```python
# Python Objects and Classes :
#! => An object is simply a collection of data (variables) and methods
(functions) that act on those data.
#! =>  Similarly, a class is a blueprint for that object.

# To create a class, use the keyword class:
class Myclass:                # Class
    text = "Coding Ideas"
    print(text)
# Now we can use the class named MyClass to create objects:
obj1 = Myclass()              # Object
print(obj1.text)
```

```
Coding Ideas
Coding Ideas
```

```python
# Classes => Templates
# Object => Instance of the class.

#! But Why do we make Class ?
# Reason => Concept of DRY.
''' DRY => Do not Repeat yourself.
    It saves time and effort.
'''

class student:
    pass           # Blank Template.

student1 = student()
student2 = student()

print(student1)
print(student2)
'''
utput =>
<__main__.student object at 0x000001F4169FD990>   #! Address of
student1
<__main__.student object at 0x000001F4169FD8A0>   #! Address of
student2
'''
```

```
<__main__.student object at 0x000001F4169FD990>
<__main__.student object at 0x000001F4169FD8A0>
```

```python
class student:
    pass           # class => Blank Template.

student1 = student()       # Object 1
student2 = student()       # Object 2
```

```
student1.name = "Varun Dahiya"
student1.std =  12
student1.sec = "A"
student2.name = "Nivedita"
student2.std = 10
student2.sub = ['Physics', 'English']

print(student1.name,student1.std, student1.sec)
print(student2.name, student2.std, student2.sub)
print(student2.sec)                  # Error : AttributeError:
'student' object has no attribute 'sec'
```

```
Varun Dahiya 12 A
Nivedita 10 ['Physics', 'English']

----------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
last)
c:\Users\PKVidyarthi\Desktop\Data Science\Notes\Classes&Objects.ipynb
Cell 3 in <cell line: 16>()
     <a
href='vscode-notebook-cell:/c%3A/Users/PKVidyarthi/Desktop/Data
%20Science/Notes/Classes%26Objects.ipynb#ch0000007?line=13'>14</a>
print(student1.name,student1.std, student1.sec)
     <a
href='vscode-notebook-cell:/c%3A/Users/PKVidyarthi/Desktop/Data
%20Science/Notes/Classes%26Objects.ipynb#ch0000007?line=14'>15</a>
print(student2.name, student2.std, student2.sub)
---> <a
href='vscode-notebook-cell:/c%3A/Users/PKVidyarthi/Desktop/Data
%20Science/Notes/Classes%26Objects.ipynb#ch0000007?line=15'>16</a>
print(student2.sec)

AttributeError: 'student' object has no attribute 'sec'
```

```
class Employee:
    leaves = 10
    pass
rohit = Employee()
varun = Employee()

rohit.name = "Rohit Mishra"
rohit.salary = 700000
rohit.role = "Full Stack Developer"

varun.name = "Varun Dahiya"
varun.salary = 450000
varun.role = "Web Developer"
```

```
print(varun.salary)                     # Output => 450000
print(rohit.role)                       # Output => Full Stack Developer

print(rohit.leaves)                     # Output => 10
print(varun.leaves)                     # Output => 10
print(Employee.leaves)                  # Output => 10
# leaves for all are same because leaves is derived inside class.
# Now try to chnage the value of leaves using object rohit (object
'rohit' is derived outside class Employee).
rohit.leaves = 20
print(rohit.leaves)                     # Output => 20
print(varun.leaves)                     # Output => 10
print(Employee.leaves)                  # Output => 10
''' rohit.leaves makes a new instance variable for object rohit.
So only rohit.leaves will change. '''

Employee.leaves = 25
print(rohit.leaves)                     # Output => 20
print(varun.leaves)                     # Output => 25
print(Employee.leaves)                  # Output => 25
''' Now leaves for varun and employee all are changed except
'leaves.rohit'.
    Because value of rohit.leaves already declared outside the class
which is 20.'''
```

```
450000
Full Stack Developer
10
10
10
10
10
10
25
25
25
```

```
# Attribute => __dict__ =>  All objects in Python have an attribute
__dict__

#! __dict__ => A dictionary object containing all attributes defined
for that object itself.
#!          => The mapping of attributes with its values is done to
generate a dictionary.

