Mandatory Part

Error Management

Carry out AT LEAST the following tests to try to stress the error management

- The repository isn't empty.
- No cheating.
- No forbidden function/library.
- There is no global variable.
- The executable is named as expected.
- Norminette shows no errors. (pip install flake8, alias norminette=flake8, use flag Norme)
- 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.)
- If an exercise is wrong, go to the next one.

Yes | No

```
yetay@u90z04s03 py4 % ls
ex00    ex01    ex02    ex03
yetay@u90z04s03 py4 % flake8
zsh: command not found: flake8
yetay@u90z04s03 py4 % . ~/.py310_init
(py310) yetay@u90z04s03 py4 % flake8
(py310) yetay@u90z04s03 py4 % find * -name "*.py" -exec grep -Hw import {} \;
ex03/new_student.py:import random
ex03/new_student.py:import string
ex03/new_student.py:from dataclasses import dataclass, field
(py310) yetay@u90z04s03 py4 %
```

ex00 Calculate my statistics

The function must take as argument an unknown list of numbers and return the mean, median, Quartile (25% and 75%) Standard Deviation or Variance according to the **kwargs asked.

Your script tester:

```
from statistics import ft_statistics

ft_statistics(42, 115, 30, 151, 6400, toto="mean", tutu="std", tata="quartil
e", titi="djdj")
print("-----")
ft_statistics(51, 735, 455, 8, 7, 74, 75, hello="median", world="var")
print("-----")
ft_statistics(5, 75, 450, 18, 597, 27474, 48575, ejfhhe='fdff', ejdjdejn="dem
iane")
print("-----")
ft_statistics(hello="std", world="median")
```

expected output:

```
$> python tester.py
mean : 1347.6
std : 2526.600926145639
quartile : [42.0, 151.0]
-----
median : 74
var : 68437.3469387755
-----
-----
ERROR
ERROR
```

Yes No

```
(py310) yetay@u90z04s03 py4 % cd ex00
(py310) yetay@u90z04s03 ex00 % cat > tester_eval.py
from statistics import ft_statistics

ft_statistics(42, 115, 30, 151, 6400, toto="mean", tutu="std", tata="quartile", titi
="djdj")
print("-----")
ft_statistics(51, 735, 455, 8, 7, 74, 75, hello="median", world="var")
print("-----")
ft_statistics(5, 75, 450, 18, 597, 27474, 48575, ejfhhe='fdff', ejdjdejn="demiane")
print("-----")
ft_statistics(hello="std", world="median")
(py310) yetay@u90z04s03 ex00 % cat > tester_eval.out
mean : 1347.6
std : 2526.600926145639
quartile : [42.0, 151.0]
-----
median : 74
var : 68437.3469387755
-----
-----
ERROR
ERROR
(py310) yetay@u90z04s03 ex00 % diff tester_eval.out <(python tester_eval.py)
1,3c1,3
< mean : 1347.6
< std : 2526.600926145639
< quartile : [42.0, 151.0]
---
> mean          : 1347.6
> std           : 2526.600926145639
> quartile      : [42.0, 151.0]
5,6c5,6
< median : 74
< var : 68437.3469387755
---
> median                : 74
> var           : 68437.3469387755
(py310) yetay@u90z04s03 ex00 %
```

The differences between the expected output and the submission output is spaces, which can be disregarded.

ex01 Outer inner

The function must take as argument a number and a function to execute,
then execute the function with the number kept in memory
You should only give valid calculations, the goal of the exercise is to
understand the closures, not to handle errors.

Your script tester:

```
from in_out import outer, square, pow

def cube(x: int | float) -> int | float:
    """Returns the cube of x"""
    return (x * x * x)

my_counter = outer(6, square)
print(my_counter())
print(my_counter())
print(my_counter())
print("---")
another_counter = outer(1.6, pow)
print(another_counter())
print(another_counter())
print(another_counter())
print("---")
again_another_counter = outer(4, cube)
print(again_another_counter())
print(again_another_counter())
print(again_another_counter())
```

expected output:

```
$> python tester.py
36.0
1296.0
1679616.0
---
2.1212505710975917
4.929279084950403
2599.697171916239
---
64
262144
18014398509481984
```

Yes No

```
(py310) yetay@u90z04s03 ex00 % cd ../ex01
(py310) yetay@u90z04s03 ex01 % cat > tester_eval.py
from in_out import outer, square, pow

def cube(x: int | float) -> int | float:
    """Returns the cube of x"""
    return (x * x * x)

my_counter = outer(6, square)
print(my_counter())
print(my_counter())
print(my_counter())
print("---")
another_counter = outer(1.6, pow)
print(another_counter())
print(another_counter())
print(another_counter())
print("---")
again_another_counter = outer(4, cube)
print(again_another_counter())
print(again_another_counter())
print(again_another_counter())
(py310) yetay@u90z04s03 ex01 % cat > tester_eval.out
36.0
1296.0
1679616.0
---
2.1212505710975917
4.929279084950403
2599.697171916239
---
64
262144
18014398509481984
(py310) yetay@u90z04s03 ex01 % diff tester_eval.out <(python tester_eval.py)
1,3c1,3
< 36.0
< 1296.0
< 1679616.0
---
> 36
> 1296
> 1679616
(py310) yetay@u90z04s03 ex01 %
```

The differences between the expected output and the submission output is float and int, which can be disregarded.

ex02 My first decorating

The CallLimit function takes an int as argument and blocks the execution of the function when the number of calls to itself exceeds the limit.

