



Curriculum

SE Foundations ^

Average: 137.49% v

You have a captain's log due before 2024-04-21 (in 1 day)! Log it now!
(/captain_logs/5596018/edit)

0x0C. Python - Almost a circle

Python

OOP

⚙ Weight: 1

📅 Project over - took place from Nov 9, 2023 6:00 AM to Nov 14, 2023 6:00 AM☑ Manual QA review was done by Deiaa Elzyat on Nov 13, 2023 1:58 PM

☑ An auto review will be launched at the deadline

In a nutshell...

- **Manual QA review:** 0.0/0 mandatory & 11.0/11 optional
- **Auto QA review:** 669.0/669 mandatory & 17.0/17 optional
- **Altogether: 200.0%**
 - Mandatory: 100.0%
 - Optional: 100.0%
 - Calculation: $100.0\% + (100.0\% * 100.0\%) == 200.0\%$

Overall comment:

It is ok working well.

Background Context



The AirBnB project is a big part of the Higher level curriculum. This project will help you be ready for it

In this project, you will review everything about Python:



- Import
- (/).• Exceptions
- Class
- Private attribute
- Getter/Setter
- Class method
- Static method
- Inheritance
- unittest
- Read/Write file

You will also learn about:

- args and kwargs
- Serialization/Deserialization
- JSON



Resources

Read or watch:

- args/kwargs (/rltoken/7gc6UzxSL81HcuAwklUbuQ)
- JSON encoder and decoder (/rltoken/rGVU9mt57rVURGnjK6n4_Q)
- unittest module (/rltoken/soictNXCPE18ASL3INoeew)
- Python test cheatsheet (/rltoken/ul9iskBCcNo5pc7j9Vy86A)

Learning Objectives

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



General

- What is Unit testing and how to implement it in a large project
- How to serialize and deserialize a Class
- How to write and read a JSON file
- What is `*args` and how to use it
- What is `**kwargs` and how to use it
- How to handle named arguments in a 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

Python Scripts

- 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 be documented: `python3 -c 'print(__import__("my_module").__doc__)'`
- All your classes should be documented: `python3 -c 'print(__import__("my_module").MyClass.__doc__)'`
- All your functions (inside and outside a class) should be documented: `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)

Python Unit Tests

- Allowed editors: `vi`, `vim`, `emacs`
- All your files should end with a new line
- All your test files should be inside a folder `tests`
- You have to use the `unittest` module (`/rltoken/soictNXCPE18ASL3INoeew`)
- All your test files should be python files (extension: `.py`)
- All your test files and folders should start with `test_`
- Your file organization in the `tests` folder should be the same as your project: ex: for `models/base.py`, unit tests must be in: `tests/test_models/test_base.py`



- All your tests should be executed by using this command: `python3 -m unittest discover tests`
- (/). You can also test file by file by using this command: `python3 -m unittest tests/test_models/test_base.py`
- We strongly encourage you to work together on test cases so that you don't miss any edge case

Tasks

0. If it's not tested it doesn't work

mandatory

Score: 100.0% (Checks completed: 100.0%)

All your files, classes and methods must be unit tested and be PEP 8 validated.

```
guillaume@ubuntu:~/ $ python3 -m unittest discover tests
.....
.....
.....
-----
Ran 189 tests in 13.135s

OK
guillaume@ubuntu:~/ $
```


Note that this is just an example. The number of tests you create can be different from the above example.

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `tests/`

☒ Done!

Check your code

 Get a sandbox

QA Review

1. Base class

mandatory

Score: 100.0% (Checks completed: 100.0%)



Write the first class `Base` :

Create a folder named `models` with an empty file `__init__.py` inside - with this file, the folder will become a Python package

Create a file named `models/base.py` :

(/)

- Class `Base` :
 - private class attribute `__nb_objects = 0`
 - class constructor: `def __init__(self, id=None):` :
 - if `id` is not `None` , assign the public instance attribute `id` with this argument value - you can assume `id` is an integer and you don't need to test the type of it
 - otherwise, increment `__nb_objects` and assign the new value to the public instance attribute `id`

This class will be the "base" of all other classes in this project. The goal of it is to manage `id` attribute in all your future classes and to avoid duplicating the same code (by extension, same bugs)

```
guillaume@ubuntu:~/ $ cat 0-main.py
#!/usr/bin/python3
""" 0-main """
from models.base import Base

if __name__ == "__main__":

    b1 = Base()
    print(b1.id)

    b2 = Base()
    print(b2.id)

    b3 = Base()
    print(b3.id)

    b4 = Base(12)
    print(b4.id)

    b5 = Base()
    print(b5.id)
```

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

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `models/base.py`, `models/__init__.py`



[\(7\) Done!](#)[Check your code](#)[> Get a sandbox](#)[QA Review](#)

2. First Rectangle

mandatory

Score: 100.0% (Checks completed: 100.0%)

Write the class `Rectangle` that inherits from `Base` :

- In the file `models/rectangle.py`
- Class `Rectangle` inherits from `Base`
- Private instance attributes, each with its own public getter and setter:
 - `__width` -> `width`
 - `__height` -> `height`
 - `__x` -> `x`
 - `__y` -> `y`
- Class constructor: `def __init__(self, width, height, x=0, y=0, id=None):`
 - Call the super class with `id` - this super call with use the logic of the `__init__` of the `Base` class
 - Assign each argument `width`, `height`, `x` and `y` to the right attribute

Why private attributes with getter/setter? Why not directly public attribute?

Because we want to protect attributes of our class. With a setter, you are able to validate what a developer is trying to assign to a variable. So after, in your class you can "trust" these attributes.

```
guillaume@ubuntu:~/ $ cat 1-main.py
#!/usr/bin/python3
""" 1-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(10, 2)
    print(r1.id)

    r2 = Rectangle(2, 10)
    print(r2.id)

    r3 = Rectangle(10, 2, 0, 0, 12)
    print(r3.id)

guillaume@ubuntu:~/ $ ./1-main.py
1
2
12
guillaume@ubuntu:~/ $
```



Repo:

- GitHub repository: alx-higher_level_programming
- (/).• Directory: 0x0C-python-almost_a_circle
- File: models/rectangle.py

☒ Done!

Check your code

QA Review

3. Validate attributes

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Rectangle` by adding validation of all setter methods and instantiation (`id` excluded):

- If the input is not an integer, raise the `TypeError` exception with the message: `<name of the attribute> must be an integer`. Example: `width must be an integer`
- If `width` or `height` is under or equals 0, raise the `ValueError` exception with the message: `<name of the attribute> must be > 0`. Example: `width must be > 0`
- If `x` or `y` is under 0, raise the `ValueError` exception with the message: `<name of the attribute> must be >= 0`. Example: `x must be >= 0`



```

guillaume@ubuntu:~/ $ cat 2-main.py
#!/usr/bin/python3

""" 2-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    try:
        Rectangle(10, "2")
    except Exception as e:
        print("[{}] {}".format(e.__class__.__name__, e))

    try:
        r = Rectangle(10, 2)
        r.width = -10
    except Exception as e:
        print("[{}] {}".format(e.__class__.__name__, e))

    try:
        r = Rectangle(10, 2)
        r.x = {}
    except Exception as e:
        print("[{}] {}".format(e.__class__.__name__, e))

    try:
        Rectangle(10, 2, 3, -1)
    except Exception as e:
        print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/ $ ./2-main.py
[TypeError] height must be an integer
[ValueError] width must be > 0
[TypeError] x must be an integer
[ValueError] y must be >= 0
guillaume@ubuntu:~/ $

```

Repo:


- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/rectangle.py

☒ Done!

Check your code

QA Review

4. Area first

 mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Rectangle` by adding the public method `def area(self):` that returns the area value of the `Rectangle` instance.

```
guillaume@ubuntu:~/ $ cat 3-main.py
#!/usr/bin/python3
""" 3-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(3, 2)
    print(r1.area())

    r2 = Rectangle(2, 10)
    print(r2.area())

    r3 = Rectangle(8, 7, 0, 0, 12)
    print(r3.area())

guillaume@ubuntu:~/ $ ./3-main.py
6
20
56
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `models/rectangle.py`

☒ Done!

Check your code

QA Review

5. Display #0

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Rectangle` by adding the public method `def display(self):` that prints in stdout the `Rectangle` instance with the character `#` - you don't need to handle `x` and `y` here.



```
guillaume@ubuntu:~/ $ cat 4-main.py
#!/usr/bin/python3

""" 4-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(4, 6)
    r1.display()

    print("----")

    r1 = Rectangle(2, 2)
    r1.display()
```

```
guillaume@ubuntu:~/ $ ./4-main.py
####
####
####
####
####
####
---
##
##
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/rectangle.py

☒ Done!

Check your code

QA Review

6. __str__

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Rectangle` by overriding the `__str__` method so that it returns `[Rectangle] (<id> <x>/<y> - <width>/<height>)`



```
guillaume@ubuntu:~/ $ cat 5-main.py
#!/usr/bin/python3

""" 5-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(4, 6, 2, 1, 12)
    print(r1)

    r2 = Rectangle(5, 5, 1)
    print(r2)

guillaume@ubuntu:~/ $ ./5-main.py
[Rectangle] (12) 2/1 - 4/6
[Rectangle] (1) 1/0 - 5/5
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/rectangle.py

☒ Done!

Check your code

QA Review

7. Display #1

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Rectangle` by improving the public method `def display(self):` to print in stdout the `Rectangle` instance with the character `#` by taking care of `x` and `y`



```

guillaume@ubuntu:~/ $ cat 6-main.py
#!/usr/bin/python3

""" 6-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(2, 3, 2, 2)
    r1.display()

    print("---")

    r2 = Rectangle(3, 2, 1, 0)
    r2.display()

guillaume@ubuntu:~/ $ ./6-main.py | cat -e
$
$
  ##$
  ##$
  ##$
---$
####$
####$
guillaume@ubuntu:~/ $

```

Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/rectangle.py

☒ Done!

Check your code

QA Review

8. Update #0

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Rectangle` by adding the public method `def update(self, *args):` that assigns an argument to each attribute:

- 1st argument should be the `id` attribute
- 2nd argument should be the `width` attribute
- 3rd argument should be the `height` attribute
- 4th argument should be the `x` attribute
- 5th argument should be the `y` attribute



This type of argument is called a "no-keyword argument" - Argument order is super important.

```

guillaume@ubuntu:~/ $ cat 7-main.py
#!/usr/bin/python3

""" Doc """

from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(10, 10, 10, 10)
    print(r1)

    r1.update(89)
    print(r1)

    r1.update(89, 2)
    print(r1)

    r1.update(89, 2, 3)
    print(r1)

    r1.update(89, 2, 3, 4)
    print(r1)

    r1.update(89, 2, 3, 4, 5)
    print(r1)

```

```

guillaume@ubuntu:~/ $ ./7-main.py
[Rectangle] (1) 10/10 - 10/10
[Rectangle] (89) 10/10 - 10/10
[Rectangle] (89) 10/10 - 2/10
[Rectangle] (89) 10/10 - 2/3
[Rectangle] (89) 4/10 - 2/3
[Rectangle] (89) 4/5 - 2/3
guillaume@ubuntu:~/ $

```

Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/rectangle.py

☒ Done!

Check your code

QA Review

9. Update #1

mandatory



Score: 100.0% (Checks completed: 100.0%)

Update the class `Rectangle` by updating the public method `def update(self, *args):` by changing the prototype to `update(self, *args, **kwargs)` that assigns a key/value argument to attributes:

- `**kwargs` can be thought of as a double pointer to a dictionary: key/value
 - As Python doesn't have pointers, `**kwargs` is not literally a double pointer – describing it as such is just a way of explaining its behavior in terms you're already familiar with
- `**kwargs` must be skipped if `*args` exists and is not empty
- Each key in this dictionary represents an attribute to the instance

This type of argument is called a “key-worded argument”. Argument order is not important.

```
guillaume@ubuntu:~/ $ cat 8-main.py
#!/usr/bin/python3
""" 8-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(10, 10, 10, 10)
    print(r1)

    r1.update(height=1)
    print(r1)

    r1.update(width=1, x=2)
    print(r1)

    r1.update(y=1, width=2, x=3, id=89)
    print(r1)

    r1.update(x=1, height=2, y=3, width=4)
    print(r1)
```

```
guillaume@ubuntu:~/ $ ./8-main.py
[Rectangle] (1) 10/10 - 10/10
[Rectangle] (1) 10/10 - 10/1
[Rectangle] (1) 2/10 - 1/1
[Rectangle] (89) 3/1 - 2/1
[Rectangle] (89) 1/3 - 4/2
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `models/rectangle.py`

☒ Done!

Check your code

QA Review



10. And now, the Square!

mandatory

Score: 100.0% (Checks completed: 100.0%)

(/)
Write the class Square that inherits from Rectangle :

- In the file models/square.py
- Class Square inherits from Rectangle
- Class constructor: def __init__(self, size, x=0, y=0, id=None)::
 - Call the super class with id, x, y, width and height - this super call will use the logic of the __init__ of the Rectangle class. The width and height must be assigned to the value of size
 - You must not create new attributes for this class, use all attributes of Rectangle - As reminder: a Square is a Rectangle with the same width and height
 - All width, height, x and y validation must inherit from Rectangle - same behavior in case of wrong data
- The overloading __str__ method should return [Square] (<id>) <x>/<y> - <size> - in our case, width or height

As you know, a Square is a special Rectangle, so it makes sense this class Square inherits from Rectangle. Now you have a Square class who has the same attributes and same methods.



```
guillaume@ubuntu:~/ $ cat 9-main.py
```

```
#!/usr/bin/python3
```

```
""" 9-main """
```

```
from models.square import Square
```

```
if __name__ == "__main__":
```

```
    s1 = Square(5)
```

```
    print(s1)
```

```
    print(s1.area())
```

```
    s1.display()
```

```
    print("---")
```

```
    s2 = Square(2, 2)
```

```
    print(s2)
```

```
    print(s2.area())
```

```
    s2.display()
```

```
    print("---")
```

```
    s3 = Square(3, 1, 3)
```

```
    print(s3)
```

```
    print(s3.area())
```

```
    s3.display()
```

```
guillaume@ubuntu:~/ $ ./9-main.py
```

```
[Square] (1) 0/0 - 5
```

```
25
```

```
#####
```

```
#####
```

```
#####
```

```
#####
```

```
#####
```

```
---
```

```
[Square] (2) 2/0 - 2
```

```
4
```

```
##
```

```
##
```

```
---
```

```
[Square] (3) 1/3 - 3
```

```
9
```

```
###
```

```
###
```

```
###
```

```
guillaume@ubuntu:~/ $
```



Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/square.py

☒ Done!

Check your code

QA Review

11. Square size

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Square` by adding the public getter and setter `size`

- The setter should assign (in this order) the `width` and the `height` - with the same value
- The setter should have the same value validation as the `Rectangle` for `width` and `height` - No need to change the exception error message (It should be the one from `width`)

```
guillaume@ubuntu:~/ $ cat 10-main.py
#!/usr/bin/python3
""" 10-main """
from models.square import Square

if __name__ == "__main__":

    s1 = Square(5)
    print(s1)
    print(s1.size)
    s1.size = 10
    print(s1)

    try:
        s1.size = "9"
    except Exception as e:
        print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/ $ ./10-main.py
[Square] (1) 0/0 - 5
5
[Square] (1) 0/0 - 10
[TypeError] width must be an integer
guillaume@ubuntu:~/ $
```



Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle

- File: `models/square.py`
(/)

☒ Done!

Check your code

QA Review

12. Square update

mandatory

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

Update the class `Square` by adding the public method `def update(self, *args, **kwargs)` that assigns attributes:

- `*args` is the list of arguments - no-keyworded arguments
 - 1st argument should be the `id` attribute
 - 2nd argument should be the `size` attribute
 - 3rd argument should be the `x` attribute
 - 4th argument should be the `y` attribute
- `**kwargs` can be thought of as a double pointer to a dictionary: key/value (keyworded arguments)
- `**kwargs` must be skipped if `*args` exists and is not empty
- Each key in this dictionary represents an attribute to the instance



```
guillaume@ubuntu:~/ $ cat 11-main.py
```

```
#!/usr/bin/python3
```

```
""" 11-main """
```

```
from models.square import Square
```

```
if __name__ == "__main__":
```

```
    s1 = Square(5)
```

```
    print(s1)
```

```
    s1.update(10)
```

```
    print(s1)
```

```
    s1.update(1, 2)
```

```
    print(s1)
```

```
    s1.update(1, 2, 3)
```

```
    print(s1)
```

```
    s1.update(1, 2, 3, 4)
```

```
    print(s1)
```

```
    s1.update(x=12)
```

```
    print(s1)
```

```
    s1.update(size=7, y=1)
```

```
    print(s1)
```

```
    s1.update(size=7, id=89, y=1)
```

```
    print(s1)
```

```
guillaume@ubuntu:~/ $ ./11-main.py
```

```
[Square] (1) 0/0 - 5
```

```
[Square] (10) 0/0 - 5
```

```
[Square] (1) 0/0 - 2
```

```
[Square] (1) 3/0 - 2
```

```
[Square] (1) 3/4 - 2
```

```
[Square] (1) 12/4 - 2
```

```
[Square] (1) 12/1 - 7
```

```
[Square] (89) 12/1 - 7
```

```
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/square.py



☒ Done!

Check your code

QA Review

12) Rectangle instance to dictionary representation

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Rectangle` by adding the public method `def to_dictionary(self):` that returns the dictionary representation of a `Rectangle` :

This dictionary must contain:

- `id`
- `width`
- `height`
- `x`
- `y`

```
guillaume@ubuntu:~/ $ cat 12-main.py
#!/usr/bin/python3
""" 12-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(10, 2, 1, 9)
    print(r1)
    r1_dictionary = r1.to_dictionary()
    print(r1_dictionary)
    print(type(r1_dictionary))

    r2 = Rectangle(1, 1)
    print(r2)
    r2.update(**r1_dictionary)
    print(r2)
    print(r1 == r2)

guillaume@ubuntu:~/ $ ./12-main.py
[Rectangle] (1) 1/9 - 10/2
{'x': 1, 'y': 9, 'id': 1, 'height': 2, 'width': 10}
<class 'dict'>
[Rectangle] (2) 0/0 - 1/1
[Rectangle] (1) 1/9 - 10/2
False
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `models/rectangle.py`



[Done!](#)[Check your code](#)[QA Review](#)

14. Square instance to dictionary representation

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Square` by adding the public method `def to_dictionary(self):` that returns the dictionary representation of a `Square` :

This dictionary must contain:

- `id`
- `size`
- `x`
- `y`

```
guillaume@ubuntu:~/ $ cat 13-main.py
#!/usr/bin/python3
""" 13-main """
from models.square import Square

if __name__ == "__main__":

    s1 = Square(10, 2, 1)
    print(s1)
    s1_dictionary = s1.to_dictionary()
    print(s1_dictionary)
    print(type(s1_dictionary))

    s2 = Square(1, 1)
    print(s2)
    s2.update(**s1_dictionary)
    print(s2)
    print(s1 == s2)
```

```
guillaume@ubuntu:~/ $ ./13-main.py
[Square] (1) 2/1 - 10
{'id': 1, 'x': 2, 'size': 10, 'y': 1}
<class 'dict'>
[Square] (2) 1/0 - 1
[Square] (1) 2/1 - 10
False
guillaume@ubuntu:~/ $
```



Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`

- File: models/square.py
- (/)

☒ Done!

Check your code

QA Review

15. Dictionary to JSON string

mandatory

Score: 100.0% (Checks completed: 100.0%)

JSON is one of the standard formats for sharing data representation.

Update the class `Base` by adding the static method `def to_json_string(list_dictionaries):` that returns the JSON string representation of `list_dictionaries`:

- `list_dictionaries` is a list of dictionaries
- If `list_dictionaries` is `None` or empty, return the string: `[]`
- Otherwise, return the JSON string representation of `list_dictionaries`

```
guillaume@ubuntu:~/ $ cat 14-main.py
#!/usr/bin/python3
""" 14-main """
from models.base import Base
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(10, 7, 2, 8)
    dictionary = r1.to_dictionary()
    json_dictionary = Base.to_json_string([dictionary])
    print(dictionary)
    print(type(dictionary))
    print(json_dictionary)
    print(type(json_dictionary))
```

```
guillaume@ubuntu:~/ $ ./14-main.py
{'x': 2, 'width': 10, 'id': 1, 'height': 7, 'y': 8}
<class 'dict'>
[{"x": 2, "width": 10, "id": 1, "height": 7, "y": 8}]
<class 'str'>
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/base.py



☒ Done!

Check your code

QA Review

16) JSON string to file

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Base` by adding the class method `def save_to_file(cls, list_objs):` that writes the JSON string representation of `list_objs` to a file:

- `list_objs` is a list of instances who inherits of `Base` - example: list of `Rectangle` or list of `Square` instances
- If `list_objs` is `None`, save an empty list
- The filename must be: `<Class name>.json` - example: `Rectangle.json`
- You must use the static method `to_json_string` (created before)
- You must overwrite the file if it already exists

```
guillaume@ubuntu:~/ $ cat 15-main.py
#!/usr/bin/python3
""" 15-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(10, 7, 2, 8)
    r2 = Rectangle(2, 4)
    Rectangle.save_to_file([r1, r2])

    with open("Rectangle.json", "r") as file:
        print(file.read())

guillaume@ubuntu:~/ $ ./15-main.py
[{"y": 8, "x": 2, "id": 1, "width": 10, "height": 7}, {"y": 0, "x": 0, "id": 2, "width": 2, "height": 4}]
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `models/base.py`

☒ Done!

Check your code

QA Review



17. JSON string to dictionary

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Base` by adding the static method `def from_json_string(json_string):` that returns the list of the JSON string representation `json_string`:

- `json_string` is a string representing a list of dictionaries
- If `json_string` is `None` or empty, return an empty list
- Otherwise, return the list represented by `json_string`

```
guillaume@ubuntu:~/ $ cat 16-main.py
#!/usr/bin/python3
""" 16-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    list_input = [
        {'id': 89, 'width': 10, 'height': 4},
        {'id': 7, 'width': 1, 'height': 7}
    ]
    json_list_input = Rectangle.to_json_string(list_input)
    list_output = Rectangle.from_json_string(json_list_input)
    print("[{}] {}".format(type(list_input), list_input))
    print("[{}] {}".format(type(json_list_input), json_list_input))
    print("[{}] {}".format(type(list_output), list_output))

guillaume@ubuntu:~/ $ ./16-main.py
[<class 'list'>] [{'height': 4, 'width': 10, 'id': 89}, {'height': 7, 'width': 1, 'id': 7}]
[<class 'str'>] [{"height": 4, "width": 10, "id": 89}, {"height": 7, "width": 1, "id": 7}]
[<class 'list'>] [{'height': 4, 'width': 10, 'id': 89}, {'height': 7, 'width': 1, 'id': 7}]
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `models/base.py`

☒ Done!

☐ Check your code

☐ QA Review

18. Dictionary to Instance

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Base` by adding the class method `def create(cls, **dictionary):` that returns an instance with all attributes already set:



- `**dictionary` can be thought of as a double pointer to a dictionary
- To use the `update` method to assign all attributes, you must create a “dummy” instance before:

- (/)
- Create a Rectangle or Square instance with “dummy” mandatory attributes (width, height, size, etc.)
 - Call update instance method to this “dummy” instance to apply your real values
- You must use the method `def update(self, *args, **kwargs)`
 - `**dictionary` must be used as `**kwargs` of the method `update`
 - You are not allowed to use `eval`

```
guillaume@ubuntu:~/ $ cat 17-main.py
#!/usr/bin/python3
""" 17-main """
from models.rectangle import Rectangle

if __name__ == "__main__":

    r1 = Rectangle(3, 5, 1)
    r1_dictionary = r1.to_dictionary()
    r2 = Rectangle.create(**r1_dictionary)
    print(r1)
    print(r2)
    print(r1 is r2)
    print(r1 == r2)
```

```
guillaume@ubuntu:~/ $ ./17-main.py
[Rectangle] (1) 1/0 - 3/5
[Rectangle] (1) 1/0 - 3/5
False
False
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `models/base.py`

☒ Done!

Check your code

QA Review

19. File to instances

mandatory

Score: 100.0% (Checks completed: 100.0%)

Update the class `Base` by adding the class method `def load_from_file(cls):` that returns a list of instances:



- The filename must be: `<Class name>.json` - example: `Rectangle.json`
- If the file doesn't exist, return an empty list
- Otherwise, return a list of instances - the type of these instances depends on `cls` (current class using this method)

- You must use the `from_json_string` and `create` methods (implemented previously)
- (/)



```

guillaume@ubuntu:~/ $ cat 18-main.py
#!/usr/bin/python3

""" 18-main """
from models.rectangle import Rectangle
from models.square import Square

if __name__ == "__main__":

    r1 = Rectangle(10, 7, 2, 8)
    r2 = Rectangle(2, 4)
    list_rectangles_input = [r1, r2]

    Rectangle.save_to_file(list_rectangles_input)

    list_rectangles_output = Rectangle.load_from_file()

    for rect in list_rectangles_input:
        print("[{}] {}".format(id(rect), rect))

    print("---")

    for rect in list_rectangles_output:
        print("[{}] {}".format(id(rect), rect))

    print("---")
    print("---")

    s1 = Square(5)
    s2 = Square(7, 9, 1)
    list_squares_input = [s1, s2]

    Square.save_to_file(list_squares_input)

    list_squares_output = Square.load_from_file()

    for square in list_squares_input:
        print("[{}] {}".format(id(square), square))

    print("---")

    for square in list_squares_output:
        print("[{}] {}".format(id(square), square))

```

```

guillaume@ubuntu:~/ $ ./18-main.py
[139785912033120] [Rectangle] (1) 2/8 - 10/7
[139785912033176] [Rectangle] (2) 0/0 - 2/4
---
[139785911764752] [Rectangle] (1) 2/8 - 10/7
[139785911764808] [Rectangle] (2) 0/0 - 2/4
---
---
[139785912058040] [Square] (5) 0/0 - 5

```



```
[139785912061848] [Square] (6) 9/1 - 7
(/)-
[139785911764976] [Square] (5) 0/0 - 5
[139785911765032] [Square] (6) 9/1 - 7
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/base.py

☒ Done!

Check your code

QA Review

20. JSON ok, but CSV?

#advanced

Score: 100.0% (Checks completed: 100.0%)

Update the class `Base` by adding the class methods `def save_to_file_csv(cls, list_objs):` and `def load_from_file_csv(cls):` that serializes and deserializes in CSV:

- The filename must be: `<Class name>.csv` - example: `Rectangle.csv`
- Has the same behavior as the JSON serialization/deserialization
- Format of the CSV:
 - Rectangle: `<id>,<width>,<height>,<x>,<y>`
 - Square: `<id>,<size>,<x>,<y>`



