

# OOP

June 28, 2020

## 1 Python - Object Oriented Programming (OOP)

<https://www.youtube.com/playlist?list=PLS1QulWo1RIZuCYHd7QUVCQbBxEhu9aDP>

```
[1]: # Pandas is not necessary for the OOP training  
import pandas as pd
```

## 2 Introduction

```
[ ]: '''
class Cab {
    cabService, make, location, numberPlate # data ; data in an object are
    ↳known as 'attributes'
    book(), arrival(), start()             # methods ; procedures/functions
}

class CabDriver {
    name, employeeID                       # data ; data in an object are
    ↳known as 'attributes'
    openDoor(), drive()                   # methods ; procedures/functions
}

class passenger {
    name, address                         # data ; data in an object are
    ↳known as 'attributes'
    openApp(), bookCab(), walk()          # methods ; procedures/functions
}
'''
```

```
[2]: class Car:
      pass
```

```
[3]: ford = Car()    # instance of the Class Car()
      honda = Car()
      audi = Car()
```

```
[4]: print('Color' + '\t' + 'Speed')

      ford.speed = 200
      ford.color = 'red'
      print(ford.color + '\t' + str(ford.speed))

      honda.speed = 220
      honda.color = 'green'
      print(honda.color + '\t' + str(honda.speed))

      audi.speed = 250
      audi.color = 'blue'
      print(audi.color + '\t' + str(audi.speed))
```

| Color | Speed |
|-------|-------|
| red   | 200   |
| green | 220   |
| blue  | 250   |

### 3 Loading instance attributes in Pandas DataFrame (nice to know)

```
[5]: df = pd.DataFrame()
```

```
[6]: df['model'] = ['Ford', 'Honda', 'Audi']  
df['speed'] = [ford.speed, honda.speed, audi.speed]  
df['color'] = [ford.color, honda.color, audi.color]
```

```
[7]: df
```

```
[7]:   model  speed  color  
0   Ford    200    red  
1  Honda    220  green  
2   Audi    250   blue
```

## 4 Back To OOP

```
[8]: print('' + '\t' + 'Color' + '\t' + 'Speed')

ford.speed = 200
ford.color = 'red'
print('Ford' + '\t' + ford.color + '\t' + str(ford.speed))
print('\nChanged to:')
print('' + '\t' + 'Color' + '\t' + 'Speed')
ford.speed = 250
ford.color = 'yellow'
print('Ford' + '\t' + ford.color + '\t' + str(ford.speed))
```

|      | Color | Speed |
|------|-------|-------|
| Ford | red   | 200   |

Changed to:

|      | Color  | Speed |
|------|--------|-------|
| Ford | yellow | 250   |

## 5 Create a new Class

```
[9]: class Rectangle:  
      pass
```

```
[10]: rect1 = Rectangle()  
      rect2 = Rectangle()
```

```
[11]: rect1.height = 20  
      rect1.width = 40  
  
      rect2.height = 30  
      rect2.width = 10
```

```
[12]: print(rect1.height * rect1.width)  
      print(rect2.height * rect2.width)
```

800

300

## 6 Re-Create Car() class

```
[13]: class Car:
      def __init__(self):  # constructor for the class (first method to be
      ↪called)
      print('the __init__ is called')
```

```
[14]: car1 = Car()
```

the \_\_init\_\_ is called

```
[15]: car2 = Car()
      car3 = Car()
```

the \_\_init\_\_ is called

the \_\_init\_\_ is called

## 7 Re-Create Car() class - Again

```
[16]: class Car:
        def __init__(self, speed, color):  # constructor for the class (first_
        ↪method to be called)
            print('the __init__ is called')
            print(speed, color)
            self.speed = speed
            self.color = color
```

```
[17]: ford = Car(200, 'red')
        honda = Car(220, 'green')
        audi = Car(250, 'blue')
```

```
the __init__ is called
200 red
the __init__ is called
220 green
the __init__ is called
250 blue
```

```
[18]: print('' + '\t' + 'Color' + '\t' + 'Speed')

        ford.speed = 200
        ford.color = 'red'
        print('Ford' + '\t' + ford.color + '\t' + str(ford.speed))
        print('Honda' + '\t' + honda.color + '\t' + str(honda.speed))
        print('Audi' + '\t' + audi.color + '\t' + str(audi.speed))
```

|       | Color | Speed |
|-------|-------|-------|
| Ford  | red   | 200   |
| Honda | green | 220   |
| Audi  | blue  | 250   |

