Monday: OOP Programming

Introduction To Object oriented programming.

Classes

A class is a blueprint to create objects. It allows us to group data and functions logically and in an easy to use manner.

Let us take an example of a *car*. A car has many **features**. It has *number of wheels, seat capacity ,fuel capacity, speed mileage, brand* etc.

A car also has many **behaviors** like *starting,stopping, accelerating, decelerating*.

When creating a blueprint of this car we will include all those features (read **attributes**) and behaviors (read **methods**). This will be our template to creating car **objects**

Objects

An object is an instance of a class. In our car example above we created a basic template or blueprint to which we can create our car objects. Objects can have different property values but implement the same attributes and methods.

We will consider these two in depth as we go along with the lessons.

Contact List application

Let us learn by creating our contact list application.

1. Create a folder on your desktop and call it Contact-List
2. Create in that folder, a file and name it `contact.py`
3. Open Atom

Creating Classes

contact.py

```
class Contact:
 """
 Class that generates new instances of contacts
 """

 pass
```

In Python we create a class using the keyword **class**

We create a class `Contact` and generate a **docstring** for it. Docstrings are strings that occur as the first statement of a named block. They allow you to document what that code block does. This is part of Python's [PEP 8 (https://www.python.org/dev/peps/pep-0008/)](https://www.python.org/dev/peps/pep-0008/) (Python Enhancement Proposal). This is a style guide that is generally accepted by the Python community on how to write and present your code

`__init__` method

The `__init__` method in Python, allows us to create new instances of a class. And it also allows us to pass in properties for the new object.

It is written with double underscores (`_`) before and after the word `init`

```
    def __init__(self,first_name,last_name,phone_number,email):

        '''
        __init__ method that helps us define properties for our objects.

        Args:
            first_name: New contact first name.
            last_name : New contact last name.
            number: New contact phone number.
            email : New contact email address.
        '''
```

Here we create the constructor and pass in 5 arguments. The first argument `self` is a special keyword in Python.

`self` Variable

Methods and functions in Python have only one major difference. Notice how we declare

`__init__` in the above example:

```
...
    def __init__(self,first_name,last_name,phone_number,email):
...
```

Methods have one extra variable added to the beginning of their parameter list: The `self` variable.

`self` is a variable that represents the instance of the object itself. If you are coming from another programming language like Java or JavaScript, the `self` variable can be likened to the `this` keyword,

Instance and Class variables

```
class Contact:
    """
    Class that generates new instances of contacts.
    """

    def __init__(self,first_name,last_name,phone_number,email):

      # docstring removed for simplicity

        self.first_name = first_name
        self.last_name = last_name
        self.phone_number = number
        self.email = email
```

Instance variables are variables that are unique to each new instance of the class. In our example above, we have created four instance variables, that take up the `firstname`, `lastname`, `number`, and `email` of our new contact.

```
class Contact:
    """
    Class that generates new instances of contacts.
    """

    contact_list = [] # Empty contact list

    def __init__(self,first_name,last_name,number,email):

      # docstring removed for simplicity

        self.first_name = first_name
        self.last_name = last_name
        self.phone_number = number
        self.email = email
```

Class variables are variables that belong to the entire class and can be accessed by all instances of the class. Here we create the `contact_list` variable that will be used to store our created contact objects .

Let us test our new class in the R.E.P.L. On the console open the R.E.P.L (on the terminal, type python3.6)

```
>>> from contact import Contact
>>> help(Contact)
```

Here we import our `Contact` class from our contact module and use the `help()`function to see the properties of our Class

```
>>>  new_contact = Contact("James","Muriuki","0712345678","james@moringaschool.com")
>>>  new_contact.first_name
'James'
```

We created a new contact object and passed in all four properties. Notice that we did not pass in an argument for `self`. The `self` variable is passed in the background by the Python Interpretor and we do not need to worry about it.

We then targeted the `firstname` property by using the object `.` dot notation.