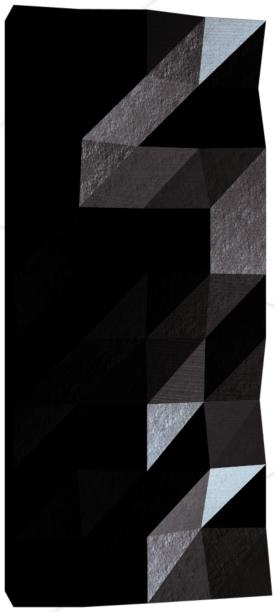

Training Piscine Python for datascience - 4

Data Oriented Design

Summary: Today, you will see some Structure Design.

Version: 1.00

Contents

I	**General rules**	**2**
II	**Specific instructions of the day**	**3**
III	**Exercise 00**	**4**
IV	**Exercise 01**	**6**
V	**Exercise 02**	**8**
VI	**Exercise 03**	**10**
VII	**Submission and peer-evaluation**	**12**

Chapter I

General rules

- You have to render your modules from a computer in the cluster either using a virtual machine:
    - You can choose the operating system to use for your virtual machine
    - Your virtual machine must have all the necessary software to realize your project. This software must be configured and installed.
- Or you can use the computer directly in case the tools are available.
    - Make sure you have the space on your session to install what you need for all the modules (use the goinfre if your campus has it)
    - You must have everything installed before the evaluations
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc) apart from undefined behaviors. If this happens, your project will be considered non functional and will receive a `0` during the evaluation.
- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded. If Deepthought is assigned to grade your work, it will be done after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.
- You must use the Python 3.10 version
- Your lib imports must be explicit, for example you must "import numpy as np". Importing "from pandas import *" is not allowed, and you will get 0 on the exercise.
- There is no global variable.
- By Odin, by Thor ! Use your brain !!!

Chapter II

Specific instructions of the day

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.

- No code in the global scope. Use functions!
- Each program must have its main and not be a simple script:

```
def main():
  # your tests and your error handling

if __name__ == "__main__":
  main()
```

- Any exception not caught will invalidate the exercices, even in the event of an error that you were asked to test.
- You can use any built-in function if it is not prohibited in the exercise.
- All your functions, class and method must have a documentation (___doc___)
- Your code must be at the norm
  - pip install flake8
  - alias norminette=flake8

Chapter III

Exercise 00

	Exercise 00
Exercice 00: Calculate my statistics	
Turn-in directory : *ex*00/	
Files to turn in : `statistics.py`	
Allowed functions : `None`	

You must take in *args a quantity of unknown number and make the Mean, Median, Quartile (25% and 75%), Standard Deviation and Variance according to the **kwargs ask.
You have to manage the errors.
The prototype of function is:

```
def ft_statistics(*args: Any, **kwargs: Any) -> None:
 #your code here
```

Your tester.py:

```
from statistics import ft_statistics

ft_statistics(1, 42, 360, 11, 64, toto="mean", tutu="median", tata="quartile")
print("-----")
ft_statistics(5, 75, 450, 18, 597, 27474, 48575, hello="std", world="var")
print("-----")
ft_statistics(5, 75, 450, 18, 597, 27474, 48575, ejfhhe="heheh", ejdjdejn="kdekem")
print("-----")
ft_statistics(toto="mean", tutu="median", tata="quartile")
```

Expected output:

```
$> python tester.py
mean : 95.6
median : 42
quartile : [11.0, 64.0]
-----
std : 17982.70124086944
var : 323377543.9183673
-----
-----
ERROR
ERROR
ERROR
$>
```

Chapter IV

Exercise 01

	Exercise 01
Exercice 01: Outer_inner	
Turn-in directory : $ex01/$	
Files to turn in : `in_out.py`	
Allowed functions : `None`	

Write a function that returns the square of argument, a function that returns the Exponentiation of argument by himself and a function that takes as argument a number and a function, it returns an object that when called returns the result of the arguments calculation.
The prototype of functions is:

```
def square(x: int | float) -> int | float:
 #your code here

def pow(x: int | float) -> int | float:
 #your code here

def outer(x: int | float, function) -> object:
          count = 0
    def inner() -> float:
     #your code here
```

Your tester.py:

```
from in_out import outer
from in_out import square
from in_out import pow

my_counter = outer(3, square)
print(my_counter())
print(my_counter())
print(my_counter())
print("---")
another_counter = outer(1.5, pow)
print(another_counter())
print(another_counter())
print(another_counter())
```

Expected output:

```
$> python tester.py
9
81
6561
---
1.8371173070873836
3.056683336818703
30.42684786675409
$>
```

We remind you that the use of global is forbidden

Chapter V

Exercise 02

	Exercise 02
Exercice 02: my first decorating	
Turn-in directory : $ex02/$	
Files to turn in : `callLimit.py`	
Allowed functions : `None`	

Write a function that takes as argument a call limit of another function and blocks its execution above a limit.
The prototype of functions is:

```
def callLimit(limit: int):
    count = 0
    def callLimiter(function):
        def limit_function(*args: Any, **kwds: Any):
         #your code here
```

Your tester.py:

```
from callLimit import callLimit

callLimit(3)
def f():
    print ("f()")

callLimit(1)
def g():
    print ("g()")

for i in range(3):
    f()
    g()
```

Expected output:

```
$> python tester.py
f()
g()
f()
Error: <function g at 0x7fabdc243ee0> call too many times
f()
Error: <function g at 0x7fabdc243ee0> call too many times
$>
```

Chapter VI

Exercise 03

	Exercise 03
Exercice 03: data class	
Turn-in directory : $ex03/$	
Files to turn in : `new_student.py`	
Allowed functions : `dataclasses, random, string`	

Write a dataclass that takes as arguments a name and nickname, set active to True, create the student login, and generate a random ID with the generate_id function.
You must not use ___str___ , ___repr___ in your class.
The prototype of function and class is:

```
import random
import string
from dataclasses import dataclass, field

def generate_id() -> str:
        return "".join(random.choices(string.ascii_lowercase, k = 15))

dataclass
class Student:
 #your code here
```

Your tester.py:

```
from new_student import Student

student = Student(name = "Edward", surname = "agle")
print(student)
```

Expected output: (id is random)

```
$> python tester.py
Student(name='Edward', surname='agle', active=True, login='Eagle', id='trannxhndgtolvh')
$>
```

The login and id should not be initializable and must return an error.

Your tester.py:

```
from new_student import Student

student = Student(name = "Edward", surname = "agle", id = "toto")
print(student)
```

Expected output:

```
$> python tester.py
...
TypeError: Student.__init__() got an unexpected keyword argument 'id'
$>
```

Chapter VII

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

`The evaluation process will happen on the computer of the evaluated group.`