

**PYTHON OOPS CONCEPT**

* **OOPs in Python** is a programming approach that focuses on using objects and classes as same as other general programming languages.
* The objects can be any real-world entities.
* Python allows developers to develop applications using the OOPs approach with the major focus on code reusability.
* It is very easy to create classes and objects in Python.

**What is a Class?**

* Class is a collection of data member and member function,here data member means variable and member function means function that define in class it also called class property(data member and member function).

**Object**

* Object is a basic runtime entity which use for the access the property of class.

**Inheritance**

* Access the property of one class to another class and its basically use for code reusablity.

**Polymorphism**

* Polymorphism means ability to take one more than one form.
* it means we can create one form’s many form.

**Encapsulation**

* Wrapping a data in a single unit its call encapsulation.
* It is used to restrict access to methods and variables.

**Data Abstraction**

* Show Only Functionality and hiding Background Detail that’s call data abstraction.
* Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonyms because data abstraction is achieved through encapsulation.
* Abstraction is used to hide internal details and show only functionalities.
* Abstracting something means to give names to things so that the name captures the core of what a function or a whole program does.

**Python Constructor**

* A constructor is a special type of method (function) which is used to initialize the instance members of the class.
* It means method and class name is same in constructor.

Constructors can be of two types.

* Parameterized Constructor
* Non-parameterized Constructor

Creating the constructor in python

* Constructor definition is executed when we create the object of this class.
* In Python, the method the **\_\_init\_\_()** simulates the constructor of the class.
* This method is called when the class is instantiated.
* It accepts the **self**-keyword as a first argument which allows accessing the attributes or method of the class.
* We can pass any number of arguments at the time of creating the class object, depending upon the **\_\_init\_\_()** definition. It is mostly used to initialize the class attributes. Every class must have a constructor, even if it simply relies on the default constructor.

**Constructor Example:**

class demo:  
 def \_\_init\_\_(self): //here create constructor  
 print("Hello from Constructor")  
   
 def show(self,name,contact):  
 print("Name :",name)  
 print("Contact :",contact)  
  
c1=demo()*#here create class object*c1.show("PRAXWARE TECHNOLOGIES",9016395600)

**Python Parameterized Constructor**

class demo:  
 def \_\_init\_\_(self,name):  
 print(name)  
  
 def show(self,name,contact):  
 print("Name :",name)  
 print("Contact :",contact)

*#here create class object AND CONSTRUCTOR*  
c1=demo("PRAXWARE TECHNOLOGIES AHMEDBAD")

c1.show("PRAXWARE TECHNOLOGIES",9016395600)

# **Accessing Attributes and Methods in Python**

* Attributes of a class are function objects that define corresponding methods of its instances.
* They are used to implement access controls of the classes.
* Attributes of a class can also be accessed using the following built-in methods and functions.
* **getattr() –** This function is used to access the attribute of object.
* **hasattr() –** This function is used to check if an attribute exists or not.
* **setattr() –** This function is used to set an attribute. If the attribute does not exist, then it would be created.
* **delattr() –** This function is used to delete an attribute. If you are accessing the attribute after deleting it raises error “class has no attribute”.

**Example:**

class attributedemo:  
 name="Praxware Technologies"  
 Address="Ahmedabad"  
 def show(self):  
 print("Name :",self.name)  
 print("Address :",self.Address)

obj=attributedemo()  
  
*# Use getattr*print(getattr(obj,'name'))  
  
*# returns true if object has attribute*print(hasattr(obj,'name'))  
  
*# sets an attribute*setattr(obj,'contact',9016395600)  
  
*# returns the value of attribute name contact*print(getattr(obj,'contact'))  
  
*# delete the attribute*print(delattr(obj,'contact'))

**Static methods**

* A static method is a method (member function) that don’t use argument self at all.
* To declare a static method, proceed it with the statement “@staticmethod”.

Example:

class staticdemo:  
 result = None  
  
 @staticmethod  
 def squere(r):  
 staticdemo.result=r\*r  
  
obj1=staticdemo()  
obj1.squere(50)  
print(staticdemo.result)  
  
obj1.squere(100)  
print(staticdemo.result)  
print(staticdemo.result)