Constructors

- Constructors are generally **used for instantiating an object (initializing all variables of an object)**
- The **task of constructors is to initialize (assign values)** to the data members of the class when an object of the class is created.
- In Python, **the __init__() method is called the constructor.**
- The constructor **is always called when an object is created.**
- Without constructors, all objects will have the same values and no object will be unique.

Syntax:

```
class classname:
    def __init__(self):
        # body of the constructor
```

Creating a Constructor

To create a constructor in Python, we need to define a special method called __init__() inside our class.

__init__()

- **The __init__ method is called as the special method in Python whose name 'init' is preceded and followed by the double underscore sign.**
- **It is called as the constructor method in Python.**
- **It is used to initializing the object of a class.**
- **It is always executed when the object of the class is created.**
- Like normal methods, **the __inti__ also contains a set of statements that are executed at the time of object creation.**

Example:

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  #Constructor or init method
  def __init__(self, name, age):
     self.name = name
     self.age = age

p1 = Person("John", 36)
p1.name
p1.age
```

Output:

```
John
36
```

Example:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " , self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Output:

Hello My name is John

Note: We can use any name instead of self.

Example:

Use the words *mysillyobject* and *abc* instead of *self*:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " , abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Output:

Hello My name is John

Types of constructors

Constructors are categorized into three types:

1. Non-Parameterized Constructor
2. Parameterized Constructor
3. Default Constructor

A. Non-Parameterized constructor:

- Constructors with no parameters other than self are called Non-Parameterized Constructors.
  - ✓ It does not accept any arguments.
  - ✓ Its definition has only one argument which is a reference to the object of the class.

Example

```
class A:
   def __init__(self):
      self.a = "Python for DS"

   def print(self):
      print(self.a)

obj = A()
obj.print()
```

Output :

Python for DS

Example:

```
class Dress:
   def __init__(self):
      self.cloth = "Cotton"
      self.type = "T-Shirt"

   def get_details(self):
      print("Cloth = “, self.cloth)
      print(“\nType =”, self.type)

t_shirt = Dress()
t_shirt.get_details()
```

Output

Cloth = Cotton

Type = T-Shirt

B. Parameterized constructor:

- Constructors with at least two parameters including self are called Parameterized Constructors.
- ✓ It takes its first argument a self parameter which is as a reference to the object of the class.
- ✓ The rest of the arguments are provided by the programmer while creating the object.

Example:

```
class A:
        # Class Variable
        x = “Python”
        y = “DS”
  def __init__(self, name, age):
        # Instance Variable
        self.name = name
        self.age = age
  def func(self):
        print("Hello:", self.name)
        print("Your age is:", self.age)

obj = A(‘ABCD’, 19)
print(obj. x)
print(obj.y)
obj.func()
```

Output:

```
Python
DS
Hello: ABCD
Your age is: 19
```

- **Example:**

```
class Dress:
  def __init__(self, cloth, type, quantity=5):
     self.cloth = cloth
     self.type = type
     self.quantity = quantity

  def get_details(self):
     print(self.quantity, self.cloth, self.type)

shirt = Dress("silk", "shirt", 20)
t_shirt = Dress("cotton", "t-shirt")

shirt.get_details()
t_shirt.get_details()
```

Output

```
20 silk shirt(s)
5 cotton t-shirt(s)
```

Example:

```
class Addition:
   first = 0
   second = 0
   answer = 0
   [
   def __init__(self, f, s):
      self.first = f
      self.second = s

   def display(self):
      print("First number = " , self.first)
      print("Second number = " , self.second)
      print("Addition of two numbers = " , self.answer)

   def calculate(self):
      self.answer = self.first + self.second

obj = Addition(1000, 2000)
obj.calculate()
obj.display()
```

Output:
First number = 1000
Second number = 2000
Addition of two numbers = 3000

C. Default Constructor in Python

- If the user do not define a constructor, then a non-parameterized constructor with an empty body is initialized while creating the object of the class. This constructor is called Default Constructor.

Example:

```
class Dress:
  cloth = "silk"
  type = "shirt"
  def details(self):
     print("A", self.cloth, self.type)

shirt = Dress()
shirt.details()
```

Output
A silk shirt

Note:
What happen if we declare the two same constructors in the class?
Internally, the object of the class will always call the last constructor if the class has multiple constructors.
Example

