**Python OOPs Concepts**

Python is an **object-oriented programming language**. What this means is we can solve a problem in Python by creating objects in our programs. In this guide, we will discuss OOPs terms such as **class**, **objects**, **methods** etc. along with the Object oriented programming features such as **inheritance**, **polymorphism**, **abstraction**, **encapsulation**.

**Object**

An object is an entity that has attributes and behaviour. For example, Ram is an object who has attributes such as height, weight, color etc. and has certain behaviours such as walking, talking, eating etc.

**Class**

A class is a blueprint for the objects. For example, Ram, Shyam, Steve, Rick are all objects so we can define a template (blueprint) class Human for these objects. The class can define the common attributes and behaviours of all the objects.

**Methods**

As we discussed above, an object has attributes and behaviours. These behaviours are called methods in programming.

**Example of Class and Objects**

In this example, we have two objects Ram and Steve that belong to the class Human  
Object attributes: name, height, weight  
Object behaviour: eating()

**Source code**

class Human:

# instance attributes

def \_\_init\_\_(self, name, height, weight):

self.name = name

self.height = height

self.weight = weight

# instance methods (behaviours)

def eating(self, food):

return "{} is eating {}".format(self.name, food)

# creating objects of class Human

ram = Human("Ram", 6, 60)

steve = Human("Steve", 5.9, 56)

# accessing object information

print("Height of {} is {}".format(ram.name, ram.height))

print("Weight of {} is {}".format(ram.name, ram.weight))

print(ram.eating("Pizza"))

print("Weight of {} is {}".format(steve.name, steve.height))

print("Weight of {} is {}".format(steve.name, steve.weight))

print(steve.eating("Big Kahuna Burger"))

**Output:**

Height of Ram is 6

Weight of Ram is 60

Ram is eating Pizza

Weight of Steve is 5.9

Weight of Steve is 56

Steve is eating Big Kahuna Burger

**How to create Class and Objects in Python**

In the previous guide, we discussed [Object-oriented programming in Python](https://beginnersbook.com/2018/03/python-oops-concepts/). In this tutorial, we will see how to create classes and objects in Python.

**Define class in Python**

A class is defined using the keyword class.

**Example**

In this example, we are creating an empty class DemoClass. This class has no attributes and methods.

The string that we mention in the triple quotes is a docstring which is an optional string that briefly explains the purpose of the class.

class DemoClass:

"""This is my docstring, this explains brief about the class"""

# this prints the docstring of the class

print(DemoClass.\_\_doc\_\_)

Output:

This is my docstring, this explains brief about the class

**Creating Objects of class**

In this example, we have a class MyNewClass that has an attribute num and a function hello(). We are creating an object obj of the class and accessing the attribute value of object and calling the method hello() using the object.

class MyNewClass:

"""This class demonstrates the creation of objects"""

# instance attribute

num = 100

# instance method

def hello(self):

print("Hello World!")

# creating object of MyNewClass

obj = MyNewClass()

# prints attribute value

print(obj.num)

# calling method hello()

obj.hello()

# prints docstring

print(MyNewClass.\_\_doc\_\_)

Output:

100

Hello World!

This class demonstrates the creation of objects

**Python Constructors – default and parameterized**

A constructor is a special kind of method which is used for initializing the instance variables during object creation. In this guide, we will see what is a constructor, types of it and how to use them in the python programming with examples.

**1. What is a Constructor in Python?**

Constructor is used for initializing the instance members when we create the object of a class.

For example:  
Here we have a instance variable num which we are initializing in the constructor. The constructor is being invoked when we create the object of the class (obj in the following example).

class DemoClass:

# constructor

def \_\_init\_\_(self):

# initializing instance variable

self.num=100

# a method

def read\_number(self):

print(self.num)

# creating object of the class. This invokes constructor

obj = DemoClass()

# calling the instance method using the object obj

obj.read\_number()

Output:

100

**1.1 Syntax of constructor declaration**

As we have seen in the above example that a constructor always has a name init and the name init is prefixed and suffixed with a double underscore(\_\_). We declare a constructor using def keyword, just like methods.

def \_\_init\_\_(self):

# body of the constructor

**2. Types of constructors in Python**

We have two types of constructors in Python.  
1. default constructor – this is the one, which we have seen in the above example. This constructor doesn’t accept any arguments.  
2. parameterized constructor – constructor with parameters is known as parameterized constructor.

**2.1 Python – default constructor example**

Note: An object cannot be created if we don’t have a constructor in our program. This is why when we do not declare a constructor in our program, python does it for us. Lets have a look at the example below.

**Example: When we do not declare a constructor**  
In this example, we do not have a constructor but still we are able to create an object for the class. This is because there is a default constructor implicitly injected by python during program compilation, this is an empty default constructor that looks like this:

def \_\_init\_\_(self):

# no body, does nothing.

**Source Code:**

class DemoClass:

num = 101

# a method

def read\_number(self):

print(self.num)

# creating object of the class

obj = DemoClass()

# calling the instance method using the object obj

obj.read\_number()

Output:

101

**Example: When we declare a constructor**  
In this case, python does not create a constructor in our program.

class DemoClass:

num = 101

# non-parameterized constructor

def \_\_init\_\_(self):

self.num = 999

# a method

def read\_number(self):

print(self.num)

# creating object of the class

obj = DemoClass()

# calling the instance method using the object obj

obj.read\_number()

Output:

999

**2.2 Python – Parameterized constructor example**

When we declare a constructor in such a way that it accepts the arguments during object creation then such type of constructors are known as Parameterized constructors. As you can see that with such type of constructors we can pass the values (data) during object creation, which is used by the constructor to initialize the instance members of that object.

class DemoClass:

num = 101

# parameterized constructor

def \_\_init\_\_(self, data):

self.num = data

# a method

def read\_number(self):

print(self.num)

# creating object of the class

# this will invoke parameterized constructor

obj = DemoClass(55)

# calling the instance method using the object obj

obj.read\_number()

# creating another object of the class

obj2 = DemoClass(66)

# calling the instance method using the object obj

obj2.read\_number()

Output:

55

66