Training Piscine Python for Data Science - 0

Starting

Summary: Today, you will learn the basics of the Python programming language.

Version: 1.3

Contents

Chapter I

General Rules

- You must submit your modules from a computer in the cluster or using a virtual machine:
  - You can choose the operating system for your virtual machine.
  - Your virtual machine must have all the necessary software to complete your project. This software must be installed and properly configured.
- Alternatively, you can use the cluster computers directly if the necessary tools are available.
  - Ensure that you have enough space in your session to install all required dependencies for the modules (use `goinfre` if your campus provides it).
  - Everything must be installed before the evaluations.
- Your functions must not terminate unexpectedly (segmentation fault, bus error, double free, etc.), except in cases of undefined behavior. If such an issue occurs, 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 these tests **do not need to be submitted and will not be graded**. They will help you easily test your work and your peers' work. These tests will be particularly useful during your defense. During the defense, you are free to use your own tests and/or those 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 occur after peer evaluations. If an error occurs in any section of your work during Deepthought's grading, the evaluation will be terminated.
- You must use Python version 3.10.
- You may use any built-in function unless explicitly prohibited in the exercise.
- Your library imports must be explicit. For example, you must use `import numpy as np`. Importing libraries using `from pandas import *` is not allowed and will result in a score of `0` for the exercise.

- Global variables are not allowed.
- By Odin, by Thor! Use your brain!!!

Chapter II

Exercise 00

	Exercise 00
Exercice 00: First python script	
Turn-in directory: $ex00/$	
Files to turn in: `Hello.py`	
Allowed functions: `None`	

You need to modify the string of each data object to display the following greetings: "Hello World", "Hello «country of your campus»", "Hello «city of your campus»", "Hello «name of your campus»"

```
ft_list  = ["Hello", "tata!"]
ft_tuple = ("Hello", "toto!")
ft_set   = {"Hello", "tutu!"}
ft_dict  = {"Hello" : "titi!"}

#your code here

print(ft_list)
print(ft_tuple)
print(ft_set)
print(ft_dict)
```

Expected output:

```
$>python Hello.py | cat -e
['Hello', 'World!']$
('Hello', 'France!')$
{'Hello', 'Paris!'}$
{'Hello': '42Paris!'}$
$>
```

Chapter III

Exercise 01

	Exercise 01
Exercice 01: First use of package	
Turn-in directory: $ex01/$	
Files to turn in: `format_ft_time.py`	
Allowed functions: `time, datetime or any other library that allows to receive the date`	

Write a script that formats the dates this way. Of course, your date will not be the same as mine, as in the example, but it must be formatted in the same way.

Expected output:

```
$>python format_ft_time.py | cat -e
Seconds since January 1, 1970: 1,666,355,857.3622 or 1.67e+09 in scientific notation$
Oct 21 2022$
$>
```

Chapter IV

Exercise 02

	Exercise 02
Exercice 02: First function python	
Turn-in directory: $ex02/$	
Files to turn in: `find_ft_type.py`	
Allowed functions: `None`	

Write a function that prints the object types and returns 42.

Here's how it should be prototyped:

```
def all_thing_is_obj(object: any) -> int:
        #your code here
```

Your tester.py:

```
from find_ft_type import all_thing_is_obj

ft_list  = ["Hello", "tata!"]
ft_tuple = ("Hello", "toto!")
ft_set   = {"Hello", "tutu!"}
ft_dict  = {"Hello" : "titi!"}

all_thing_is_obj(ft_list)
all_thing_is_obj(ft_tuple)
all_thing_is_obj(ft_set)
all_thing_is_obj(ft_dict)
all_thing_is_obj("Brian")
all_thing_is_obj("Toto")
print(all_thing_is_obj(10))
```

Expected output:

```
$>python tester.py | cat -e
List : <class 'list'>$
Tuple : <class 'tuple'>$
Set : <class 'set'>$
Dict : <class 'dict'>$
Brian is in the kitchen : <class 'str'>$
Toto is in the kitchen : <class 'str'>$
Type not found$
42$
$>
```

Running your function alone does nothing.

Expected output:

```
$>python find_ft_type.py | cat -e
$>
```

Chapter V

Exercise 03

	Exercise 03
Exercice 03: NULL not found	
Turn-in directory: $ex03/$	
Files to turn in: `NULL_not_found.py`	
Allowed functions: `None`	

Write a function that prints the object type of all types of "Null".
Return 0 if it goes well and 1 in case of error.
Your function needs to print all types of "Null".

Here's how it should be prototyped:

```
def NULL_not_found(object: any) -> int:
 #your code here
```

Your tester.py:

```
from NULL_not_found import NULL_not_found

Nothing = None
Garlic  = float("NaN")
Zero    = 0
Empty   = ""
Fake    = False

NULL_not_found(Nothing)
NULL_not_found(Garlic)
NULL_not_found(Zero)
NULL_not_found(Empty)
NULL_not_found(Fake)
print(NULL_not_found("Brian"))
```

Expected output:

```
$>python tester.py | cat -e
Nothing: None <class 'NoneType'>$
Cheese: nan <class 'float'>$
Zero: 0 <class 'int'>$
Empty: <class 'str'>$
Fake: False <class 'bool'>$
Type not Found$
1$
$>
```

Running your function alone does nothing.

Expected output:

```
$>python NULL_not_found.py | cat -e
$>
```

Chapter VI

Exercise 04

	Exercise 04
Exercice 04: The Even and the Odd	
Turn-in directory: $ex04/$	
Files to turn in: `whatis.py`	
Allowed functions: `sys or any other library that allows to receive the args`	

Create a script that takes a number as an argument, checks whether it is odd or even, and prints the result.

If more than one argument is provided or if the argument is not an integer, print an **AssertionError**.

Expected output:

```
$> python whatis.py 14
I'm Even.
$>
$> python whatis.py -5
I'm Odd.
$>
$> python whatis.py
$>
$> python whatis.py 0
I'm Even.
$>
$> python whatis.py Hi!
AssertionError: argument is not an integer
$>
$> python whatis.py 13 5
AssertionError: more than one argument is provided
$>
```