

## 19. PYTHON – OBJECT ORIENTED PROGRAMMING

### Table of Contents

<b>1. Is Python follows Functional approach or Object-oriented approach?</b>	2
<b>2. Features of Object-Oriented Programming System</b>	2
<b>3. class:</b>	3
3.1. Def1:	3
3.2. Def2:	3
<b>4. How to define or create a class?</b>	4
<b>5. Brief discussion about class</b>	4
<b>6. object</b>	7
6.1. Why should we create an object?	7
6.2. What is an object?	7
6.3. Syntax to create an object	8
<b>7. Constructor</b>	10
7.1. What is the main purpose of constructor?	10
7.2. When constructor will be executed?	11
7.3. How many times Constructor will executes?	11
7.4. Types of constructors	12
7.5. Constructor without parameters	12
7.6. Creating parameterised constructor	13
<b>8. Difference between method and constructor</b>	17
<b>9. Instance variables:</b>	18
9.1. What is instance variable?	18
9.2. Separate copy instance variable for every object	18
9.3. Declaring & accessing instance variables	19
<b>10. Instance methods</b>	20
<b>11. self pre-defined variable</b>	21

### 19. PYTHON – OBJECT ORIENTED PROGRAMMING

- ✓ Object-Oriented Programming is a methodology to design software by using classes and objects.
- ✓ It simplifies the software development and maintenance by providing the below features,

#### 1. Is Python follows Functional approach or Object-oriented approach?

- ✓ Python supports both functional and object-oriented programming.

#### 2. Features of Object-Oriented Programming System

- ✓ class
- ✓ object
- ✓ constructor
- ✓ Inheritance & etc...

### 3. class:

#### 3.1. Def1:

- ✓ A class is a model for creating an object and it does not exist physically.

#### 3.2. Def2:

- ✓ A class is a specification (idea/plan/theory) of properties and actions of objects.

#### Syntax

```
class NameOfTheClass:  
    1. constructor  
    2. properties (attributes)  
    3. actions (behaviour)
```

- ✓ We can create class by using **class** keyword.
- ✓ class can contain,
  - constructor
  - properties
  - actions
- ✓ Properties also called as variables.
- ✓ Actions also called as methods.

### 4. How to define or create a class?

- ✓ A python class may contain the below things,

#### Syntax

```
class NameOfTheClass:  
    """ documentation string """
```

1. Constructor
2. Variables
  1. instance variables
3. Methods
  1. instance methods

### 5. Brief discussion about class

- ✓ We can create class by using **class** keyword.
- ✓ class keyword follows the **name of the class**.
- ✓ After name of the class we should give **colon :** symbol.
- ✓ After **:** colon symbol in next line we should provide the **indentation**, otherwise we will get error.
- ✓ class can contain,
  - **Constructor** are used for initialization purpose
  - **Variables** are used to represent the data.
    - **instance** variables
  - **Methods** are used to represent actions.
    - **instance** methods

### Class naming convention

- ✓ While writing a class we need to follow the naming convention to meet real time standards,
  - class names should start with upper case and remaining letters are in lower case.
    - **Example:** Student
  - If name having multiple words, then every inner word should start with upper case letter.
    - **Example:** StudentInfo

### Note

- ✓ Documentation string represents description of the class. Within the class doc string is always optional.

**Program**     Define a class  
**Name**        demo1.py

```
class Employee:
    def display(self):
        print("Hello My name is Daniel")
```

**output**

### Make a note

- ✓ In above program, when we run then we will not get any output because we didn't call display method
- ✓ Above program Employee represents a **class** which is defined by developer.
- ✓ Developer defined only one method as display(self)
- ✓ Method we can define by using **def** keyword.
- ✓ Methods means it's just like a functions to perform an operations

### Kind info:

- ✓ Writing a class is not enough; we should know how to use the variables and methods.

### So,

- ✓ We need to create an object to access instance data(variables/methods) of a class.

### 6. object

#### 6.1. Why should we create an object?

- ✓ As per requirement we used to define variables and methods in a class.
- ✓ These variables and methods hold the data or values.
- ✓ When we create an object for a class, then only data will be store for the data members of a class.

#### 6.2. What is an object?

##### Definition 1:

- ✓ Instance of a class is known as an object.
  - Instance is a mechanism to allocate enough memory space for data members of a class.

