(/)

Curriculum
SE Foundations
Average: **108.76%**

0x0A. Python - Inheritance

Python **OOP** **Inheritance**

By: Guillaume

Weight: 1

Project over - took place from Jun 12, 2023 6:00 AM to Jun 13, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 106.5/170 mandatory & 21.0/21 optional
- **Altogether: 125.3%**
  - Mandatory: 62.65%
  - Optional: 100.0%
  - Calculation: 62.65% + (62.65% * 100.0%) == **125.3%**

Resources

Read or watch:

- Inheritance (/rltoken/ct-bhZHBxfE-aHYQoAcscQ)
- Multiple inheritance (/rltoken/qq52YyYhDIbKBneA-u0PKw)
- Inheritance in Python (/rltoken/RJVbH9PvRlwDkBxcTIoVOQ)
- Learn to Program 10 : Inheritance Magic Methods (/rltoken/CFBGj9h1gP3eNLnEm2Ehhg)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/UJKcx5DE4cRGNq4Ayi-g9g), **without the help of Google**:

Help

General

- Why Python programming is awesome
- What is a superclass, baseclass or parentclass
- What is a subclass
- How to list all attributes and methods of a class or instance
- When can an instance have new attributes
- How to inherit class from another
- How to define a class with multiple base classes
- What is the default class every class inherit from
- How to override a method or attribute inherited from the base class
- Which attributes or methods are available by heritage to subclasses
- What is the purpose of inheritance
- What are, when and how to use `isinstance`, `issubclass`, `type` and `super` built-in functions

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`

Python Test Cases

- Allowed editors: `vi`, `vim`, `emacs`
- All your files should end with a new line
- All your test files should be inside a folder `tests`
- All your test files should be text files (extension: `.txt`)
- All your tests should be executed by using this command: `python3 -m doctest ./tests/*`
- 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)
- We strongly encourage you to work together on test cases, so that you don't miss any edge case

Documentation

- Do not use the words `import` or `from` inside your comments, the checker will think you try to import some modules

Quiz questions

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

Tasks

0. Lookup

mandatory

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

Write a function that returns the list of available attributes and methods of an object:

- Prototype: `def lookup(obj):`
- Returns a `list` object
- You are not allowed to import any module

(7)

```
guillaume@ubuntu:~/0x0A$ cat 0-main.py
#!/usr/bin/python3
lookup = __import__('0-lookup').lookup

class MyClass1(object):
    pass

class MyClass2(object):
    my_attr1 = 3
    def my_meth(self):
        pass

print(lookup(MyClass1))
print(lookup(MyClass2))
print(lookup(int))

guillaume@ubuntu:~/0x0A$ ./0-main.py
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format_
_', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt_
_', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '_
_setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format_
_', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt_
_', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '_
_setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'my_attr1',
'my_meth']
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr_
_', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floor
div__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__h
ash__', '__index__', '__init__', '__int__', '__invert__', '__le__', '__lshift__', '_
_lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__',
'__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__
repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round
__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor_
_', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truedi
v__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_byte
s', 'imag', 'numerator', 'real', 'to_bytes']
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `0-lookup.py`

Done! | Help | Check your code | Get a sandbox | QA Review

1. My list

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Write a class `MyList` that inherits from `list`:

- Public instance method: `def print_sorted(self):` that prints the list, but sorted (ascending sort)
- You can assume that all the elements of the list will be of type `int`
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0A$ cat 1-main.py
#!/usr/bin/python3
MyList = __import__('1-my_list').MyList

my_list = MyList()
my_list.append(1)
my_list.append(4)
my_list.append(2)
my_list.append(3)
my_list.append(5)
print(my_list)
my_list.print_sorted()
print(my_list)

guillaume@ubuntu:~/0x0A$ ./1-main.py
[1, 4, 2, 3, 5]
[1, 2, 3, 4, 5]
[1, 4, 2, 3, 5]
guillaume@ubuntu:~/0x0A$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `1-my_list.py, tests/1-my_list.txt`

Done? | Help | Check your code | Ask for a new correction | Get a sandbox | QA Review

2. Exact same object

mandatory

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

Write a function that returns `True` if the object is *exactly* an instance of the specified class ; otherwise `False`.

- Prototype: `def is_same_class(obj, a_class):`

- You are not allowed to import any module

(/)

```
guillaume@ubuntu:~/0x0A$ cat 2-main.py
#!/usr/bin/python3
is_same_class = __import__('2-is_same_class').is_same_class

a = 1
if is_same_class(a, int):
    print("{} is an instance of the class {}".format(a, int.__name__))
if is_same_class(a, float):
    print("{} is an instance of the class {}".format(a, float.__name__))
if is_same_class(a, object):
    print("{} is an instance of the class {}".format(a, object.__name__))

guillaume@ubuntu:~/0x0A$ ./2-main.py
1 is an instance of the class int
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `2-is_same_class.py`

Done! | Help | Check your code | Get a sandbox | QA Review

3. Same class or inherit from

mandatory

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

Write a function that returns `True` if the object is an instance of, or if the object is an instance of a class that inherited from, the specified class ; otherwise `False` .

- Prototype: `def is_kind_of_class(obj, a_class):`
- You are not allowed to import any module

(/)

```
guillaume@ubuntu:~/0x0A$ cat 3-main.py
#!/usr/bin/python3
is_kind_of_class = __import__('3-is_kind_of_class').is_kind_of_class

a = 1
if is_kind_of_class(a, int):
    print("{} comes from {}".format(a, int.__name__))
if is_kind_of_class(a, float):
    print("{} comes from {}".format(a, float.__name__))
if is_kind_of_class(a, object):
    print("{} comes from {}".format(a, object.__name__))

guillaume@ubuntu:~/0x0A$ ./3-main.py
1 comes from int
1 comes from object
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `3-is_kind_of_class.py`

Done! | Help | Check your code | Get a sandbox | QA Review

4. Only sub class of

mandatory

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

Write a function that returns `True` if the object is an instance of a class that inherited (directly or indirectly) from the specified class ; otherwise `False`.

- Prototype: `def inherits_from(obj, a_class):`
- You are not allowed to import any module

(7)

```
guillaume@ubuntu:~/0x0A$ cat 4-main.py
#!/usr/bin/python3
inherits_from = __import__('4-inherits_from').inherits_from

a = True
if inherits_from(a, int):
    print("{} inherited from class {}".format(a, int.__name__))
if inherits_from(a, bool):
    print("{} inherited from class {}".format(a, bool.__name__))
if inherits_from(a, object):
    print("{} inherited from class {}".format(a, object.__name__))

guillaume@ubuntu:~/0x0A$ ./4-main.py
True inherited from class int
True inherited from class object
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `4-inherits_from.py`

Done! | Help | Check your code | Get a sandbox | QA Review

5. Geometry module

mandatory

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

Write an empty class `BaseGeometry`.

- You are not allowed to import any module

(7)

```
guillaume@ubuntu:~/0x0A$ cat 5-main.py
#!/usr/bin/python3
BaseGeometry = __import__('5-base_geometry').BaseGeometry

bg = BaseGeometry()

print(bg)
print(dir(bg))
print(dir(BaseGeometry))

guillaume@ubuntu:~/0x0A$ ./5-main.py
<5-base_geometry.BaseGeometry object at 0x7f2050c69208>
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format_
_', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt_
_', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '_
_setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format_
_', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt_
_', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '_
_setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `5-base_geometry.py`

Done! | Help | Check your code | Get a sandbox | QA Review

6. Improve Geometry

mandatory

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

Write a class `BaseGeometry` (based on `5-base_geometry.py`).

- Public instance method: `def area(self):` that raises an `Exception` with the message `area() is not implemented`
- You are not allowed to import any module

(7)

```
guillaume@ubuntu:~/0x0A$ cat 6-main.py
#!/usr/bin/python3
BaseGeometry = __import__('6-base_geometry').BaseGeometry

bg = BaseGeometry()

try:
    print(bg.area())
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/0x0A$ ./6-main.py
[Exception] area() is not implemented
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `6-base_geometry.py`

Done! | Help | Check your code | Get a sandbox | QA Review

7. Integer validator

mandatory

Score: 50.0% (*Checks completed: 100.0%*)

Write a class `BaseGeometry` (based on `6-base_geometry.py`).

- Public instance method: `def area(self):` that raises an `Exception` with the message `area() is not implemented`
- Public instance method: `def integer_validator(self, name, value):` that validates `value` :
  - you can assume `name` is always a string
  - if `value` is not an integer: raise a `TypeError` exception, with the message `<name> must be an integer`
  - if `value` is less or equal to 0: raise a `ValueError` exception with the message `<name> must be greater than 0`
- You are not allowed to import any module

(7)

```
guillaume@ubuntu:~/0x0A$ cat 7-main.py
#!/usr/bin/python3
BaseGeometry = __import__('7-base_geometry').BaseGeometry

bg = BaseGeometry()

bg.integer_validator("my_int", 12)
bg.integer_validator("width", 89)

try:
    bg.integer_validator("name", "John")
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

try:
    bg.integer_validator("age", 0)
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

try:
    bg.integer_validator("distance", -4)
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/0x0A$ ./7-main.py
[TypeError] name must be an integer
[ValueError] age must be greater than 0
[ValueError] distance must be greater than 0
guillaume@ubuntu:~/0x0A$
```

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `7-base_geometry.py, tests/7-base_geometry.txt`

Done! | Help | Check your code | Get a sandbox | QA Review

8. Rectangle

mandatory

Score: 50.0% (*Checks completed: 100.0%*)

Write a class `Rectangle` that inherits from `BaseGeometry` (`7-base_geometry.py`).

