# Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

## Create a Class

To create a class, use the keyword `class`:

**Example :**

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
print(MyClass)
```

**Output :**

```
<class '__main__.MyClass'>
```

## Create Object

Now we can use the class named MyClass to create objects:

**Example :**

Create an object named p1, and print the value of x:

```
class MyClass:
  x = 5
p1 = MyClass()
print(p1.x)
```

**Output :**

```
5
```

## The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

**Example :**

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

```
class Person:

  def __init__(self, name, age):

    self.name = name

    self.age = age

p1 = Person("John", 36)

print(p1.name)

print(p1.age)
```

**Output :**

```
John
36
```

**Note:** The __init__() function is called automatically every time the class is being used to create a new objec

## Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

**Example :**

Insert a function that prints a greeting, and execute it on the p1 object:

```
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:** The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

## The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

**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
```

## Modify Object Properties

You can modify properties on objects like this:

**Example :**

Set the age of p1 to 40:

```
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.age = 40
print(p1.age)
```

**Output :**

```
40
```

# Python Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

**Parent class** is the class being inherited from, also called base class.

**Child class** is the class that inherits from another class, also called derived class.

Inheritance is the capability of one class to derive or inherit the properties from another class. The benefits of inheritance are:

1. It represents real-world relationships well.

2. It provides reusability of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.

3. It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A. Below is a simple example of inheritance in Python.

Different forms of Inheritance:

1. **Single inheritance:** When a child class inherits from only one parent class, it is called single inheritance.

**Example :**

```
class Person(object):
# Constructor
    def __init__(self, name):
        self.name = name
# To get name
    def getName(self):
        return self.name
# To check if this person is an employee
    def isEmployee(self):
        return False
# Inherited or Subclass (Note Person in bracket)
class Employee(Person):
# Here we return true
    def isEmployee(self):
        return True
# Driver code
emp = Person("Geek1") # An Object of Person
print(emp.getName(), emp.isEmployee())
emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

**Output :**

```
Geek1 False
Geek2 True
```

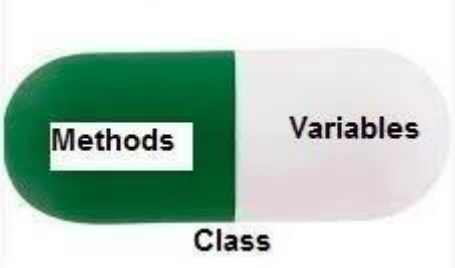
**2. Multiple inheritance:** When a child class inherits from multiple parent classes, it is called multiple inheritance. Unlike Java and like C++, Python supports multiple inheritance. We specify all parent classes as a comma-separated list in the bracket.

**3.**

## Encapsulation

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as **private variable**.

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.




Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name "sales section". Using encapsulation also hides the data. In this example, the data of the sections like sales, finance, or accounts are hidden from any other section.

# Polymorphism

Polymorphism means multiple forms. In python we can find the same operator or function taking multiple forms. It also useful in creating different classes which will have class methods with same name. That helps in re using a lot of code and decreases code complexity.

## Polymorphism in operators

The + operator can take two inputs and give us the result depending on what the inputs are. In the below examples we can see how the integer inputs yield an integer and if one of the input is float then the result becomes a float. Also for strings, they simply get concatenated. This happens automatically because of the way the + operator is created in python.

**Example :**

```
a = 23
b = 11
c = 9.5
s1 = "Hello"
s2 = "There!"
print(a + b)
print(type(a + b))
print(b + c)
print(type (b + c))
print(s1 + s2)
print(type(s1 + s2))
```

**Output :**

```
34
20.5
```

## Polymorphism in in-built functions

We can also see that different python functions can take inputs of different types and then process them differently. When we supply a string value to len() it counts every letter in it. But if we five tuple or a dictionary as an input, it processes them differently.

**Example :**

```
str = 'Hi There !'
tup = ('Mon','Tue','wed','Thu','Fri')
```

```
lst = ['Jan','Feb','Mar','Apr']
dict = {'1D':'Line','2D':'Triangle','3D':'Sphere'}
print(len(str))
print(len(tup))
print(len(lst))
print(len(dict))
```

**Output**

```
10
5
4
3
```