In [1]:
```
import sys
import keyword
import operator
from datetime import datetime
import os
```

In [2]:
```
print(keyword.kwlist) # List all Python Keywords
```

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'aw
ait', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'fi
nally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonloca
l', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

In [3]:
```
len(keyword.kwlist) # Python contains 35 keywords
```

Out[3]:
```
36
```

In [4]:
```
1var = 10 # Identifier can't start with a digit
```

```
  Input In [4]
    1var = 10 # Identifier can't start with a digit
     ^
SyntaxError: invalid syntax
```

In [5]:
```
val2@ = 35 # Identifier can't use special symbols
```

```
  Input In [5]
    val2@ = 35 # Identifier can't use special symbols
          ^
SyntaxError: invalid syntax
```

In [6]:
```
import = 125 # Keywords can't be used as identifiers
```

```
  Input In [6]
    import = 125 # Keywords can't be used as identifiers
           ^
SyntaxError: invalid syntax
```

In [7]:
```
val2 = 10
```

In [8]:
```
val_ = 99
```

In [9]:
```
# Single line comment
val1 = 10
```

In [10]:
```
# Multiple
# line
# comment
val1 = 10
```

In [11]:
```
# Single line statement
p1 = 10 + 20
p1
```

Out[11]: 30

In [12]:
```
# Single line statement
p2 = ['a' , 'b' , 'c' , 'd']
p2
```

Out[12]: ['a', 'b', 'c', 'd']

In [13]:
```
# Multiple line statement
p1 = 20 + 30 \
 + 40 + 50 +\
 + 70 + 80
p1
```

Out[13]: 290

In [15]:
```
# Multiple line statement
p2 = ['a' ,
 'b' ,
 'c' ,
 'd'
 ]
p2
```

Out[15]: ['a', 'b', 'c', 'd']

In [16]:
```
p = 10
if p == 10:
 print ('P is equal to 10') # correct indentation
```

```
P is equal to 10
```

In [21]:
```
# if indentation is skipped we will encounter "IndentationError: expected an inde
p = 10
if p == 10:
print ('P is equal to 10')
```

```
  Input In [21]
    print ('P is equal to 10')
    ^
IndentationError: expected an indented block
```

In [22]:
```
for i in range(0,5):
 print(i)
```

```
0
1
2
3
4
```

In [24]:
```
# if indentation is skipped we will encounter "IndentationError: expected an inde
for i in range(0,5):

```

```
0
1
2
3
4
```

In [25]:
```
for i in range(0,5): print(i) # correct indentation but less readableprint(i)
```

```
0
1
2
3
4
```

In [26]:
```
j=20
for i in range(0,5):
 print(i) # inside the for loop
print(j) # outside the for loop
```

```
0
1
2
3
4
20
```

In [ ]:
```
#Docstrings
#1) Docstrings provide a convenient way of associating documentation with functio
methods or modules.
#2) They appear right after the definition of a function, method, class, or modul

```

In [34]:
```
def square(num):
 '''Square Function :- This function will return the square of a number'''
 return num**2
```

In [35]:
```
square(2)
```

Out[35]: 4

In [36]:
```
square.__doc__ # We can access the Docstring using __doc__ method
```

Out[36]:
```
'Square Function :- This function will return the square of a number'
```

In [63]:
```
def evenodd(num):
 '''evenodd Function :- This function will test whether a numbr is Even or Odd'''
 if num % 2 == 0:
  print("Even Number")
 else:
  print("Odd Number")
```

In [64]:
```
evenodd(3)
```

```
Odd Number
```

In [65]:
```
evenodd(2)
```

```
Even Number
```

In [66]:
```
evenodd.__doc__
```

Out[66]:
```
'evenodd Function :- This function will test whether a numbr is Even or Odd'
```

In [ ]:
```
#Variables
#A Python variable is a reserved memory location to store values.A variable is cr
#you first assign a value to it.
```

In [67]:
```
p = 30
```

In [68]:
```
'''
id() function returns the “identity” of the object.
The identity of an object - Is an integer
 - Guaranteed to be unique
 - Constant for this object during its lifetime.
'''
id(p)
```

Out[68]:
```
2300729781456
```

In [69]:
```
hex(id(p)) # Memory address of the variable
```

Out[69]:
```
'0x217ae2e6cd0'
```

In [70]:
```
p = 20 #Creates an integer object with value 20 and assigns the variable p to po
q = 20 # Create new reference q which will point to value 20. p & q will be poin
r = q # variable r will also point to the same location where p & q are pointing
p , type(p), hex(id(p)) # Variable P is pointing to memory location '0x7fff6d71a3
```

Out[70]:
```
(20, int, '0x217ae2e6b90')
```

In [71]:
```
q , type(q), hex(id(q))
```

Out[71]:
```
(20, int, '0x217ae2e6b90')
```

In [72]:
```
r , type(r), hex(id(r))
```

Out[72]:
```
(20, int, '0x217ae2e6b90')
```

In [73]:
```
p = 20
p = p + 10 # Variable Overwriting
p
```

Out[73]:
```
30
```

In [74]:
```
intvar = 10 # Integer variable
floatvar = 2.57 # Float Variable
strvar = "Python Language" # String variable
print(intvar)
print(floatvar)
print(strvar)
```

```
10
2.57
Python Language
```

In [75]:
```
intvar , floatvar , strvar = 10,2.57,"Python Language" # Using commas to separate
print(intvar)
print(floatvar)
print(strvar)
```

```
10
2.57
Python Language
```

In [76]:
```
p1 = p2 = p3 = p4 = 44 # All variables pointing to same value
print(p1,p2,p3,p4)
```

```
44 44 44 44
```

In [77]:
```
val1 = 10 # Integer data type
print(val1)
print(type(val1)) # type of object
print(sys.getsizeof(val1)) # size of integer object in bytes
print(val1, " is Integer?", isinstance(val1, int)) # val1 is an instance of int c
```

```
10
<class 'int'>
28
10  is Integer? True
```

In [78]:
```
val2 = 92.78 # Float data type
print(val2)
print(type(val2)) # type of object
print(sys.getsizeof(val2)) # size of float object in bytes
print(val2, " is float?", isinstance(val2, float)) # Val2 is an instance of float
```

```
92.78
<class 'float'>
24
92.78  is float? True
```

In [79]:
```
val3 = 25 + 10j # Complex data type
print(val3)
print(type(val3)) # type of object
print(sys.getsizeof(val3)) # size of float object in bytes
print(val3, " is complex?", isinstance(val3, complex)) # val3 is an instance of c
```

```
(25+10j)
<class 'complex'>
32
(25+10j)  is complex? True
```

In [80]:
```
sys.getsizeof(int()) # size of integer object in bytes
```

Out[80]:
```
24
```

In [81]:
```
sys.getsizeof(float()) # size of float object in bytes
```

Out[81]:
```
24
```

In [82]:
```
sys.getsizeof(complex()) # size of complex object in bytes
```

Out[82]:
```
32
```

In [83]:
```
bool1 = True
```

In [84]:
```
bool2 = False
```

In [85]:
```
print(type(bool1))
```

```
<class 'bool'>
```

In [86]:
```
print(type(bool2))
```

```
<class 'bool'>
```

In [87]:
```
isinstance(bool1, bool)
```

Out[87]:
```
True
```

In [88]:
```
bool(0)
```

Out[88]:
```
False
```

In [89]:
```
bool(1)
```

Out[89]:
```
True
```

In [90]:
```
bool(None)
```

Out[90]:
```
False
```

In [91]:
```
bool (False)
```

Out[91]:
```
False
```

In [92]:
```
str1 = "HELLO PYTHON"
print(str1)
```

```
HELLO PYTHON
```

In [93]:
```
mystr = 'Hello World' # Define string using single quotes
print(mystr)
```

```
Hello World
```

In [94]:
```
mystr = "Hello World" # Define string using double quotes
print(mystr)
```

```
Hello World
```

In [95]:
```
mystr = '''Hello
 World ''' # Define string using triple quotes
print(mystr)
```

```
Hello
 World
```

In [96]:
```
mystr = """Hello
 World""" # Define string using triple quotes
print(mystr)
```

```
Hello
 World
```

In [97]:
```
mystr = ('Happy '
 'Monday '
 'Everyone')
print(mystr)
```

```
Happy Monday Everyone
```

In [98]:
```
mystr2 = 'Woohoo '
mystr2 = mystr2*5
mystr2
```

Out[98]:
```
'Woohoo Woohoo Woohoo Woohoo Woohoo '
```

In [99]:
```
len(mystr2) # Length of string
```

Out[99]: 35

In [100]:
```
str1
```

Out[100]: 'HELLO PYTHON'

In [101]:
```
str1[0] # First character in string "str1"
```

Out[101]: 'H'

In [102]:
```
str1[len(str1)-1] # Last character in string using len function
```

Out[102]: 'N'

In [103]:
```
str1[-1] # Last character in string
```

Out[103]: 'N'

In [104]:
```
str1[6] #Fetch 7th element of the string
```

Out[104]: 'P'

In [105]:
```
str1[5]
```

Out[105]: ' '

In [106]:
```
str1[0:5] # String slicing - Fetch all characters from 0 to 5 index location excl
```

