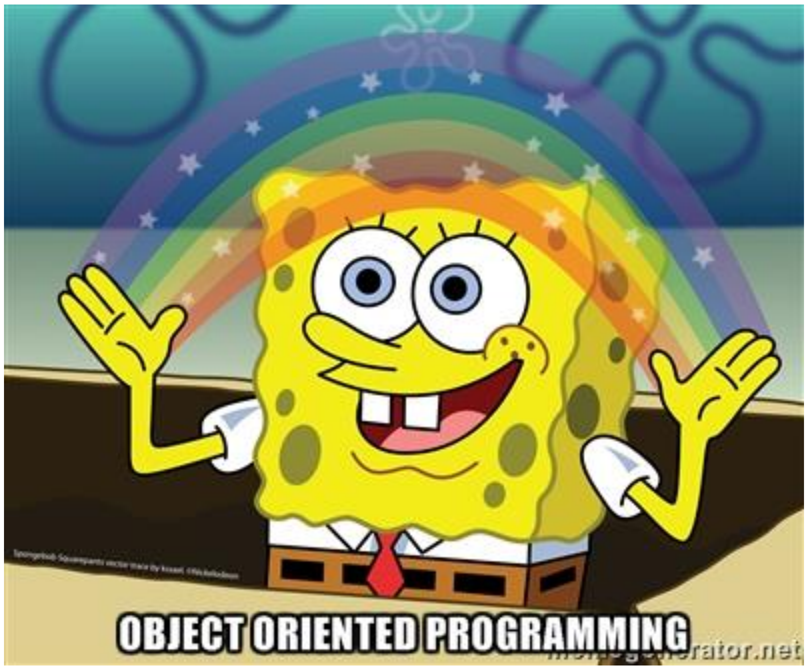
0x06. Python - Classes and Objects

PythonOOP

- By: Guillaume
- Weight: 1
- Project over - took place from Jun 21, 2022 6:00 AM to Jun 22, 2022 6:00 AM
- An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 50.05/77 mandatory & 40.95/63 optional
- **Altogether: 107.25%**
    - Mandatory: 65.0%
    - Optional: 65.0%
    - Calculation: 65.0% + (65.0% * 65.0%) == **107.25%**



Background Context

OOP is a totally new concept for all of you (especially those who think they know about it :)). It's VERY important that you read at least all the material that is listed bellow (and skip what we recommend you to skip, you will see them later in the curriculum).

As usual, make sure you type (never copy and paste), test, understand all examples shown in the following links (including those in the video), test again etc. The biggest and most important takeaway of this project is: experiment by yourself OOP, play with it!

Read or watch the below resources in the order presented.

Resources

Read or watch:

- Object Oriented Programming (*Read everything until the paragraph "Inheritance" excluded. You do NOT have to learn about class attributes, `classmethod` and `staticmethod` yet*)
- Object-Oriented Programming (*Please* *be careful**: in most of the following paragraphs, the author shows things the way you should not use or write a class in order to help you better understand some concepts and how everything works in Python 3. Make sure you read everything in the following paragraphs: General Introduction, First-class Everything, A Minimal Class in Python, Attributes (You DON'T have to learn about class attributes), Methods, The `__init__` Method, "Data Abstraction, Data Encapsulation, and Information Hiding," "Public, Protected, and Private Attributes"*)
- Properties vs. Getters and Setters
- Learn to Program 9 : Object Oriented Programming
- Python Classes and Objects
- Object Oriented Programming

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone, **without the help of Google**:

General

- Why Python programming is awesome
- What is OOP
- "first-class everything"
- What is a class
- What is an object and an instance
- What is the difference between a class and an object or instance
- What is an attribute
- What are and how to use public, protected and private attributes
- What is `self`
- What is a method
- What is the special `__init__` method and how to use it
- What is Data Abstraction, Data Encapsulation, and Information Hiding
- What is a property

- What is the difference between an attribute and a property in Python
- What is the Pythonic way to write getters and setters in Python
- How to dynamically create arbitrary new attributes for existing instances of a class
- How to bind attributes to object and classes
- What is the `__dict__` of a class and/or instance of a class and what does it contain
- How does Python find the attributes of an object or class
- How to use the `getattr` function

