W200 - Classes - Notes about getter/setter in Python.

(opinions are my own, so if I'm wrong, please just lmk! - GB)

Many folk see the idea of "get/set" methods in any OOD computer language as a way of hiding variables (or making them "private") to control access to the variables. Python doesn't really do this because there are still ways to access an attribute by using its name.

In this note, we'll focus on the idea of making a variable private … and then demonstrate the many ways Python lets us do this. Why are there multiple ways? Beats me :). In addition to what you read about and heard about in the asynch ... In brief:

- **getters** are methods that let us control access to private attributes of a class - returning the value of some private attribute.
- **setters** are the methods to control the setting of a private attribute value.

Several techniques for this …

1). The first example demonstrates data encapsulation - an OOP approach that captures and controls all the attributes and methods in a class - like having a room full of people and only one door.

```
class encapsulationDemoToHideData:
      def __init__(self, a):        # constructor
            # this is a private variable or property in Python
            # the left dundle __ indicates this
            self.__a = a

      # to get the properties using an option - use a "getter"
      def get_a(self):
            return self.__a
      # change the value of this "a"
      def set_a(self, a):
            self.__a = a
```

Example:

```
mydemo = encapsulationDemoToHideData(10)
print(mydemo.get_a())
mydemo.set_a(45)
print(mydemo.get_a())
```

2). As above but now without using a getter/setter. We access the private variable directly using dot notation:

```
class demoPrivate:
      def __init__(self, a):
            self.a = a
# this will return the value without a setter/getter…
dp = print(demoPrivate(50))
print(dp.a)
```

What way is better? Depends on what you're doing … if you need private vars, implement the class using setters, getters.

3). Property:

Use when you want to have some conditions to set the value of an attribute.

E.g., say we want to control the value of data passed as a parameter, such as making sure we pass an even and positive value; otherwise as a default set the value to 2.

```
class demoClass1:
      ''' calls the set_a() method to set value by checking certain conditions'''
      def __init__(self, a):
            self.set_a(a)

            # getter method to set the properties
            def get_a(self):
                  return self.__a
                  # notice the left dundle __

            def set_a(self, a):
                  # condition to check:
                  if a > 0 and a % 2 == 0:
                        self.__a = a
                  else:
                        self.__a = 2
```

Demo:

```
dc1 = demoClass(10)
print(dc1.get_a())
```

3a) Using the @property decorator

```
class propertyDemo1:
      def __init__(self, var):
            # init the attribute
            self.a = var

      @property
      def a(self):
            return self.__a

      # the attribute name and the method name must be the same which
      # is used to set the var for the attribute
      @a.setter
      def a(self, var):
            if var > 0 and var % 2 == 0:
                  self.__a = var
            else:
                  self.__a = 2
```

@property is used to get the value of a private attribute without using any getter methods. We have to put an @property in front of the method where we return the private var.

To set the value of the private variable, we use the **@method_name.setter** in front of the method. We have to use it as a setter.

Demo:

```
pd1 = propertyDemo1(47)
print(pd1.a)
# the above will return 47
```

`@a.setter` will set the value of `a` by checking the conditions mentioned in the method.

3b) Another way to use the @property

```
class propDemo2:
      def __init__(self, var):
            # call the set_a() method to set the value "a" by checking
            # certain conditions exist.
            self.set_a(var)

      # getter method to get the properties using an object
      def get_a(self):
            return self.__a

      # setter method to change the value 'a' using an object
      def set_a(self, var):
            # check conditions:
            if var > 0 and var %2 == 0:
                  self.__a = var
            else:
                  self.__a = 2

      a = property(__get_a, __set_a)
```

Demo:

```
pd2 = propDemo2( sqrt(2) )
# returns 2
```

filename: getterSetterPython.pdf