## 8 Encapsulation

```
[19]: class Car:
        def __init__(self, speed, color):  # constructor for the class (first
        ↪method to be called)
            self.__speed = speed          # '__' (double underscore makes
        ↪attribute private)
            self.__color = color          # '__' (double underscore makes
        ↪attribute private)

        def set_speed(self, value):
            self.__speed = value

        def get_speed(self):
            return self.__speed

        def set_color(self, value):
            self.__color = value

        def get_color(self):
            return self.__color
```

```
[20]: ford = Car(200, 'red')
        honda = Car(220, 'green')
        audi = Car(250, 'blue')
```

```
[21]: print(ford.get_speed())
        print(ford.get_color())

        ford.set_speed(250)
        ford.set_color('gray')

        print('\nChanged to:')
        print(ford.get_speed())
        print(ford.get_color())
```

200  
red

Changed to:  
250  
gray



## 9 Re-Create Rectangle() class - Again

```
[22]: class Rectangle:
      def __init__(self, height, width):
          self.__height = height
          self.__width = width

      def totals(self):
          return self.__height * self.__width
```

```
[23]: rect1 = Rectangle(20, 60)
      rect2 = Rectangle(50, 40)
```

```
[24]: print(rect1.totals())
      print(rect2.totals())
```

1200

2000

## 10 Hello Example - Private variables

```
[25]: class Hello:
      def __init__(self, name):
          self.a = 10
          self._b = 20
          self.__c = 30

      def public_method(self):
          print(self.a)
          print(self.__c)
          print('public method')
          self.__private_method()

      def __private_method(self):
          print('private method')
```

```
[26]: hoi = Hello('Lex')
```

```
[27]: hoi.public_method()
```

```
10
30
public method
private method
```

## 11 Inheritance

```
[28]: class Polygon:
    __width = None
    __height = None

    def set_values(self, width, height):
        self.__width = width
        self.__height = height

    def get_width(self):
        return self.__width

    def get_height(self):
        return self.__height

class Rectangle(Polygon):
    def area(self):
        return self.get_width() * self.get_height()

class Triangle(Polygon):
    def area(self):
        return self.get_width() * self.get_height() / 2
```

```
[29]: rect = Rectangle()
      tri = Triangle()
```

```
[30]: rect.set_values(50, 40)
```

```
[31]: tri.set_values(50,40)
```

```
[32]: print(rect.area())
      print(tri.area())
```

```
2000
1000.0
```

## 12 From here start: main.py

```
[ ]: # go to the terminal and launch main.py
```

```
[ ]: # Inheritance / Multiple Inheritance
```

```
[ ]: # after that, come back here
```

## 13 Understanding: Python super() Function

```
[33]: class Parent:
        def __init__(self, name):
            print('Parent __init__', name)

        class Parent2:
            def __init__(self, name):
                print('Parent2 __init__', name)

        class Child(Parent, Parent2):
            def __init__(self):
                print('Child __init__')
                super().__init__('Kas ten Braber')
```

```
[34]: child = Child()
```

```
Child __init__
Parent __init__ Kas ten Braber
```

```
[35]: print(Child.__mro__)    # mro = Method Resolution Order
```

```
(<class '__main__.Child'>, <class '__main__.Parent'>, <class
'__main__.Parent2'>, <class 'object'>)
```

```
[36]: class Child(Parent2, Parent):
        def __init__(self):
            print('Child __init__')
            super().__init__('Kas ten Braber')
```

```
[37]: print(Child.__mro__)    # mro = Method Resolution Order
```

```
(<class '__main__.Child'>, <class '__main__.Parent2'>, <class
'__main__.Parent'>, <class 'object'>)
```

```
[38]: class Child(Parent2, Parent):
        def __init__(self):
            print('Child __init__')
            Parent2.__init__(self, 'Kas ten Braber')
            Parent.__init__(self, 'Arie Bombarie')
```

```
[39]: print(Child.__mro__)    # mro = Method Resolution Order
```

```
(<class '__main__.Child'>, <class '__main__.Parent2'>, <class
'__main__.Parent'>, <class 'object'>)
```

```
[40]: child = Child()
```

```
Child __init__  
Parent2 __init__ Kas ten Braber  
Parent __init__ Arie Bombarie
```

## 14 Composition and Aggregation

### Composition

```
[41]: class Salary:
    def __init__(self, pay, bonus):
        self.pay = pay
        self.bonus = bonus

    def annual_salary(self):
        return (self.pay * 12) + self.bonus

class Employee:
    def __init__(self, name, age, pay, bonus):
        self.name = name
        self.age = age
        self.obj_salary = Salary(pay, bonus)

    def total_salary(self):
        return self.obj_salary.annual_salary()
```

```
[42]: emp1 = Employee('Lex', 54, 10000, 2000)
```

```
[43]: print(emp1.total_salary())
```

