Object oriented thinking

* Many of today's most popular programming languages are object oriented, but that's not the only way to program. To understand when and why using object oriented languages can be beneficial, it helps to compare it with a common alternative. Procedural programming languages like plain C.
* In procedural code, the program is written as a long series of operations to execute.
* Now, some of that might be organized intonamed functions or sub-routines to make the code modular and maintainable, but the end goal is really just to get from Point A to Point B to complete some task.
* It's a straight forward approach that I like to relate to writing a recipe for a cookbook.
* The program or recipe to say, bake a cake, would list the sequence of steps you need to follow.
* Mix the ingredients together, pour them into a cake pan, and put it in the oven. Just execute those steps in that order and voila, a cake.
* I've found that new programmers have a tendency to write code in this procedural manner because it's easy to think of simple programs in terms of sequential steps. –
* Right, and to approach that same task of baking a cake in an object oriented mannerrather than describing a sequence of steps, I'll describe each of the objects in my kitchen, the pan, the oven, and the mixer, and what each one can do.
* So, instead of writing a single large program, my object oriented code is split apart into several self contained objects.
* The idea here is that we can talk about and use these programmed objects similar to objects in the real world.
* The mixer can mix ingredients together. I can pour the mix into the pan and the oven can bake whatever I give it.
* One of the main advantages of using an object oriented approach is code re-usability. If we want to make something other than a cake,perhaps muffins, I've already created the functionality to mix and bake things in the mixer and the oven. So I can reuse those objects.

# **Python Class and Objects**

* A class is a virtual entity and can be seen as a blueprint of an object.
* The class came into existence when it instantiated. Let's understand it by an example.
* Suppose a class is a prototype of a building. A building contains all the details about the floor, doors, windows, etc. we can make as many buildings as we want, based on these details. Hence, the building can be seen as a class, and we can create as many objects of this class.
* On the other hand, the object is the instance of a class. The process of creating an object can be called as instantiation.
* Every object has state and behaviour
* State (properties, characteristics, attributes, data, variables)
* Behaviour (action that an object does). Object shows behaviour based on state.
* Beaviour is defined by methods (not functions)
* Explain class and object with particular phone. (Design is class, instance is object)
* Any number of objects can be created from the class.

## Defining a Class in Python

Like function definitions begin with the keyword [def](https://www.programiz.com/python-programming/keyword-list#def), in Python, we define a class using the keyword [class](https://www.programiz.com/python-programming/keyword-list#class).

The first string is called docstring and has a brief description about the class. Although not mandatory, this is recommended.

Here is a simple class definition.

class MyNewClass:

'''This is a docstring. I have created a new class'''

pass

example: Computer class

class Computer:

def config(self):

print(“i5, 16gb ram, 1TB”)

finding the type of object

**class** Comp:  
 **def** display(self):  
 print(**"hello"**)  
c1=Comp()  
print(type(c1))  
c2=Comp()  
print(type(c2))

ACCESSING METHOD OF CLASS

**class** Comp:  
 **def** display(self):  
 print(**"hello"**)  
c1=Comp()  
display()

Traceback (most recent call last):

File "C:/Users/i81191/PycharmProjects/untitled2/MyPython/tk.py", line 5, in <module>

display()

NameError: name 'display' is not defined

Process finished with exit code 1

**class** Comp:  
 **def** display(self):  
 print(**"hello"**)  
c1=Comp()  
Comp.display()

Traceback (most recent call last):

File "C:/Users/i81191/PycharmProjects/untitled2/MyPython/tk.py", line 5, in <module>

Comp.display()

TypeError: display() missing 1 required positional argument: 'self'

Process finished with exit code 1

**class** Comp:  
 **def** display(self):  
 print(**"hello"**)  
c1=Comp()  
Comp.display(c1)

hello

Process finished with exit code 0

Other way:

**class** Comp:  
 **def** display(self):  
 print(**"hello"**)  
c1=Comp()  
c1.display()

NOTE: self is the object that you are referring to

\_\_init\_\_() method in class

* As we know each object has attributes and methods
* Similar to constructor in other OOP languages like c++,java
* It is a special method that gets called automatically whenever the object is created (instantiated)
* Used to initialize the object
* If you need any arguments to be associated with any object, you need to assign them to self.

**class** Comp:  
 **def** \_\_init\_\_(self,cpu,ram):  
 self.cpu=cpu  
 self.ram=ram  
 **def** display(self):  
 print(**"cpu is{} and ram is {}"**.format(self.cpu,self.ram))  
c1=Comp(**"i5"**,8)  
c2=Comp(**"i7"**,16)  
c1.display()  
c2.display()