```
print("Hello World")
```

```
Hello World
```

```
x = 6
y = "Ashir Khan"
print(x)
print(y)
```

```
6
Ashir Khan
```

```
x = 5
x = "Hamza" #x is now a type of string previously it was integer
print(x)
```

```
Hamza
```

```
x = str(5)    #5 will be taken as string
y = int(6)
z = float(7)
print(x)
print(y)
print(z)
```

```
5
6
7.0
```

Variable names are case Senstive in Python

```
a = 6
A = "Sally"
print(a)
print(A)
```

```
6
Sally
```

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

```
x = y = z = "Ashir Khan"
print(x)
print(y)
print(z)
```

```
Ashir Khan
Ashir Khan
Ashir Khan
```

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
apple
banana
cherry
```

```
#Printing In a reverse order
fruits = ["apple", "banana", "cherry"]
z, x, y = fruits
print(x)
print(y)
print(z)
```

```
banana
cherry
apple
```

```
x = "Ashir"
y = "is"
z = "Super Star"
print(x, y, z)
```

```
Ashir is Super Star
```

```
x = "Ashir"
y = "is "
z = "Super Star"
print(x + y + z)
     #NO auto spacing
```

```
Ashiris Super Star
```

```
x = 5
y = 5
print(x + y)
```

```
10
```

```
x = 5
y = "Ashir"
print(x + y)
#In the print() function, when you try to combine a string and a number with the + operator
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[58], line 3
      1 x = 5
      2 y = "Ashir"
----> 3 print(x + y)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
x = 5
y = "Ashir"
print(x , y)

#you can do like this
```

```
5 Ashir
```

```
#Creating a variable outside of a function, and use it inside the function

x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

```
Python is awesome
```

```
#Create a variable inside a function, with the same name as the global variable
x = "I dont know"
def myfunc():
    x = "fantastic"
    print("python is " + x)
myfunc()

print("python is " + x)
```

```
python is fantastic
python is I dont know
```

```
x = "I dont Know"
def myfunc():
    global x
    x = "fantastic as per global keyword"
    print("python is " + x)
myfunc()

print("python is " + x)
```

```
python is fantastic as per global keyword
python is fantastic as per global keyword
```

```
print(10+20)

#Operator Precedence
print((6 + 3) - (6 + 3))

#Multiplication * has higher precedence than addition +, and therefor multiplications are e
print(100 + 5 * 3)

#Addition + and subtraction - has the same precedence, and therefor we evaluate the express
print(5 + 4 - 7 + 3)
```

```
30
0
115
5
```

```
List = ["Ashir", "Pookie", "Laiba"]
print(List)
```

```
['Ashir', 'Pookie', 'Laiba']
```

```
#List allows duplicate values
List = ["Ashir" , "Hussain" , "Aimal" , "Ashir"]
print(List)
```

```
['Ashir', 'Hussain', 'Aimal', 'Ashir']
```

```
#Length of the List

List = ["Ashir" , "Hussain" , "Aimal" , "Ashir"]
print(len(List))
```

```
4
```

```
#List item can be of any type


List1 = [1 , 2 , 3]
List2 = ["Ashir" ,"Tayyab" ,"Aimal"]
List3 = ['a' , 'b' , 'c']
List4 = [True , False , True]
print(List1)
print(List2)
print(List3)
print(List4)
```

```
[1, 2, 3]
```

```
['Ashir', 'Tayyab', 'Aimal']
['a', 'b', 'c']
[True, False, True]
```

```
#List can be of any type unlike array

List = ["Ashir", 'a', True, "male"]

print(List)
```

```
['Ashir', 'a', True, 'male']
```

```
#List is actually a data type

mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

```
<class 'list'>
```

```
#You can print the list items

mylist = ["ashir", "laiba", "aimal"]
print(mylist[1])
```

```
laiba
```

```
#Negative Indexing

mylist = ["ashir", "laiba", "aimal"]
print(mylist[-1])
```

```
aimal
```

```
#Range of a List

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

print(List[2:5])
```

```
['Hamza', 'Tayab', 'Mustafa']
```

```
#This will print from start but not the 5th that is khubaib

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

print(List[:5])
```

```
['Aimal', 'Ashir', 'Hamza', 'Tayab', 'Mustafa']
```

```
#THis will print from 5 that is khubaib till end
```

```
List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

print(List[5:])
```

```
['Khubaib', 'Hussain']
```

```
#Negative Indexing

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

print(List[-7:-3])
```

```
['Aimal', 'Ashir', 'Hamza', 'Tayab']
```

```
#Checking The List if items exist

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
if "Mustafa" in List:
    print(" Yes Mustafa is in the List")
```

```
 Yes Mustafa is in the List
```

```
#Changin the Items of the List

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

List[2] = "Usama"

print(List)
```

```
['Aimal', 'Ashir', 'Usama', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain']
```

