

Python In-class Programming Assignment

Classes With Private Properties

In our first lesson on classes, we learned that you can access the properties of a class directly using *dot* notation. For our `Midshipman` class, we performed operations like this:

```
class Midshipman:
    def __init__(self, alpha, company):
        self.alpha = alpha
        self.company = company

mid1 = Midshipman(211234, 13)
mid2 = Midshipman(219999, 8)
print(mid1.alpha, mid1.company)
print(mid2.alpha, mid2.company)
```

*Direct access of a data field in an object, while possible, is not a good practice for two reasons: (1) Data may be tampered-with and (2) The class becomes difficult to maintain and vulnerable to bugs.*¹

Imagine this scenario:

```
class Circle:
    def __init__(self, radius):
        self.radius = radius

c1 = Circle(5)
print(c1.radius)
c1.radius = -1
print(c1.radius) # Is that a valid radius for a circle?
```

The preferred method to manage properties is to make them private and use *getter* and *setter* methods to manage them. We create private properties in our classes by preceding the property name with double underscore (`__`), and we create *getter* and *setter* methods using a particular naming convention.

Here's how the `Circle` class example would be modified using *getters* and *setters*:

¹ Liang pp. 227

Python In-class Programming Assignment

Classes With Private Properties

```
class Circle:

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

    def getRadius(self):
        return self.__radius

    def setRadius(self,radius):
        if (type(radius) == int or type(radius) == float) and radius >= 0:
            self.__radius = radius
        else:
            self.__radius = 0

c1 = Circle(5)
print(c1.getRadius())
c1.setRadius(25)
print(c1.getRadius())
c1.setRadius("Navy")
print(c1.getRadius()) # What happens here?
```

Create a **Rectangle** class using *setters*, *getters* and private properties. Your class should implement the following methods:

```
__init__(self,length,width)
setLengthWidth(self,length,width):
getLengthWidth(self): returns a tuple in the form (Length,width)
perimeter(self):
area(self):
```

- Test your **Rectangle** class by creating a short program that creates a new **Rectangle** object with **length = 13** and **width = 5**.
- Compute and print the perimeter and area of your **Rectangle** object using the appropriate **Rectangle** class methods.
- Now change the dimensions of your **Rectangle** object to **length = 8** and **width = 3** using your *setter* (don't create a new object).
- Compute and print the perimeter and area of your (now-changed) **Rectangle** object.
- Finally, use your *getter* to print the dimensions of your **Rectangle** object.