# Python Programming


**RGM College of Engineering & Technology (Autonomous)**

Department of Computer Science & Engineering

# PYTHON'S OBJECT ORIENTED PROGRAMMING - 3

# Guido Van Rossum

# Agenda:

I. Basic Idea about Types of Variables - Instance, Static and Local

II. Basic Idea about Types of Methods - Instance, Class and Static

III. Various places to declare Instance Variables

IV. How to access instance variables?

V. How to delete instance variables?

# I. Types of Variables used inside Python class

**Inside Python class 3 types of variables are allowed.**

1. Instance Variables (Object Level Variables)
2. Static Variables (Class Level Variables)
3. Local variables (Method Level Variables)

# 1. Instance Variables:

- If the value of a variable is varied from object to object, then such type of variables are called instance variables (or) object level variables.

  **For example**, 'name' and 'rollno' of the student.

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

**Eg:**

```
class student:
    def __init__(self,name,rollno):
        self.name = name
        self.rollno = rollno
```

## 2. Static Variables:

- If the value of a variable is not varied from object to object, then it is not recommended to declare those variables as instance variables. We have to declare at class level as static variables (or) class level variables. For example, '**school_name**' of students.
- In the case of instance variables, for every object a separate copy will be created. But in the case of static variable at class level only one copy will be created and shared by all objects of that class.
- Most of the times, static variable should be declared within the class directly.
- We can access static variables either by class name or by object reference. But recommended to use class name.

**Eg:**

```
class student:
    school_name = 'RGM'                    # Static variable
    def __init__(self,name,rollno):
        self.name = name
        self.rollno = rollno               # Instance Variables
```

## 3. Local Variables or Method level Variables:

- Sometimes to meet temporary requirements of programmer, we can declare variables inside a method directly, such type of variables are called local variables (or) temporary variables (or) Method level Variables.
- Local variables will be created at the time of method execution and destroyed once method completes it's execution.
- Local variables of a method cannot be accessed from outside of method.

**Eg:**

```
class student:
    college_name = 'RGM'                    # Static variable
    def __init__(self,name,rollno):
        self.name = name
        self.rollno = rollno               # Instance Variables

    def info(self):
        x = 10                             # Local Variables
        for i in range(x):                 # 'i' is also local variable
            print(i)
```

## II. Basic Idea about Types of Methods - Instance, Class and Static

There are three types of methods are allowed with in the python class:

1. Instance Method
2. Class Method
3. Static Method

# 1. Instance Method

- The name itself indicates that, Instance means object. That is, Object related method is called Instance Method.
- Inside a method, if we are trying to access instance variables, then that method always talks about a particular object and that method should be declared as an instance method.
- The first argument to the Instance method is always self, which is the reference variable to the current object.
- Inside instance method declaration, we have to pass self variable.

**Blind Rule:** For any method, if the first argument is self, that is an Instance Method.

## 2. Class Method

- Inside a method, if we are using only static variables and if we are not using any instance variable, then that method is no way related to a particular object and it is class level method. Such type of methods we have to declare as class methods.
- We have to declare class method with @classmethod decorator.
- The first argument to the class method is always **cls**, which is reference variable to the class object.
- For every class, one special object will be created by PVM to maintain class level information, which is nothing but class level object. **cls** is the reference variable pointing to that class level object.

## Demo Program on Class Methods:

```
class Animal:
    legs=4
    @classmethod
    def walk(cls,name):
        print('{} walks with {} legs...'.format(name,cls.legs))

Animal.walk('Dog')
Animal.walk('Cat')
```

```
Dog walks with 4 legs...
Cat walks with 4 legs...
```

## Another Example:

```
class Test:
    @classmethod
    def m1(cls):
        print(id(cls))

print(id(Test))

Test.m1()
```

```
1509338637784
1509338637784
```

## 3. Static Method

- Inside a method, if we are not using any instance or static variables, such type of methods are general utility methods and these methods we have to declare as static Methods.
- Static methods should be declared by using @staticmethod decorator.

## Demo Program on various types of Methods:

```
class student:
    college_name = "RGM"

    def __init__(self,name,rollno):
        self.name = name
        self.rollno = rollno

    def getStudentInfo(self):                    # Instance Method
        print('Student Name : ',self.name)
        print("Student Roll No. : ",self.rollno)

    @classmethod                                 # Class Method
    def getCollegeInfo(cls):
        print("College Name : ",cls.college_name)

    @statimethod                                 # Static Method
    def getSum(a,b):
        return a+b
```

# III. Various places to declare Instance Variables

**Key Points to Recap:**

- If the value of a variable is varied from object to object, then such type of variables are called instance variables.
- For every object a separate copy of instance variables will be created.
- Most of the times Instance variables will be declared inside the constructor by using self variable.

**Where we can declare Instance variables:**

1. Inside Constructor by using self variable
2. Inside Instance Method by using self variable
3. Outside of the class by using object reference variable

## 1. Inside Constructor by using self variable:

- We can declare instance variables inside a constructor by using self keyword.
- Once we creates object, automatically these variables will be added to the object.