```
#Changing the range of multiple Items Values you can add multiple at a time

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
List[2:4] = ["Confidence" , "Calculator" , "Website" , "test"]

print(List)
```

```
['Aimal', 'Ashir', 'Confidence', 'Calculator', 'Website', 'test', 'Mustafa', 'Khubaib', 'Hus
```

```
#Inserting the values in the List

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
List.insert(2, "Ashir Khan ")

print(List)
```

```
['Aimal', 'Ashir', 'Ashir Khan ', 'Hamza', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain']
```

```
#Adding Items in the List
```

```
List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
List.append("Usssssaaaaama Khaaan")

print (List)
```

```
['Aimal', 'Ashir', 'Hamza', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain', 'Usssssaaaaama Khaaan'
```

```
#Extending the list
#Add the elements of List2 to List1:
List1 = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
List2 = ["Mubashir putar" , "Puneet Super Star" , "Bakhti Rehman"]

List1.extend(List2)
print(List1)
```

```
['Aimal', 'Ashir', 'Hamza', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain', 'Mubashir putar', 'Pur
```

```
#Add any iterable to the list

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
thistuple = ("puneet Super Star" , "Mubashir Putar")

List.extend(thistuple)

print(List)
```

```
['Aimal', 'Ashir', 'Hamza', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain', 'puneet Super Star',
```

```
#Remove Item from the List

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

List.remove("Hamza")

print(List)
```

```
['Aimal', 'Ashir', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain']
```

```
#Remove the first occurance of Ashir

List = ["Aimal" , "Ashir" , "Hamza" , "Ashir" , "Mustafa" , "Khubaib" , "Hussain"]

List.remove("Ashir")

print(List)
```

```
['Aimal', 'Hamza', 'Ashir', 'Mustafa', 'Khubaib', 'Hussain']
```

```
#Remove specified Index from the list

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
```

```
List.pop(2)

print(List)
```

```
['Aimal', 'Ashir', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain']
```

```
#If you donot specify the item in pop the last item will automaticaly be removed

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

List.pop()

print(List)
```

```
['Aimal', 'Ashir', 'Hamza', 'Tayab', 'Mustafa', 'Khubaib']
```

```
#Dell Key also removes the specified Index


List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

del List[1]

print(List)
```

```
['Aimal', 'Hamza', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain']
```

```
#Dell can also delete the whole list


List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

del List
print(List)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[161], line 7
      4 List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
      6 del List
----> 7 print(List)

NameError: name 'List' is not defined
```

```
#you can also clear the list

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
List.clear()
```

```
print(List)
```

```
[]
```

```
#Printing the Elements of the List


List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

for List in List:
    print(List)
```

```
Aimal
Ashir
Hamza
Tayab
Mustafa
Khubaib
Hussain
```

```
#you can also print list by reffering to their number

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

for List in range(len(List)):
    print(List)
```

```
0
1
2
3
4
5
6
```

```
#While Loop

List = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

i = 0

while i < len(List):
    print(List[i])
    i+=1
```

```
Aimal
Ashir
```

```
Hamza
Tayab
Mustafa
Khubaib
Hussain
```

```
#without List COmprehension

Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

newNames = []

for x in Names:
    if "s" in x:
        newNames.append(x)

print(newNames)
```

```
['Ashir', 'Mustafa', 'Hussain']
```

```
#With list comprehension

Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]

newNames = [x for x in Names if "s" in x]

print (newNames)
```

```
['Ashir', 'Mustafa', 'Hussain']
```

```
#Only accept items that are not Aimal

Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
newNames= [x for x in Names if x!= "Tayab"]

print(newNames)
```

```
['Aimal', 'Ashir', 'Hamza', 'Mustafa', 'Khubaib', 'Hussain']
```

```
#New List without if

Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
newNames= [x for x in Names ]

print(newNames)
```

```
['Aimal', 'Ashir', 'Hamza', 'Tayab', 'Mustafa', 'Khubaib', 'Hussain']
```

```
#Range Function
```

```
newNames= [x for x in range(10)]

print(newNames)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
#Range Example with condition


newNames= [x for x in range(10) if x<5]

print(newNames)
```

```
[0, 1, 2, 3, 4]
```

```
#Convert each into uper case
Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
newNames= [x .upper() for x in Names ]

print(newNames)
```

```
['AIMAL', 'ASHIR', 'HAMZA', 'TAYAB', 'MUSTAFA', 'KHUBAIB', 'HUSSAIN']
```

```
#Set all the values in new to another word

Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
newNames= ["Puneet Super Star" for x in Names ]

print(newNames)
```

```
['Puneet Super Star', 'Puneet Super Star', 'Puneet Super Star', 'Puneet Super Star', 'Puneet
```

```
#Return Zaid Ghori instead of Mustafa
Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
newNames = [x if x != "Mustafa" else "Zaid Ghori" for x in Names]

print(newNames)
```

```
['Aimal', 'Ashir', 'Hamza', 'Tayab', 'Zaid Ghori', 'Khubaib', 'Hussain']
```

```
#Sorting the List string


Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
Names.sort()
print(Names)
```