Out[106]: 'HELLO'

In [107]:
```
str1[6:12] # String slicing - Retreive all characters between 6 - 12 index loc ex
```

Out[107]: 'PYTHON'

In [108]:
```
str1[-4:] # Retreive last four characters of the string
```

Out[108]: 'THON'

In [109]:
```
str1[-6:] # Retreive last six characters of the string
```

Out[109]: 'PYTHON'

In [110]:
```
str1[:4] # Retreive first four characters of the string
```

Out[110]: 'HELL'

In [111]:
```
str1[:6] # Retreive first six characters of the string
```

Out[111]: 'HELLO '

In [112]:
```
str1
```

Out[112]:
```
'HELLO PYTHON'
```

In [113]:
```
#Strings are immutable which means elements of a string cannot be changed once th
str1[0:5] = 'HOLAA'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [113], in <cell line: 2>()
      1 #Strings are immutable which means elements of a string cannot be chang
ed once th
----> 2 str1[0:5] = 'HOLAA'

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

In [114]:
```
del str1 # Delete a string
print(srt1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [114], in <cell line: 2>()
      1 del str1 # Delete a string
----> 2 print(srt1)

NameError: name 'srt1' is not defined
```

In [115]:
```
# String concatenation
s1 = "Hello"
s2 = "Asif"
s3 = s1 + s2
print(s3)
```

```
HelloAsif
```

In [116]:
```
# String concatenaion
s1 = "Hello"
s2 = "Asif"
s3 = s1 + " " + s2
print(s3)
```

```
Hello Asif
```

In [117]:
```
mystr1 = "Hello Everyone"
```

In [118]:
```
# Iteration
for i in mystr1:
 print(i)
```

```
H
e
l
l
o

E
v
e
r
y
o
n
e
```

In [119]:
```
for i in enumerate(mystr1):
 print(i)
```

```
(0, 'H')
(1, 'e')
(2, 'l')
(3, 'l')
(4, 'o')
(5, ' ')
(6, 'E')
(7, 'v')
(8, 'e')
(9, 'r')
(10, 'y')
(11, 'o')
(12, 'n')
(13, 'e')
```

In [120]:
```
list(enumerate(mystr1)) # Enumerate method adds a counter to an iterable and retu
```

Out[120]:
```
[(0, 'H'),
 (1, 'e'),
 (2, 'l'),
 (3, 'l'),
 (4, 'o'),
 (5, ' '),
 (6, 'E'),
 (7, 'v'),
 (8, 'e'),
 (9, 'r'),
 (10, 'y'),
 (11, 'o'),
 (12, 'n'),
 (13, 'e')]
```

In [121]:
```
# String membership
mystr1 = "Hello Everyone"
print ('Hello' in mystr1) # Check whether substring "Hello" is present in string
print ('Everyone' in mystr1) # Check whether substring "Everyone" is present in s
print ('Hi' in mystr1) # Check whether substring "Hi" is present in string "mysrt
```

```
True
True
False
```

In [122]:
```
"""
The partition() method searches for a specified string and splits the string into
 - The first element contains the part before the argument string.
 - The second element contains the argument string.
 - The third element contains the part after the argument string.
"""
str5 = "Natural language processing with Python and R and Java"
L = str5.partition("and")
print(L)
```

```
('Natural language processing with Python ', 'and', ' R and Java')
```

In [123]:
```
"""
The rpartition() method searches for the last occurence of the specified string a
containing three elements.
 - The first element contains the part before the argument string.
 - The second element contains the argument string.
 - The third element contains the part after the argument string.
"""
str5 = "Natural language processing with Python and R and Java"
L = str5.rpartition("and")
print(L)
```

```
('Natural language processing with Python and R ', 'and', ' Java')
```

In [124]:
```
mystr2 = " Hello Everyone "
mystr2
```

Out[124]:
```
' Hello Everyone '
```

In [125]:
```
mystr2.strip() # Removes white space from begining & end
```

Out[125]:
```
'Hello Everyone'
```

In [126]:
```
mystr2.rstrip() # Removes all whitespaces at the end of the string
```

Out[126]:
```
' Hello Everyone'
```

In [127]:
```
mystr2.lstrip() # Removes all whitespaces at the begining of the string
```

Out[127]:
```
'Hello Everyone '
```

In [128]:
```
mystr2 = "*********Hello Everyone**********All the Best**********"
mystr2
```

Out[128]:
```
'*********Hello Everyone**********All the Best**********'
```

In [129]:
```
mystr2.strip('*') # Removes all '*' characters from begining & end of the string
```

Out[129]:
```
'Hello Everyone**********All the Best'
```

In [130]:
```
mystr2.rstrip('*') # Removes all '*' characters at the end of the string
```

Out[130]:
```
'*********Hello Everyone**********All the Best'
```

In [131]:
```
mystr2.lstrip('*') # Removes all '*' characters at the begining of the string
```

Out[131]:
```
'Hello Everyone**********All the Best**********'
```

In [132]:
```
mystr2 = " Hello Everyone "
```

In [133]:
```
mystr2.lower() # Return whole string in lowercase
```

Out[133]:
```
' hello everyone '
```

In [134]:
```
mystr2.upper() # Return whole string in uppercase
```

Out[134]:
```
' HELLO EVERYONE '
```

In [135]:
```
mystr2.replace("He" , "Ho") #Replace substring "He" with "Ho"
```

Out[135]:
```
' Hollo Everyone '
```

In [136]:
```
mystr2.replace(" " , "") # Remove all whitespaces using replace function
```

Out[136]:
```
'HelloEveryone'
```

In [137]:
```
mystr5 = "one two Three one two two three"
```

In [138]:
```
mystr5.count("one") # Number of times substring "one" occurred in string.
```

Out[138]:
```
2
```

In [139]:
```
mystr5.count("two") # Number of times substring "two" occurred in string.
```

Out[139]:
```
3
```

In [140]:
```
mystr5.startswith("one") # Return boolean value True if string starts with "one"
```

Out[140]:
```
True
```

In [141]:
```
mystr5.endswith("three") # Return boolean value True if string ends with "three"
```

Out[141]:
```
True
```

In [142]:
```
mystr4 = "one two three four one two two three five five six seven six seven one"
```

In [143]:
```
mylist = mystr4.split() # Split String into substrings
mylist
```

Out[143]:
```
['one',
 'two',
 'three',
 'four',
 'one',
 'two',
 'two',
 'three',
 'five',
 'five',
 'six',
 'seven',
 'six',
 'seven',
 'one']
```

In [144]:
```
# Combining string & numbers using format method
item1 = 40
item2 = 55
item3 = 77
res = "Cost of item1 , item2 and item3 are {} , {} and {}"
print(res.format(item1,item2,item3))
```

```
Cost of item1 , item2 and item3 are 40 , 55 and 77
```

In [145]:
```
# Combining string & numbers using format method
item1 = 40
item2 = 55
item3 = 77
res = "Cost of item3 , item2 and item1 are {2} , {1} and {0}"
print(res.format(item1,item2,item3))
```

```
Cost of item3 , item2 and item1 are 77 , 55 and 40
```

In [150]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.center(100) # center align the string using a specific character as t
print(str2)
```

```
                                                WELCOME EVERYONE
```

In [151]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.center(100,'*') # center align the string using a specific character
print(str2)
```

```
****************************************** WELCOME EVERYONE ******************
**********
```

In [152]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.rjust(50) # Right align the string using a specific character as the
print(str2)
```

```
                                          WELCOME EVERYONE
```

In [155]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.rjust(50,'*') # Right align the string using a specific character ('
print(str2)
```

```
****************************** WELCOME EVERYONE
```

In [156]:
```
str2 = " WELCOME EVERYONE "
str2 = str2.ljust(50,'*') # Right align the string using a specific character ('
print(str2)
```

```
 WELCOME EVERYONE ******************************
```

In [157]:
```
str4 = "one two three four five six seven"
loc = str4.find("five") # Find the location of word 'five' in the string "str4"
print(loc)
```

```
19
```

In [158]:
```
str4 = "one two three four five six seven"
loc = str4.index("five") # Find the location of word 'five' in the string "str4"
print(loc)
```

```
19
```

In [159]:
```
mystr6 = '123456789'
print(mystr6.isalpha()) # returns True if all the characters in the text are lett
print(mystr6.isalnum()) # returns True if a string contains only letters or numb
print(mystr6.isdecimal()) # returns True if all the characters are decimals (0-9)
print(mystr6.isnumeric()) # returns True if all the characters are numeric (0-9)
```

```
False
True
True
True
```

In [160]:
```
mystr6 = 'abcde'
print(mystr6.isalpha()) # returns True if all the characters in the text are lett
print(mystr6.isalnum()) # returns True if a string contains only letters or numb
print(mystr6.isdecimal()) # returns True if all the characters are decimals (0-9)
print(mystr6.isnumeric()) # returns True if all the characters are numeric (0-9)
```

```
True
True
False
False
```

In [161]:
```
mystr6 = 'abc12309'
print(mystr6.isalpha()) # returns True if all the characters in the text are lett
print(mystr6.isalnum()) # returns True if a string contains only letters or numb
print(mystr6.isdecimal()) # returns True if all the characters are decimals (0-9)
print(mystr6.isnumeric()) # returns True if all the characters are numeric (0-9)
```

```
False
True
False
False
```

In [162]:
```
mystr7 = 'ABCDEF'
print(mystr7.isupper()) # Returns True if all the characters are in upper case
print(mystr7.islower()) # Returns True if all the characters are in lower case
```

```
True
False
```

In [163]:
```
mystr8 = 'abcdef'
print(mystr8.isupper()) # Returns True if all the characters are in upper case
print(mystr8.islower()) # Returns True if all the characters are in lower case
```

```
False
True
```

In [164]:
```
str6 = "one two three four one two two three five five six one ten eight ten nine
loc = str6.rfind("one") # last occurrence of word 'one' in string "str6"
print(loc)
```

```
51
```

In [165]:
```
loc = str6.rindex("one") # last occurrence of word 'one' in string "str6"
print(loc)
```

```
51
```

In [166]:
```
txt = " abc def ghi "
txt.rstrip()
```

Out[166]:
```
' abc def ghi'
```

In [167]:
```
txt = " abc def ghi "
txt.lstrip()
```

Out[167]:
```
'abc def ghi '
```

In [168]:
```
txt = " abc def ghi "
txt.strip()
```

Out[168]:
```
'abc def ghi'
```

In [169]:
```
#Using double quotes in the string is not allowed.
mystr = "My favourite TV Series is "Game of Thrones""
```

```
  Input In [169]
    mystr = "My favourite TV Series is "Game of Thrones""
                                       ^
SyntaxError: invalid syntax
```

In [170]:
```
#Using escape character to allow illegal characters
mystr = "My favourite series is \"Game of Thrones\""
print(mystr)
```

```
My favourite series is "Game of Thrones"
```

In [171]:
```
list1 = [] # Empty List
```

In [172]:
```
print(type(list1))
```

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

In [173]:
```
list2 = [10,30,60] # List of integers numbers
```

In [174]:
```
list3 = [10.77,30.66,60.89] # List of float numbers
```

In [175]:
```
list4 = ['one','two' , "three"] # List of strings
```

In [176]:
```
list5 = ['Asif', 25 ,[50, 100],[150, 90]] # Nested List
```

In [177]:
```
list6 = [100, 'Asif', 17.765] # List of mixed data type
```

In [178]:
```
list7 = ['Asif', 25 ,[50, 100],[150, 90] , {'John' , 'David'}]
```

In [179]:
```
len(list6) #Length of list
```

Out[179]:
```
3
```

In [180]:
```
list2[0] # Retreive first element of the list
```

Out[180]:
```
10
```

In [182]:
```
list4[0] # Retreive first element of the list
```

Out[182]:
```
'one'
```

In [183]:
```
list4[0][0] # Nested indexing - Access the first character of the first list elem
```

Out[183]:
```
'o'
```

In [184]:
```
list4[-1] # Last item of the list
```

Out[184]:
```
'three'
```

In [185]:
```
list5[-1] # Last item of the list
```

Out[185]:
```
[150, 90]
```

In [186]:
```
mylist = ['one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight']
```

In [187]:
```
mylist[0:3] # Return all items from 0th to 3rd index location excluding the item
```

Out[187]:
```
['one', 'two', 'three']
```

In [188]:
```
mylist[2:5] # List all items from 2nd to 5th index location excluding the item at
```

Out[188]:
```
['three', 'four', 'five']
```

In [189]:
```
mylist[:3] # Return first three items
```

Out[189]:
```
['one', 'two', 'three']
```

In [190]:
```
mylist[:2] # Return first two items
```

Out[190]:
```
['one', 'two']
```

In [191]:
```
mylist[-3:] # Return last three items
```

Out[191]:
```
['six', 'seven', 'eight']
```

In [192]:
```
mylist[-2:] # Return last two items
```

Out[192]:
```
['seven', 'eight']
```

In [193]:
```
mylist[-1] # Return last item of the list
```

Out[193]:
```
'eight'
```

In [194]:
```
mylist[:] # Return whole list
```

Out[194]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [195]:
```
mylist
```

Out[195]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [196]:
```
mylist.append('nine') # Add an item to the end of the list
mylist
```

Out[196]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [197]:
```
mylist.insert(9,'ten') # Add item at index location 9
mylist
```

Out[197]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

In [198]:
```
mylist.insert(1,'ONE') # Add item at index location 1
mylist
```

Out[198]:
```
['one',
 'ONE',
 'two',
 'three',
 'four',
 'five',
 'six',
 'seven',
 'eight',
 'nine',
 'ten']
```

In [199]:
```
mylist.remove('ONE') # Remove item "ONE"
mylist
```

Out[199]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

In [200]:
```
mylist.pop() # Remove last item of the list
mylist
```

Out[200]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [202]:
```
mylist.pop(7) # Remove item at index location 8
mylist
```

Out[202]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven']
```

In [204]:
```
del mylist[6] # Remove item at index location 7
mylist
```

Out[204]:
```
['one', 'two', 'three', 'four', 'five', 'six']
```

In [205]:
```
# Change value of the string
mylist[0] = 1
mylist[1] = 2
mylist[2] = 3
mylist
```

Out[205]:
```
[1, 2, 3, 'four', 'five', 'six']
```

In [206]:
```
mylist.clear() # Empty List / Delete all items in the list
mylist
```

Out[206]:
```
[]
```

In [207]:
```
del mylist # Delete the whole list
mylist
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [207], in <cell line: 2>()
      1 del mylist # Delete the whole list
----> 2 mylist

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

In [208]:
```
mylist = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [209]:
```
mylist1 = mylist # Create a new reference "mylist1"
```

In [210]:
```
id(mylist) , id(mylist1) # The address of both mylist & mylist1 will be the same
```

Out[210]:
```
(2300846094720, 2300846094720)
```

In [211]:
```
mylist2 = mylist.copy() # Create a copy of the list
```

In [212]:
```
id(mylist2) # The address of mylist2 will be different from mylist because mylist
```

Out[212]:
```
2300846166656
```

In [213]:
```
mylist[0] = 1
```

In [214]:
```
mylist
```

Out[214]:
```
[1, 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [215]:
```
mylist1 # mylist1 will be also impacted as it is pointing to the same list
```

Out[215]:
```
[1, 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [216]:
```
mylist2 # Copy of list won't be impacted due to changes made on the original list
```

Out[216]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [217]:
```
list1 = ['one', 'two', 'three', 'four']
list2 = ['five', 'six', 'seven', 'eight']
```

In [218]:
```
list3 = list1 + list2 # Join two lists by '+' operator
list3
```

Out[218]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [219]:
```
list1.extend(list2) #Append list2 with list1
list1
```

Out[219]: ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

In [220]:
```
list1
```

Out[220]: ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

In [221]:
```
'one' in list1 # Check if 'one' exist in the list
```

Out[221]: True

In [222]:
```
'ten' in list1 # Check if 'ten' exist in the list
```

Out[222]: False

In [223]:
```
if 'three' in list1: # Check if 'three' exist in the list
 print('Three is present in the list')
else:
 print('Three is not present in the list')
```

```
Three is present in the list
```

In [224]:
```
if 'eleven' in list1: # Check if 'eleven' exist in the list
 print('eleven is present in the list')
else:
 print('eleven is not present in the list')
```

```
eleven is not present in the list
```

In [225]:
```
list1
```

Out[225]: ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

In [226]:
```
list1.reverse() # Reverse the list
list1
```

Out[226]: ['eight', 'seven', 'six', 'five', 'four', 'three', 'two', 'one']

In [227]:
```
list1 = list1[::-1] # Reverse the list
list1
```

Out[227]: ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

In [228]:
```
mylist3 = [9,5,2,99,12,88,34]
mylist3.sort() # Sort list in ascending order
mylist3
```

Out[228]: [2, 5, 9, 12, 34, 88, 99]

In [229]:
```
mylist3 = [9,5,2,99,12,88,34]
mylist3.sort(reverse=True) # Sort list in descending order
mylist3
```

Out[229]: [99, 88, 34, 12, 9, 5, 2]

In [230]:
```
mylist4 = [88,65,33,21,11,98]
sorted(mylist4)
```

Out[230]: [11, 21, 33, 65, 88, 98]

In [231]:
```
mylist4
```

Out[231]: [88, 65, 33, 21, 11, 98]

In [232]:
```
list1
```

Out[232]: ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

In [233]:
```
for i in list1:
 print(i)
```

```
one
two
three
four
five
six
seven
eight
```

In [234]:
```
for i in enumerate(list1):
 print(i)
```

```
(0, 'one')
(1, 'two')
(2, 'three')
(3, 'four')
(4, 'five')
(5, 'six')
(6, 'seven')
(7, 'eight')
```

In [235]:
```
list10 =['one', 'two', 'three', 'four', 'one', 'one', 'two', 'three']
```

In [236]:
```
list10.count('one') # Number of times item "one" occurred in the list.
```

Out[236]: 3

In [237]:
```
list10.count('two') # Occurence of item 'two' in the list
```

Out[237]: 2

In [238]:
```
list10.count('four') #Occurence of item 'four' in the list
```

Out[238]:
```
1
```

In [ ]:
```
#All / Any
#The all() method returns:
#True - If all elements in a list are true
#False - If any element in a list is false
#The any() function returns True if any element in the list is True. If not, any(
```

In [239]:
```
L1 = [1,2,3,4,0]
```

In [240]:
```
all(L1) # Will Return false as one value is false (Value 0)
```

Out[240]:
```
False
```

In [241]:
```
any(L1) # Will Return True as we have items in the list with True value
```

Out[241]:
```
True
```

In [242]:
```
L2 = [1,2,3,4,True,False]
```

In [243]:
```
all(L2) # Returns false as one value is false
```

Out[243]:
```
False
```

In [244]:
```
any(L2) # Will Return True as we have items in the list with True value
```

Out[244]:
```
True
```

In [245]:
```
L3 = [1,2,3,True]
```

In [246]:
```
all(L3) # Will return True as all items in the list are True
```

Out[246]:
```
True
```

In [247]:
```
any(L3) # Will Return True as we have items in the list with True value
```

Out[247]:
```
True
```

In [ ]:
```
#List Comprehensions
#List Comprehensions provide an elegant way to create new lists.
#It consists of brackets containing an expression followed by a for clause, then
```

In [248]:
```
mystring = "WELCOME"
mylist = [ i for i in mystring ] # Iterating through a string Using List Comprehe
mylist
```

Out[248]:
```
['W', 'E', 'L', 'C', 'O', 'M', 'E']
```

In [249]:
```
mylist1 = [ i for i in range(40) if i % 2 == 0] # Display all even numbers betwee
mylist1
```

Out[249]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38]
```

In [250]:
```
mylist2 = [ i for i in range(40) if i % 2 == 1] # Display all odd numbers betweer
mylist2
```

Out[250]:
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39]
```

In [251]:
```
mylist3 = [num**2 for num in range(10)] # calculate square of all numbers betweer
mylist3
```

Out[251]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [252]:
```
#List all numbers divisible by 3 , 9 & 12 using nested "if" with List Comprehensi
mylist4 = [i for i in range(200) if i % 3 == 0 if i % 9 == 0 if i % 12 == 0]
mylist4
```

Out[252]:
```
[0, 36, 72, 108, 144, 180]
```

In [255]:
```
# Odd even test
l1 = [print("{} is Even Number".format(i)) if i%2==0 else print("{} is odd numbe
l1
```

```
  Input In [255]
    l1 = [print("{} is Even Number".format(i)) if i%2==0 else print("{} is odd
 number".format(i)) if i%2==1]

 ^
SyntaxError: invalid syntax
```

In [256]:
```
# Extract numbers from a string
mystr = "One 1 two 2 three 3 four 4 five 5 six 6789"
numbers = [i for i in mystr if i.isdigit()]
numbers
```

Out[256]:
```
['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

In [257]:
```
# Extract letters from a string
mystr = "One 1 two 2 three 3 four 4 five 5 six 6789"
numbers = [i for i in mystr if i.isalpha()]
numbers
```

Out[257]:
```
['O',
 'n',
 'e',
 't',
 'w',
 'o',
 't',
 'h',
 'r',
 'e',
 'e',
 'f',
 'o',
 'u',
 'r',
 'f',
 'i',
 'v',
 'e',
 's',
 'i',
 'x']
```

In [258]:
```
tup1 = () # Empty tuple
```

In [259]:
```
tup2 = (10,30,60) # tuple of integers numbers
```

In [260]:
```
tup3 = (10.77,30.66,60.89) # tuple of float numbers
```

In [261]:
```
tup4 = ('one','two' , "three") # tuple of strings
```

In [262]:
```
tup5 = ('Asif', 25 ,(50, 100),(150, 90)) # Nested tuples
```

In [263]:
```
tup6 = (100, 'Asif', 17.765) # Tuple of mixed data types
```

In [264]:
```
tup7 = ('Asif', 25 ,[50, 100],[150, 90] , {'John' , 'David'} , (99,22,33))
```

In [265]:
```
len(tup7) #Length of list
```

Out[265]:
```
6
```

In [266]:
```
tup2[0] # Retreive first element of the tuple
```

Out[266]:
```
10
```

In [267]:
```
tup4[0] # Retreive first element of the tuple
```

Out[267]:
```
'one'
```

In [268]:
```
tup4[0][0] # Nested indexing - Access the first character of the first tuple elem
```

Out[268]:
```
'o'
```

In [269]:
```
tup4[-1] # Last item of the tuple
```

Out[269]:
```
'three'
```

In [270]:
```
tup5[-1] # Last item of the tuple
```

Out[270]:
```
(150, 90)
```

In [ ]:
```
# Loop through a tuple
```

In [6]:
```
tup3
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [6], in <cell line: 1>()
----> 1 tup3

NameError: name 'tup3' is not defined
```

del mytuple # Deleting entire tuple object is possible

In [7]:
```
mytuple
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [7], in <cell line: 1>()
----> 1 mytuple

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

In [8]:
```
mytuple = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight')
```

In [9]:
```
mytuple
```

Out[9]:
```
('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

In [10]:
```
for i in mytuple:
 print(i)
```

```
one
two
three
four
five
six
seven
eight
```

In [11]:
```
for i in enumerate(mytuple):
 print(i)
```

```
(0, 'one')
(1, 'two')
(2, 'three')
(3, 'four')
(4, 'five')
(5, 'six')
(6, 'seven')
(7, 'eight')
```

In [12]:
```
mytuple1 =('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

In [13]:
```
mytuple1.count('one') # Number of times item "one" occurred in the tuple.
```

Out[13]: 3

In [14]:
```
mytuple1.count('two') # Occurence of item 'two' in the tuple
```

Out[14]: 2

In [15]:
```
mytuple1.count('four') #Occurence of item 'four' in the tuple
```

Out[15]: 1

In [16]:
```
mytuple
```

Out[16]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

In [17]:
```
'one' in mytuple # Check if 'one' exist in the list
```

Out[17]: True

In [18]:
```
'ten' in mytuple # Check if 'ten' exist in the list
```

Out[18]: False

In [19]:
```
if 'three' in mytuple: # Check if 'three' exist in the list
 print('Three is present in the tuple')
else:
 print('Three is not present in the tuple')
```

```
Three is present in the tuple
```

In [20]:
```
if 'eleven' in mytuple: # Check if 'eleven' exist in the list
 print('eleven is present in the tuple')
else:
 print('eleven is not present in the tuple')
```

```
eleven is not present in the tuple
```

In [21]:
```
mytuple
```

Out[21]:
```
('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

In [22]:
```
mytuple.index('one') # Index of first element equal to 'one'
```

Out[22]:
```
0
```

In [23]:
```
mytuple.index('five') # Index of first element equal to 'five
```

Out[23]:
```
4
```

In [24]:
```
mytuple1
```

Out[24]:
```
('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

In [25]:
```
mytuple1.index('one') # Index of first element equal to 'one'
```

Out[25]:
```
0
```

In [26]:
```
mytuple2 = (43,67,99,12,6,90,67)
```

In [27]:
```
sorted(mytuple2)
```

Out[27]:
```
[6, 12, 43, 67, 67, 90, 99]
```

In [28]:
```
sorted(mytuple2, reverse=True) # Sort in descending orde
```

Out[28]:
```
[99, 90, 67, 67, 43, 12, 6]
```

In [ ]:
```
#Sets
#1) Unordered & Unindexed collection of items.
#2) Set elements are unique. Duplicate elements are not allowed.
#3) Set elements are immutable (cannot be changed).
#4) Set itself is mutable. We can add or remove items from it.
```

In [29]:
```
myset = {1,2,3,4,5} # Set of numbers
myset
```

Out[29]:
```
{1, 2, 3, 4, 5}
```

In [30]:
```
len(myset) #Length of the set
```

Out[30]:
```
5
```

In [31]:
```
my_set = {1,1,2,2,3,4,5,5}
my_set
```

Out[31]:
```
{1, 2, 3, 4, 5}
```

In [32]:
```
myset1 = {1.79,2.08,3.99,4.56,5.45} # Set of float numbers
myset1
```

Out[32]:
```
{1.79, 2.08, 3.99, 4.56, 5.45}
```

In [33]:
```
myset2 = {'Asif' , 'John' , 'Tyrion'} # Set of Strings
myset2
```

Out[33]:
```
{'Asif', 'John', 'Tyrion'}
```

In [34]:
```
myset3 = {10,20, "Hola", (11, 22, 32)} # Mixed datatypes
myset3
```

Out[34]:
```
{(11, 22, 32), 10, 20, 'Hola'}
```

In [35]:
```
myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable items like lis
myset3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [35], in <cell line: 1>()
----> 1 myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable item
s like list
      2 myset3

TypeError: unhashable type: 'list'
```

In [36]:
```
myset4 = set() # Create an empty set
print(type(myset4))
```

```
<class 'set'>
```

In [39]:
```
my_set1 = set(('one' , 'two' , 'three' , 'four'))
my_set1
```

Out[39]:
```
{'four', 'one', 'three', 'two'}
```

In [40]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
for i in myset:
 print(i)
```

```
six
one
four
two
five
three
eight
seven
```

In [41]:
```
for i in enumerate(myset):
 print(i)
```

```
(0, 'six')
(1, 'one')
(2, 'four')
(3, 'two')
(4, 'five')
(5, 'three')
(6, 'eight')
(7, 'seven')
```

In [42]:
```
myset
```

Out[42]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [43]:
```
'one' in myset # Check if 'one' exist in the set
```

Out[43]:
```
True
```

In [44]:
```
'ten' in myset # Check if 'ten' exist in the set
```

Out[44]:
```
False
```

In [45]:
```
if 'three' in myset: # Check if 'three' exist in the set
 print('Three is present in the set')
else:
 print('Three is not present in the set')
```

```
Three is present in the set
```

In [46]:
```
if 'eleven' in myset: # Check if 'eleven' exist in the list
 print('eleven is present in the set')
else:
 print('eleven is not present in the set')
```

```
eleven is not present in the set
```

In [47]:
```
myset
```

Out[47]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [48]:
```
myset.add('NINE') # Add item to a set using add() method
myset
```

Out[48]:
```
{'NINE', 'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [49]:
```
myset.update(['TEN' , 'ELEVEN' , 'TWELVE']) # Add multiple item to a set using u
myset
```

Out[49]:
```
{'ELEVEN',
 'NINE',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [51]:
```
myset.remove('ELEVEN') # remove item in a set using remove() method
myset
```

Out[51]:
```
{'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [53]:
```
myset.discard('eight') # remove item from a set using discard() method
myset
```

Out[53]:
```
{'TWELVE', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [54]:
```
myset.clear() # Delete all items in a set
myset
```

Out[54]:
```
set()
```

In [55]:
```
del myset # Delete the set object
myset
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [55], in <cell line: 2>()
      1 del myset # Delete the set object
----> 2 myset

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

In [56]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
myset
```

Out[56]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

In [57]:
```
myset1 = myset # Create a new reference "myset1"
myset1
```

Out[57]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

In [58]:
```
myset
```

Out[58]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

In [59]:
```
myset1
```

Out[59]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

In [60]:
```
id(myset) , id(myset1) # The address of both myset & myset1 will be the same as
```

Out[60]: (2177425025280, 2177425025280)

In [61]:
```
my_set = myset.copy() # Create a copy of the list
my_set
```

Out[61]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

In [62]:
```
id(my_set) # The address of my_set will be different from myset because my_set is
```

Out[62]: 2177425145920

In [63]:
```
myset.add('nine')
myset
```

Out[63]: {'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}

In [64]:
```
myset1 # myset1 will be also impacted as it is pointing to the same Set
```

Out[64]: {'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}

In [65]:
```
my_set # Copy of the set won't be impacted due to changes made on the original Se
```

Out[65]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [66]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
C = {8,9,10}
```

In [67]:
```
A | B # Union of A and B (All elements from both sets. NO DUPLICATES)
```

Out[67]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [68]:
```
A.union(B) # Union of A and B
```

Out[68]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [69]:
```
A.union(B, C) # Union of A, B and C.
```

Out[69]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [70]:
```
"""
Updates the set calling the update() method with union of A , B & C.
For below example Set A will be updated with union of A,B & C.
"""
A.update(B,C)
A
```

Out[70]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [71]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [72]:
```
A & B # Intersection of A and B (Common items in both sets)
```

Out[72]:
```
{4, 5}
```

In [73]:
```
A.intersection(B) Intersection of A and B
```

```
  Input In [73]
    A.intersection(B) Intersection of A and B
                      ^
SyntaxError: invalid syntax
```

In [74]:
```
"""
Updates the set calling the intersection_update() method with the intersection of
For below example Set A will be updated with the intersection of A & B.
"""
A.intersection_update(B)
A
```

Out[74]:
```
{4, 5}
```

In [75]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [76]:
```
A - B # set of elements that are only in A but not in B
```

Out[76]: {1, 2, 3}

In [77]:
```
A.difference(B) # Difference of sets
```

Out[77]: {1, 2, 3}

In [78]:
```
B- A # set of elements that are only in B but not in A
```

Out[78]: {6, 7, 8}

In [79]:
```
B.difference(A)
```

Out[79]: {6, 7, 8}

In [80]:
```
"""
Updates the set calling the difference_update() method with the difference of set
For below example Set B will be updated with the difference of B & A.
"""
B.difference_update(A)
B
```

Out[80]: {6, 7, 8}

In [81]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [82]:
```
A ^ B # Symmetric difference (Set of elements in A and B but not in both. "EXCLUD
```

Out[82]: {1, 2, 3, 6, 7, 8}

In [83]:
```
A.symmetric_difference(B) # Symmetric difference of sets
```

Out[83]: {1, 2, 3, 6, 7, 8}

In [84]:
```
"""
Updates the set calling the symmetric_difference_update() method with the symmetr
For below example Set A will be updated with the symmetric difference of A & B.
"""
A.symmetric_difference_update(B)
A
```

Out[84]: {1, 2, 3, 6, 7, 8}

In [85]:
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,20,30,40}
```

In [86]:
```
B.issubset(A) # Set B is said to be the subset of set A if all elements of B are
```

Out[86]:
```
True
```

In [87]:
```
A.issuperset(B) # Set A is said to be the superset of set B if all elements of B
```

Out[87]:
```
True
```

In [88]:
```
C.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common el
```

Out[88]:
```
True
```

In [89]:
```
B.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common el
```

Out[89]:
```
False
```

In [90]:
```
B.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common el
```

Out[90]:
```
False
```

In [ ]:
```
# Other Builtin functions
```

In [1]:
```
A
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <cell line: 1>()
----> 1 A

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

In [2]:
```
A = {1,2,3,4,5,6,7,8,9}
```

In [3]:
```
A
```

Out[3]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [4]:
```
sum(A)
```

Out[4]:
```
45
```

In [5]:
```
max(A)
```

Out[5]:
```
9
```

In [6]:
```
min(A)
```

Out[6]:
```
1
```

In [8]:
```
len(A)
```

Out[8]:
```
9
```

In [9]:
```
list(enumerate(A))
```

Out[9]:
```
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]
```

In [10]:
```
D= sorted(A,reverse=True)
D
```

Out[10]:
```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [11]:
```
sorted(D)
```

Out[11]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [ ]:
```
#Dictionary
#Dictionary is a mutable data type in Python.
#A python dictionary is a collection of key and value pairs separated by a colon
#in curly braces {}.
#Keys must be unique in a dictionary, duplicate values are allowed.
```

In [12]:
```
mydict = dict() # empty dictionary
mydict
```

Out[12]:
```
{}
```

In [13]:
```
mydict = {} # empty dictionary
mydict
```

Out[13]:
```
{}
```

In [14]:
```
mydict = {1:'one' , 2:'two' , 3:'three'} # dictionary with integer keys
mydict
```

Out[14]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [15]:
```
mydict = dict({1:'one' , 2:'two' , 3:'three'}) # Create dictionary using dict()
mydict
```

Out[15]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [16]:
```
mydict = {'A':'one' , 'B':'two' , 'C':'three'} # dictionary with character keys
mydict
```

Out[16]:
```
{'A': 'one', 'B': 'two', 'C': 'three'}
```

In [17]:
```
mydict = {1:'one' , 'A':'two' , 3:'three'} # dictionary with mixed keys
mydict
```

Out[17]:
```
{1: 'one', 'A': 'two', 3: 'three'}
```

In [18]:
```
mydict.keys() # Return Dictionary Keys using keys() method
```

Out[18]:
```
dict_keys([1, 'A', 3])
```

In [19]:
```
mydict.values() # Return Dictionary Values using values() method
```

Out[19]:
```
dict_values(['one', 'two', 'three'])
```

In [20]:
```
mydict.items() # Access each key-value pair within a dictionary
```

Out[20]:
```
dict_items([(1, 'one'), ('A', 'two'), (3, 'three')])
```

In [21]:
```
mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria']} # dictionary with
mydict
```

Out[21]:
```
{1: 'one', 2: 'two', 'A': ['asif', 'john', 'Maria']}
```

In [23]:
```
mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria'], 'B':('Bat' , 'cat'
mydict
```

Out[23]:
```
{1: 'one', 2: 'two', 'A': ['asif', 'john', 'Maria'], 'B': ('Bat', 'cat')}
```

In [24]:
```
mydict = {1:'one' , 2:'two' , 'A':{'Name':'asif' , 'Age' :20}, 'B':('Bat' , 'cat'
mydict
```

Out[24]:
```
{1: 'one', 2: 'two', 'A': {'Name': 'asif', 'Age': 20}, 'B': ('Bat', 'cat')}
```

In [25]:
```
keys = {'a' , 'b' , 'c' , 'd'}
mydict3 = dict.fromkeys(keys) # Create a dictionary from a sequence of keys
mydict3
```

Out[25]:
```
{'b': None, 'c': None, 'a': None, 'd': None}
```

In [26]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = 10
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of k
mydict3
```

