Classes!

Lab 08

Recap: Classes

- A class is a template or blueprint for an object.
  - Describes what information is stored in an object (by attribute values)
  - What operations can be performed on an object (by methods)
- An object is an instance of a class.

```
class name_of_the_class:
# constructor/initializer
# other methods
```

- Defining a Class
  - Classes consist of:
    - state variables (sometimes called instance variables)
    - methods (functions that are linked to a particular instance of the class)

What do they have and what do they do?

- Classes consist of:
  - state variables (sometimes called instance variables) – these are what the object defined by a class “remembers"
  - methods (functions that are linked to a particular instance of the class) – these are what an object defined by a class can “do”
- For example:

```
class my_class:
    def __init__(self, name):
        self.__name = name
        #this is what objects defined by this class will remember

    # this is what objects defined by this class can do
    def say_name(self):
        print(self.__name) # will print the name of the object
```

Methods

- Methods define what an object can do
- Methods are like functions the object can run with information about itself.
- Method parameters always start with “self” this is how you can refer to object variables (attributes)
- Special methods start and end with two underscores (e.g. “__init__”, “__str__”) python knows to look out for these methods when trying to do particular things
- Method Example:

```
class Cat:

 def __init__(self, name): # this method tells python how to make Cat objects
  self.__name = name # tells python cat objects have an attribute “__name”

 def meow_n_times(self, n): #this method which tells a cat object how to meow
  print(self.__name + “ says: ” + “meow ” * n)
```

Constructors/Initializer

- Each class should contain a constructor method
  - Name of the method is __init__
  - First parameter must always be self and is mandatory.
  - Self is a reference to the object that we are creating
  - Other parameters are optional.
- Leading __ before attribute name is used to refer to **private** attributes.
- Default values mentioned in parameter list (age=50).

```
class Person:
    def __init__(self, name, age=50):
        self.__name = name
        self.__age  = age
```

```
p1 = Person('Peter', 62)
p2 = Person('Mary')
p3 = Person('John', 24)
```

Other methods: Accessors & Mutators

- Accessors
  - Used to return attribute values to client code.
    - Remember to use self to access attributes.
  - Don't forget to return a value.

```
def get_name(self):
    return self.__name
```

- Mutators
  - Used by client code to **update** or **modify** attribute values .
    - Usually no return statement.
    - Remember that attributes are accessed using self.

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

Other methods:

```
def increment_age(self):
    self.__age+=1
```

- Other methods:
  - Used by client code to update or modify attribute values .
    - Usually no return statement.
    - Remember that attributes are accessed using self.

- Other methods: return an Object
  - Used by client code to return a new instance of itself.
    - With return statement
    - Remember to create a new instance and return it

```
def create_person(self):
    return Person(self.__name, 0)
```

```
def create_person(self):
    return (self.__name, 0)
```

Note: it returns a tuple, not a Person object.

Coderunner Tips

- Q6: You can use the “get_discriminant” method you wrote in Q5 for this question by calling self.get_discriminant()
- Q7: You can use the “has_solution” method you wrote in Q6 for this question by calling self.has_solution()
- Q8: You can use the four methods from Q7, Q6, and Q5 to help with this!
- Q12: You will have to make a new tuple to change the position (tuples can’t be modified)

Two Common Errors With Classes!

- "TypeError: my_method() takes 0 positional arguments but 1 was given"

  Did you forget "self" as the first parameter?

  e.g. def __init__(name, age): rather than def __init__(self, name, age):

- "AttributeError: 'MyClass' object has no attribute 'my_attribute'"

  Did you forget to add two underscores to an attribute name?

  e.g. return self.my_attribute rather than return self.__my_attribute