```
['Aimal', 'Ashir', 'Hamza', 'Hussain', 'Khubaib', 'Mustafa', 'Tayab']
```

```
#Sorting numbers
```

```
numbers = [44, 77, 22,88,33,11,55,99,66,00]
numbers.sort()
print(numbers)
```

```
[0, 11, 22, 33, 44, 55, 66, 77, 88, 99]
```

```
#Descending Order Sort

numbers = [44, 77, 22,88,33,11,55,99,66,00]
numbers.sort(reverse = True)



#Sorting the List string in descending
Names = ["Aimal" , "Ashir" , "Hamza" , "Tayab" , "Mustafa" , "Khubaib" , "Hussain"]
Names.sort(reverse = True)
print(Names)
```

```
[99, 88, 77, 66, 55, 44, 33, 22, 11, 0]
['Tayab', 'Mustafa', 'Khubaib', 'Hussain', 'Hamza', 'Ashir', 'Aimal']
```

```
#Case Insensitive Sorting



Names = ["Aimal" , "ashir" , "hamza" , "tayab" , "Mustafa" , "Khubaib" , "hussain"]
Names.sort()
print(Names)
Names.sort(reverse = True)
print(Names)
```

```
['Aimal', 'Khubaib', 'Mustafa', 'ashir', 'hamza', 'hussain', 'tayab']
['tayab', 'hussain', 'hamza', 'ashir', 'Mustafa', 'Khubaib', 'Aimal']
```

```
#Copy the List using copy() method

Names = ["Aimal" , "ashir" , "hamza" , "tayab" , "Mustafa" , "Khubaib" , "hussain"]
newNames = Names.copy()
print(newNames)
```

```
['Aimal', 'ashir', 'hamza', 'tayab', 'Mustafa', 'Khubaib', 'hussain']
```

```
#Copy the List using list() method

Names = ["Aimal" , "ashir" , "hamza" , "tayab" , "Mustafa" , "Khubaib" , "hussain"]
newNames = list(Names)
print(newNames)
```

```
['Aimal', 'ashir', 'hamza', 'tayab', 'Mustafa', 'Khubaib', 'hussain']
```

```
#Copy the List using Operator Slice


Names = ["Aimal" , "ashir" , "hamza" , "tayab" , "Mustafa" , "Khubaib" , "hussain"]
newNames = Names[:]
print(newNames)
```

```
['Aimal', 'ashir', 'hamza', 'tayab', 'Mustafa', 'Khubaib', 'hussain']
```

```
#joining the List

List1 = ['a', 'b', 'c']
List2 = [1, 2, 3]

List3 = List1 + List2

print(List3)
```

```
['a', 'b', 'c', 1, 2, 3]
```

```
#Joining the list Method 2
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

```
#Joining the list Method 3

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

```
#Printing the Tuple

newTuple = ('Ashir' , 'Aimal' , 'Tayab')

print(newTuple)
```

```
('Ashir', 'Aimal', 'Tayab')
```

```
#Length Of a tuple
```

```
newTuple = ('Ashir' , 'Aimal' , 'Tayab')
print(len(newTuple))
```

```
3
```

```
#One item tuple, remember the comma:

tuple1 = ('Ashir' , )
print(type(tuple1))

tuple1 = ('Ashir')
print(type(tuple1))
```

```
<class 'tuple'>
<class 'str'>
```

```
#Tuple String int boolean type

tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)

print(tuple1 + tuple2 + tuple3 )
```

```
('apple', 'banana', 'cherry', 1, 5, 7, 9, 3, True, False, False)
```

```
#It is also possible to use the tuple() constructor to make a tuple.

newTuple = ("Ashir", "Aimal", "Tayab")
print(newTuple)
```

```
('Ashir', 'Aimal', 'Tayab')
```

```
#Printing the Item in the tuple

newTuple = ("Ashir", "Aimal", "Tayab", "Khubaib" , "Mustafa", "Hamza" , "Hussain" , "Usama")
print(newTuple[1])
#Negative Indexing
print(newTuple[-3])

#Return the 3rd 4th and 5th Item
print(newTuple[3:6])


#By leaving out the start value, the range will start at the first item:
print(newTuple[:3])


#By leaving out the end value, the range will go on to the end of the tuple:
```

```
print(newTuple[3:])

#Specify negative indexes if you want to start the search from the end of the tuple:
print(newTuple[-4:-2])
```

```
Aimal
Hamza
('Khubaib', 'Mustafa', 'Hamza')
('Ashir', 'Aimal', 'Tayab')
('Khubaib', 'Mustafa', 'Hamza', 'Hussain', 'Usama')
('Mustafa', 'Hamza')
```

```
#Check if the item is available in the tuple

newTuple = ("Ashir", "Aimal", "Tayab", "Khubaib" , "Mustafa", "Hamza" , "Hussain" , "Usama")
if "Mustafa" in newTuple:
    print("Yes Mustafa is in the tuple")
```

