SCALE FOR PROJECT PYTHON - 3 - OOP (/PROJECTS/PYTHON-3-OOP)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere.42yerevan.am:vogsphere/intra-uuid-2cf69fe5-26
```

Introduction

- Remain polite, courteous, respectful and constructive throughout the evaluation process. The well-being of the community depends on it.

- Identify with the person (or the group) evaluated the eventual dysfunctions of the work. Take the time to discuss and debate the problems you have identified.

- You must consider that there might be some difference in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade him/her as honestly as possible. The pedagogy is valid only and only if peer-evaluation is conducted seriously.

Guidelines

- Only grade the work that is in the student or group's GiT repository.

- Double-check that the GiT repository belongs to the student or the group. Ensure that the work is for the relevant project and also check that "git clone" is used in an empty folder.

- Check carefully that no malicious aliases was used to fool you and make you evaluate something other than the content of the official repository.

- To avoid any surprises, carefully check that both the evaluating

and the evaluated students have reviewed the possible scripts used
to facilitate the grading.

- If the evaluating student has not completed that particular
project yet, it is mandatory for this student to read the
entire subject prior to starting the defence.

- Use the flags available on this scale to signal an empty repository,
non-functioning program, cheating, and so forth.
In these cases, the grading is over and the final grade is 0,
or -42 in case of cheating. However, except the exception of cheating, you
are encouraged to continue to discuss your work even if the later is in
progress in order to identify any issues that may have caused the project
failure and avoid repeating the same mistake in the future.

- Remember that for the duration of the defense, no other unexpected,
premature, or uncontrolled termination of the program, else the final
grade is 0 for the exercise, and continue the evaluation.

- You should never have to edit any file except the configuration file if
the latter exists. If you want to edit a file, take the time to explain why
with the evaluated student and make sure both of you agree on this.

- Lib imports must be explicit, for example importing "from pandas import *"
is not allowed, you must put 0 to the exercise and continue the evaluation.

- Your exercises are going to be evaluated by other students,
make sure that your variable names and function names are appropriate and civil.

Attachments

subject.pdf (https://cdn.intra.42.fr/pdf/pdf/82034/en.subject.pdf)

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

---

ex00 GOT S1E9

There must be an abstract class of Character and a class of stark which inherits from Character and which can take a first_name as first parameter and can change the health state of the Character with a method which passes is_alive from True to False.

Your script tester:

```
from S1E9 import Character, Stark

Ned = Stark("Ned")
print(Ned.__dict__)
print(Ned.is_alive)
Ned.die() # stupefaction and crying
print(Ned.is_alive)
print(Ned.__doc__)
print(Ned.die.__doc__)
print(Ned.__init__.__doc__)
```

expected output: (docstrings can be different)

```
$> python tester.py
{'first_name': 'Ned', 'is_alive': True}
True
False
docstring for Class
docstring for Constructor
docstring for Method
```

It must be impossible to instantiate an abstract class, the code below should make an error.

```
from S1E9 import Character

hodor = Character("hodor")
```

expected output: (if there is no error it means that the abstract class has not been used and put 0 to the exercise)

```
TypeError: Can't instantiate abstract class Character with abstract metho
d
```

✓ Yes ✕ No

ex01 GOT S1E7 & S4E10

You must be able to instantiate classes with the classical method and with the bound method technique.

Your script tester:

```
from S1E7 import Baratheon, Lannister

Robert = Baratheon("Robert")
print(Robert.__dict__)
print(Robert.is_alive)
Robert.die()
print(Robert.is_alive)
print(Robert.__doc__)
print("---")
Cersei = Lannister("Cersei")
print(Cersei.__dict__)
print(Cersei.__str__)
print(Cersei.__repr__)
print(Cersei.is_alive)
print(Cersei.__doc__)
print("---")
Tywin = Lannister.create_lannister("Tywin", True)
Tywin.die()
print(f"Name : {Tywin.first_name, type(Tywin).__name__}, Alive : {Tywin.i
s_alive}")
```

expected output: (docstrings can be different)

```
$> python tester.py
{'first_name': 'Robert', 'is_alive': True, 'family_name': 'Baratheon', 'e
yes': 'brown', 'hair': 'dark'}
True
False
Representing the Baratheon family.
print("---")
{'first_name': 'Cersei', 'is_alive': True, 'family_name': 'Lannister', 'e
yes': 'blue', 'hair': 'light'}
<bound method Lannister.__str__ of Vector: ('Lannister', 'blue', 'light')
>
<bound method Lannister.__repr__ of Vector: ('Lannister', 'blue', 'ligh
t')>
True
Representing the Lannister family.
---
Name : ('Tywin', 'Lannister'), Alive : False
```

✓ Yes ✗ No

ex02 Now it's weird!

The class must manage the diamond heritage correctly and we must be able
to modify the attributes of the king with getter and setter

Your script tester:

```
from DiamondTrap import King
from S1E7 import Baratheon, Lannister
from S1E9 import Character, Stark

Joffrey = King("Joffrey")
print(Joffrey.__dict__)
print(Joffrey.__doc__)
Joffrey.set_eyes("blue")
Joffrey.set_hairs("light")
print(Joffrey.__dict__)
if(isinstance(Joffrey, King) and issubclass(King, Character) and issubcla
ss(King, Lannister) and issubclass(King, Baratheon)):
    print("OK")
else:
    print("Something seems fishy, look at the code to see if the class ki
ng is inherited from the previous exercises")
```

expected output: (docstrings can be different)

```
$> python tester.py
{'first_name': 'Joffrey', 'is_alive': True, 'family_name': 'Baratheon',
'eyes': 'brown', 'hair': 'dark'}
Representing the legitimate king :D
{'first_name': 'Joffrey', 'is_alive': True, 'family_name': 'Baratheon',
'eyes': 'blue', 'hair': 'light'}
OK
```

✓ Yes ✗ No

ex03 Calculate my vector

You have to give only valid calculation, the goal of the exercise is to do magic/special methods, not error management, but you can try the division by 0.

Your script tester:

```
from ft_calculator import calculator

v1 = calculator([0.0, 1.0, 2.0, 3.0])
v1 + 5
v1 * 5
v1 - 5
v1 / 5
```

expected output:

```
$> python tester.py
[5.0, 6.0, 7.0, 8.0]
[25.0, 30.0, 35.0, 40.0]
[20.0, 25.0, 30.0, 35.0]
[4.0, 5.0, 6.0, 7.0]
```

✓ Yes ✗ No

ex04 Calculate my dot product

You have to give only valid calculation, the goal of the exercise is to do magic/special methods, not error management.

Your script tester:

```
from ft_calculator import calculator

a = [7, 8, 42]
b = [125, 3, 0]
calculator.dotproduct(a,b)
calculator.add_vec(a,b)
calculator.sous_vec(a,b)
```

expected output:

```
$> python tester.py
Dot product is: 899
Add Vector is : [132.0, 11.0, 42.0]
Sous Vector is: [-118, 5.0, 42.0]
$>
```

✓ Yes ✗ No

Ratings

Don't forget to check the flag corresponding to the defense

✔ Ok ★ Outstanding project

Empty work Incomplete work Norme Cheat Crash ⚠ Concerning situation

⊘ Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation