Functions

1. Inbuilt functions
2. User-defined functions
3. Recursion
4. Lambda Expressions
5. High Order Functions

1. Inbuilt functions

max()- to find the maximum in the sequence

In [1]:
```
list1 = [1,2,3,4]
max(list1)
```

Out[1]:
```
4
```

In [2]:
```
max('Python')
```

Out[2]:
```
'y'
```

min()- to find the minimum in the sequence

In [3]:
```
min("Python")
```

Out[3]:
```
'P'
```

len()- to find the length of the squence

In [4]:
```
len('Python')
```

Out[4]:
```
6
```

sum()- to get the sum of the integer items in the list or tuples or sets

In [7]:
```
list1 = [2,4,3] # list
sum(list1)
```

Out[7]:
```
9
```

In [9]:
```
tup1= (2,3,4) #tuple
sum(tup1)
```

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

In [10]:
```
set1={2,3,4} #set
sum(set1)
```

Out[10]:
```
9
```

In [11]:
```
#in dictionary sum will give the sum of all the keys in the dictionary
dic = {1:10,2:20,3:30} #dictionary
sum(dic) #sum of all the keys
```

```
6
```

Out[11]:

Strings - Immutable

Can acess each character of a string through indexing or slicing

Operations in strings

+ operator --> Concatenate the strings

In [12]:
```
s1 = 'Hello'
s2 = 'Python'
s1+s2
```

Out[12]:
```
'HelloPython'
```

*** operator**

In [13]:
```
#Will give the twice Hello
s1 * 2
```

Out[13]:
```
'HelloHello'
```

In [14]:
```
'A'*4 # print 4 times A
```

Out[14]:
```
'AAAA'
```

Membership operator

In [15]:
```
's' in 'Python'
```

Out[15]:
```
False
```

In [16]:
```
's' not in 'Python'
```

Out[16]:
```
True
```

String Methods

capitalize() Converts the first character to upper case

In [20]:
```
s1 = 'Python is a scriPting Language'
s1.capitalize()
```

Out[20]:
```
'Python is a scripting language'
```

swapcase() Swaps cases, lower case becomes upper case and vice versa

In [2]:
```
s1.swapcase()
```

Out[2]:
```
'pYTHON IS A SCRIpTING lANGUAGE'
```

title() Converts the first character of each word to upper case

In [27]:
```
s1.title()
```

Out[27]: 'Python Is A Scripting Language'

lower() Converts a string into lower case

In [29]:
```
s1.lower()
```

Out[29]: 'python is a scripting language'

upper() Converts a string into upper case

In [28]:
```
s1.upper()
```

Out[28]: 'PYTHON IS A SCRIPTING LANGUAGE'

replace() Returns a string where a specified value is replaced with a specified value

In [31]:
```
s1.replace('s','x')
```

Out[31]: 'Python ix a xcriPting Language'

find() Searches the string for a specified value and returns the position of where it was found

In [11]:
```
s1.find('z')
```

Out[11]: -1

In [12]:
```
s1.find('a')
```

Out[12]: 10

In [13]:
```
s1.find('p')
```

Out[13]: -1

In [14]:
```
s1.find('P')
```

Out[14]: 0

In [15]:
```
s1.find('L')
```

Out[15]: 22

In [16]:
```
s1.find('l')
```

Out[16]: -1

rfind() - find the last occurence of a specified value and returns the index number if found

In [26]:
```
s1.rfind('a')
```

Out[26]: 27

In [27]:
```
#finds the first occurence of the character
s1.find('a')
```

Out[27]: 10

In [28]:
```
s1.rfind('x')
```

Out[28]:
```
-1
```

endswith() Returns true if the string ends with the specified value

In [17]:
```
s1.endswith('e')
```

Out[17]:
```
True
```

In [18]:
```
s1.endswith('.')
```

Out[18]:
```
False
```

startswith() Returns true if the string starts with the specified value

In [19]:
```
s1.startswith('P')
```

Out[19]:
```
True
```

In [20]:
```
s1.startswith('p')
```

Out[20]:
```
False
```

split() Splits the string at the specified separator, and returns a list

In [22]:
```
s1.split()
```

Out[22]:
```
['Python', 'is', 'a', 'scriPting', 'Language']
```

In [23]:
```
s1.split('s')
```

Out[23]:
```
['Python i', ' a ', 'criPting Language']
```

In [26]:
```
s2 = 'apple, banana, orange, pear'
s2.split(',')
```

Out[26]:
```
['apple', ' banana', ' orange', ' pear']
```

join() Converts the elements of an iterable into a string

In [40]:
```
list1 = ['BMW','Mercedes','Jaguar','Audi']
list1
```

Out[40]:
```
['BMW', 'Mercedes', 'Jaguar', 'Audi']
```

In [56]:
```
'--'.join(list1)
```

Out[56]:
```
'BMW--Mercedes--Jaguar--Audi'
```

count() Returns the number of times a specified value occurs in a string

In [58]:
```
s1
```

Out[58]:
```
'Python is a scriPting Language'
```

In [61]:
```
s1.count('a')
```

Out[61]:
```
3
```

rstrip() - to remove the specified value from the right side, by default it removes spaces