```
Yes Mustafa is in the tuple
```

```
#Changing the values of the tuple


#Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutabl
#But there is a workaround. You can convert the tuple into a list, change the list, and cont

newTuple = ("Ashir", "Aimal", "Tayab", "Khubaib" )
x = list(newTuple)
x[3] = "Hamza"
print(type(x))
newTuple = tuple(x)

print(type(newTuple))
print(newTuple)
```

```
<class 'list'>
<class 'tuple'>
('Ashir', 'Aimal', 'Tayab', 'Hamza')
```

```
#Adding items in the tuple
#You have to change the tuple into the list

tuple1 = ("Ashir", "Aimal", "Mustafa")
print("Print  beforeappending:")
print(tuple1)
x = list(tuple1)
x.append("Khubaib")
tuple1 = tuple(x)
```

```
print("\nPrint after appending:")
print(tuple1)
```

```
Print  beforeappending:
('Ashir', 'Aimal', 'Mustafa')

Print after appending:
('Ashir', 'Aimal', 'Mustafa', 'Khubaib')
```

```
#Adding tuple to the tuple

tuple1 = ("Ashir", "Aimal", "Mustafa")


tuple2 = ("Khubaib" , "Tayyab" , "Al-Wazeeri")

print("Print  before adding:")
print(tuple1)

tuple1 += tuple2

print("\nPrint  after adding:")
print(tuple1)
```

```
Print  before adding:
('Ashir', 'Aimal', 'Mustafa')

Print  after adding:
('Ashir', 'Aimal', 'Mustafa', 'Khubaib', 'Tayyab', 'Al-Wazeeri')
```

```
#Removing the item from the tuple

tuple1 = ("Ashir", "Aimal", "Mustafa")
x = list(tuple1)
x.remove("Ashir")
tuple1 = tuple(x)
print(tuple1)
```

```
('Aimal', 'Mustafa')
```

```
#Deleting the tuple

tuple1 = ("Ashir", "Aimal", "Mustafa")
del tuple1

print(tuple1)

#Raising an error because tuple does not exist now
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[14], line 6
      3 tuple1 = ("Ashir", "Aimal", "Mustafa")
      4 del tuple1
----> 6 print(tuple1)

NameError: name 'tuple1' is not defined
```

```
#Paking a tuple
#When we assign values to the tuples we actually packing the tuple

tuple1 = ("Ashir", "Aimal", "Mustafa")

#unpacking
(Ashir , Aimal , Mustafa) = tuple1

print(Ashir)
print(Aimal)
print(Mustafa)
```

```
Ashir
Aimal
Mustafa
```

```
#Using ASTERIC in the last for Unpacking

Cars = ("Bugati", "Tesla", "Range Rover", "Mercedes", "BMW" , "Supra" , "Mark-X" )

(Pink, Pookie, *AllOther) = Cars

print(Pink)
print(Pookie)
print(AllOther)
```

```
Bugati
Tesla
['Range Rover', 'Mercedes', 'BMW', 'Supra', 'Mark-X']
```

```
#Using ASTERIC Other than Last One  for Unpacking

Cars = ("Bugati", "Tesla", "Range Rover", "Mercedes", "BMW" , "Supra" , "Mark-X" )

(Pink, *Pookie, Green) = Cars

print(Pink)
print(Pookie)
print(Green)
```

```
Bugati
['Tesla', 'Range Rover', 'Mercedes', 'BMW', 'Supra']
Mark-X
```

```
#Using a Loop through a tuple

Cars = ("Bugati", "Tesla", "Range Rover", "Mercedes", "BMW" , "Supra" , "Mark-X" )

for x in Cars:
    print(x)
```

```
Bugati
Tesla
Range Rover
Mercedes
BMW
Supra
Mark-X
```

```
#Looping through the Index Numbers

Cars = ("Bugati", "Tesla", "Range Rover", "Mercedes", "BMW" , "Supra" , "Mark-X" )
for x in range(len(Cars)):
  print(Cars[x])
```

```
Bugati
Tesla
Range Rover
Mercedes
BMW
Supra
Mark-X
```

```
#Printing all the items using While Loop

Cars = ("Bugati", "Tesla", "Range Rover", "Mercedes", "BMW" , "Supra" , "Mark-X" )
i = 0
while i < len(Cars):
  print(Cars[i])
  i = i + 1
```

