



**Marwadi**  
University

01CE1705-Programming with Python

# Unit-6 Classes and Objects

Prof. Chetan Chudasama  
Computer Engineering Department



# Outline

- Classes and objects
- Importance of self
- `__init__()` method
- Instance Variable
- Access Specifier
- Instance Method
- Class Method

# class

- A Python class is a group of attributes and methods.

What is Attribute:-Attributes are represented by variable that contains data.

What is Method:-Method performs an actions or task. It is similar to function.

## Defining a class in python

- Like function definitions begin with the def keyword in Python, class definitions begin with a class keyword.
- The first string inside the class is called docstring and has a brief description of the class. Although not mandatory, this is highly recommended.

Example:

```
class Student:
```

```
    """this template contain student name,age,rollno"""
```

## Rules:

- The class name can be any valid identifier.
- It can't be python reserved word.

- A valid class name starts with a letter, followed by any number of letter, numbers or underscores.
- A class name generally starts with Capital Letter.

### Object:-

- Object is class type variable or class instance. To use a class, we should create an object to the class.
- Object creation represent allotting memory necessary to store actual data of the variables.
- Each time we created an object of a class a copy of variables defined in the class is created.
- In other words we can say that each object of a class has its own copy of data members defined in the class.

**Syntax:-**object\_name=class\_name()

**Example:**

- We can think of a class as a sketch of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.
- As many houses can be made from a house's blueprint, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called instantiation.

**Example:**

```
class Employee:
    def __init__(self,fname,lname,pay):
        self.fname=fname
        self.lname=lname
        self.pay=pay
        self.email=fname+'.'+lname+'@marwadiuniversity.ac.in'
```

```
def fullName(self):  
    return '{ } { }'.format(self.fname,self.lname)  
emp_1=Employee('Rajesh','Patel',45000)  
emp_2=Employee('Harsh','Raval',41000)  
print(emp_1.fullName())  
print(emp_2.fullName())
```

### **How it works**

```
emp_1=Employee()
```

- A block of memory is allocated on heap. The size of allocated memory is to be decided from the attributes and methods available in the class(Employee).
- After allocating memory block, the special method `__init__()` is called internally. This methods stores data into the variables.
- The allocated memory location address of the object is returned.
- The memory location is passed to self.

## **Accessing class member using object**

- We can access variable and method of a class using class object.

Example:-


`object_name.variable_name`

`object_name.method_name()`

`object_name.method_name(parameter_list)`



# self parameter

- Whenever we create a class in Python, the programmer needs a way to access its attributes and methods. In most languages, there is a fixed syntax assigned to refer to attributes and methods; for example, C++ uses this keyword for reference.
- In Python, the word self is the first parameter of methods that represents the instance of the class. Therefore, in order to call attributes and methods of a class, the programmer needs to use self.
- It binds the attributes with the given arguments.
- The reason why we use self is that Python does not use the '@' syntax to refer to instance attributes.
- In Python, we have methods that make the instance to be passed automatically, but not received automatically. 
- **self is not a keyword** in Python. It is a parameter in function and the user can use a different parameter name in place of it. Although it is advisable to use self because it increases the readability of code.

# \_\_init\_\_ method

- It is one of the reserved methods in python.
- In object oriented programming, it is known as a constructor.
- The task of constructor is to initialize(assign values) to the data member of the class when an object of the class is created.
- Like methods, a constructor also contains a collection of statements that are executed at the time of object creation.
- The method is useful to do any initialization you want to do with your object.
- It is used only within classes.

Example:

```
class Person:
```

```
    def __init__(self, name, age):
```

```
        self.name = name
```

```
        self.age = age
```

```
p1 = Person("Vaibhav", 30)
```

```
print(p1.name,p1.age)
```

- The `__init__()` method has two underscores (`__`) on each side. Therefore, the `__init__()` is often called **dunder init**. The name comes abbreviation of the double underscores `init`.
- The double underscores at both sides of the `__init__()` method indicate that **Python will use the method internally**. In other words, you should not explicitly call this method.
- If the `__init__` has parameters other than the `self`, you need to pass the corresponding arguments when creating a new object. Otherwise, you'll get an error.

### **`__init__` method with default parameters**

- The `__init__()` method's parameters can have default values.

