  ['a', 'a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'c', 'c', 'c']

assert hastur([]) == []
assert hastur(['a']) == ['a']
assert hastur(['a','-']) == ['a','a']
assert hastur(['a','-','b']) == ['a','a','b']
assert hastur(['a','-','-']) == ['a','a','a']
assert hastur(['a','b','-']) == ['a','b','b']
assert hastur(['c','-','-','a']) == ['c','c','c','a']
assert hastur(['c','-','-','-','b','-','a','-']) == ['c','c','c','c','b','b','a','a']
assert hastur(['a','-','-','-','b','-','-','c','-','-','-','-']) == ['a', 'a', 'a', 'a
↪', 'b', 'b', 'b', 'c', 'c', 'c', 'c', 'c']
```

Challenge - Tsathoggua

Given a list of characters, RETURN the **length** of longest sequence of contiguous equal characters

- **DO NOT** use `.count()` method
- **DO NOT** use dictionaries

[4]:
```
def tsat(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = tsat(['a','a','a','b','b','c','c','c','c','c','e','e','e','f','f','f','f'])
print(result)  # 5

assert tsat([]) == 0
assert tsat(['a']) == 1
assert tsat(['a','b']) == 1
assert tsat(['a','b','b']) == 2
assert tsat(['a','a','b']) == 2
assert tsat(['a','b','c']) == 1
assert tsat(['c','c','a']) == 2
assert tsat(['c','c','c']) == 3
assert tsat(['c','c','a','a','a','b','b']) == 3
assert tsat(['c','c','a','a','a','a','b','b']) == 4
assert tsat(['d','e','c','c','c','c','c','a','b','b']) == 5
assert tsat(['e','e','e','e','b','b','b','b']) == 4
assert tsat(['g','e','a','b','b','c','c']) == 2
```

Challenge - Yig

Given lists of numbers `la` and `lb` of equal size, RETURN a NEW list with the pairwise greatest numbers

[5]:
```
def yig(la, lb):
    raise Exception('TODO IMPLEMENT ME !')

result = yig([5,2,7,2,4,8], [9,4,3,5,2,5])
print(result)  # [9,4,7,5,4,8]

assert yig([], []) == []
assert yig([3], [4]) == [4]
assert yig([5], [2]) == [5]
assert yig([0], [0]) == [0]
assert yig([0,4], [7,3]) == [7,4]
assert yig([9,4], [7,8]) == [9,8]
assert yig([5,2,7,2,4,8], [9,4,3,5,2,5]) == [9,4,7,5,4,8]
```

Challenge - Shub-Niggurath

Given a list `la` and a list of numbers `lb` such that the `sum(lb) == len(la)`, RETURN a NEW list of lists in which elements of `la` are grouped according to the sizes expressed in `lb`.

[6]:
```
def shub(la, lb):
    raise Exception('TODO IMPLEMENT ME !')

result = shub([5,9,6,2,3,1,4,7,5,2], [1,4,2,3])
print(result) # [[5],[9,6,2,3],[1,4],[7,5,2]]

assert shub([],[]) == []
assert shub([3],[1]) == [[3]]
assert shub([3,4],[2]) == [[3,4]]
assert shub([5,8],[1,1]) == [[5],[8]]
assert shub([5,8,6],[1,2]) == [[5],[8,6]]
assert shub([5,8,6],[2,1]) == [[5,8],[6]]
assert shub([5,8,6],[1,1,1]) == [[5],[8],[6]]
assert shub([5,8,6,0],[1,1,2]) == [[5],[8],[6,0]]
assert shub([5,8,6,0],[2,2]) == [[5,8],[6,0]]
assert shub([7,4,6,2,9,4,2,1],[3,5]) == [[7,4,6],[2,9,4,2,1]]
assert shub([5,9,6,2,3,1,4,7,5,2], [1,4,2,3]) == [[5],[9,6,2,3],[1,4],[7,5,2]]
```

Challenge - Shoggoth

Given a list of progressively nested lists which ultimately always contain at most two characters, RETURN a NEW list with all the characters

[7]:
```
def sho(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = sho(['s',['h',['o',['g','g'],'o'],'t'],'h'])
print(result)  # ['s','h','o','g','g','o','t','h']