- Instantiation with `width` and `height` : `def __init__(self, width, height):`
  - `width` and `height` must be private. No getter or setter
  - `width` and `height` must be positive integers, validated by `integer_validator`

(7)

```
guillaume@ubuntu:~/0x0A$ cat 8-main.py
#!/usr/bin/python3
Rectangle = __import__('8-rectangle').Rectangle

r = Rectangle(3, 5)

print(r)
print(dir(r))

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

try:
    r2 = Rectangle(4, True)
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/0x0A$ ./8-main.py
<8-rectangle.Rectangle object at 0x7f6f488f7eb8>
['_Rectangle__height', '_Rectangle__width', '__class__', '__delattr__', '__dict__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt_
_', '__hash__', '__init__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', 'area', 'integer_validator']
[AttributeError] 'Rectangle' object has no attribute 'width'
[TypeError] height must be an integer
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `8-rectangle.py`

☑ Done! | Help | Check your code | >_ Get a sandbox | QA Review

9. Full rectangle

mandatory

Score: 50.0% (*Checks completed: 100.0%*)

Write a class `Rectangle` that inherits from `BaseGeometry` (`7-base_geometry.py`). (task based on `8-rectangle.py`)

- Instantiation with `width` and `height`: `def __init__(self, width, height):`:

(/)

  - `width` and `height` must be private. No getter or setter
  - `width` and `height` must be positive integers validated by `integer_validator`
- the `area()` method must be implemented
- `print()` should print, and `str()` should return, the following rectangle description: `[Rectangle] <width>/<height>`

```
guillaume@ubuntu:~/0x0A$ cat 9-main.py
#!/usr/bin/python3
Rectangle = __import__('9-rectangle').Rectangle

r = Rectangle(3, 5)

print(r)
print(r.area())

guillaume@ubuntu:~/0x0A$ ./9-main.py
[Rectangle] 3/5
15
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `9-rectangle.py`

Done! | Help | Check your code | Get a sandbox | QA Review

10. Square #1

mandatory

Score: 50.0% (*Checks completed: 100.0%*)

Write a class `Square` that inherits from `Rectangle` (`9-rectangle.py`):

- Instantiation with `size`: `def __init__(self, size):`:
  - `size` must be private. No getter or setter
  - `size` must be a positive integer, validated by `integer_validator`
- the `area()` method must be implemented

(7)

```
guillaume@ubuntu:~/0x0A$ cat 10-main.py
#!/usr/bin/python3
Square = __import__('10-square').Square

s = Square(13)

print(s)
print(s.area())

guillaume@ubuntu:~/0x0A$ ./10-main.py
[Rectangle] 13/13
169
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `10-square.py`

Done! | Help | Check your code | Get a sandbox | QA Review

11. Square #2

mandatory

Score: 50.0% (*Checks completed: 100.0%*)

Write a class `Square` that inherits from `Rectangle` (`9-rectangle.py`). (task based on `10-square.py`).

- Instantiation with `size` : `def __init__(self, size):` :
  - `size` must be private. No getter or setter
  - `size` must be a positive integer, validated by `integer_validator`
- the `area()` method must be implemented
- `print()` should print, and `str()` should return, the square description: `[Square] <width>/<height>`

(7)

```
guillaume@ubuntu:~/0x0A$ cat 11-main.py
#!/usr/bin/python3
Square = __import__('11-square').Square

s = Square(13)

print(s)
print(s.area())

guillaume@ubuntu:~/0x0A$ ./11-main.py
[Square] 13/13
169
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `11-square.py`

Done! | Help | Check your code | Get a sandbox | QA Review

12. My integer

#advanced

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

Write a class `MyInt` that inherits from `int`:

- `MyInt` is a rebel. `MyInt` has `==` and `!=` operators inverted
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0A$ cat 100-main.py
#!/usr/bin/python3
MyInt = __import__('100-my_int').MyInt

my_i = MyInt(3)
print(my_i)
print(my_i == 3)
print(my_i != 3)

guillaume@ubuntu:~/0x0A$ ./100-main.py
3
False
True
guillaume@ubuntu:~/0x0A$
```

No test cases needed
(/)

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `100-my_int.py`

Done! | Help | Check your code | Get a sandbox | QA Review

13. Can I?

#advanced

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

Write a function that adds a new attribute to an object if it's possible:

- Raise a `TypeError` exception, with the message `can't add new attribute` if the object can't have new attribute
- You are not allowed to use `try/except`
- You are not allowed to import any module

```
guillaume@ubuntu:~/0x0A$ cat 101-main.py
#!/usr/bin/python3
add_attribute = __import__('101-add_attribute').add_attribute

class MyClass():
    pass

mc = MyClass()
add_attribute(mc, "name", "John")
print(mc.name)

try:
    a = "My String"
    add_attribute(a, "name", "Bob")
    print(a.name)
except Exception as e:
    print("[{}] {}".format(e.__class__.__name__, e))

guillaume@ubuntu:~/0x0A$ ./101-main.py
John
[TypeError] can't add new attribute
guillaume@ubuntu:~/0x0A$
```

No test cases needed

Repo:

- GitHub repository: `alx-higher_level_programming`
- Directory: `0x0A-python-inheritance`
- File: `101-add_attribute.py`

(/)

Done! | Help | Check your code | Get a sandbox | QA Review