```
class Student:
   def __init__(self):
      print("The First Constructor")
   def __init__(self):
      print("The second contructor")
 st = Student()
```

Output:
The second contructor

Additional Points:

Example:

```
class car:
   def __init__(self,modelname, year):
      self.modelname = modelname
      self.year = year
   def display(self):
      print(self.modelname,self.year)

c1 = car("Toyota", 2016)
c1.display()
```

Example:

```
#Creating more than one object of a class
class employee():
   def __init__(self,name,age,id,salary):   //creating a function
      self.name = name // self is an instance of a class
      self.age = age
      self.salary = salary
      self.id = id

emp1 = employee("harshit",22,1000,1234) //creating objects
emp2 = employee("arjun",23,2000,2234)
```

Note: We can use any name instead of self.

Example:

```
#Use the words mysillyobject and abc instead of self:
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " , abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Output:

```
Hello My name is John
```

Note:

Example:

```
#Passing the wrong number of arguments.
class Car:
   def __init__(self, name, mileage):
      self.name = name
      self.mileage = mileage
Honda = Car("Honda City")
print(Honda)
```

Output:

Error

Example:

```
#Order of the arguments
class Car:
   def __init__(self, name, mileage):
      self.name = name
      self.mileage = mileage

   def description(self):
      return f"The {self.name} car gives the mileage of {self.mileage}km/l"
Honda = Car(24.1,"Honda City")
print(Honda.description())
```

Note:

To add two numbers using function:

```
def add_number(n1,n2):
   sum = n1 + n2;
   return sum;
num1 = 30
num2 = 20
print("The sum of two number is",add_number(num1,num2))
```

To add two numbers using function inside a class:

```
class MyClass:
    a = 10
    b = 20
    def add(self):
        sum = self.a + self.b
        print(sum)

ob = MyClass()
print(ob.a)
print(ob.b)
ob.add()
```

To add two numbers using Constructors:

```
class add:
   def __init__(self,number1,number2):
      self.addition = number1 + number2

addnumbers = add(10,15)
print(addnumbers.addition)
```

Note:

Constructors with Positional Arguments

Positional arguments pass values to parameters depending on their position.

#Example: Positional Arguments

```
class Dress:
  def __init__(self, type, price):
    self.type = type
    self.price = price
  def details(self):
    print("A", self.type, "costs Rs." , self.price)
shirt = Dress("shirt", 50)
shirt.details()
```

Output

A shirt costs Rs.50

Constructors with keyword arguments

Arguments which should be passed by using a parameter name and an equal to sign are called keyword arguments. They are independent of position and dependent on the name of the parameter.

Example:

```
class Dress:
  def __init__(self, type, price):
    self.type = type
    self.price = price

  def details(self):
    print("A", self.type, "costs Rs." , self.price)

t_shirt = Dress(price=150, type="t-shirt")
print(t_shirt.details())
```

Constructors with Arbitrary Arguments

Multiple arguments which are passed into a single indefinite length tuple are called arbitrary arguments. We use arbitrary arguments when we don't know the number of arguments that will be passed to the function.

Example:

```
class Dress:
  def __init__(self, *types):
    self.types = types
  def details(self):
    return self.types
t_shirt = Dress("shirt", "t-shirt")
print(t_shirt.details())
```

Arbitrary Arguments

```
class Dress:
  def __init__(self, *types):
    self.types = types
  def details(self):
    return self.types
t_shirt = Dress("shirt", "t-shirt")
print(t_shirt.details())

class Square:
  def __init__(self, length):
    self.length = length
    self.area = self.find_area()
  def find_area(self):
    return self.length * self.length
obj = Square(5)
print(obj.area)
```

Count Objects Using Constructor in Python

```
class Dress:
  def __init__(self):
     Dress.no_of_dresses += 1
  no_of_dresses = 0
print("Before Creating objects")
print(f"No of objects = {Dress.no_of_dresses}")
for _ in range(10):  # To create 10 objects
  Dress()
print("After Creating objects")
print(f"No of objects = {Dress.no_of_dresses}")
```

Output:

```
Before Creating objects
No of objects = 0
After Creating objects
No of objects = 10
```

Delete the Object

We can delete the properties of the object or object itself by using the del keyword.

Example

```
class Employee:
   id = 10
   name = "John"
   def display(self):
      print("ID: %d \nName: %s" % (self.id, self.name))

emp = Employee()
 # Deleting the property of object
del emp.id
# Deleting the object itself
del emp
emp.display()
```