#Python Notes

__/__/____

* Classes and Objects :-

Python is an object Ori ented Programming language.

Concept of OOPs in Python :- Python (OOPs) is a Programming Pradigm that uses objects and classes in Programming.

- It aims to implement real-world entities like inheritance, encapsulation, polymorphisms etc.

Main Concepts of OOPs :-

1. Class
2. Object
3. Inheritance
4. Polymorphism
5. Encapsulation.
6. Data hiding and abstraction
7. Dynamic Binding.

Here we will discuss each of this concept in brief :-

_/_/_

1. Class :-

- A class is a collection of objects.
- It is a logical entity that contains some attributes and methods.
- For example :- If you have an employee class, then it should contain an $\boxed{\text{employee}}$ class, then it should contain an attribute and method i.e. an id, name, email, age, salary etc.

2 Object :-

- The object is an entity that has a state and behaviour.
- It may be any real-world object like the mouse, keyboard, chair, table, pen etc.

3. Inheritance :-

- In which objects of one class inherit the properties of objects of another class.

__/__/____

- The new class is known as derived class and old class is known as a base class or Parent class.

- It Provides the re- usability of code.

④ Polymorphism :-

- Polymorphism contain two words "poly" and "morphs".

  Poly → many

  morph → shapes

- It refers to the ability to take more than one form.

- For example :- the addition of two numbers will result into sum however the addition of two strings results into concatenation of strings. It means '+' used for different purpose refers to operator overloading.

⑤ Data hiding and abstraction :-

Abstraction is used to hide internal details and show only functionalities.

- It refers to represent the necessary features without including the background details.

(6) Encapsulation :-

- The wrapping of data and functions into a single unit is known as encapsulation.

- It is used to restrict access to methods and variables.

(7) Dynamic binding :-

- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time.

__/__/____

* Designing Classes

- A class can be defined by using the keyword "class".

- The first string inside the class is called docstring and has a brief description of the class.

Syntax :-
```
class ClassName:
    ''' documentation string '''
    class details.
```

Example:
```
class Dog:
    pass
```

* Creating objects :-

- The instance of class is called object.

- Creating an object in Python is very simple.

- The syntax of defining an object is given as follows :-

```
ObjName = ClassName(arglist)
```

__/__/____

Example:

```
stu = Student('abc', 11) # first object
                           of class Student
```

Program

```
class Person:

    age = 10

    def greet(self):
        print('jpwebdevelopers')

    P1 = Person()
    print(Person.greet)
    print(P1.greet)
    P1.greet()
```

Self Parameter:-

- The self parameter is a reference to the current instance of class and it is used to access variables that belongs to the class.

- It does not have to be named $\boxed{\text{self}}$, you can call it whatever you like, but it has be the first parameter of any function in the class.

__/__/____

* Accessing Attributes :-

The attributes of a class can be accessed by using the dot (·) operator.

Attribute :- That the Properties defined in any class are called its Attribut

So, attributes are the values or function that is associated with any class or data type.

Accessing attribute of class objects using dot (".") operator

```
stu1. displayStudent()
stu2 . display. Student( )
```

(The class methods can be accessed).

Program

```
class Data(object):
    var1 = 2
    def _init_(self, i_var):
        self.i_var = i_var

foo = Data(3)
baz = Data (4)
print( foo.var1 , foo.i_var)
print (baz.var1, baz.i_var)
```

__/__/____

* Editing Class Attributes :-

- In Python, the Programmer can add, delete or modify class attributes.

- It is very simple procedure, which can be performed by using the class object name with the class attribute by using the dot (.) operator.

Program to Modifying object Properties :-

```
class emp:
      def _init_ (self, name, age):
         self.name = name
         self.age  = age

      def myfun (self):
            print ("my name is " + self.name)

  emp1  = employee (" Ram ", 31)
  emp1 .age = 40
  print (emp1.age)
```

Program to Delete Object Properties :-

```
class emp:
```

```
def __init__(self, name, age):

        self.name  = name
        self.age   = age

 def myfunc (self):
        print("My name is" + self.name)

emp1  = employee(" Ram", 31)
del  emp1.age
print(emp1.age)
```

_/_/_

* Built-in Class Attributes in Python :-

- Every Python class keeps following built-in-attributes and they can be accessed using dot operator.
- It is to be noted that the class built-in-attributes are accessed with the classname, dot (.) operator.

Attribute	Description
dict	Dictionary containing the class's namespace.
doc	Class documentation string or none, if undefined.
name	Class name.
module	Module name in which the class is defined. This attribute is "_main_" in interactive mode.
bases	A Possibly empty tuple containing the base classes, in Order of their occurrence in the base class list.

__/__/____

Program of using built-in-class Attributes.

```
class Emp:
    empCount = 0

    def _init_ (self, name, salary):
      self.name = name
      self.salary = salary
      Emp.  .empCount += 1

    def displayCount(self):
      print "Total Employee" Emp.  .empCount

    def displayEmployee (self):
      print "Name:", self.name, "Salary", self.salary

print "Employee.__doc__:", Emp.__doc__
print "Employee.__name__:", Emp.__name__
print "Employee.__module__:", Emp.__module__
print "Employee.__bases__:", Emp.__bases__
print "Employee.__dict__:", Emp.__dict__
```

__/__/____

* Garbage Collection / Destroying Objects :-

- Python deletes unwanted objects (built in types or class instances) automatically to free the memory space. The process by which python periodically frees and reclaim blocks of memory are in use is called Garbage Collection.

- Python provides a special method $\boxed{\_del\_()}$ called a destructor, that is called when the instance is about to be destoryed.

Syntax :- def _del_(self):
↑
argument

Example :-
```
class ABC:
    def __init_ (self):
        Print ("The Object is created")
    def __del__ (self):
        Print ("The Object is destroyed")
Obj 1 = ABC()
Obj 2 = Obj 1
Obj 3 = Obj 1

Print (" Set Obj1 to None ...")
Obj 1 = None
```

```
Print ("Set Obj2 to None_ ")
Obj2 = None
print ("Set Obj3 to None --")
Obj3 = None
```

Output :-

```
The Object is Created
 Set Obj1 to None
Set Obj2 to None
Set Obj3 to None
The Object is destroyed
```

After Creation of the Object the (init) methoc is called, when the object is destroyed due to Garbage collection --del-- (del) method is called.