Out[26]:
```
{'b': 10, 'c': 10, 'a': 10, 'd': 10}
```

In [27]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = [10,20,30]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of k
mydict3
```

Out[27]:
```
{'b': [10, 20, 30], 'c': [10, 20, 30], 'a': [10, 20, 30], 'd': [10, 20, 30]}
```

In [28]:
```
value.append(40)
mydict3
```

Out[28]:
```
{'b': [10, 20, 30, 40],
 'c': [10, 20, 30, 40],
 'a': [10, 20, 30, 40],
 'd': [10, 20, 30, 40]}
```

In [ ]:
```
#Accessing Items
```

In [29]:
```
mydict = {1:'one' , 2:'two' , 3:'three' , 4:'four'}
mydict
```

Out[29]:
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [30]:
```
mydict[1] # Access item using key
```

Out[30]:
```
'one'
```

In [31]:
```
mydict.get(1) # Access item using get() method
```

Out[31]:
```
'one'
```

In [32]:
```
mydict1 = {'Name':'Asif' , 'ID': 74123 , 'DOB': 1991 , 'job' :'Analyst'}
mydict1
```

Out[32]:
```
{'Name': 'Asif', 'ID': 74123, 'DOB': 1991, 'job': 'Analyst'}
```

In [33]:
```
mydict1['Name'] # Access item using key
```

Out[33]:
```
'Asif'
```

In [34]:
```
mydict1.get('job') # Access item using get() metho
```

Out[34]:
```
'Analyst'
```

In [35]:
```
mydict1['DOB'] = 1992 # Changing Dictionary Items
mydict1['Address'] = 'Delhi'
mydict1
```

Out[35]:
```
{'Name': 'Asif',
 'ID': 74123,
 'DOB': 1992,
 'job': 'Analyst',
 'Address': 'Delhi'}
```

In [36]:
```
dict1 = {'DOB':1995}
mydict1.update(dict1)
mydict1
```

Out[36]:
```
{'Name': 'Asif',
 'ID': 74123,
 'DOB': 1995,
 'job': 'Analyst',
 'Address': 'Delhi'}
```

In [46]:
```
mydict1
```

Out[46]:
```
{'Name': 'Asif',
 'ID': 74123,
 'DOB': 1995,
 'job': 'Analyst',
 'Address': 'Delhi'}
```

In [47]:
```
mydict1.popitem() # A random item is removed
```

Out[47]:
```
('Address', 'Delhi')
```

In [48]:
```
mydict1
```

Out[48]:
```
{'Name': 'Asif', 'ID': 74123, 'DOB': 1995, 'job': 'Analyst'}
```

In [50]:
```
del[mydict1['job']] # Removing item using del method
mydict1
```

Out[50]:
```
{'Name': 'Asif', 'DOB': 1995}
```

In [51]:
```
mydict1.clear() # Delete all items of the dictionary using clear method
mydict1
```

Out[51]:
```
{}
```

In [52]:
```
del mydict1 # Delete the dictionary object
mydict1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [52], in <cell line: 2>()
      1 del mydict1 # Delete the dictionary object
----> 2 mydict1

NameError: name 'mydict1' is not defined
```

In [ ]:
```
#Copy Dictionary
```

In [53]:
```
mydict = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict
```

Out[53]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [54]:
```
mydict1 = mydict # Create a new reference "mydict1"
```

In [55]:
```
id(mydict) , id(mydict1) # The address of both mydict & mydict1 will be the same
```

Out[55]:
```
(1918737565824, 1918737565824)
```

In [56]:
```
mydict2 = mydict.copy() # Create a copy of the dictionary
```

In [57]:
```
id(mydict2) # The address of mydict2 will be different from mydict because mydict
```

Out[57]:
```
1918766803328
```

In [58]:
```
mydict['Address'] = 'Mumbai'
```

In [59]:
```
mydict
```

Out[59]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}
```

In [60]:
```
mydict1 # mydict1 will be also impacted as it is pointing to the same dictionary
```

Out[60]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}
```

In [61]:
```
mydict2 # Copy of list won't be impacted due to the changes made in the original
```

Out[61]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [ ]:
```
# Loop through a Dictionary
```

In [62]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki' }
mydict1
```

Out[62]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [63]:
```
for i in mydict1:
 print(i , ':' , mydict1[i]) # Key & value pair
```

```
Name : Asif
ID : 12345
DOB : 1991
Address : Hilsinki
```

In [64]:
```
for i in mydict1:
 print(mydict1[i]) # Dictionary items
```

```
Asif
12345
1991
Hilsinki
```

In [ ]:
```
#Dictionary Membership
```

In [65]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[65]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [66]:
```
'Name' in mydict1 # Test if a key is in a dictionary or not.
```

Out[66]:
```
True
```

In [67]:
```
'Asif' in mydict1 # Membership test can be only done for keys.
```

Out[67]:
```
False
```

In [68]:
```
'ID' in mydict1
```

Out[68]:
```
True
```

In [69]:
```
'Address' in mydict1
```

Out[69]:
```
False
```

In [ ]:
```
#All / Any
#The all() method returns:
#True - If all all keys of the dictionary are true
#False - If any key of the dictionary is false
#The any() function returns True if any key of the dictionary is True. If not, ar
```

In [70]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[70]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [71]:
```
all(mydict1) # Will Return true as one value is true (Value 0)
```

Out[71]:
```
True
```

In [77]:
```
any(mydict1) # Will Return True as we have items in the dictionary with True val
```

Out[77]:
```
True
```

In [73]:
```
mydict1[0] = 'test1'
mydict1
```

Out[73]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst', 0: 'test1'}
```

In [74]:
```
all(mydict1) # Returns false as one value is false
```

Out[74]:
```
False
```

In [75]:
```
any(mydict1) # Will Return True as we have items in the dictionary with True va
```

Out[75]:
```
True
```

In [ ]:
```
#Dictionary Comprehension
```

In [76]:
```
double = {i:i*2 for i in range(10)} #double each value using dict comprehension
double
```

Out[76]:
```
{0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
```

In [78]:
```
square = {i:i**2 for i in range(10)}
square
```

Out[78]:
```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

In [79]:
```
key = ['one' , 'two' , 'three' , 'four' , 'five']
value = [1,2,3,4,5]
mydict = {k:v for (k,v) in zip(key,value)} # using dict comprehension to create d
mydict
```

Out[79]:
```
{'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5}
```

In [80]:
```
mydict1 = {'a':10 , 'b':20 , 'c':30 , 'd':40 , 'e':50}
mydict1 = {k:v/10 for (k,v) in mydict1.items()} # Divide all values in a dictionc
mydict1
```

Out[80]:
```
{'a': 1.0, 'b': 2.0, 'c': 3.0, 'd': 4.0, 'e': 5.0}
```

In [81]:
```
str1 = "Natural Language Processing"
mydict2 = {k:v for (k,v) in enumerate(str1)} # Store enumerated values in a dict
mydict2
```

Out[81]:
```
{0: 'N',
 1: 'a',
 2: 't',
 3: 'u',
 4: 'r',
 5: 'a',
 6: 'l',
 7: ' ',
 8: 'L',
 9: 'a',
 10: 'n',
 11: 'g',
 12: 'u',
 13: 'a',
 14: 'g',
 15: 'e',
 16: ' ',
 17: 'P',
 18: 'r',
 19: 'o',
 20: 'c',
 21: 'e',
 22: 's',
 23: 's',
 24: 'i',
 25: 'n',
 26: 'g'}
```

In [82]:
```
str1 = "abcdefghijklmnopqrstuvwxyz"
mydict3 = {i:i.upper() for i in str1} # Lower to Upper Case
mydict3
```

Out[82]:
```
{'a': 'A',
 'b': 'B',
 'c': 'C',
 'd': 'D',
 'e': 'E',
 'f': 'F',
 'g': 'G',
 'h': 'H',
 'i': 'I',
 'j': 'J',
 'k': 'K',
 'l': 'L',
 'm': 'M',
 'n': 'N',
 'o': 'O',
 'p': 'P',
 'q': 'Q',
 'r': 'R',
 's': 'S',
 't': 'T',
 'u': 'U',
 'v': 'V',
 'w': 'W',
 'x': 'X',
 'y': 'Y',
 'z': 'Z'}
```

In [83]:
```
str1 = "abcdefghijklmnopqrstuvwxyz"
mydict3 = {i:i.upper() for i in str1} # Lower to Upper Case
mydict3
```

Out[83]:
```
{'a': 'A',
 'b': 'B',
 'c': 'C',
 'd': 'D',
 'e': 'E',
 'f': 'F',
 'g': 'G',
 'h': 'H',
 'i': 'I',
 'j': 'J',
 'k': 'K',
 'l': 'L',
 'm': 'M',
 'n': 'N',
 'o': 'O',
 'p': 'P',
 'q': 'Q',
 'r': 'R',
 's': 'S',
 't': 'T',
 'u': 'U',
 'v': 'V',
 'w': 'W',
 'x': 'X',
 'y': 'Y',
 'z': 'Z'}
```

In [84]:
```
mystr4 = "one two three four one two two three five five six seven six seven one"
```

In [85]:
```
mylist = mystr4.split() # Split String into substrings
mylist
```

Out[85]:
```
['one',
 'two',
 'three',
 'four',
 'one',
 'two',
 'two',
 'three',
 'five',
 'five',
 'six',
 'seven',
 'six',
 'seven',
 'one']
```

