Recap and language details for Python exercises

4 - OOP: classes, attributes, methods, magic methods

Object Oriented Programming

Object Oriented Programming

Python provides features that support object-oriented programming (OOP):

- In OOP the focus is on the creation of objects which contain both data (**attributes**) and functionality (**methods**).
- An object is an *instance* of a **class**; Classes describe what the objects will be (a class is an object's specification, or definition).
- You can use the same class as a blueprint for creating multiple different objects.
- Classes are created using the keyword **class** and an indented block, which contains the class attributes (data) and methods (which are like functions). Ex.:

```
class className:
    <Attributes>
    <Methods>
```

Initialization

The `__init__` method is called when, using the class name as a function, an instance (an object) of the class is created.

- All methods must have self as their first parameter but you do not need to include it when you call the methods. **Within a method definition, self refers to the instance calling the method**.
- In the `__init__` method, self is used to set the **initial values** of an instance's fundamental attributes.

```python
class Dog:
  def __init__(self, name, color):
    self.name = name
    self.color = color

dog_instance = Dog('Fido', 'brown')
```

Attributes

- Instances of a class have **attributes**: pieces of data associated with them.
- Attributes can be accessed by putting a dot and the attribute name after an instance.

```
class Dog:
  legs = 4 # class attribute
  def __init__(self, name, color):
    self.name = name
    self.color = color

dog1 = Dog('Fido', 'brown')
dog2 = Dog('Spotty', 'white')

print("The dog is called", dog1.name)
print("The dog has color", dog2.color)
```

Class Attributes

- Classes can also have class attributes, created by assigning variables within the body of the class.
- The value is the same for all the instances of a class, and can be modified
- Class attributes can be accessed usually by putting a dot and the attribute name after the name of the class, both inside and outside the class.

```python
class Dog:
  legs = 4 # class attribute
  def __init__(self, name, color):
    self.name = name
    self.color = color

dog1 = Dog('Fido', 'brown')

print("All dogs have", Dog.legs, "gambe")
```

Class Attributes: attention

```python
class Dog:
  legs = 4 # class attribute
  def __init__(self, name, color):
    self.name = name
    self.color = color

dog = Dog('Fido', 'brown')
print(dog.legs) # accessing class attribute in a non standard way
dog.legs = 3 # creating a new instance attribute
""" Disclaimer: no animals were harmed
              by the execution of this code """
print(dog.legs) # instance shadows class attribute
print(Dog.legs) # accessing class attribute
```

See on PythonTutor

Class Methods

- Classes can have methods defined to add functionality to them. They express an action that is possible on that object.

A method differs from a function in two aspects:

- It belongs to a class (and it is defined within a class)
- The first parameter in the definition of a method has to be a reference to the instance which called the method. This parameter is usually called `self`.[1]

Methods are accessed using the same dot syntax as attributes.

[1] Technically, the parameter name `self` is just a convention: it could be changed to anything else. However, this convention is universally followed: it is wise to stick to it.

Class Methods: Example

```python
class Dog:
  def __init__(self, name, color):
    self.name = name
    self.color = color
  def bark(self):
    print('Woof!')

fido = Dog('Fido', 'brown')
fido.bark()
```

```
>>>
Woof!
>>>
```

Class Methods: Example (2)

```python
class Robot:
  def __init__(self, name = None):
      self.name = name
  def say_hi(self):
      if self.name:
          print('Hi, I am ' + self.name)
      else:
          print('Hi, I am a robot without a name')

x = Robot()
x.say_hi()
y = Robot('Marvin')
y.say_hi()
```

```
>>>
Hi, I am a robot without a name
Hi, I am Marvin
>>>
```

Private Attributes and Methods

- Private attributes and methods can only be accessed and modified / called from within a class
- Private attributes and methods might be used to restrict external access to data and functionalities of a class
- Attributes and methods are made private by prefixing their names with a **double underscore**:

Private Attributes and Methods

```python
class Spam:
    __egg = 8 # private attribute
    def print_egg(self): # public method
        self.__increase()
        print(self.__egg)
    def __increase(self): # private method
        self.__egg += 1

s = Spam() # new instance of class Spam
```

```
>>> s.print_egg()
9
>>> print(s.__egg)
AttributeError: 'Spam' object has no attribute '__egg'
>>> s.__increase()
AttributeError: 'Spam' object has no attribute '__increase'
```

“Private” Attributes and Methods

Note however, attributes and methods are not really private:

```python
print(s._Spam__egg)
s._Spam__increase()
print(s._Spam__egg)
```

```
9
10
```

Magic methods

Methods (staring and finishing with a `__`, a *double under* or *dunder*) that, if implemented, provide special functionalities for the class. Just a few examples:

- `__eq__(self, other)` Defines behavior for the equality operator, ==.
- `__add__(self, other)` Implements addition.
- `__sub__(self, other)` Implements subtraction.
- `__mul__(self, other)` Implements multiplication.
- `__floordiv__(self, other)` Implements integer division using the // operator.
- `__div__(self, other)` Implements division using the / operator.
- `__str__(self)` Defines behavior for when str() is called on an instance of your class. Returns a “nice”, “human”, “readable” string representation of the object.
- `__repr__(self)` Returns a string that shoul correspond to the Python istruction useful for creating that instance.
- `__len__(self)` Returns the length of the object

Is an object instance of a class?

We can use the special function **isinstance** to determine if an object is an instance of a certain class

```
d = Dog('Fido', 'brown')
c = Cat('Lulu', 'grey')

isinstance(d,Dog) #True
isinstance(c,Dog) #False

```