OO Programming with Python

A simple Python class

```
class Robot:

    def __init__(self,
                 name=None,
                 build_year=None):
        self.name = name
        self.build_year = build_year

    def say_hi(self):
        if self.name:
            print("Hi, I am " + self.name)
        else:
            print("Hi, I am a robot without a name")
        if self.build_year:
            print("I was built in " + str(self.build_year))
        else:
            print("It's not known, when I was created!")
```

```
x = Robot("Henry", 2008)
y = Robot()
y.name = "Marvin"
x.say_hi()
y.say_hi()
```

```
Hi, I am Henry
I was built in 2008
Hi, I am Marvin
It's not known, when I was created!
```

"But, but, but, but, but ... ", we can hear them howling and screaming, "But there is NO data ENCAPSULATION!"

Notice how data members are accessed directly.

```
z = Robot()
z.name = "Frodo"
z.build_year = -1
z.say_hi()
```

```
Hi, I am Frodo
I was built in -1
```

Public, - Protected-, and Private Attributes

```
class A():

    def __init__(self):
        self.__priv = "I am private"
        self._prot = "I am protected"
        self.pub = "I am public"


x = A()
x.pub
```

```
'I am public'
```

```
x.__priv
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-f75b36b98afa> in <cell line: 1>()
----> 1 x.__priv

AttributeError: 'A' object has no attribute '__priv'
```

EXPLAIN ERROR

Properties vs. Getters and Setters

A Javaesque way

```
class Robot:

    def __init__(self, name=None, build_year=None):
        self.__name = name
        self.__build_year = self.set_build_year(build_year)

    def set_name(self, name):
        self.__name = name

    def get_name(self):
        return self.__name

    def set_build_year(self, by):
      if (by >= 0):
        return  by
```

```
        else:
          return None

    def get_build_year(self):
        return self.__build_year

    def __str__(self):
        return "Name: " + self.__name + ", Build Year: " +  str(self.__build_year)


if __name__ == "__main__":
    x = Robot("Marvin", 1979)
    y = Robot("Caliban", -1943)
    for rob in [x, y]:
        print(rob)
```

```
Name: Marvin, Build Year: 1979
Name: Caliban, Build Year: None
```

But Python offers a solution to this problem. The solution is called properties!

A Pythonic way

```
class P:
    def __init__(self,x):
        self.x = x


p1 = P(42)
p2 = P(4711)
p1.x
```

```
42
```

```
p1.x = 47
p1.x = p1.x + p2.x
p1.x
```

```
4758
```

The attribute x can have values between 0 and 1000. If a value larger than 1000 is assigned, x should be set to 1000. Correspondingly, x should be set to 0, if the value is less than 0.

```
p1 = P(42)
p1.x = 1001
p1.x
```

```
1001
```

```
class P:
```

```
        def __init__(self,x):
            self.x = x

        @property
        def x(self):
            return self.__x

        @x.setter
        def x(self, x):
            if x < 0:
                self.__x = 0
            elif x > 1000:
                self.__x = 1000
            else:
                self.__x = x


p1 = P(42)
p1.x = 1001
p1.x
```

```
1000
```

```
class Robot:

    def __init__(self, name=None, build_year=None):
        self.name = name
        self.build_year = build_year

    @property
    def build_year(self):
        return self.__build_year

    @build_year.setter
    def build_year(self, by):
      if by < 0:
            self.__build_year = None
      else:
            self.__build_year = by

    def __str__(self):
        return "Name: " + self.name + ", Build Year: " +  str(self.__build_year)


if __name__ == "__main__":
    x = Robot("Marvin", 1979)
    y = Robot("Caliban", -1943)
    for rob in [x, y]:
         print(rob)
```

```
Name: Marvin, Build Year: 1979
Name: Caliban, Build Year: None
```