##### Definition 2:

- ✓ Grouped item is known as an object.
  - Grouped item is a variable which stores more than one value.

##### Definition 3:

- ✓ Real world entities are called as objects.

#### Make some notes

- ✓ An object exists physically in this world, but class does not exist.

### 6.3. Syntax to create an object

#### Syntax

```
nameoftheobject = nameoftheclass()
```

#### Example

```
emp = Employee()
```

#### Program Name

Creating a class and object  
demo2.py

```
class Employee:
    def display(self):
        print("Hello my name is Daniel")

emp = Employee()
emp.display()
```

#### output

```
Hello my name is Daniel
```



**Program Name**      Creating a class and object  
demo2.py

```
class Employee:
    def display(self):
        print("Hello my name is Daniel")

    def teaching(self):
        print("I like teaching")

emp = Employee()

emp.display()
emp.teaching()
```

**output**

```
Hello my name is Daniel
I like teaching
```

### Make a note

- ✓ We can create object for class.
- ✓ In the above example **emp** is object name.
  - emp is just like a variable
- ✓ above example, display(self) is instance method.
  - To access instance method, we should create an object
  - So, we are accessing instance methods by using object name

### 7. Constructor

- ✓ Constructor is a special kind of method in python.
- ✓ So, we can create constructor by using **def** keyword
- ✓ The name of the constructor should be **\_\_init\_\_(self)**
  - Two underscore symbols before and after init with self as parameter
- ✓ self should be first parameter in constructor,

#### Syntax

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

#### 7.1. What is the main purpose of constructor?

- ✓ The main purpose of constructor is to initialize instance variables.

### 7.2. When constructor will be executed?

- ✓ Constructor will be executed automatically at the time of object creation.

**Program Name**      Creating a constructor  
demo3.py

```
class Employee:
    def __init__(self):
        print("constructor is executed")

emp = Employee()
```

**output**  
constructor is executed

### 7.3. How many times Constructor will executes?

- ✓ If we create object in two times then constructor will execute two times.

**Program Name**      Creating a constructor  
demo3.py

```
class Employee:
    def __init__(self):
        print("constructor is executed")

emp1 = Employee()
emp2 = Employee()
```

**output**  
constructor is executed  
constructor is executed

### 7.4. Types of constructors

- ✓ Based on parameters constructors can be divided into two types,
  1. Constructor without parameters
  2. Constructor with parameters

### 7.5. Constructor without parameters

- ✓ If constructor having no parameters, then at least it should contain **self** as one parameter.

#### Syntax

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

**Program**      Creating a constructor  
**Name**          demo3.py

```
class Employee:
    def __init__(self):
        print("constructor is executed")

emp = Employee()
```

**output**  
constructor is executed

### 7.6. Parameterised constructor

- ✓ Based on requirement constructor can contain any number of parameters.

### 7.6. Creating parameterised constructor

- ✓ By default, first parameter should be self to constructor.
- ✓ Constructor can contain more parameters along with **self**
- ✓ If constructor having more parameters, then the first parameter should be **self** and remaining parameters will be next.

#### Syntax

```
class NameOfTheClass:  
    def __init__(self, parameter1, parameter2):  
        body of the constructor
```

### Note: One parameterised constructor

**Program Name**      One parameterised constructor  
demo6.py

```
class Employee:  
    def __init__(self, number):  
        self.number= number  
        print("Employee id is: ", self.number)
```

```
e1 = Employee(1)  
e2 = Employee(2)  
e3 = Employee(3)
```

#### output

```
Employee id is: 1  
Employee id is: 2  
Employee id is: 3
```

### Note: One parameterised constructor

- ✓ If constructor having one parameter, then during object creation we need to pass one value.

### Can i write a constructor and an instance method in a single program?

- ✓ Yes we can write constructor and instance method both in single program.
- ✓ Here constructor purpose is to initialize instance variables, and method purpose is to perform operations.

### Two parameterised constructor

**Program Name**      One parameterised constructor and instance method  
demo7.py

```
class Employee:
    def __init__(self, number):
        self.number = number

    def display(self):
        print("Employee id is:", self.number)

e1 = Employee(1)
e2 = Employee(2)
e3 = Employee(3)

e1.display()
e2.display()
e3.display()
```

**output**

```
Employee id is: 1
Employee id is: 2
Employee id is: 3
```

### Note: Access instance variable in instance method

- ✓ Inside instance method we can access instance variables by using self.