Example:-

```
class Person:
```

```
    def __init__(self, name, age=22):
```

```
        self.name = name
```

```
        self.age = age
```

```
p1=Person("Vaibhav")
```

# Instance Variable



- If the value of variable different from object to object, then such type of variables are called instance variable.
- For every object,seperate copy of instance variable is created.
- Instance variables are not shared by objects.
- Every object has its own copy of the instance attribute.
- This means that for each object of class, the instance variable value is different.
- Instance variables are used within the instance method.
- We can access the instance variable using the object and dot(.) operator.

## Example:

class Student:

```
    def __init__(self,rollNo):  
        self.rollNo=rollNo#instance variable
```

```
stud_1=Student(17)
print(stud_1.rollNo)#Access instance variable
stud_2=Student(18)
print(stud_2.rollNo)
```

- As mention in above code, instance variable are declared inside a method using the self keyword.
- We can use constructor to define and initialize the instance variable.
- When we created an object, we passed the values to the instance variables using a constructor.
- We can modify the value of the instance variable and assign a new value to it using object reference.
- When you change the instance variable's values of one object, the changes will not be reflected in the remaining objects because every object maintains a separate copy of the instance variable.

## Way to access Instance Variable

There are two ways to access the instance variable of class:

using self keyword

using getattr() method

} Imp

Example 1:-Access instance variable in the instance method using self keyword

class Student:

```
def __init__(self,name,age):
```

```
    self.name=name
```

```
    self.age=age
```

```
def show(self):
```

```
    print("Name is",self.name,"age is",self.age)
```

```
stud_1=Student("Raj",25)
```

```
stud_1.show()
```

## Example 2:-Access instance variable using getattr()

- Pass the object reference and instance variable name to the getattr() method to get the value of an instance variable.

Syntax:-getattr(Object, 'instance\_variable')

```
class Student:
```

```
    def __init__(self,name,age):
```

```
        self.name=name
```

```
        self.age=age
```

```
stud_1=Student("Raj",25)
```

```
print("Name is",getattr(stud_1,'name'))#use getattr instead of stud_1.name
```

```
print("Age is",getattr(stud_1,'age'))
```

## Instance Variables Naming Conventions

- Instance variable names should be all lower case.
- Words in an instance variable name should be separated by an underscore.
- Non-public instance variables should begin with a single underscore.

## Delete Instance Variable

- In Python, we use the del statement and delattr() function to delete instance variable.

Example 1:-Delete variable using del keyword

```
class Student:
```

```
    def __init__(self,name,age):
```

```
        self.name=name
```

```
        self.age=age
```

```
stud_1=Student("jay",23)
```

```
del stud_1.name
```

```
print(stud_1.name)#we try to access the deleted attribute,it raises an attribute error.
```



Example 2:-delete variable using delattr() function

```
class Student:
```

```
    def __init__(self,name,age):
```

```
        self.name=name
```

```
        self.age=age
```

```
stud_1=Student("jay",23)
```

```
delattr(stud_1,"name")#delattr(object_name,variable_name)
```

```
print(stud_1.name)
```

# Class Variable

- If the value of a variable is not different from object to object, such types of variables are called **class variables or static variables**.
- Class variables are **shared by all objects of a class**.
- Class variables are declared when a class is being constructed. They are **not** defined **inside any methods** of a class.

## Example:-

- In Student class, we can have different instance variables such as name and roll number because each student's name and roll number are different. But, if we want to include the school name in the student class, we must use the class variable instead of an instance variable as the school name is the same for all students.
- In Employee class, we can have different instance variable such as first\_name, last\_name, salary. But if we want to add salary\_increment (same for all employee) and number\_of\_employee, we must use class variable.

## Create class variables:-

It is placed below class header and before the constructor and other methods.

Example:-

```
class Employee:
```

```
    raise_amount=1.04
```

```
    num_of_employee=0
```

```
    def __init__(self,first,last,pay):
```

```
        self.first=first
```

```
        self.last=last
```

```
        self.pay=pay
```

```
        self.email=first+'.'+last+'@marwadiuniversity.ac.in'
```

```
        Employee.num_of_employee+=1
```

```
def show(self):  
    self.pay=int(self.pay*Employee.raise_amount)  
    print("Employee name "+self.first+" "+self.last)  
    print("Email "+self.email)  
    print("salary",self.pay)
```

```
emp_1=Employee("Rajesh","Patel",55000)
```

```
emp_1.show()
```

- class variable can store any type of data such as list,tuple,dictionary.