In [86]:
```
mylist1 = set(mylist) # Unique values in a list
mylist1 = list (mylist1)
mylist1
```

Out[86]:
```
['four', 'seven', 'six', 'two', 'five', 'one', 'three']
```

In [95]:
```
# Calculate frequenct of each word
count1 = [0] * len(mylist1)
mydict5 = dict()
for i in range(len(mylist1)):
    for j in range(len(mylist)):
     if mylist1[i] == mylist[j]:
        count1[i] += 1
    mydict5[mylist1[i]] = count1[i]
print(mydict5)
```

```
{'four': 1, 'seven': 2, 'six': 2, 'two': 3, 'five': 2, 'one': 3, 'three': 2}
```

In [ ]:
```
#Operators
#Operators are special symbols in Python which are used to perform operations on
```

In [98]:
```
a = 5
b = 2
x = 'Asif'
y = 'Bhat'
# Addition
c = a + b
print('Addition of {} and {} will give :- {}\n'.format(a,b,c))
#Concatenate string using plus operator
z = x+y
print ('Concatenate string \'x\' and \'y\' using \'+\' operaotr :- {}\n'.format(z

# Subtraction
c = a - b
print('Subtracting {} from {} will give :- {}\n'.format(b,a,c))

# Multiplication
c = a * b
print('Multiplying {} and {} will give :- {}\n'.format(a,b,c))

# Division
c = a / b
print('Dividing {} by {} will give :- {}\n'.format(a,b,c))

# Modulo of both number
c = a % b
print('Modulo of {} , {} will give :- {}\n'.format(a,b,c))

# Power
c = a ** b
print('{} raised to the power {} will give :- {}\n'.format(a,b,c))
# Division(floor)
c = a // b
print('Floor division of {} by {} will give :- {}\n'.format(a,b,c))
```

```
Addition of 5 and 2 will give :- 7

Concatenate string 'x' and 'y' using '+' operaotr :- AsifBhat

Subtracting 2 from 5 will give :- 3

Multiplying 5 and 2 will give :- 10

Dividing 5 by 2 will give :- 2.5

Modulo of 5 , 2 will give :- 1

5 raised to the power 2 will give :- 25

Floor division of 5 by 2 will give :- 2
```

In [ ]:
```
#Comparison Operators
#Comparison operators are used to compare values.
```

In [99]:
```
x = 20
y = 30
print('Is x greater than y :- ',x>y)
print('\nIs x less than y :- ',x<y)
print('\nIs x equal to y :- ',x==y)
print('\nIs x not equal to y :- ',x!=y)
print('\nIs x greater than or equal to y :- ',x>=y)
print('\nIs x less than or equal to y :- ',x<=y)
```

```
Is x greater than y :-  False

Is x less than y :-  True

Is x equal to y :-  False

Is x not equal to y :-  True

Is x greater than or equal to y :-  False

Is x less than or equal to y :-  True
```

In [100]:
```
a = 'Asif'
b = 'Bhat'
c = 'Asif'
a == b , a ==c , a != b # Comparison operators on string
```

Out[100]:
```
(False, True, True)
```

In [ ]:
```
#Logical Operators
```

In [101]:
```
x = True
y = False
print('Logical AND operation :- ',x and y) # True if both values are true
print('Logical OR operation :- ',x or y) # True if either of the values is true
print('NOT operation :- ',not x ) # True if operand is false
```

```
Logical AND operation :-  False
Logical OR operation :-  True
NOT operation :-  False
```

In [ ]:
```
#Bitwise operators
#Bitwise operators act on bits and performs bit by bit operation.
```

In [103]:
```
x = 18 # binary form 10010
y = 6 # binary form 00110
print('Bitwise AND operation - {}'.format(x&y))
print('Bitwise OR operation - {}'.format(x|y))
print('Bitwise XOR operation - {}'.format(x^y))
print('Bitwise NOT operation - {}'.format(~x))
print('Bitwise right shift operation - {}'.format(x>>2))
print('Bitwise left shift operation - {}'.format(x<<2))
```

```
Bitwise AND operation - 2
Bitwise OR operation - 22
Bitwise XOR operation - 20
Bitwise NOT operation - -19
Bitwise right shift operation - 4
Bitwise left shift operation - 72
```

In [ ]:
```
#Assignment Operators
```

In [104]:
```
x = 10
print('Initialize x with value 10 (x=10)) :- ',x)
x+=20 # x = x+20
print ('Add 20 to x :- ',x)
x-=20 # x = x-20
print ('subtract 20 from x :- ',x)
x/=10 # x = x/10
print ('Divide x by 10 :- ',x)
x*=10 # x = x/10
print ('Multiply x by 10 :- ',x)
x = int(x)
x**=2 # x = x/10
print ('x raised to the power 2 :- ',x)
x%=2
print ('Modulo Division :- ',x)
x = 20
x//=3
print ('Floor Division :- ',x)
x&=2
print('Bitwise AND :- ',x)
x|=2
print('Bitwise OR :- ',x)
x^=2
print('Bitwise XOR :- ',x)
x = 10
x<<=2
print('Bitwise left shift operation',x)
x>>=2
```

```
Initialize x with value 10 (x=10)) :-  10
Add 20 to x :-  30
subtract 20 from x :-  10
Divide x by 10 :-  1.0
Multiply x by 10 :-  10.0
x raised to the power 2 :-  100
Modulo Division :-  0
Floor Division :-  6
Bitwise AND :-  2
Bitwise OR :-  2
Bitwise XOR :-  0
Bitwise left shift operation 40
```

In [ ]:
```
#Membership Operators
#Membership Operators are used to test whether a value / variable is present in a
```

In [105]:
```
mystr = 'Asif Ali Bhat'
'Asif' in mystr , 'John' in mystr
```

Out[105]: (True, False)

In [106]:
```
mystr = 'Asif Ali Bhat'
'Asif' not in mystr , 'John' not in mystr
```

Out[106]: (False, True)

In [ ]:
```
#Functions
#A function is a block of organized code written to carry out a specified task.
#Functions help break our program into smaller and modular chunks for better reac
#Information can be passed into a function as arguments.
#Parameters are specified after the function name inside the parentheses.
#We can add as many parameters as we want. Parameters must be separated with a co
#A function may or may not return data.
#In Python a function is defined using the def keyword
```

In [107]:
```
def myfunc():
 print("Hello Python Lovers")
myfunc()
```

```
Hello Python Lovers
```

In [109]:
```
def details(name,userid,country): # Function to print User details
 print('Name :- ', name)
 print('User ID :- ', userid)
 print('Country :- ',country)

details('Asif' , 'asif123' , 'India')
```

```
Name :-  Asif
User ID :-  asif123
Country :-  India
```

In [111]:
```
def details(name,userid,country): # Function to print User details
 print('Name :- ', name)
 print('User ID  :- ', userid)
 print('Country :- ',country)
details('Asif' , 'asif123' , 'India')
```

```
Name :-  Asif
User ID  :-  asif123
Country :-  India
```

In [138]:
```
def even_odd (num): #Even odd test
 """This function will check whether a number is even or odd"""
    if num % 2 ==0:
        print (num, ' is even number')
    else:
        print (num, ' is odd number')
even_odd(3)
even_odd(4)
print(even_odd.__doc__) # Print function documentation string
```

```
  Input In [138]
    if num % 2 ==0:
    ^
IndentationError: unexpected indent
```

In [139]:
```
def fullname (firstname , middlename ,lastname): #Concatenate Strings
 fullname = "{} {} {}".format(firstname,middlename,lastname)
 print (fullname)
fullname('Asif' , 'Ali' , 'Bhat')
```

```
Asif Ali Bhat
```

In [140]:
```
def fullname (firstname , middlename ,lastname): #Concatenate Strings
 fullname = "{} {} {}".format(firstname,middlename,lastname)
 print (fullname)
fullname(lastname = 'Bhat' , middlename='Ali' , firstname='Asif') # Keyword Argu
```

```
Asif Ali Bhat
```

In [141]:
```
fullname ('Asif') # This will throw error as function is expecting 3 arguments.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [141], in <cell line: 1>()
----> 1 fullname ('Asif')

TypeError: fullname() missing 2 required positional arguments: 'middlename' and
'lastname'
```

In [143]:
```
def myfunc(city = 'Mumbai'):
 print('Most Populous City :- ', city)

myfunc() # When a function is called without an argument it will use default val
```

```
Most Populous City :-  Mumbai
```

In [144]:
```
var1 = 100 # Variable with Global scope.
def myfunc():
 print(var1) # Value 100 will be displayed due to global scope of var1

myfunc()
print(var1)
```

```
100
100
```

In [155]:
```
def myfunc1():
    var2 = 10 # Variable with Local scope
print(var2)
def myfunc2():
print(var2) # This will throw error because var2 has a local scope. Var2 is 10
myfunc1()
myfunc2()
```

```
  File <tokenize>:3
    print(var2)
    ^
IndentationError: unindent does not match any outer indentation level
```

In [156]:
```
var1 = 100 # Variable with Global scope.
def myfunc():
 var1 = 99 # Local scope
 print(var1)

myfunc()
print(var1) # The original value of var1 (100) will be retained due to global sc
```

```
99
100
```

