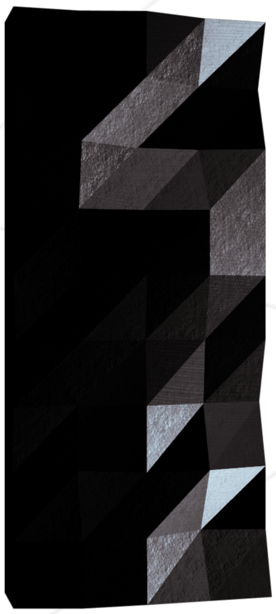

Module 02 – Piscine Python for Data Science

Intro to Python: OOP skills

Summary: Today we will help you acquire a basic knowledge of the OOP approach in Python.

Contents

I Foreword 2

II Instructions 3

III Specific instructions of the day 4

IV Exercise 00 : Simple class 5

V Exercise 01 : Method 7

VI Exercise 02 : Constructor 8

VII Exercise 03 : Nested class 10

VIII Exercise 04 : Inheritance 12

IX Exercise 05 : Config and the main program 13

X Exercise 06 : Logging 15

Chapter I

Foreword

A common complaint to data scientists is that they write shitcode (by the way, only for educational purposes you may find a lot of examples of Python shitcode here, provided strictly for educational purposes). Why? Because the average data scientist uses a lot of inefficient techniques and hard coded variables and neglects object-oriented programming. Do not be like them.

Here are the top few examples from the website mentioned above:

- How to get the absolute value in just 6 lines of python

```
def absolute\_value(value):
    if str(value)[0]=='-':
        value = -1 * value
        return value
    else:
        return value
```

- How to evaluate the factorial of 40000 in approximately 1 second:

```
for module in next\_possible\_modules:
    import math; math.factorial(40000) # approx. a 1 second operation
    end\_time = start\_time + timedelta(minutes=module.duration)
```

- Gotta check that date

```
if (SelectionAndTimeData[1] < 2000 or \
        SelectionAndTimeData[2] < 1 or SelectionAndTimeData[2] > 12 or \
        SelectionAndTimeData[3] < 1 or SelectionAndTimeData[3] > 31 or \
        SelectionAndTimeData[4] < 0 or SelectionAndTimeData[4] > 24 or \
        SelectionAndTimeData[5] < 0 or SelectionAndTimeData[5] > 60 or \
        SelectionAndTimeData[2] < 0 or SelectionAndTimeData[2] >60):

    print('*****************************************************************')
    print(' Entered date is not valid ')
    print('*****************************************************************')
```

Chapter II

Instructions

- Use this page as your only reference. Do not listen to any rumors or speculations about how to prepare your solution.
- Here and further on we use Python 3 as the only correct version of Python.
- The solutions for python exercises (module01, module02, module03) must include the following block in the end: if __name__ == '__main__'.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave any additional files in your directory other than those explicitly specified in the subject. It is recommended that you modify your .gitignore to avoid any accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. If that fails, try your neighbor on the left.
- Your reference material: peers / Internet / Google.
- You can ask questions in Slack.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!

Chapter III

Specific instructions of the day

- No code in the global scope. Use functions!
- Each file must end with a function call in a condition similar to:

```
if __name__ == \'__main__\':
    # your tests and your error handling
```

- Any exception not caught will invalidate your work, even in the event of an error that you were asked to test.
- No imports are allowed, except those explicitly mentioned in the section "Authorized functions" of the title block of each exercise.
- Any built-in function is allowed.

Chapter IV

Exercise 00 : Simple class

	Exercise 00
Simple class	
Turn-in directory : *ex*00/	
Files to turn in : first_class.py	
Allowed functions : all imports are restricted	

This is going to be an easy warm-up exercise to get you started with object-oriented programming in Python.

- Create a python script called first_class.py that contains a class called Must_read. It does the only thing reads the file data.csv and prints it. You can hardcode the name of the csv file inside the class. Put print() inside your class (you will learn about methods and constructors later, forget about them in this exercise).
- data.csv contains the following data (you can create the file any way you want):

```
head,tail
0,1
1,0
0,1
1,0
0,1
0,1
0,1
1,0
1,0
0,1
1,0
```

Example of launching the script:

```
$ python3 first_class.py
head,tail
0,1
1,0
0,1
1,0
0,1
0,1
```

```
0,1
1,0
1,0
0,1
1,0
```

Chapter V

Exercise 01 : Method

	Exercise 01
Method	
Turn-in directory : *ex*01/	
Files to turn in : first_method.py	
Allowed functions : all imports are is restricted	

In the previous exercise, you managed to create a class. To be honest, nobody creates such classes in real life. Classes usually help to unite different functions with a common topic and common parameters. That is a better way to organize them. In this case, functions are called methods.

- In this exercise you need to move the code from the body of the class to the method of that class with the name file_reader(). Methods are like functions - they can return something. Classes are unable to do that. So you need to replace print() with return() in the method. Change the name of the class to Research.
- The script still must have the exact same behavior. It needs to display the content of the file data.csv. Save the script with the name first_method.py.

Chapter VI

Exercise 02 : Constructor

Exercise 02
Constructor
Turn-in directory : *ex*02/
Files to turn in : first_constructor.py
Allowed functions : import sys, import os

It was not a very good idea to hardcode the name of the file in the method. It would be great if we could give the path to the file as a parameter of the script. It would be great if we did not have to put the path in every method that we bring in later. There is a solution. There can be a constructor in python classes: __init__(). It is the method that runs first when the instance of a class is instantiated.