### **Accessing class variable:-**

- We can access static variables either by class name or by object reference, but it is recommended to use the class name.

- In Python, we can access the class variable in the following places
  - Access inside the constructor by using either self parameter or class name.
  - Access class variable inside instance method by using either self or class name
  - Access from outside of class by using either object reference or class name.

### **Modify Class Variables**

- Generally, we assign value to a class variable inside the class declaration. However, we can change the value of the class variable either in the class or outside of class.

# Access Specifier



- In high-level programming languages like C++, Java, etc., private, protected, and public keywords are used to control the access of class variables.
- Python has no such keywords.
- In python we use underscore "\_" symbol to specify the access modifier for specific data members and member functions in the class.
- Access modifiers decide whether other classes can use the variables or functions of a specific class or not.
- In Python, there are three types of access modifiers.

Public Access Modifier

Protected Access Modifier

Private Access Modifier

**public:-**

- All the variables and methods in python are by **default public**.
- Any instance variable in a class followed by the 'self' keyword ie. self.var\_name are public.
- Any member declared as public can be accessed from outside the class through an object.
- If variable is public we can modify their value.

**protected:-**

- The data member of the class is declared protected by adding a single underscore “\_”.
- The members of a class that are declared protected are only accessible to a class derived from it.
- We can access protected variable outside the class and modify their value.
- Actually, protected access modifiers are designed so that responsible programmer would identify by their name convention and do the required operation only on that protected class members or class methods.

**Example:-**

```
class A:
    _rollNo=9
    def _abc(self):
        print("I am from class A")
class B(A):
    def show(self):
        print(self._rollNo)
        self._abc()
obj_1=B()
obj_1.show()
```

**Output:-**

9

I am from class A



## **private**

- A double underscore “\_\_” makes the variable private.
- Private members of a class (variables or methods) are those members which are only accessible inside the class. We cannot use private members outside of class.
- It is also not possible to inherit the private members of any class (parent class) to derived class (child class).
- So, to access the private members of a class we have **name mangling** of private variables. Every member with a double underscore will be changed to ” **object.\_class\_\_variable** “ and then it can be accessed from outside the class.

```
class A:
    __rollNo=18
    def __abc(self):
        print("I am from class A")
    def show(self):
        self.__abc()#We are able to access private method in same class
class B(A):
    def display(self):
        print(self.__rollNo)#Not able to access private variable in other classes
        self.__abc()#Not able to access private method in other classes.
obj_1=B()
obj_1.display()
```

# Instance Method

- If we use **instance variable inside a method**, such method are called instance methods.
- Instance method is bound to the object of the class.
- To work with instance method we use self keyword. We use the self keyword as the first parameter to a method. The self refers to the current object.
- Any method we create in a class will automatically be created as an instance method unless we explicitly tell python that it is a class or static method.

Example:

```
class Student:
    def __init__(self,name,age):
        self.name=name#instance variable
        self.age=age#instance variable
    def show(self):#instance method
        print("Name",self.name,"age",self.age)

stud_1=Student("vaibhav",30)
stud_1.show()#call instance method
```

- We use an object and dot (.) operator to execute the block of code defined in the instance method.
- Inside any instance method, we can use self to access variable or method. We are unable to access it without a self parameter.
- An instance method can freely access attributes or even modify the value of attribute by using the self parameter.
- By using `self.__class__` we can access class attribute and change class state. Therefore instance method gives us control of changing the object as well as the class state.

# Static Method



- Static method don't pass anything automatically. They don't pass instance or class.
- It behave just like regular function except we include them in our class because they have some logical connection with the class.
- A static method is bound to the class and not the object of the class. Therefore, we can call it using the class name.
- Any method we create in a class will automatically be created as an instance method. We must explicitly tell Python that it is a static method using the `@staticmethod` decorator
- It `consume less memory as compare to instance method`.

Note:-Let's assume you have ten employee objects and if you create `gather_requirement()` as a instance method then Python have to create a ten copies of this method (separate for each object) which will consume more memory. On the other hand static method has only one copy per class.

```
class Employee:
    @staticmethod
    def is_workday(day):
        if day.weekday() == 5 or day.weekday() == 6:
            return False
        return True
import datetime
my_date=datetime.date(2016,7,10)
print(Employee.is_workday(my_date))
```

Thank You