```
Bugati
Tesla
Range Rover
Mercedes
BMW
Supra
Mark-X
```

```
#Joining The Touples

tuple1 = ("Bugati", "Tesla", "Range Rover")
tuple2 = ( "Mercedes", "BMW" , "Supra" , "Mark-X" )

tuple3 = tuple1 + tuple2

print(tuple3)
```

```
('Bugati', 'Tesla', 'Range Rover', 'Mercedes', 'BMW', 'Supra', 'Mark-X')
```

```
#Multiplying the Tuples

tuple1 = ("Bugati", "Tesla", "Range Rover")

myNewTuple = tuple1*2

print(myNewTuple)
```

```
('Bugati', 'Tesla', 'Range Rover', 'Bugati', 'Tesla', 'Range Rover')
```

```
#Printing the Sets

pythonSet = {"Bugati", "Tesla", "Range Rover", "Mercedes", "BMW" , "Supra" , "Mark-X"}

print(pythonSets)
print(type(pythonSet))
```

```
{True, 3, 'Bugati', 'Tesla', 'Range Rover'}
<class 'set'>
```

```
#True and 1 is considered the same value:
pythonSet = {"Bugati", "Tesla", "Range Rover", True , 1, 3}
print("\nTrue and 1 is considered the same value ")
print(pythonSet)

#False and 0 are concidered to be the same
pythonSet2 = {"Bugati", "Tesla", "Range Rover", False , 0 , }
print("\nFalse and 0 are concidered to be the same")
print(pythonSet2)


#To get the number of Items in the set
print("\n#To get the number of Items in the set")
print(len(pythonSet2))


#Set Items can be of any data type
set1 = {True , False , True , True}
```

```python
set2 = {1, 5, 7, 9, 3}
set3 = {"Bugati", "Tesla", "Range Rover"}
print("\nSet Items can be of any data type")
print(set1)
print(set2)
print(set3)

#Set Items can be of Mix data Types
set4 = {"Bugati", "Tesla", "Range Rover", False , 0 , 2 }
print("\nSet Items can be of Mix data Types")
print(set4)


#type of a set
print("\nType: ")
print(type(set4))


#It is also possible to use the set() constructor to make a set.
Fruit = set(("apple", "banana", "cherry")) # note the double round-brackets
print("\n Set constructor to make a set")
print(Fruit)
```

```
True and 1 is considered the same value
{True, 3, 'Bugati', 'Tesla', 'Range Rover'}

False and 0 are concidered to be the same
{False, 'Tesla', 'Range Rover', 'Bugati'}

#To get the number of Items in the set
4

Set Items can be of any data type
{False, True}
{1, 3, 5, 7, 9}
{'Tesla', 'Range Rover', 'Bugati'}

Set Items can be of Mix data Types
{False, 2, 'Bugati', 'Tesla', 'Range Rover'}

Type:
<class 'set'>

 Set constructor to make a set
{'apple', 'banana', 'cherry'}
```

```python
#Accessing the Set using for loop

pythonSet = {"Bugati", "Tesla", "Range Rover", "Mercedes", "BMW" , "Supra" , "Mark-X"}

for x in pythonSet:
    print(x)


#To check if it is present in the set
print("\nChecking if Mercedes is in the set: ")
print("Mercedes" in pythonSet)

#Checking if it is  not present in the set
print("\nChecking if Mercedes is not present in the set: ")

print("Mercedes" not in pythonSet)
```

```
BMW
Mercedes
Bugati
Supra
Tesla
Range Rover
Mark-X

Checking if Mercedes is in the set:
True

Checking if Mercedes is not present in the set:
False
```

```python
#Adding into the set

pythonSet = {"Bugati", "Tesla", "Range Rover", "Mercedes"}
pythonSet.add("BMW")
print(pythonSet)
```

```
{'BMW', 'Mercedes', 'Bugati', 'Tesla', 'Range Rover'}
```

```python
#Updating a set
cars = {"Ferrari", "Lamborghini", "McLaren"}
sportsCars = {"Porsche", " Martin", "Mustang"}

cars.update(sportsCars)

print(cars)
```

```
{' Martin', 'Mustang', 'McLaren', 'Porsche', 'Lamborghini', 'Ferrari'}
```

```
#Add elements of a list to at set:

carSet = {"Bugatti", "Tesla", "Range Rover"}
newCars = ["BMW", "Supra", "Mustang"]

carSet.update(newCars)

print(carSet)
```

```
{'BMW', 'Mustang', 'Bugatti', 'Supra', 'Tesla', 'Range Rover'}
```

```
#Removing Items from the set

carSet = {"Bugatti", "Tesla", "Range Rover"}

carSet.remove("Tesla")

print(carSet)
```

```
{'Bugatti', 'Range Rover'}
```

```
#Using Discard Method

carSet = {"Bugatti", "Tesla", "Range Rover"}

carSet.discard("Tesla")

print(carSet)
```

```
{'Bugatti', 'Range Rover'}
```

```
#Remove a random item by using the pop() method:

carSet = {"Bugatti", "Tesla", "Range Rover"}

x = carSet.pop()

print(x)

print(carSet)
```

```
Tesla
{'Bugatti', 'Range Rover'}
```

```
#Using a clear Method
carSet = {"Bugatti", "Tesla", "Range Rover"}

carSet.clear()

print(carSet)
```

```
set()
```

```
#Using Del Method

carSet = {"Bugatti", "Tesla", "Range Rover"}

del carSet

print(carSet)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[62], line 7
      3 carSet = {"Bugatti", "Tesla", "Range Rover"}
      5 del carSet
----> 7 print(carSet)