class Employee:
    leaves = 10
    pass
```

```
rohit = Employee()
varun = Employee()

rohit.name = "Rohit Mishra"
rohit.salary = 700000
rohit.role = "Full Stack Developer"

varun.name = "Varun Dahiya"
varun.salary = 450000
varun.role = "Web Developer"

print(rohit.__dict__)
print(varun.__dict__)

{'name': 'Rohit Mishra', 'salary': 700000, 'role': 'Full Stack
Developer'}
{'name': 'Varun Dahiya', 'salary': 450000, 'role': 'Web Developer'}


# Call by value and Call by reference in Python.
# OR
# Call by Object and Call by Object reference in Python.

#! Mutable Object => An object whose internal state can be changed is
called mutable.
# => for example a list, a set, and a dictionary.

#! Immutable Object => An object whose internal state cannot be
changed is called immutable.
# => for example a number, a string, and a tuple.

# Difference between mutable and Immutable objects:

str = "Coding Ideas"
lst = [10, 20, 30, 40]
print(str)                  # Output => Coding Ideas
print(lst)                  # Output => [10, 20, 30, 40]
lst[1] = lst[1] + lst[2]
print(lst)                  # Output => [10, 50, 30, 40]

# Values of list can be changed so list is mutable in Python.

str[1] = "Gate"             # Error => TypeError: 'str' object does
not support item assignment
print(str)                  # Cannot be printed because error in
above line.

# 'str' object does not support item assignment. Hence string is
immutable in Python.
```

```
Coding Ideas
[10, 20, 30, 40]
[10, 50, 30, 40]

----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
c:\Users\PKVidyarthi\Desktop\Data Science\Notes\Classes&Objects.ipynb
Cell 3 in <cell line: 12>()
      <a
href='vscode-notebook-cell:/c%3A/Users/PKVidyarthi/Desktop/Data
%20Science/Notes/Classes%26Objects.ipynb#ch0000002?line=7'>8</a>
print(lst)                 # Output => [10, 50, 30, 40]
     <a
href='vscode-notebook-cell:/c%3A/Users/PKVidyarthi/Desktop/Data
%20Science/Notes/Classes%26Objects.ipynb#ch0000002?line=9'>10</a> #
Values of list can be change so list is mutable in Python.
---> <a
href='vscode-notebook-cell:/c%3A/Users/PKVidyarthi/Desktop/Data
%20Science/Notes/Classes%26Objects.ipynb#ch0000002?line=11'>12</a>
str[1] = "Gate"
     <a
href='vscode-notebook-cell:/c%3A/Users/PKVidyarthi/Desktop/Data
%20Science/Notes/Classes%26Objects.ipynb#ch0000002?line=12'>13</a>
print(str)

TypeError: 'str' object does not support item assignment
```

```python
# Call by value and Call by reference in Python.
# OR
# Call by Object and Call by Object reference in Python.

#! Call by value => Just a copy of the original variable is passed so
the original variable cannot be changed.
#! Call by reference => The variable itself is passed so the original
variable may be changed (altered).


#! call by value

str = "Coding and Gate"
def teststr(str):
    str = "Coding Ideas!"
    print("Inside the Function: ", str)
teststr(str)                            # passing value not an
address.
print("Outside the Function: ",str)     # changing the value of 'str'
inside the function does not affect here.
```

```
Inside the Function:  Coding Ideas!
Outside the Function:  Coding and Gate


#! call by reference

def testlist(lst):
     lst[1] = 100
     print("Inside the Function", lst)
mylist = [10,20,30,40]

testlist(mylist)                        # passing the address of 'lst'
not the value.
print("Outside the Function:", mylist)
# So, after changing the value of 'lst inside the function,
# value of 'mylist' also be changed.

Inside the Function [10, 100, 30, 40]
Outside the Function: [10, 100, 30, 40]

#Define or call a function that returns a list
def myfunc(a):
  return a
myfunc([5,2,3,7,4]) #Call by value
myfunc([8,11,15,17,25])   #Call by value

''' return => Returns only last function. '''
# Output => [8, 11, 15, 17, 25]

[8, 11, 15, 17, 25]

def myfunc(a):
      print(a)
myfunc([5,2,3,7,4])
myfunc([8,11,15,17,25])

''' print => Prints all the functions.