Modify your code in the following way:

- Inside the class Research create an __init__() method that takes the path to the file that needs to be read as an argument.
- Modify the method file_reader(). This method does almost the same thing as in the previous exercise - just reads the file and returns its data. The difference is that the path to the file should be used from the __init__() method.
- If a file with a different structure was given, and your program cannot read it, raise an exception. The correct file contains a header with two strings delimited by a comma. There are one or more lines after that that contain either 0 or 1 and never both of them delimited by a comma.
- Modify the main program. The script must still have the exact same behavior. The path to the file should be given as an argument to the script. It needs to display the content of the file data.csv. Save the script with the name first_constructor.py.

Example of launching the script:

```
$ python3 first_class.py data.csv
head,tail
0,1
1,0
```

```
0,1
1,0
0,1
0,1
0,1
1,0
1,0
0,1
1,0
```

Chapter VII

Exercise 03 : Nested class

	Exercise 03
Nested class	
Turn-in directory : *ex*03/	
Files to turn in : first_nest.py	
Allowed functions : import sys, import os	

Let us go further with OOP in Python. Can a class be inside another class? Sure, why not? We can still benefit from it by giving our code a clearer structure by uniting several methods in one nested class.

What you need to do in this exercise:

- Modify the file_reader() method by adding one more argument has_header with the default value True. You should use it if your file has a header, if it is not - it should be False. The return of this method in this exercise is not a string anymore but a list of lists [0, 1] or [1, 0]. So the argument has_header influences the logic of how to process the file. In both cases, the return should be the same without a header.

- Create a nested class called Calculations without a constructor. In that class, create two methods: counts() and fractions(). The method counts() takes data from file_reader() as an argument and returns the count of heads and tails, for example, 3 and 7. The method fractions() takes counts of head and tails as arguments and calculates fractions in percents, for example, 30 and 70.

- The script should display:

    - the data from file_reader()
    - the counts from counts()
    - the fractions from fractions()

Here is an example:

```
$ python3 first \_nest.py data.csv
[[0, 1], [1, 0], [0, 1], [1, 0], [0, 1], [0, 1], [0, 1], [1, 0], [1, 0], [0, 1], [1, 0], [0, 1]]
5 7
41.66666666666667 58.333333333333336
```

Chapter VIII

Exercise 04 : Inheritance

	Exercise 04
Inheritance	
Turn-in directory : *ex*04/	
Files to turn in : first_child.py	
Allowed functions : import sys, from random import randint	

You have one class with many useful methods and you need another class with all or some of those methods? No problem! Inherit one from the other.

What you need to do in this exercise:

- In the previous exercise, you had the argument data in your method counts(). Let us move it to the constructor of the class Calculations. The same data might be useful for the future methods of this class, right?
- Create a new class called Analytics, inherited from Calculations.
- In the new class, create two methods:
  - predict_random() that takes the number of predictions that it should return and returns a list of lists of predicted observations of heads and tails: if heads equals 1, then tails equals 0 and vice versa: [[1, 0], [1, 0], [0, 1]]
  - predict_last() that just returns the last item of the data from file_reader() (this method has the same functionality as in the previous exercise), it should be a list.
- The script should display:
  - the data from file_reader()
  - the counts from counts()
  - the fractions from fractions()
  - the list of lists from predict_random() for the 3 steps
  - the list from predict_last()

Chapter IX

Exercise 05 : Config and the main program

	Exercise 05
Config and the main program	
Turn-in directory : *ex*05/	
Files to turn in : config.py, analytics.py, make_report.py	
Allowed functions : import os, from random import randint	

Ok. Now we need to make our code even clearer. We need to transfer all the logic of the script into a different file. And the second thing we need to do is move all the parameters into a config file. We will import the config file and our module file into the main program script.

The same things in detail:

- create a file called config.py where you will store all the external parameters like num_of_steps for predict_random()
- delete the logic after block if __name__ == '__main__' from your script from the previous exercise,
- rename that script analytics.py
- add to the class Analytics a method that saves any given result to a file with a given extension like save_file(data, name of file, 'txt')
- create a new file called make_report.py where the whole logic of your program will be written, the result saved in the file should look like this (you may need additional methods to add to analytics.py):

```
Report
We have made 12 observations from tossing a coin: 5 of them were tails and 7 of
them were heads. The probabilities are 41.67\% and 58.33\%, respectively. Our
forecast is that in the next 3 observations we will have: 1 tail and 2 heads.
```

The template of the text must be stored in config.py.

In this exercise config.py may have code in the global scope (for variables).

In this exercise config.py and analytics.py do not have to contain the block if __name__ == '__main__'.

Chapter X

Exercise 06 : Logging

	Exercise 06
Logging	
Turn-in directory : *ex*06/	
Files to turn in : config.py, analytics.py, make_report.py	
Allowed functions : import os, from random import randint, import logging, import requests (or urllib), import json	

- By now you have written your own module containing several classes which contain several methods, a program that uses that module and a config file. But what if there are some problems during production that you will need to debug? How are you going to do it? That is right! You need to log it. So the first task of the exercise is to each and every method in all the classes should log useful information for debugging. You need to store this in the file analytics.log. The format is a date, time, and a message delimited by a space:

  ```
  2020-05-01 22:16:16,877 Calculating the counts of heads and tails
  ```

- The second task is to write a method in the Research class that sends a message to a Slack channel using webhooks. The message should contain: “The report has been successfully created” or “The report hasn't been created due to an error”. Yeah, we know that you do not have admin rights to create a custom integration in the School workspace, but be creative, create your own Slack workspace!
- In this exercise, config.py may have code in the global scope (for variables).
- In this exercise, config.py and analytics.py do not have to contain the block if __name__ == '__main__'.