### Two parameterised constructor

**Program Name** Two parameterised constructor and instance method  
demo8.py

```
class Employee:
    def __init__(self, number, name):
        self.number = number
        self.name = name

    def display(self):
        print("Hello my id is :", self.number)
        print("My name is :", self.name)

e1=Employee(1, 'Daniel')
e1.display()

e2=Employee(2, 'Arjun')
e2.display()
```

### Output

```
Hello my id is: 1
My name is: Daniel

Hello my id is: 2
My name is: Arjun
```

### Note: Two parameterised constructor

- ✓ If constructor having two parameters, then during object creation we need to pass two values

### Three parameterised constructor

**Program Name**      Three parameterised constructor and instance method  
demo9.py

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

    def display(self):
        print("Hello my id is :", self.number)
        print("My name is :", self.name)
        print("My age is sweet :", self.age)

e1=Employee(1, 'Daniel', 16)
e1.display()

e2=Employee(2, 'Arjun', 17)
e2.display()

e3=Employee(3, 'Prasad', 18)
e3.display()
```

### Output

```
Hello my id is: 1
My name is: Daniel
My age is sweet: 16

Hello my id is: 2
My name is: Arjun
My age is sweet: 17

Hello my id is :3
My name is: Prasad
My age is sweet: 18
```



### Note: Three parameterised constructor

- ✓ If constructor having three parameters, then during object creation we need to pass three values.

### 8. Difference between method and constructor

Method	Constructor
✓ Methods are used to perform operations or actions	✓ Constructors are used to initialize the instance variables.
✓ Method name can be any name.	✓ Constructor name should be <code>__init__(self)</code>
✓ Methods we should call explicitly to execute	✓ Constructor automatically executed at the time of object creation.

### 9. Instance variables:

#### 9.1. What is instance variable?

- ✓ If the value of a variable is changing from object to object such type of variables is called as instance variables.

#### 9.2. Separate copy instance variable for every object

- ✓ For every object a separate copy of instance variables will be created.

**Program**      Instance variables  
**Name**          demo10.py

```
class Student:
    def __init__(self, name, number):
        self.name=name
        self.number=number

s1 = Student('Daniel', 101)
s2 = Student('Prasad', 102)

print("Studen1 info:")
print("Name: ", s1.name)
print("Id : ", s1.number)

print("Studen2 info:")
print("Name: ", s2.name)
print("Id : ", s2.number)
```

**Output**

```
Studen1 info:
Name: Daniel
Id: 101

Studen2 info:
Name: Prasad
Id: 102
```

### 9.3. Declaring & accessing instance variables

- ✓ We can declare instance variables inside constructor
- ✓ We can access instance variables by using object name

**Program Name**      Initializing instance variables inside Constructor  
demo11.py

```
class Employee:
    def __init__(self):
        self.eno = 10
        self.ename = "Daniel"
        self.esal = 10000

emp = Employee()

print("Employee number:", emp.eno)
print("Employee name:", emp.ename)
print("Employee salary:", emp.esal)
```

**output**

```
Employee number: 10
Employee name : Daniel
Employee salary : 10000
```

### 10. Instance methods

- ✓ Instance methods are methods which act upon the instance variables of the class.
- ✓ Instance methods are bound with instances or objects, that's why called as instance methods.
- ✓ The first parameter for instance methods is **self** variable.
- ✓ Along with **self** variable it can contains other variables as well.

**Program**      Instance methods  
**Name**          demo13.py

```
class Demo:
    def __init__(self, a):
        self.a=a

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

d=Demo(10)
d.m()
```

**Output**  
10

### 11. self pre-defined variable

- ✓ self is a predefined variable in python, this variable belongs to current class object.
  - self variable we can use to create below things,
    - Constructor
    - Instance variable
    - Instance methods
- ✓ Constructor
  - By using self, we can initialize the instance variables inside constructor `__init__(self)`
- ✓ Instance variable
  - By using self, we can declare and access instance variables,
- ✓ Instance methods
  - By using self, we can create instance methods.