Day 02 – Piscine Python for Data Science

Intro to Python: OOP skills

Summary: This day will help you to get the basic knowledge about OOP approach in Python.

Contents

Chapter I

Preamble

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). Why? Because an average data scientist uses a lot of inefficient techniques, hard coded variables, and neglects object-oriented programming. Do not be like them.

Just the top examples from the website mentioned above:

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

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

- How to evaluate 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 the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Here and further we use `Python 3` as the only correct version of Python.
- The python files for python exercises (d01, d02, d03) must have a 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 in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid 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. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google.
- Remember to discuss on the Intra Piscine forum.
- 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 be ended by a function call in a condition similar to:

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

- Any exception not caught will invalidate the work, even in the event of an error that was asked you to test.
- No imports allowed, except those explicitly mentioned in the section “Authorized functions” of the title block of each exercise.
- The interpreter to use is `Python 3`.
- Any built-in function is allowed.

Chapter IV

Exercise 00

Exercise 00 : Simple class
Directory to store your solution : `ex00/`
Files to be in the directory : `first_class.py`
Authorized functions : `any import is restricted`

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

Create a python script `first_class.py` that contains a class `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
```

The 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

Exercise 01 : Method
Directory to store your solution : `ex01/`
Files to be in the directory : `first_method.py`
Authorized functions : `any import 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 get together different functions united by a common topic and common parameters. That is a better way to organize them. In this case, functions are called methods.

In the 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

Exercise 02 : Constructor
Directory to store your solution : `ex02/`
Files to be in the directory : `first_constructor.py`
Authorized 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 to the script. It would be great, if we would not have to put the path in every method that we come with later. There is a solution. In python classes may have a constructor `__init__()`. It is a method that runs first when the instance of a class is instantiated.

Modify your code in the following way:

- Inside the class `Research` create a method `__init__()` that takes as an argument the path to the file that needs to be read.
- 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 (`split(',')` should work with the file, `splitlines()` should work with the file).
- Modify the main program. 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_constructor.py`.

Chapter VII

Exercise 03

Exercise 03 : Nested class
Directory to store your solution : `ex03/`
Files to be in the directory : `first_nest.py`
Authorized 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 a clearer structure of our code by uniting several methods in one nested class.

What you need to do in this exercise:

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

The example is below:

```
$ 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

Exercise 04 : Inheritance
Directory to store your solution : `ex04/`
Files to be in the directory : `first_child.py`
Authorized 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 the class, right?
- Create a new class `Analytics` inherited from `Calculations`.
- In the new class create two methods:
    - `predict_random()` that takes as an argument the number of predictions that it should return and returns a list of lists of predicted observations of heads and tails: if heads equal 1, then tails equal 0 and vice versa: `[[1, 0], [1, 0], [0, 1]]`
    - `predict_last()` that just returns the last item of the data from `file_reader()`, 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 3 steps,
    - the list from `predict_last()`.

Chapter IX

Exercise 05

Exercise 05 : Config and the main program
Directory to store your solution : `ex05/`
Files to be in the directory : `config.py, analytics.py, make_report.py`
Authorized 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 in a different file. And the second thing is we need to move all the parameters in a config file. In the main program script, we will import the config file and our module file.

The same things in details:

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

```
Report
We have made 12 observations of 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.
```

Chapter X

Exercise 06

Exercise 06 : Logging
Directory to store your solution : `ex06/`
Files to be in the directory : `config.py, analytics.py, make_report.py`
Authorized functions : `import os, from random import randint, import logging, import requests (or urllib), import json`

By now you wrote your own module containing several classes containing several methods, a program that uses that module and a config file. But what if during the production there will be some problems 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: each and every method in all the classes should log useful information for debugging. You need to store it 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. Write a method in 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!