**Eg 1:**

```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20       # Instance Variables

t = Test()
print(t.__dict__)   # __dict__ is a predefined dictionary, contains Instance variables

                    # their corresponding values of a particular object.
```

```
{'a': 10, 'b': 20}
```

**Eg 2:**

```
class Employee:

    def __init__(self):
        self.eno=100
        self.ename='Karthikeya'
        self.esal=100000

e=Employee()
print(e.__dict__)
```

```
{'eno': 100, 'ename': 'Karthikeya', 'esal': 100000}
```

## 2. Inside Instance Method by using self variable:

- We can also declare instance variables inside instance method by using self variable.
- If any instance variable declared inside instance method, that instance variable will be added once we call that method.
- If we are not calling that instance method then those variables won't be added to the object. See the below example code,

**I.**

```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20       # Instance Variables

    def m1(self):
        self.c = 30       # Instance Variable

t = Test()
print(t.__dict__)
```

```
{'a': 10, 'b': 20}
```

**II.**

```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20        # Instance Variables

    def m1(self):
        self.c = 30        # Instance Variable

t = Test()
t.m1()
print(t.__dict__)
```

```
{'a': 10, 'b': 20, 'c': 30}
```

### 3. Outside of the class by using object reference variable:

- We can also add instance variables outside of a class to a particular object.
- In other programming languages like Java, it is not possible.

**Eg:**

```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20       # Instance Variables

    def m1(self):
        self.c = 30       # Instance Variable

t = Test()          # a,b will be added to object 't'
t.m1()              # c will be added to object 't'
t.d = 40            # d will be added to object 't'

print(t.__dict__)   # 4 instance variables will be displayed.
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
```

- **Very Important Point**
- In Python, the number of instance variables are varied from an object to object.
- But, in java, the number of instance variables are same for every object.

**Eg:**

```
class Test:
    def __init__(self):
        self.a = 10
        self.b = 20       # Instance Variables

    def m1(self):
        self.c = 30       # Instance Variable

t1 = Test()               # a,b will be added to object 't1'
t1.m1()                   # c will be added to object 't'
t1.d = 40                 # d will be added to object 't'

t2 = Test()

print("t1 dict : ",t1.__dict__)
print("t2 dict : ",t2.__dict__)
```

```
t1 dict :  {'a': 10, 'b': 20, 'c': 30, 'd': 40}
t2 dict :  {'a': 10, 'b': 20}
```

## IV. How to access instance variables?

- We can access instance variables with in the class by using self variable.
- Outside of the class by using object reference.

**Eg 1:**

```
class Test:
    def __init__(self):
        self.a=10
        self.b=20

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

t=Test()
t.display()
```

```
10
20
```

**Eg 2:**

```
class Test:
    def __init__(self):
        self.a=10
        self.b=20

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

t=Test()
t.display()
print(t.a,t.b)
```

```
10
20
10 20
```

## V. How to delete instance variables?

1. Within a class we can delete instance variable as follows:

**Syntax:**

del self.variableName

**Eg:** del self.a

2. From outside of class we can delete instance variables as follows:

**Syntax:**

del objectreference.variableName

**Eg:** del t.a

3. We can delete multiple instance variables also.

**Eg:** del t.a,t.b,t.c

**Note:** In java, this facility is not available.

## Eg 1:

```
class Test:
    def __init__(self):
        self.a=10
        self.b=20
        self.c=30
        self.d=40

    def m1(self):
        del self.d

t=Test()
print(t.__dict__)          # a,b,c,d
t.m1()
print(t.__dict__)          # a,b,c
del t.c
print(t.__dict__)          # a,b
```

```
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
{'a': 10, 'b': 20, 'c': 30}
{'a': 10, 'b': 20}
```

## Eg 2:

```
class Test:
    def __init__(self):
        self.a=10
        self.b=20
        self.c=30
        self.d=40

    def m1(self):
        del self.d

t=Test()
print("t :",t.__dict__)          # a,b,c,d
t.m1()
print("t :", t.__dict__)          # a,b,c
t1 = Test()
del t1.b,t1.d
print("t1 :", t1.__dict__)          # a,c
```

```
t : {'a': 10, 'b': 20, 'c': 30, 'd': 40}
t : {'a': 10, 'b': 20, 'c': 30}
t1 : {'a': 10, 'c': 30}
```

**Note:**

- The instance variables which are deleted from one object, will not be deleted from other objects.

**Eg:**

```
class Test:
    def __init__(self):
        self.a=10
        self.b=20
        self.c=30
        self.d=40

t1=Test()
t2=Test()
del t1.a
print(t1.__dict__)
print(t2.__dict__)
```

```
{'b': 20, 'c': 30, 'd': 40}
{'a': 10, 'b': 20, 'c': 30, 'd': 40}
```

- **Very Important Point:**
  - If we change the values of instance variables of one object then those changes won't be reflected to the remaining objects, because for every object we are separate copy of instance variables are available.

**Eg:**

```
class Test:
    def __init__(self):
        self.a=10
        self.b=20

t1=Test()
t1.a=888
t1.b=999
t2=Test()
print('t1:',t1.a,t1.b)
print('t2:',t2.a,t2.b)
```

```
t1: 888 999
t2: 10 20
```

# Any question?

If you try to practice programs yourself, then you will learn many things automatically

Spend few minutes and then enjoy the study

# Thank You