NameError: name 'carSet' is not defined
```

```
#Using Loop for Printing the Values:

carSet = {"Bugatti", "Tesla", "Range Rover"}

for x in carSet:
  print(x)
```

```
Tesla
Bugatti
Range Rover
```

```
#Joining the Set

set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{1, 2, 3, 'b', 'a', 'c'}
```

```
#Joning Using | symbol

set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1 | set2
print(set3)
```

```
{1, 2, 3, 'b', 'a', 'c'}
```

```
#Joining Multiple Sets
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"Khubaib", "Aimal"}
set4 = {"Supra", "RAnge ROver", "Bugati"}

myset = set1.union(set2, set3, set4)
print(myset)
```

```
{1, 2, 3, 'c', 'a', 'RAnge ROver', 'Supra', 'Bugati', 'Khubaib', 'b', 'Aimal'}
```

```
#Joining Multiple Sets
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"Khubaib", "Aimal"}
set4 = {"Supra", "RAnge ROver", "Bugati"}

myset = set1 | set2 | set3 |set4
print(myset)
```

```
{1, 2, 3, 'c', 'a', 'RAnge ROver', 'Supra', 'Bugati', 'Khubaib', 'b', 'Aimal'}
```

```
#Joining A Set and a Tuple

x = {"a", "b", "c"}
y = (1, 2, 3)

z = x.union(y)
print(z)
```

```
{1, 2, 3, 'c', 'b', 'a'}
```

```
#Updating a Set

carSet = {"Bugatti", "Tesla", "Range Rover"}
carModels = {1, 2, 3}

carSet.update(carModels)
print(carSet)
```

```
{1, 2, 'Bugatti', 3, 'Tesla', 'Range Rover'}
```

```
#Intersection Of a Set

carSet1 = {"Bugatti", "Tesla", "Range Rover"}
carSet2 = {"BMW", "Porsche", "Bugatti"}

carSet3 = carSet1.intersection(carSet2)
print(carSet3)
```

```
{'Bugatti'}
```

```
#Using & to join 2 sets

carSet1 = {"Bugatti", "Tesla", "Range Rover"}
carSet2 = {"BMW", "Porsche", "Bugatti"}

carSet3 = carSet1 & carSet2
print(carSet3)
```

```
{'Bugatti'}
```

```
#INtersection

carSet1 = {"Bugatti", 1, "Tesla", 0, "Range Rover"}
carSet2 = {False, "BMW", 1, "Bugatti", 2, True}

carSet3 = carSet1.intersection(carSet2)

print(carSet3)
```

```
{False, 1, 'Bugatti'}
```

```
#Using Difference Method

carSet1 = {"Bugatti", "Tesla", "Range Rover"}
carSet2 = {"BMW", "Porsche", "Bugatti"}

carSet3 = carSet1.difference(carSet2)

print(carSet3)
```

```
{'Tesla', 'Range Rover'}
```

```
#Use - to join two sets:
carSet1 = {"Bugatti", "Tesla", "Range Rover"}
carSet2 = {"BMW", "Porsche", "Bugatti"}

carSet3 = carSet1 - carSet2
print(carSet3)
```

```
{'Tesla', 'Range Rover'}
```

```
#Use the difference_update() method to keep the items that are not present in both sets:


carSet1 = {"Bugatti", "Tesla", "Range Rover"}
carSet2 = {"BMW", "Porsche", "Bugatti"}
```

```
carSet1.difference_update(carSet2)

print(carSet1)
```

```
{'Tesla', 'Range Rover'}
```

```
#symettric Difference

carSet1 = {"Bugatti", "Tesla", "Range Rover"}
carSet2 = {"BMW", "Porsche", "Bugatti"}

carSet3 = carSet1.symmetric_difference(carSet2)

print(carSet3)
```

```
{'BMW', 'Porsche', 'Tesla', 'Range Rover'}
```

```
#Using ^ function for joining the set

carSet1 = {"Bugatti", "Tesla", "Range Rover"}
carSet2 = {"BMW", "Porsche", "Bugatti"}

carSet3 = carSet1 ^ carSet2
print(carSet3)
```

```
{'BMW', 'Porsche', 'Tesla', 'Range Rover'}
```

```
carSet1 = {"Bugatti", "Tesla", "Range Rover"}
carSet2 = {"BMW", "Porsche", "Bugatti"}

carSet1.symmetric_difference_update(carSet2)

print(carSet1)
```

```
{'BMW', 'Porsche', 'Tesla', 'Range Rover'}
```

```
#Printing the Dictionary

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
print(type(thisdict))
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
<class 'dict'>
```

```
#Accessing the items of the dictionary
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

```
Ford
```

```
#Overwrite duplicate values

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

```
#String int and list data types
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
print(thisdict)
```

```
{'brand': 'Ford', 'electric': False, 'year': 1964, 'colors': ['red', 'white', 'blue']}
```