122000

## Aggregation

```
[44]: class Salary:
        def __init__(self, pay, bonus):
            self.pay = pay
            self.bonus = bonus

        def annual_salary(self):
            return (self.pay * 12) + self.bonus

class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.obj_salary = salary

    def total_salary(self):
        return self.obj_salary.annual_salary()
```

```
[45]: salary = Salary(15000, 10000)
emp = Employee('Lex', 54, salary)
print(emp.total_salary())
```

190000



## 15 Abstract Classes

```
[46]: from abc import ABC, abstractmethod
```

```
[47]: class Shape(ABC):
    @abstractmethod
    def area(self): pass

    @abstractmethod
    def perimeter(self): pass

    class Square(Shape):
        def __init__(self, side):
            self.__side = side

        def area(self):
            return self.__side * self.__side

        def perimeter(self):
            return 4 * self.__side
```

```
[48]: square = Square(5)
```

```
[49]: print(square.area())           # square    5 = 5 x 5 = 25
      print(square.perimeter())      # perimeter 5 = 4 x 5 = 20
```

25

20

## 16 Decorators

```
[50]: def decorator_func(func):  
    def wrapper_func():  
        print('X' * 11)  
        func()  
        print('Y' * 11)  
  
    return wrapper_func  
  
def say_hello():  
    print('Hello World')
```

```
[51]: hello = decorator_func(say_hello)  
hello()
```

```
XXXXXXXXXXXXX  
Hello World  
YYYYYYYYYYYYY
```

```
[52]: say_hello()
```

```
Hello World
```

```
[53]: def decorator_X(func):  
    def wrapper_func():  
        print('X' * 11)  
        func()  
        print('X' * 11)  
  
    return wrapper_func  
  
def decorator_Y(func):  
    def wrapper_func():  
        print('Y' * 11)  
        func()  
        print('Y' * 11)  
  
    return wrapper_func  
  
@decorator_Y  
@decorator_X  
def say_hello():  
    print('Hello World')
```

```
[54]: say_hello()
```

```
YYYYYYYYYYYYY
XXXXXXXXXXXXX
Hello World
XXXXXXXXXXXXX
YYYYYYYYYYYYY
```

```
[55]: def decorator_divide(func):
      def wrapper_func(a, b):
          print('divide', a, 'and', b)
          if b == 0 :
              print('division with zero is not allowed')
              return
          return a / b

      return wrapper_func

      @decorator_divide
      def divide(x, y):
          return x / y
```

```
[56]: print(divide(15, 5))
```

```
divide 15 and 5
3.0
```

```
[57]: print(divide(15, 0))
```

```
divide 15 and 0
division with zero is not allowed
None
```

```
[58]: from time import time

def timing(func):
    def wrapper_func(*args, **kwargs):
        start = time()
        result = func(*args, **kwargs)
        end = time()
        print('Elapsed time: {}'.format(end - start))
        return result

    return wrapper_func

@timing
def my_func(num):
    sum = 0
    for i in range(num + 1):
        sum += 1

    return sum
```

```
[59]: print(my_func(200000000))
```

```
Elapsed time: 13.048908710479736
2000000001
```

## 17 Operator Overloading

```
[60]: print(type(2))
      print(type(2.0))
      print(type('2.0'))
      print(type(True))
      print(2 + 2)
      print('2' + '2')
      print('2' * 3)
      print(2 * 3)
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
4
22
222
6
```

```
[61]: class Number:
      def __init__(self, num):
          self.num = num
```

```
[62]: n1 = Number(1)
      n2 = Number(2)
```

```
[63]: n1 + n2
```

```

      □
↳ -----
      ↳
      TypeError                                Traceback (most recent call↳
      ↳last)

      <ipython-input-63-7fb1059652d4> in <module>
      ----> 1 n1 + n2

      TypeError: unsupported operand type(s) for +: 'Number' and 'Number'
```

```
[64]: class A: pass
```

```
[65]: dir(A)
```

```
[65]: ['__class__',  
      '__delattr__',  
      '__dict__',  
      '__dir__',  
      '__doc__',  
      '__eq__',  
      '__format__',  
      '__ge__',  
      '__getattribute__',  
      '__gt__',  
      '__hash__',  
      '__init__',  
      '__init_subclass__',  
      '__le__',  
      '__lt__',  
      '__module__',  
      '__ne__',  
      '__new__',  
      '__reduce__',  
      '__reduce_ex__',  
      '__repr__',  
      '__setattr__',  
      '__sizeof__',  
      '__str__',  
      '__subclasshook__',  
      '__weakref__']
```