```

guillaume@ubuntu:~/ $ cat 100-main.py
#!/usr/bin/python3

""" 100-main """

from models.rectangle import Rectangle
from models.square import Square

if __name__ == "__main__":

    r1 = Rectangle(10, 7, 2, 8)
    r2 = Rectangle(2, 4)
    list_rectangles_input = [r1, r2]

    Rectangle.save_to_file_csv(list_rectangles_input)

    list_rectangles_output = Rectangle.load_from_file_csv()

    for rect in list_rectangles_input:
        print("[{}] {}".format(id(rect), rect))

    print("---")

    for rect in list_rectangles_output:
        print("[{}] {}".format(id(rect), rect))

    print("---")
    print("---")

    s1 = Square(5)
    s2 = Square(7, 9, 1)
    list_squares_input = [s1, s2]

    Square.save_to_file_csv(list_squares_input)

    list_squares_output = Square.load_from_file_csv()

    for square in list_squares_input:
        print("[{}] {}".format(id(square), square))

    print("---")

    for square in list_squares_output:
        print("[{}] {}".format(id(square), square))

```

```

guillaume@ubuntu:~/ $ ./100-main.py
[140268695797600] [Rectangle] (1) 2/8 - 10/7
[140268695797656] [Rectangle] (2) 0/0 - 2/4
---
[140268695529008] [Rectangle] (1) 2/8 - 10/7
[140268695528952] [Rectangle] (2) 0/0 - 2/4
---
---
[140268695822520] [Square] (5) 0/0 - 5

```



```
[140268695826328] [Square] (6) 9/1 - 7
(/)-
[140268695529232] [Square] (5) 0/0 - 5
[140268695529176] [Square] (6) 9/1 - 7
guillaume@ubuntu:~/ $
```

Repo:

- GitHub repository: alx-higher_level_programming
- Directory: 0x0C-python-almost_a_circle
- File: models/

☒ Done!

Check your code

QA Review

21. Let's draw it

#advanced

Score: 100.0% (Checks completed: 100.0%)

Update the class `Base` by adding the static method `def draw(list_rectangles, list_squares):` that opens a window and draws all the `Rectangles` and `Squares` :

- You must use the Turtle graphics module ([rltoken/d16zMqYw0c7eQje2XgFvFg](https://pypi.org/project/turtle/))
- To install it: `sudo apt-get install python3-tk`
- To make the GUI available outside your vagrant machine, add this line in your Vagrantfile:
`config.ssh.forward_x11 = true`
- No constraints for color, shape etc... be creative!

```
guillaume@ubuntu:~/ $ cat 101-main.py
#!/usr/bin/python3
""" 101-main """
from models.base import Base
from models.rectangle import Rectangle
from models.square import Square

if __name__ == "__main__":

    list_rectangles = [Rectangle(100, 40), Rectangle(90, 110, 30, 10), Rectangle(20, 25, 110, 80)]
    list_squares = [Square(35), Square(15, 70, 50), Square(80, 30, 70)]

    Base.draw(list_rectangles, list_squares)

guillaume@ubuntu:~/ $ ./101-main.py
....
```



- Uncommented line in `/etc/ssh/ssh_config` that said `# ForwardX11 no` and change `no` to `yes` .
- Then added line `config.ssh.forward_agent = true` to my Vagrantfile in addition to `config.ssh.forward_x11 = true` .

- Halted my vm with `vagrant halt` and started it back up with `vagrant up --provision` then `vagrant ssh`.
- If you get an error that looks like `/usr/bin/xauth: timeout in locking authority file /home/vagrant/.Xauthority`, then enter `rm .Xauthority` (you may have to `sudo`).
- Logout and restart the vm with `vagrant up --provision`.
- Test with `xeyes`. If Xquartz is installed on the Mac OS it should open in an Xquartz window.

It is your responsibility to request a review for this task from a peer before the project's deadline. If no peers have been reviewed, you should request a review from a TA or staff member.

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0C-python-almost_a_circle`
- File: `models/base.py`

☒ Done!

QA Review

Ready for a new peer review

Copyright © 2024 ALX, All rights reserved.