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be interpreted/compiled on Ubuntu 20.04 LTS using python3 (version 3.8.5)
- All your files should end with a new line
- The first line of all your files should be exactly `#!/usr/bin/python3`
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the pycodestyle (version `2.8.*`)
- All your files must be executable
- The length of your files will be tested using `wc`
- All your modules should have a documentation (`python3 -c 'print(__import__("my_module").__doc__)'`)
- All your classes should have a documentation (`python3 -c 'print(__import__("my_module").MyClass.__doc__)'`)
- All your functions (inside and outside a class) should have a documentation (`python3 -c 'print(__import__("my_module").my_function.__doc__)'` and `python3 -c 'print(__import__("my_module").MyClass.my_function.__doc__)'`)
- A documentation is not a simple word, it's a real sentence explaining what's the purpose of the module, class or method (the length of it will be verified)

More Info

Documentation is now mandatory! Each module, class, and method must contain docstring as comments. Example Google Style Python Docstrings

Quiz questions

Great! You've completed the quiz successfully! Keep going! (Show quiz)

Tasks

0. My first square

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write an empty class `Square` that defines a square:

- You are not allowed to import any module

```
guillaume@ubuntu:~/0x06$ cat 0-main.py
#!/usr/bin/python3
Square = __import__('0-square').Square

my_square = Square()
print(type(my_square))
print(my_square.__dict__)

guillaume@ubuntu:~/0x06$ ./0-main.py
<class '0-square.Square'>
{}
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `0-square.py`

Done! Help Check your code Get a sandbox QA Review

1. Square with size
mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Square` that defines a square by: (based on `0-square.py`)

- Private instance attribute: `size`
- Instantiation with `size` (no type/value verification)
- You are not allowed to import any module

Why?

Why `size` is private attribute?

The size of a square is crucial for a square, many things depend of it (area computation, etc.), so you, as class builder, must control the type and value of this attribute. One way to have the control is to keep it privately. You will see in next tasks how to get, update and validate the size value.

```
guillaume@ubuntu:~/0x06$ cat 1-main.py
#!/usr/bin/python3
Square = __import__('1-square').Square

my_square = Square(3)
print(type(my_square))
print(my_square.__dict__)

try:
    print(my_square.size)
except Exception as e:
    print(e)

try:
    print(my_square.__size)
except Exception as e:
    print(e)

guillaume@ubuntu:~/0x06$ ./1-main.py
<class '1-square.Square'>
```

```
{'_Square__size': 3}
'Square' object has no attribute 'size'
'Square' object has no attribute '__size'
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `1-square.py`

Done! Help Check your code Get a sandbox QA Review

2. Size validation

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Square` that defines a square by: (based on `1-square.py`)

- Private instance attribute: `size`
- Instantiation with optional `size`: `def __init__(self, size=0):`
  - `size` must be an integer, otherwise raise a `TypeError` exception with the message `size must be an integer`
  - if `size` is less than `0`, raise a `ValueError` exception with the message `size must be >= 0`
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x06$ cat 2-main.py
#!/usr/bin/python3
Square = __import__('2-square').Square

my_square_1 = Square(3)
print(type(my_square_1))
print(my_square_1.__dict__)

my_square_2 = Square()
print(type(my_square_2))
print(my_square_2.__dict__)

try:
    print(my_square_1.size)
except Exception as e:
    print(e)

try:
    print(my_square_1.__size)
except Exception as e:
    print(e)

try:
    my_square_3 = Square("3")
    print(type(my_square_3))
    print(my_square_3.__dict__)
except Exception as e:
    print(e)

try:
    my_square_4 = Square(-89)
    print(type(my_square_4))
    print(my_square_4.__dict__)
except Exception as e:
    print(e)

guillaume@ubuntu:~/0x06$ ./2-main.py
<class '2-square.Square'>
{'_Square__size': 3}
<class '2-square.Square'>
{'_Square__size': 0}
'Square' object has no attribute 'size'
'Square' object has no attribute '__size'
size must be an integer
size must be >= 0
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `2-square.py`

Done! Help Check your code Get a sandbox QA Review

3. Area of a square

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Square` that defines a square by: (based on `2-square.py`)

