

Python Notes

//_

* Classes and Objects :-

Python is an object oriented programming language.

Concept of OOPs in Python :- Python (OOPs) is a programming paradigm 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 :-

(Download handwritten notes www.jpwebdevelopers.in)

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 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.

o The new class is known as derived class and old class is known as a base class or parent class.

o It provides the re-usability of code.

④ Polymorphism :-
=====

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

Poly → many
morph → shapes

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

o 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

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

- o The instance of class is called object.
- o Creating an object in Python is very simple.
- o 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('jwebdevelopers')
```

```
    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 self, you can call it whatever you like, but it has to 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 Attribute. 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.displayStudent()
```

(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)
```

ipwebdevelopers

* 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
<code>__dict__</code>	Dictionary containing the class's namespace.
<code>__doc__</code>	Class documentation string or none, if undefined.
<code>__name__</code>	Class name.
<code>__module__</code>	Module name in which the class is defined. This attribute is " <code>__main__</code> " in interactive mode.
<code>__bases__</code>	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__
```


//_

```
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) method is called, when the object is destroyed due to garbage collection -- del -- (del) method is called.