In [29]:
```
s1 = 'Hello '
s1.rstrip()
```

Out[29]:
```
'Hello'
```

In [30]:
```
s1 = 'HEllo###'
s1.rstrip('#')
```

Out[30]:
```
'HEllo'
```

lstrip() - to remove the specified value from the left side, by default it removes spaces

In [31]:
```
s1 = '  Hello '
s1.lstrip()
```

Out[31]:
```
'Hello '
```

In [33]:
```
s1 = '####HEllo###'
s1.lstrip('#')
```

Out[33]:
```
'HEllo###'
```

In [ ]:
```
###### strip() - to remove the specified value from the both sides, by default it remove
```

In [35]:
```
s1 = '  Hello '
s1.strip()
```

Out[35]:
```
'Hello'
```

In [39]:
```
s1 = '@###HEllo###'
s1.strip('@#')
```

Out[39]:
```
'HEllo'
```

In [40]:
```
s1 = '@#HEllo###'
s1.strip('#@')
```

Out[40]:
```
'HEllo'
```

index() Searches the string for a specified value and returns the position of where it was found

In [41]:
```
s1 = 'Hello'
s1.index('e')
```

Out[41]:
```
1
```

In [42]:
```
s1.index('z')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13240\757536014.py in <module>
----> 1 s1.index('z')

ValueError: substring not found
```

format() Formats specified values in a string

In [47]:
```
l1 = [1,2,3]
'Hello, how are you {}'.format(l1[2])
```

Out[47]: 'Hello, how are you 3'

islower() Returns True if all characters in the string are lower case

In [48]:
```
s1 = 'hello'
s1.islower()
```

Out[48]: True

In [49]:
```
s1 = "Hello"
s1.islower()
```

Out[49]: False

isupper() Returns True if all characters in the string are upper case

In [51]:
```
s1 = "HELLO"
s1.isupper()
```

Out[51]: True

In [50]:
```
s1 = 'Hello'
s1.isupper()
```

Out[50]: False

isnumeric() Returns True if all characters in the string are numeric

In [52]:
```
s1 = '123'
s1.isnumeric()
```

Out[52]: True

In [53]:
```
s1='Hello123'
s1.isnumeric()
```

Out[53]: False

isspace() Returns True if all characters in the string are whitespaces

In [54]:
```
s1='    '
s1.isspace()
```

Out[54]: True

In [55]:
```
s1='H    '
s1.isspace()
```

Out[55]: False

istitle() Returns True if the string follows the rules of a title

In [58]:
```
s1= "Hello World"
s1.istitle()
```

Out[58]: True

In [59]:
```
s1= "Hello world"
s1.istitle()
```

Out[59]: False

isdigit() Returns True if all characters in the string are digits

In [60]:
```
s1 = '123'
s1.isdigit()
```

Out[60]: True

In [61]:
```
s1 = '12a3'
s1.isdigit()
```

Out[61]: False

isalnum() Returns True if all characters in the string are alphanumeric

In [62]:
```
s1 = 'avc123'
s1.isalnum()
```

Out[62]: True

In [63]:
```
s1 = 'avc'
s1.isalnum()
```

Out[63]: True

isalpha() Returns True if all characters in the string are in the alphabet

In [65]:
```
s1 = 'avc'
s1.isalpha()
```

Out[65]: True

In [67]:
```
s1 = '12as'
s1.isalpha()
```

Out[67]: False

In [68]:
```
s1 = '@as'
s1.isalpha()
```

Out[68]: False

Lists -> mutable

Operations in Lists

+ operator --> Concatenate the lists

In [69]:
```
l1 = [1,2,3]
l2 = [7,8,9]
l1 + l2
```

Out[69]: [1, 2, 3, 7, 8, 9]