- Private instance attribute: `size`
- Instantiation with optional `size`: `def __init__(self, size=0):`
  - `size` must be an integer, otherwise raise a `TypeError` exception with the message `size must be an integer`
  - if `size` is less than `0`, raise a `ValueError` exception with the message `size must be >= 0`
- Public instance method: `def area(self):` that returns the current square area
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x06$ cat 3-main.py
#!/usr/bin/python3
Square = __import__('3-square').Square

my_square_1 = Square(3)
print("Area: {}".format(my_square_1.area()))

try:
    print(my_square_1.size)
except Exception as e:
    print(e)

try:
    print(my_square_1.__size)
except Exception as e:
    print(e)
```

```
my_square_2 = Square(5)
print("Area: {}".format(my_square_2.area()))

guillaume@ubuntu:~/0x06$ ./3-main.py
Area: 9
'Square' object has no attribute 'size'
'Square' object has no attribute '__size'
Area: 25
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `3-square.py`

Done! Help Check your code Get a sandbox QA Review

4. Access and update private attribute

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Square` that defines a square by: (based on `3-square.py`)

- Private instance attribute: `size`:
  - property `def size(self):` to retrieve it
  - property setter `def size(self, value):` to set it:
    - `size` must be an integer, otherwise raise a `TypeError` exception with the message `size must be an integer`
    - if `size` is less than `0`, raise a `ValueError` exception with the message `size must be >= 0`
- Instantiation with optional `size`: `def __init__(self, size=0):`
- Public instance method: `def area(self):` that returns the current square area
- You are not allowed to import any module

Why?

Why a getter and setter?