## 18 A real example

```
[66]: import math
```

```
[67]: class Circle:
    def __init__(self, radius):
        self.__radius = radius

    def setRadius(self, radius):
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def area(self):
        return math.pi * self.__radius ** 2

    def __add__(self, circle_object):
        return Circle(self.__radius + circle_object.__radius)

    def __lt__(self, circle_object):
        return (self.__radius < circle_object.__radius)

    def __gt__(self, circle_object):
        return (self.__radius > circle_object.__radius)

    def __mul__(self, circle_object):
        return (self.__radius * circle_object.__radius)

    def __str__(self):
        return 'Circle area = ' + str(self.area())
```

```
[68]: c1 = Circle(2)
      c2 = Circle(3)
      c3 = c1 + c2
      c4 = c1 < c2
      c5 = c1 > c2
      c6 = c1 * c2
```

```
[69]: print(c1.getRadius())
      print(c2.getRadius())
      print(c3.getRadius())
      #print(c4.getRadius())
      #print(c5.getRadius())
      #print(c6.getRadius())
```

2  
3  
5

```
[70]: print(c1 < c2)
      print(c1 > c2)
      print(c1 * c2)
      print(c3 < c2)
      print(c3 > c2)
      print(c3 * c1)
      print(c3 * c2)
```

True  
False  
6  
False  
True  
10  
15

```
[71]: print(dir(c1))
      print()
      print(dir(c2))
      print()
      print(dir(c3))
      print()
      print(dir(c4))
      print()
      print(dir(c5))
      print()
      print(dir(c6))
```

```
['_Circle__radius', '__add__', '__class__', '__delattr__', '__dict__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
 '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
 '__module__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 '__weakref__', 'area', 'getRadius', 'setRadius']
```

```
['_Circle__radius', '__add__', '__class__', '__delattr__', '__dict__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
```



```

'__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
'__module__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'area', 'getRadius', 'setRadius']

```

```

['_Circle__radius', '__add__', '__class__', '__delattr__', '__dict__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
'__module__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'area', 'getRadius', 'setRadius']

```

```

['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__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__', '__truediv__', '__trunc__', '__xor__', 'as_integer_ratio',
'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator',
'real', 'to_bytes']

```

```

['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__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__', '__truediv__', '__trunc__', '__xor__', 'as_integer_ratio',
'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator',
'real', 'to_bytes']

```

```

['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__',
'__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__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__', '__truediv__', '__trunc__', '__xor__', 'as_integer_ratio',  
'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator',  
'real', 'to_bytes']
```

```
[72]: print(str(c1))  
      print(str(c2))  
      print(str(c3))
```

```
Circle area = 12.566370614359172  
Circle area = 28.274333882308138  
Circle area = 78.53981633974483
```

## 19 How to use the Python Debugger

```
[ ]: # open a terminal and execute: python -m pdb debugging.py
```

```
[ ]: # pdb is the Python Debugger
```

```
[ ]: # at the pdb prompt, the following commands where executed
```

```
[ ]: '''  
(Pdb) help          (or: h)  
(Pdb) help next     (or: h n)  
(Pdb) where         (or: w)  
(Pdb) next          (or: n)  
(Pdb) press <enter> (<enter> repeats the last command 'n' or 'next')  
(Pdb) press <enter> (asking for input)  
(Pdb) press <ctrl-c>  
(Pdb) continue      (or: c)  !!! 'c' doesn't work for me here 'continue' does  
(Pdb) 2  
(Pdb) 4  
(Pdb) next  
(Pdb) 3  
(Pdb) print(x)  
(Pdb) next  
(Pdb) 4  
(Pdb) print(y)  
(Pdb) whatis x  
<class 'str'>  
(Pdb) step  
(Pdb) next  
(Pdb) next  
(Pdb) continue  
'''
```

```
[ ]: # make corrections to the code, we pass 'strings' in place of 'integers'
```

```
[ ]: # correct the 2 'input' lines of code with: int(input())
```

```
[ ]: '''
(Pdb) break 9
Breakpoint 1 at /home/lboerhoop/src/OOP-Python/debugging.py:9
(Pdb) continue
Num 1 :
(Pdb) 3
Num 2 :
(Pdb) 4
z = add(x, y)      (breakpoint)
(Pdb) whatis x
<class 'str'>      (still is a 'string', the debugger needs a reload after code_
    ↳ is edited)
(Pdb) quit
'''
```

```
[ ]: # created debugging2.py
```

```
[ ]: # start debugging2.py: python debugging2.py
```

```
[ ]: # after filling in the numbers, the debugger kicks in
```

```
[ ]: # another way to debug, in the terminal:
```

```
[ ]: '''
python
>>> import debugging3
>>> import pdb
>>> pdb.run('debugging3.main()')
> <string>(1)<module>()
(Pub) next
Num 1 :
(Pub) 2
Num 2 :
(Pub) 3
5
--Return--
> <string>(1)<module>()->None
(Pdb) quit
'''
```

## 20 End Of Training