Your script tester.py:

```
from callLimit import callLimit

@callLimit(0)
def h():
    print ("h()")

@callLimit(2)
def j():
    print ("j()")

@callLimit(1)
def k():
    print ("k()")

for i in range(2):
    h()
    j()
    k()
```

expected output:

```
$> python tester.py
Error:  <function h at 0x7fb993152f70> call too many times
j()
k()
Error:  <function h at 0x7fb993152f70> call too many times
j()
Error:  <function k at 0x7fb9930f01f0> call too many times
```

Yes No

```
(py310) yetay@u90z04s03 ex01 % cd ../ex02
(py310) yetay@u90z04s03 ex02 % cat > tester_eval.py
from callLimit import callLimit

@callLimit(0)
def h():
    print ("h()")

@callLimit(2)
def j():
    print ("j()")

@callLimit(1)
def k():
    print ("k()")

for i in range(2):
    h()
    j()
    k()
(py310) yetay@u90z04s03 ex02 % cat > tester_eval.out
Error:  <function h at 0x7fb993152f70> call too many times
j()
k()
Error:  <function h at 0x7fb993152f70> call too many times
j()
Error:  <function k at 0x7fb9930f01f0> call too many times
(py310) yetay@u90z04s03 ex02 % diff tester_eval.out <(python tester_eval.py)
1c1
< Error:  <function h at 0x7fb993152f70> call too many times
---
> Error: <function h at 0x7f97f41835b0> call too many times
4c4
< Error:  <function h at 0x7fb993152f70> call too many times
---
> Error: <function h at 0x7f97f41835b0> call too many times
6c6
< Error:  <function k at 0x7fb9930f01f0> call too many times
---
> Error: <function k at 0x7f97f41837f0> call too many times
(py310) yetay@u90z04s03 ex02 %
```

The difference between the expected output and the submission output is only the space and the memory address (which is expected to be different anyway), hence can be disregarded.

ex03 data class

The data class that takes as Arguments a name and a nickname, create the student's login, and generate a random ID with the generate_id function.
You can change the active state, pass it to False.

```
from new_student import Student

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

expected output: (id is random)

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

The login and id should not be initializable and must return an error.
Your script tester.py:

```
from new_student import Student

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

expected output:

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

Yes No

```
(py310) yetay@u90z04s03 ex02 % cd ../ex03
(py310) yetay@u90z04s03 ex03 % cat > tester_eval.py
from new_student import Student

student = Student(name = "Edmund", surname = "agle")
print(student)
(py310) yetay@u90z04s03 ex03 % cat > tester_eval.out
Student(name='Edmund', surname='agle', active=True, login='Eagle', id='hhkjdjtxeccjb
xh')
(py310) yetay@u90z04s03 ex03 % diff tester_eval.out <(python tester_eval.py)
1c1
< Student(name='Edmund', surname='agle', active=True, login='Eagle', id='hhkjdjtxecc
jbxh')
---
> Student(name='Edmund', surname='agle', active=True, login='Eagle', id='mbkkihejlzc
scet')
(py310) yetay@u90z04s03 ex03 %
```

The difference between the expected output and the submission output is the ID (which is expected to be different), hence can be disregarded.

```
(py310) yetay@u90z04s03 ex03 % cat > tester_eval2.py
from new_student import Student

student = Student(name = "Edward", surname = "agle", login = "pigeon")
print(student)
(py310) yetay@u90z04s03 ex03 % python tester_eval2.py
Traceback (most recent call last):
  File "/Users/yetay/core/peer_eval/20230927153339-lewlee/py4/ex03/tester_eval2.py",
 line 3, in <module>
    student = Student(name = "Edward", surname = "agle", login = "pigeon")
TypeError: Student.__init__() got an unexpected keyword argument 'login'
(py310) yetay@u90z04s03 ex03 %
```

Bahaviour for initializing the Student class directly is as expected (error is thrown).

Conclusion

Leave a comment on this evaluation

Lewis knows the subject requirements well and was able to show us how he complied with all of them during the defense. Thank you for your time, Lewis. I learned as much as I can, but I think I need some time to digest some of the concept. I am sure your pointers will be useful when I work on the exercises later!!!