Constructor

Class functions that begin with double underscore __ are also called special functions as they have special meaning.

Of one particular interest is the __init__() function. This special function gets called whenever a new object of that class is instantiated.

Self Parameter

When we call a method of this object as myobj.method(arg1, arg2), this is automatically converted by Python into myClass.method(myobject, arg1, arg2) - this is all the special self is about.

~~init method~~

~~The~~ e.g

```
class GFG:
        def __init__(self, name, company)
                self.name = name
                self.company = company
        def show(self):
                print("Hello my name is " + self.name
                      "work in "+ self.company + ".
obj = GFG ("John", "OPPO")
obj.show()
```

The self parameter does not call it to be self, You can use any other name instead of it. Here we change the self to the word someone and the output will be the same.

__init__() method

The __init__ method is similar to constructors in C++ and Java. Constructors are used to initializing the object's state. Like methods, a constructor also contains a collection of statements (i.e. that are executed at the time of object creation. It runs as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

1) # Sample class with init method

```
class Person:
    # init method or constructor
    def __init__(self, name):
        self.name = name
    # Sample Method
    def say_hi(self):
        print('Hello, my name is', self.name)
p = Person("Nikhil")
p.say_hi()
```

Output

```
Hello, my name is Nikhil
```

__str__() metod

Python has a particular method called __str__ that is used to define how a class object should be represented as a string. When a class object is used to create a string using the built-in function print() and str(), the __str__() functi is automatically used. You can alter how object of a class are represented in strings by defining the __str__() method.

```
class GFG:
    def __init__(self, name, company)
        self.name = name
        self.company = company
    def __str__(self):
        return f"My name is {self.name} and
                I work in {self.company}."
my_obj = GFG("John", "OPPO")
print(my_obj)
```

Output

```
My name is John and I work in OPPO.
```

Instance Variables

Instance variables are for data, unique to each instance and class variables are for attributes and methods shared by all instances of the class. Instance variables are variables whose value is assigned inside a constructer or method with self whereas class variables are variables whose value is assigned in the class.

(*) Variable inside method or constructer are ~~class~~ called instance.

OOPS

Modularity

Modularity in OOP refers to grouping components with related functionality into a single unit. This helps in robustness, readability and reusability.