Output :
[5, 2, 3, 7, 4]
[8, 11, 15, 17, 25]
'''

[5, 2, 3, 7, 4]
[8, 11, 15, 17, 25]

# Question: Define a function mean and calculate the mean of a list.
# eg: Input => [2, 4, 10, 9, 5]
#     Output => 6.0
''' Solution:  '''
def mean(lst):
    m = sum(lst) / len(lst)
    print(m)
mean([2, 4, 10, 9, 5])

''' Explanation:
sum(lst) => sum of 2, 4, 10, 9, 5 => 30
len(lst) => length of list => 5
m = 30/5 => 6.0
'''
```

```
6.0
```

```
# Calculating mean of list:
def mean(lst):
    m = sum(lst) / len(lst)
    print(m)

mean([2, 4, 10, 9, 5])
mean([10, 20, 25, 45, 60, 125 ])
mean([1, 2, 3, 4, 5])
```

```
6.0
47.5
3.0
```

```
#Define a class and print your name, profession, marital status, and
age
class status:
    name = 'PK'
    profession = "Student"
    ms = "Single"
    age = 23
print(status.name)
print(status.profession)
print(status.ms)
print(status.age)
```

```
PK
Student
Single
23
```

```
# Create object (using class 'status')
obj1 = status()
print(obj1.name)
print(obj1.profession)
print(obj1.ms)
print(obj1.age)
```

```
PK
Student
```

```
Single
23
```

```python
# CONSTRUCTORS in Python
'''
=> Constructors are generally used for representing by an object.

=> The task of constructors is to 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 and
   it is always called when an object is created.

=> In Python, the method the __init__() simulates the constructor of
the class.
=> It accepts the self-keyword as a first argument which allows
accessing the attributes of the class.
'''
# Syntax:
'''
def __init__(self):
    # body of the constructor
'''
```

```python
# Create a constructor/def__init__
class sum2:
  def __init__(self,a,b):
    self.a = a
    self.b = b
  def add1(self):
    c = self.a + self.b
    return c
obj2 = sum2(5,6)
obj2.add1() #Calling by reference
```

```
11
```

```python
print(obj2.a)
print(obj2.b)
```

```
5
6
```

```python
obj2.a = 30
obj2.b = 50
obj2.add1()
```

```
80
```

```python
#Q - define a class and constructor and solve division
class div:
  def __init__(self,a,b):
    self.a = a
    self.b = b
  def divide1(self):
    d = self.a/self.b
    return d
obj3 = div(60,3)
obj3.divide1()
```

```
20.0
```

```
# Practice Questions:
#! Q1. Solve Pythagoras theorem using class and typecast function.
#! Q2. Solve Einstein's formula using both with & without type cast.
```

```
# Solution 1. Solve Pythagoras theorem using class and typecast
function.
''' Solution: '''
class Pythagoras:
    def __init__(self, base, perp):
        self.b = base
        self.p = perp
    def cal(self):
        hyp = (self.b*self.b + self.p*self.p)**(1/2)
        print("Base:",self.b)
        print("Perpendicular:",self.p)
        print("Hypotenuse:",hyp)
        print("-----------------")          # As a Divider

obj = Pythagoras(3,4)
obj.cal()

obj1 = Pythagoras(40, 9)
obj1.cal()

obj2 = Pythagoras(8, 15)
obj2.cal()
```

```
Base: 3
Perpendicular: 4
Hypotenuse: 5.0
-----------------
Base: 40
Perpendicular: 9
Hypotenuse: 41.0
-----------------
Base: 8
Perpendicular: 15
Hypotenuse: 17.0
-----------------
```

```
# Solution 2. Solve Einstein's formula using both with & without type
cast.
class Energy:
    def __init__(self, mass):
        self.m = mass
    def calEnergy(self):
        c = 3*(10**8)
        e = self.m*(c*c)
        return(e)
obj = Energy(10)
obj.calEnergy()
```

```
900000000000000000
```

```
# Create a class name MyList and display the list.
class MyList:
    def __init__(self, lst):
        self.lst = lst
    def display(self):
        return self.lst
obj3 =MyList([10, 20, 30, 40])
obj3.display()
```

```
[10, 20, 30, 40]
```

```
# Create a class name MyList1 and print the mean of a list.
class MyList1:
    def __init__(self, lst1):                 # Definate Constructor
        self.lst1 = lst1
    def mean(self):
        mean = sum(self.lst1)/len(self.lst1)
        return mean
obj4 =MyList1([3, 15, 12, 10, 5, 9])
obj4.mean()
```

```
9.0
```

```
#Connector - Connecting object of one class to another class.

obj4.lst1 = obj3.lst
print(obj4.lst1)


obj4.mean()
```

```
[10, 20, 30, 40]
```

```
25.0
```