Public instead of Private Attribute

- Will the value of "OurAtt" be needed by the possible users of our class?
- If not, we can or should make it a private attribute.
- If it has to be accessed, we make it accessible as a public attribute
- We will define it as a private attribute with the corresponding property, if and only if we have to do some checks or transformation of the data. (As an example, you can have a look again at our class P, where the attribute has to be in the interval between 0 and 1000, which is ensured by the property "x")
- Alternatively, you could use a getter and a setter, but using a property is the Pythonic way to deal with it!

```
class OurClass:

    def __init__(self, a):
        self.OurAtt = a


x = OurClass(10)
print(x.OurAtt)
```

```
10
```

```
class OurClass:

    def __init__(self, a):
        self.OurAtt = a

    @property
    def OurAtt(self):
        return self.__OurAtt

    @OurAtt.setter
    def OurAtt(self, val):
        if val < 0:
            self.__OurAtt = 0
        elif val > 1000:
            self.__OurAtt = 1000
        else:
            self.__OurAtt = val


x = OurClass(10)
print(x.OurAtt)
```

```
10
```

Inheritance

```
class Robot:
```

```
    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print("Hi, I am " + self.name)

class PhysicianRobot(Robot):
    pass

x = Robot("Marvin")
y = PhysicianRobot("James")

print(x, type(x))
print(y, type(y))

y.say_hi()
```

```
<__main__.Robot object at 0x7a657556b2e0> <class '__main__.Robot'>
<__main__.PhysicianRobot object at 0x7a657556a560> <class '__main__.PhysicianRobot'>
Hi, I am James
```

```
class Robot:

    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print("Hi, I am " + self.name)

class PhysicianRobot(Robot):

    def __init__(self, name, speciality = None):
      super().__init__(name)
      self.speciality = speciality

    def say_hi(self):
        #Robot.say_hi(self)
        super().say_hi()
        print("and I am a", self.speciality)


doc = PhysicianRobot("Dr. Frankenstein", "Peditrician")
doc.say_hi()
```

```
Hi, I am Dr. Frankenstein
and I am a Peditrician
```

Function Overload?

```
def f(n):
    return n + 42

def f(n,m):
    return n + m + 42

print(f(3, 4))
```

```
49
```

```
print(f(3))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-b58c3bbce83e> in <cell line: 1>()
----> 1 print(f(3))

TypeError: f() missing 1 required positional argument: 'm'
```

EXPLAIN ERROR

Abstract class

```
from abc import ABC, abstractmethod

class Shape(ABC):

    def __init__(self, in_shapename ="no name yet"):
        self.shapename = in_shapename

    def __str__(self):
        return "\nShape Name: " + self.shapename + "\nArea: " + str(self.computeArea())

    @abstractmethod
    def computeArea(self):
        pass

    @abstractmethod
    def computePerimeter(self):
        pass

class Rectangle(Shape):

    def __init__(self, in_shapename, in_width = 0, in_length = 0):
        super().__init__(in_shapename)
        self.width = in_width
        self.length = in_length

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, in_width):
        if in_width < 0:
            print("illegal value")
        else:
            self.__width = in_width

    @property
    def length(self):
        return self.__length
```

```
        @length.setter
        def length(self, in_length):
            if in_length < 0:
                print("illegal value")
            else:
                self.__length = in_length

        def __str__(self):
            return super().__str__() + "\nWidth: " + str(self.__width) + "\nLength:" + str(self.__length)


        def computeArea(self):
            return self.__width * self.__length

        def computePerimeter(self):
            return (self.__width + self.__length) * 2



    if __name__ == "__main__":

        # s = Shape() not allowed.
        print("\n******Testing Rectangle class******")
        rectangle = Rectangle("Basketball Court", 94, 50)
        print(rectangle)
        print("\nChange the widht to 100")
        rectangle.width = 100
        print(rectangle)
        print()
        print("\nChange the height to -50")
        rectangle.height = -50
        print(rectangle)
        print()
        rectangle_1 = Rectangle("Basketball Court", -94, 50)
```

```
******Testing Rectangle class******

Shape Name: Basketball Court
Area: 4700
Width: 94
Length:50

Change the widht to 100

Shape Name: Basketball Court
Area: 5000
Width: 100
Length:50


Change the height to -50

Shape Name: Basketball Court
Area: 5000
Width: 100
Length:50

illegal value
```