```
#Can be also used as a constructor

thisdict = dict(name = "Khubaib", age = 22, country = "Pakistan")
print(thisdict)
```

```
{'name': 'Khubaib', 'age': 22, 'country': 'Pakistan'}
```

```
#Access the dictionary Items

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

```
#Get a list of the keys:

x = thisdict.keys()
print(x)
```

```
Mustang
dict_keys(['brand', 'model', 'year'])
```

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change


#Get a list of the values:

x = thisdict.values()
print(x)
```

```
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year', 'color'])
dict_values(['Ford', 'Mustang', 1964])
```

```
#Make a change in the original dictionary, and see that the values list gets updated as wel

car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 2020])
```

```
#Add a new item to the original dictionary, and see that the values list gets updated as wel

car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["color"] = "red"

print(x) #after the change



#Get a list of the key:value pairs
x = thisdict.items()
print(x)
```

```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 1964, 'red'])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

```
#Adding Items to dict

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

```
#Updating an Item in dictionary

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

```
thisdict.update({"color": "red"})
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

```
#using pop method to remove items in the dictionary
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

```
#popitem() removes the last item in the dictionary

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)


{'brand': 'Ford', 'model': 'Mustang'}

#Dell will delete the specified key name
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

```
#The del keyword can also delete the dictionary completely:

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[110], line 9
      3 thisdict = {
      4   "brand": "Ford",
      5   "model": "Mustang",
      6   "year": 1964
      7 }
      8 del thisdict
----> 9 print(thisdict)

NameError: name 'thisdict' is not defined
```

```
#The clear() method empties the dictionary:

thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

```
{}
```

```
#Printing all key names in the dictionary, one by one:

x = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in x:
  print(x)
```

```
brand
model
year
```

```
# printing the values

x = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in x.values():
  print(x)
```

```
Ford
Mustang
1964
```

```
#Using key method

mydict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

for x in mydict.keys():
  print(x)
```

```
brand
model
year
```

```
#Loop through both keys and values, by using the items() method:

mydict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in mydict.items():
  print(x, y)
```

```
brand Ford
model Mustang
year 1964
```

```
#Coping a Dict
Olddict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

print(Olddict)
mydict = mydict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

```
#Make a copy of a dictionary with the dict() function:
```

```
oldDict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(oldDict)
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

```
#Create a dictionary that contain three dictionaries:

myfamily = {
  "child1" : {
    "name" : "Ashir",
    "year" : 2002
  },
  "child2" : {
    "name" : "Hamza",
    "year" : 2004
  },
  "child3" : {
    "name" : "Tayyab",
    "year" : 2006
  }
}

print(myfamily)
```

```
{'child1': {'name': 'Ashir', 'year': 2002}, 'child2': {'name': 'Hamza', 'year': 2004}, 'chil
```

```
#Create three dictionaries, then create one dictionary that will contain the other three di

child1 = {
  "name" : "Ashir",
  "year" : 2004
}
child2 = {
  "name" : "Tayab",
  "year" : 2007
}
child3 = {
  "name" : "Hamza",
  "year" : 2006
}

myfamily = {
  "child1" : child1,
```

```
  "child2" : child2,
  "child3" : child3
}

print(myfamily)
```

```
{'child1': {'name': 'Ashir', 'year': 2004}, 'child2': {'name': 'Tayab', 'year': 2007}, 'chil
```

```
#Access the items in the nested Dictionary

print(myfamily["child2"]["name"])
```

```
Tayab
```

```
#Loop through Nested Dictionaries


for x, obj in myfamily.items():
  print(x)

  for y in obj:
    print(y + ':', obj[y])
```

```
child1
name: Ashir
year: 2004
child2
name: Tayab
year: 2007
child3
name: Hamza
year: 2006
```

```
a = 20
b = 10

if a>b:
    print("a is greater than b")
```

```
a is greater than b
```

```
#Elif in the Python

a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
a and b are equal
```

```
#The else keyword catches anything which isn't caught by the preceding conditions.


a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

```
#Short Hand if
if a > b: print("a is greater than b")
```

```
a is greater than b
```

```
#One line if else statement:

a = 2
b = 330
print("A") if a > b else print("B")
```

```
B
```

```
#One line if else statement, with 3 conditions:

a = 330
b = 330
print("A") if a > b else print("B") if a == b else print("C")
```

```
B
```

```
#Checking And Keyword
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

```
Both conditions are True
```

```
#Checking OR condition

a = 200
b = 33
c = 500
```

```
if a > b or a > c:
  print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