Reminder: `size` is a private attribute. We did that to make sure we control the type and value. Getter and setter methods are not 100% Python, but more OOP. With them, you will be able to validate the assignment of a private attribute and also define how getting the attribute value will be available from outside - by copy? by assignment? etc. Also, adding type/value validation in the setter will centralize the logic, since you will do it in only one place.

```
guillaume@ubuntu:~/0x06$ cat 4-main.py
#!/usr/bin/python3
Square = __import__('4-square').Square

my_square = Square(89)
print("Area: {} for size: {}".format(my_square.area(), my_square.size))

my_square.size = 3
print("Area: {} for size: {}".format(my_square.area(), my_square.size))

try:
    my_square.size = "5 feet"
    print("Area: {} for size: {}".format(my_square.area(), my_square.size))
except Exception as e:
    print(e)

guillaume@ubuntu:~/0x06$ ./4-main.py
Area: 7921 for size: 89
Area: 9 for size: 3
size must be an integer
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `4-square.py`

Done! Help Check your code Get a sandbox QA Review

5. Printing a square

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Square` that defines a square by: (based on `4-square.py`)

- Private instance attribute: `size`:
  - property `def size(self):` to retrieve it
  - property setter `def size(self, value):` to set it:

        - `size` must be an integer, otherwise raise a `TypeError` exception with the message `size must be an integer`
        - if `size` is less than `0`, raise a `ValueError` exception with the message `size must be >= 0`
- Instantiation with optional `size`: `def __init__(self, size=0):`
- Public instance method: `def area(self):` that returns the current square area
- Public instance method: `def my_print(self):` that prints in stdout the square with the character `#`:
    - if `size` is equal to 0, print an empty line
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x06$ cat 5-main.py
#!/usr/bin/python3
Square = __import__('5-square').Square

my_square = Square(3)
my_square.my_print()

print("--")

my_square.size = 10
my_square.my_print()

print("--")

my_square.size = 0
my_square.my_print()

print("--")

guillaume@ubuntu:~/0x06$ ./5-main.py
###
###
###
--
##########
##########
```

```
##########
##########
##########
##########
##########
##########
##########
##########
--

--
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `5-square.py`

Done! Help Check your code Get a sandbox QA Review

6. Coordinates of a square

mandatory

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Square` that defines a square by: (based on `5-square.py`)

- Private instance attribute: `size`:
  - property `def size(self):` to retrieve it
  - property setter `def size(self, value):` to set it:
    - `size` must be an integer, otherwise raise a `TypeError` exception with the message `size must be an integer`
    - if `size` is less than `0`, raise a `ValueError` exception with the message `size must be >= 0`
- Private instance attribute: `position`:
  - property `def position(self):` to retrieve it
  - property setter `def position(self, value):` to set it:
    - `position` must be a tuple of 2 positive integers, otherwise raise a `TypeError` exception with the message `position must be a tuple of 2 positive integers`
- Instantiation with optional `size` and optional `position`: `def __init__(self, size=0, position=(0, 0)):`
- Public instance method: `def area(self):` that returns the current square area

- Public instance method: `def my_print(self):` that prints in stdout the square with the character `#`:
  - if `size` is equal to 0, print an empty line
  - `position` should be use by using space - **Don't fill lines by spaces** when `position[1] > 0`
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x06$ cat 6-main.py
#!/usr/bin/python3
Square = __import__('6-square').Square

my_square_1 = Square(3)
my_square_1.my_print()

print("--")

my_square_2 = Square(3, (1, 1))
my_square_2.my_print()

print("--")

my_square_3 = Square(3, (3, 0))
my_square_3.my_print()

print("--")

guillaume@ubuntu:~/0x06$ ./6-main.py | tr " " "_" | cat -e
###$
###$
###$
--$
$
_###$
_###$
_###$
```

```
--$
___###$
___###$
___###$
--$
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `6-square.py`

Done! Help Check your code Get a sandbox QA Review

7. Singly linked list

#advanced

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Node` that defines a node of a singly linked list by:

- Private instance attribute: `data`:
  - property `def data(self):` to retrieve it
  - property setter `def data(self, value):` to set it:
    - `data` must be an integer, otherwise raise a `TypeError` exception with the message `data must be an integer`
- Private instance attribute: `next_node`:
  - property `def next_node(self):` to retrieve it
  - property setter `def next_node(self, value):` to set it:
    - `next_node` can be `None` or must be a `Node`, otherwise raise a `TypeError` exception with the message `next_node must be a Node object`
- Instantiation with `data` and `next_node`: `def __init__(self, data, next_node=None):`

And, write a class `SinglyLinkedList` that defines a singly linked list by:

- Private instance attribute: `head` (no setter or getter)
- Simple instantiation: `def __init__(self):`
- Should be printable:
  - print the entire list in stdout
  - one node number by line
- Public instance method: `def sorted_insert(self, value):` that inserts a new `Node` into the correct sorted position in the list (increasing order)
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x06$ cat 100-main.py
#!/usr/bin/python3
SinglyLinkedList = __import__('100-singly_linked_list').SinglyLinkedList

sll = SinglyLinkedList()
sll.sorted_insert(2)
sll.sorted_insert(5)
sll.sorted_insert(3)
sll.sorted_insert(10)
sll.sorted_insert(1)
sll.sorted_insert(-4)
sll.sorted_insert(-3)
sll.sorted_insert(4)
sll.sorted_insert(5)
sll.sorted_insert(12)
sll.sorted_insert(3)
print(sll)

guillaume@ubuntu:~/0x06$ ./100-main.py
-4
-3
1
2
3
3
4
5
5
10
12
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `100-singly_linked_list.py`

Done! Help Check your code Get a sandbox QA Review

8. Print Square instance

#advanced

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Square` that defines a square by: (based on `6-square.py`)

- Private instance attribute: `size`:
  - property `def size(self):` to retrieve it
  - property setter `def size(self, value):` to set it:
    - `size` must be an integer, otherwise raise a `TypeError` exception with the message `size must be an integer`
    - if `size` is less than `0`, raise a `ValueError` exception with the message `size must be >= 0`
- Private instance attribute: `position`:
  - property `def position(self):` to retrieve it
  - property setter `def position(self, value):` to set it:
    - `position` must be a tuple of 2 positive integers, otherwise raise a `TypeError` exception with the message `position must be a tuple of 2 positive integer`
- Instantiation with optional `size` and optional `position`: `def __init__(self, size=0, position=(0, 0)):`
- Public instance method: `def area(self):` that returns the current square area
- Public instance method: `def my_print(self):` that prints in stdout the square with the character `#`:
  - if `size` is equal to 0, print an empty line
  - `position` should be use by using space
- Printing a `Square` instance should have the same behavior as `my_print()`
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x06$ cat 101-main.py
#!/usr/bin/python3
Square = __import__('101-square').Square

my_square = Square(5, (0, 0))
print(my_square)

print("--")
```

```
my_square = Square(5, (4, 1))
print(my_square)

guillaume@ubuntu:~/0x06$ ./101-main.py | tr " " "_" | cat -e
#####$
#####$
#####$
#####$
#####$
--$
$
____#####$
____#####$
____#####$
____#####$
____#####$
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `101-square.py`

Done! Help Check your code Get a sandbox QA Review

9. Compare 2 squares

#advanced

Score: 65.0% (*Checks completed: 100.0%*)

Write a class `Square` that defines a square by: (based on `4-square.py`)

- Private instance attribute: `size`:
  - property `def size(self):` to retrieve it
  - property setter `def size(self, value):` to set it:
    - `size` must be a number (float or integer), otherwise raise a `TypeError` exception with the message `size must be a number`
    - if `size` is less than `0`, raise a `ValueError` exception with the message `size must be >= 0`
- Instantiation with `size`: `def __init__(self, size=0):`

- Public instance method: `def area(self):` that returns the current square area
- `Square` instance can answer to comparators: `==`, `!=`, `>`, `>=`, `<` and `<=` based on the square area
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x06$ cat 102-main.py
#!/usr/bin/python3
Square = __import__('102-square').Square

s_5 = Square(5)
s_6 = Square(6)

if s_5 < s_6:
    print("Square 5 < Square 6")
if s_5 <= s_6:
    print("Square 5 <= Square 6")
if s_5 == s_6:
    print("Square 5 == Square 6")
if s_5 != s_6:
    print("Square 5 != Square 6")
if s_5 > s_6:
    print("Square 5 > Square 6")
if s_5 >= s_6:
    print("Square 5 >= Square 6")

guillaume@ubuntu:~/0x06$ ./102-main.py
Square 5 < Square 6
Square 5 <= Square 6
Square 5 != Square 6
guillaume@ubuntu:~/0x06$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `102-square.py`

Done! Help Check your code Get a sandbox QA Review

10. ByteCode -> Python #5

#advanced

Score: 65.0% (*Checks completed: 100.0%*)

Write the Python class `MagicClass` that does exactly the same as the following Python bytecode:

```
Disassembly of __init__:
 10           0 LOAD_CONST               1 (0)
              3 LOAD_FAST                0 (self)
              6 STORE_ATTR               0 (_MagicClass__radius)

 11           9 LOAD_GLOBAL              1 (type)
             12 LOAD_FAST                1 (radius)
             15 CALL_FUNCTION            1 (1 positional, 0 keyword pair)
             18 LOAD_GLOBAL              2 (int)
             21 COMPARE_OP               9 (is not)
             24 POP_JUMP_IF_FALSE       60
             27 LOAD_GLOBAL              1 (type)
             30 LOAD_FAST                1 (radius)
             33 CALL_FUNCTION            1 (1 positional, 0 keyword pair)
             36 LOAD_GLOBAL              3 (float)
             39 COMPARE_OP               9 (is not)
             42 POP_JUMP_IF_FALSE       60

 12          45 LOAD_GLOBAL              4 (TypeError)
             48 LOAD_CONST               2 ('radius must be a number')
             51 CALL_FUNCTION            1 (1 positional, 0 keyword pair)
             54 RAISE_VARARGS            1
             57 JUMP_FORWARD             0 (to 60)

 13     >>   60 LOAD_FAST                1 (radius)
             63 LOAD_FAST                0 (self)
             66 STORE_ATTR               0 (_MagicClass__radius)
             69 LOAD_CONST               3 (None)
             72 RETURN_VALUE
```

```
Disassembly of area:
 17           0 LOAD_FAST                0 (self)
              3 LOAD_ATTR                0 (_MagicClass__radius)
              6 LOAD_CONST               1 (2)
              9 BINARY_POWER
             10 LOAD_GLOBAL              1 (math)
             13 LOAD_ATTR                2 (pi)
             16 BINARY_MULTIPLY
             17 RETURN_VALUE

Disassembly of circumference:
 21           0 LOAD_CONST               1 (2)
              3 LOAD_GLOBAL              0 (math)
              6 LOAD_ATTR                1 (pi)
              9 BINARY_MULTIPLY
             10 LOAD_FAST                0 (self)
             13 LOAD_ATTR                2 (_MagicClass__radius)
             16 BINARY_MULTIPLY
             17 RETURN_VALUE
```

- Tip: Python bytecode

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x06-python-classes`
- File: `103-magic_class.py`

Done! Help Check your code Get a sandbox QA Review