assert sho([]) == []
assert sho(['a','a']) == ['a','a']
assert sho(['a',['b','c'],'d']) == ['a','b','c','d']
assert sho(['a',['b',['z'],'c'],'d']) == ['a','b','z','c','d']
assert sho(['a',['b',['x','y'],'c'],'d']) == ['a','b','x','y','c','d']
assert sho(['a',['b',['x',['y'],'z'],'c'],'d']) == ['a','b','x','y','z','c','d']
assert sho(['s',['h',['o',['g','g'],'o'],'t'],'h']) == ['s','h','o','g','g','o','t','h
↪']
```

Challenge - Azathoth

Given a list of arbitrarily nested lists which eventually contain strings, RETURN a NEW list of strings

HINT: you need double recursion, both in first element depth and in following elements in the sequence

[8]:
```
def aza(lst):
    raise Exception('TODO IMPLEMENT ME !')

result = aza([[['a']],[['z'],['a'],[['t','h']]],['o','t','h']])
print(result) # ['a','z','a','t','h','o','t','h']

assert aza([]) == []
assert aza(['a']) == ['a']
assert aza([['b']]) == ['b']
assert aza([[['c']]]) == ['c']
assert aza(['x','y']) == ['x','y']
assert aza([['x'],'y']) == ['x','y']
assert aza(['x',['y']]) == ['x','y']
assert aza([['a'],['x']]) == ['a','x']
assert aza([['a'],[['x']]]) == ['a','x']
assert aza(['q','e','d']) == ['q','e','d']
assert aza(['q',['e','d']]) == ['q','e','d']
assert aza([['q','e'],'d']) == ['q','e','d']
assert aza([['q','e','d']]) == ['q','e','d']
assert aza([[['q'],'e',['d']]]) == ['q','e','d']
assert aza([[['a']],[['z'],['a'],[['t','h']]],['o','t','h']]) == ['a','z','a','t','h',
↪'o','t','h']
```

References

- *Unaussprechlichen Kulten*, by Friedrich von Junzt
- *The Pnakotic Manuscripts*, unknown authors

7.4 Algorithm analysis

7.4.1 Introduction

Life's too easy so let's complicate it, shall we?

Are you ready to fight the invisible enemy of computational complexity? If not, please read the references first:

- Erik Dassi's algorithms slides 1 and 2
- Python DS Chapter 2.6: Algorithm analysis[345]

[345] https://runestone.academy/runestone/books/published/pythonds/AlgorithmAnalysis/toctree.html

7.4.2 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 Python DS book Chapter 2.6: Lists[346]:

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	$O(nk)$
remove	$O(n)$	min	$O(n)$
iteration	$O(n)$	max	$O(n)$
len(lst)	$O(1)$	sum	$O(n)$
equality	$O(n)$		

Fast or not?

Given a list `x` of n characters, for each of the following expressions, try giving the complexity:

1. ```
   x[n // 2]
   ```
2. ```
   x[n // 2] = 'd'
   ```
3. ```
   x.append('c')
   ```
4. ```
   x.insert(0, 'd')
   ```
5. ```
   x + x
   ```
6. ```
   x[3:5]
   ```
7. ```
   x.sort()
   ```
8. ```
   len(x)
   ```
9. ```
   x.pop(1)
   ```
10. ```
    x.pop(-1)
    ```
11. ```
    'z' in x
    ```
12. ```
    x.extend(1000)
    ```

[346] https://runestone.academy/runestone/books/published/pythonds/AlgorithmAnalysis/Lists.html