```
#Checking NOT

a = 33
b = 200
if not a > b:
  print("a is NOT greater than b")
```

```
a is NOT greater than b
```

```
#Nested If

x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

```
Above ten,
and also above 20!
```

```
#The Pass Statement
a = 33
b = 200

if b > a:
  pass

i = 1
while i < 10:
  print(i)
  i += 1
```

```
1
2
3
4
5
6
7
8
9
```

```
#Break Statement
i = 1
while i <10:
  print(i)
  if i == 5:
    break
  i += 1
```

```
1
2
3
4
5
```

```
#continue Statement
i = 0
while i < 10:
  i += 1
  if i == 3:
    continue
  print(i)
```

```
1
2
4
5
6
```

```
#Print the message once the ci=ondition is false

i = 1
while i < 10:
  print(i)
  i += 1
else:
  print("i is no longer less than 10")
```

```
1
2
3
4
5
6
7
8
9
i is no longer less than 10
```

```
#Printing through for  Loop
```

```
Names = ["Hamza", "Ashir", "Aimal"]
for x in Names:
  print(x)
```

```
Hamza
Ashir
Aimal
```

```
#Loop through the letters in the word :

for x in "Ashir":
  print(x)
```

```
A
s
h
i
r
```

```
#Exit the loop when x is some given value:

list1 = ["Ashir", "Aimal", "Hamza"]
for x in list1:
  print(x)
  if x == "Aimal":
    break
```

```
Ashir
Aimal
```

```
#Exit the loop when x is "Hamza", but this time the break comes before the print:

list1 = ["Aimal", "Hamza", "Ashir"]
for x in list1:
  if x == "Hamza":
    break
  print(x)
```

```
Aimal
```

```
#Do not print Hamza:

Names = ["Tayab", "Hamza", "Ahmad"]
for x in Names:
  if x == "Hamza":
    continue
  print(x)
```

```
Tayab
```

Ahmad

```
#Range Function


for x in range(2, 6):
  print(x)
```

```
2
3
4
5
```

```
#Increment the sequence with 3 (default is 1):

for x in range(2, 30, 3):
  print(x)
```

```
2
5
8
11
14
17
20
23
26
29
```

```
#Print a message when complete a range
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

```
#Break the loop when x is 3, and see what happens with the else block:

for x in range(6):
  if x == 3: break
  print(x)
else:
```

```
  print("Finally finished!")
```

```
0
1
2
```

```
#Print each adjective for every fruit:

adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

```
for x in [0, 1, 2]:
  pass
```

```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

```
#Function With Arguments

def my_function(name):
  print(name + " Khan")

my_function("Ashir")
my_function("Aimal")
my_function("Tayab")
```

```
Ashir Khan
Aimal Khan
Tayab Khan
```

```
#TWO agruments
```

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Ashir", "Khan")
```

```
Ashir Khan
```

```
#If the number of arguments is unknown, add a * before the parameter name:

def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Aimal", "Tayab", "Mustafa")
```

```
The youngest child is Mustafa
```

```
#You can also send arguments with the key = value syntax.

def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Alina", child2 = "Tayab", child3 = "Ashir")
```

```
The youngest child is Ashir
```

```
#the number of keyword arguments is unknown, add a double ** before the parameter name:

def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Muhammad", lname = "Ashir")
```

```
His last name is Ashir
```

```
#The following example shows how to use a default parameter value.

def my_function(country = "Pakistan"):
  print("I am from " + country)

my_function("Bangladesh")
my_function("India")
my_function()
my_function("America")
```

```
I am from Bangladesh
I am from India
I am from Pakistan
I am from America
```

```
#Passing a List as an Argument
```

```
def my_function(food):

  for x in food:
    print(x)

Names = ["tayab", "Ashir", "ahmad"]

my_function(Names)
```

```
tayab
Ashir
ahmad
```

```
#Return Values
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

```
def myfunction():
  pass
```

```
#Positional-Only Arguments

def my_function(x, /):
  print(x)

my_function(3)
```

```
3
```

```
#Keyword-Only Arguments

def my_function(*, x):
  print(x)

my_function(x = 3)
```

```
3
```

```
#Positional-Only Arguments
def my_function(a, b, /, *, c, d):
  print(a + b + c + d)
```

```
my_function(5, 6, c = 7, d = 8)
```

```
26
```

```
#Recursion

def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("Recursion Example Results:")
tri_recursion(6)
```

```
Recursion Example Results:
1
3
6
10
15
21

21
```

```
#creating a class using key-word class
class MyClass:
  x = 5

#creating an object

p1 = MyClass()
print(p1.x)
```

```
5
```

```
#The __init__() Function

class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("Ashir", 26)

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

```
Ashir
26
```

```
#The __str__() function:

class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("Ashir", 20)

print(p1)
```

```
Ashir(20)
```

```
#Object Methods

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("ASHIR KHAN", 36)
p1.myfunc()
```

```
Hello my name is ASHIR KHAN
```

```
#Modify Object Properties

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

```
40
```

```
#Delete Object Properties
del p1.age
print(p1.age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[103], line 2
      1 #Delete Object Properties
----> 2 del p1.age
```

```
      3 print(p1.age)

AttributeError: 'Person' object has no attribute 'age'
```

```
#deleting Object

del p1

#Pass Function
class Person:
  pass
```