In [157]:
```
list1 = [11,22,33,44,55]
def myfunc(list1):
 del list1[0]
print('"List1" before calling the function:- ',list1)
myfunc(list1) # Pass by reference (Any change in the parameter within the functi
print('"List1" after calling the function:- ',list1)
```

```
"List1" before calling the function:-  [11, 22, 33, 44, 55]
"List1" after calling the function:-  [22, 33, 44, 55]
```

In [158]:
```
list1 = [11,22,33,44,55]
def myfunc(list1):
 list1.append(100)
print('"List1" before calling the function:- ',list1)
myfunc(list1) # Pass by reference (Any change in the parameter within the functi
print('"List1" after calling the function:- ',list1)
```

```
"List1" before calling the function:-  [11, 22, 33, 44, 55]
"List1" after calling the function:-  [11, 22, 33, 44, 55, 100]
```

In [159]:
```
list1 = [11,22,33,44,55]
def myfunc(list1):
 list1 = [10,100,1000,10000] # link of 'list1' with previous object is broken
print('"List1" before calling the function:- ',list1)
myfunc(list1) # Pass by reference (Any change in the parameter within the functi
print('"List1" after calling the function:- ',list1)
```

```
"List1" before calling the function:-  [11, 22, 33, 44, 55]
"List1" after calling the function:-  [11, 22, 33, 44, 55]
```

In [160]:
```
def swap(a,b):
 temp = a
 a = b # link of 'a' with previous object is broken now as new object is
 b = temp # link of 'b' with previous object is broken now as new object is
a = 10
b = 20
swap(a,b)
a,b
```

Out[160]: (10, 20)

In [162]:
```
def factorial(num): # Calculate factorial of a number using recursive function c
 if num <=1 :
    return 1
 else:
    return num * factorial(num-1)

factorial(4)
```

Out[162]: 24

In [ ]:
```
def add(num): # Sum of first n natural numbers
 if num == 0:
    return 0
 else:
    return num + add(num-1)
add(5) # Sum of first five natural numbers (1,2,3,4,5)
```

In [3]:
```
def fiboacci(num):
 if num <= 1:
    return num
 if num == 2:
    return 1
 else:
    return(fiboacci(num-1) + fiboacci(num-2))
nums = int(input("How many fibonacci numbers you want to generate -"))
for i in range(nums):
 print(fiboacci(i)) # Generate Fibonacci series
```

```
How many fibonacci numbers you want to generate -10
0
1
1
2
3
5
8
13
21
34
```

In [4]:
```
def add(a,b,c):
    return a+b+c

print(add(10,20,30)) # Sum of two numbers
```

```
60
```

In [ ]:
```
#args & kwargs
```

In [9]:
```
print(add(1,2,3,4)) '''This will throw below error as this function will only tak
If we want to make argument list dynamic then *args wil come in picture'''
```

```
  Input In [9]
    print(add(1,2,3,4)) '''This will throw below error as this function will on
ly take
                        ^
SyntaxError: invalid syntax
```

In [13]:
```
def some_args(arg_1, arg_2, arg_3):
 print("arg_1:", arg_1)
 print("arg_2:", arg_2)
 print("arg_3:", arg_3)
my_list = [2, 3]
some_args(1, *my_list)
```

```
arg_1: 1
arg_2: 2
arg_3: 3
```

In [43]:
```
def add1(*args):
    return sum(args)
  print(add(1,2,3))
 print(add(1,2,3,4))              ------------------------> wrong
 print(add(1,2,3,4,5))
 print(add(1,2,3,4,5,6))
 print(add(1,2,3,4,5,6,7))
```

```
  File <tokenize>:3
    print(add(1,2,3))
    ^
IndentationError: unindent does not match any outer indentation level
```

In [44]:
```
list1 = [1,2,3,4,5,6,7]
tuple1 = (1,2,3,4,5,6,7)
add1(*list1) , add1(*tuple1) #tuple & list items will be passed as argument list
```

Out[44]:
```
(28, 28)
```

In [46]:
```
list1 = [1,2,3,4,5,6,7]
list2 = [1,2,3,4,5,6,7]
list3 = [1,2,3,4,5,6,7]
list4 = [1,2,3,4,5,6,7]
add1(*list1 , *list2 , *list3 , *list4 ) #All four lists are unpacked and each ir
```

Out[46]:
```
112
```

In [47]:
```
def UserDetails(*args):
 print(args)
UserDetails('Asif' , 7412 , 41102 , 33 , 'India' , 'Hindi')
''' For the above example we have no idea about the parameters passed e.g 7412 ,
 In such cases we can take help of Keyworded arguments (**kwargs) '''
```

```
('Asif', 7412, 41102, 33, 'India', 'Hindi')
```

Out[47]:
```
' For the above example we have no idea about the parameters passed e.g 7412 ,
\n In such cases we can take help of Keyworded arguments (**kwargs) '
```

In [48]:
```
def UserDetails(**kwargs):
 print(kwargs)
UserDetails(Name='Asif' , ID=7412 , Pincode=41102 , Age= 33 , Country= 'India')
```

```
{'Name': 'Asif', 'ID': 7412, 'Pincode': 41102, 'Age': 33, 'Country': 'India'}
```

In [62]:
```
def UserDetails(**kwargs):
    for key,val in kwargs.items():
    print("{} :- {}".format(key,val))
UserDetails(Name='Asif', ID=7412 , Pincode=41102 , Age= 33 , Country= 'India', 'h
```

```
  Input In [62]
    print("{} :- {}".format(key,val))
    ^
IndentationError: expected an indented block
```

In [63]:
```
mydict = {'Name': 'Asif', 'ID': 7412, 'Pincode': 41102, 'Age': 33, 'Country': 'In
UserDetails(**mydict)
```

```
{'Name': 'Asif', 'ID': 7412, 'Pincode': 41102, 'Age': 33, 'Country': 'India'}
```

In [80]:
```
def UserDetails(licenseNo, *args , phoneNo=0 , **kwargs): # Using all four argum
 print('License No :- ', licenseNo)
 j=''
for i in args:
    j = j+i
  print('Full Name :-', j)
  print('Phone Number:- ',phoneNo)                          ------------------> pg68
for key,val in kwargs.items():
 print("{} :- {}".format(key,val))


name = ['Asif' , ' ' , 'Ali' , ' ','Bhat']
mydict = {'Name': 'Asif', 'ID': 7412, 'Pincode': 41102, 'Age': 33, 'Country': 'In
UserDetails('BHT145' , *name , phoneNo=1234567890,**mydict )
```

```
  File <tokenize>:6
    print('Full Name :-', j)
    ^
IndentationError: unindent does not match any outer indentation level
```

In [81]:
```
def UserDetails(licenseNo, *args , phoneNo=0, **kwargs): # Using all four argumer
 print('Nothing')
```

In [82]:
```
def UserDetails(licenseNo, **kwargs , *args): # This will fail. *args MUST come b
 print('Nothing')
```

```
  Input In [82]
    def UserDetails(licenseNo, **kwargs , *args): # This will fail. *args MUST
 come b
                                                  ^
SyntaxError: invalid syntax
```

In [83]:
```
#The below function will fail. Default argument/positional argument (licenseNo) M
def UserDetails(ID = 1, licenseNo, *args):
 print('Nothing')
```

```
  Input In [83]
    def UserDetails(ID = 1, licenseNo, *args):
                                       ^
SyntaxError: non-default argument follows default argument
```

In [ ]:
```
#Lamda
```

In [1]:
```
addition = lambda a : a + 10 # This lambda function adds value 10 to an argument
print(addition(5))
```

```
15
```

In [2]:
```
product = lambda a, b : a * b #This lambda function takes two arguments (a,b) and
print(product(5, 6))
```

```
30
```

In [3]:
```
addition = lambda a, b, c : a + b + c #This lambda function takes three argument
print(addition(5, 6, 2))
```

```
13
```

In [4]:
```
res = (lambda *args: sum(args)) # This lambda function can take any number of arg
res(10,20) , res(10,20,30,40) , res(10,20,30,40,50,60,70)
```

Out[4]: (30, 100, 280)

In [5]:
```
res1 = (lambda **kwargs: sum(kwargs.values())) # This lambda function can take ar
res1(a = 10 , b= 20 , c = 30) , res1(a = 10 , b= 20 , c = 30, d = 40 , e = 50)
```

Out[5]: (60, 150)

In [6]:
```
res1 = (lambda **kwargs: sum(kwargs.values())) # This lambda function can take ar
res1(a = 10 , b= 20 , c = 30) , res1(a = 10 , b= 20 , c = 30, d = 40 , e = 50)
```

Out[6]: (60, 150)

In [1]:
```
# User defined function to find product of numbers
def product(nums):
  total = 1
for i in nums:
 total * = i
  return total
# This lambda function can take any number of arguments and return thier product.
res1 = (lambda **kwargs: product(kwargs.values()))
res1(a = 10 , b= 20 , c = 30) , res1(a = 10 , b= 20 , c = 30, d = 40 , e = 50)
```

```
  Input In [1]
    total * = i
            ^
SyntaxError: invalid syntax
```

Type *Markdown* and LaTeX: α^2