*** operator in list**

In [119…
```
l1 = [1,2,3]
l1 * 2
```

Out[119]:
```
[1, 2, 3, 1, 2, 3]
```

In [120…
```
l1 = ['P',"y"]
l1 * 3
```

Out[120]:
```
['P', 'y', 'P', 'y', 'P', 'y']
```

Membership operator

In [167…
```
l1 = [1,2,3]
3 in l1
```

Out[167]:
```
True
```

In [168…
```
3 not in l1
```

Out[168]:
```
False
```

Lists Methods

Lists methods make changes in the original list because lists are mutuable

Can acess each element in lists through indexing or slicing

append() Adds an element at the end of the list

In [72]:
```
l1 = [1,2,3]
l1.append(9)
print(l1)
```

```
[1, 2, 3, 9]
```

clear() Removes all the elements from the list

In [73]:
```
l1 = [1,2,3]
l1.clear()
print(l1)
```

```
[]
```

copy() Returns a copy of the list

In [74]:
```
l1 = [1,2,3]
l2 = l1.copy() # copy the entire list in l2
print(l2)
```

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

count() Returns the number of elements with the specified value

In [76]:
```
l1 = [1,2,3]
l1.count(2)
```

Out[76]:
```
1
```

In [79]:
```
l1 = [1,2,3,3,2,1,4,5,2]
l1.count(2)
```

Out[79]: 3

In [81]:
```
l1 = [1,2,3,3,2,1,4,5,2]
l1.count(9) # if number not exist in the list then it returns 0
```

Out[81]: 0

extend() Add the elements of a list (or any iterable), to the end of the current list

In [80]:
```
l1 = [1,2,3]
l1.extend([7,8,9]) #extend adds the iterable
print(l1)
```

```
[1, 2, 3, 7, 8, 9]
```

In [84]:
```
l1 = [1,2,3]
l1.extend('python') #extend adds the iterable
print(l1)
```

```
[1, 2, 3, 'p', 'y', 't', 'h', 'o', 'n']
```

In [83]:
```
l1 = [1,2,3]
l1.append([7,8,9]) #append will add only one item
print(l1)
```

```
[1, 2, 3, [7, 8, 9]]
```

In [86]:
```
l1 = [1,2,3]
l1.append('python')
print(l1)
```

```
[1, 2, 3, 'python']
```

index() Returns the index of the first element with the specified value

In [88]:
```
l1 = [1,2,3]
l1.index(3)
```

Out[88]: 2

In [89]:
```
l1 = [1,2,3]
l1.index(9) #Gives error if not found
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13240\2079114118.py in <module>
      1 l1 = [1,2,3]
----> 2 l1.index(9)

ValueError: 9 is not in list
```

insert() Adds an element at the specified position

In [91]:
```
l1 = [1,2,3]
# First argument will be position and second argument will be element that has to insert
l1.insert(0,'python')
print(l1)
```

```
['python', 1, 2, 3]
```

In [92]:
```
l1 = [1,2,3]
# First argument will be position and second argument will be element that has to insert
l1.insert(7,'python') #if given the position greater than the length of the list it will
print(l1)
```

```
[1, 2, 3, 'python']
```

pop() Removes the element at the specified position

In [93]:
```
l1 = [1,2,3]
l1.pop()
print(l1)
```

```
[1, 2]
```

In [94]:
```
l1 = [1,2,3]
l1.pop(1) #index number as an argument
print(l1)
```

```
[1, 3]
```

remove() Removes the first item with the specified value

In [96]:
```
l1 = [1,2,11,23,4,3]
l1.remove(1) #specified element as an argument
print(l1)
```

```
[2, 11, 23, 4, 3]
```

In [97]:
```
l1 = [1,2,1,23,1,4,3]
l1.remove(1) #specified element as an argument
print(l1)  #Will only remove the first occurence of the given item
```

```
[2, 1, 23, 1, 4, 3]
```

In [98]:
```
l1 = [1,2,11,23,4,3]
l1.remove(9) #specified element as an argument
print(l1) # will throw an error if could not found
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13240\4265535957.py in <module>
      1 l1 = [1,2,11,23,4,3]
----> 2 l1.remove(9) #specified element as an argument
      3 print(l1)

ValueError: list.remove(x): x not in list
```

reverse() Reverses the order of the list

In [99]:
```
l1 = [1,2,3]
l1.reverse()
print(l1)
```

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

sort() Sorts the list

In [101…
```
l1 = [1200,212,300]
l1.sort()
print(l1)
```

```
[212, 300, 1200]
```

Tuples -> Immutable

Can acess each element in tuple through indexing or slicing

Operations in Tuples

+ operator --> Concatenate the tuples

In [118…
```
t1 = (1,2,3)
t2 = (3,4,5)
t1 + t2
```

Out[118]:
```
(1, 2, 3, 3, 4, 5)
```

*** operator in list**

In [121…
```
t2 = (3,4,5)
t2*3
```

Out[121]:
```
(3, 4, 5, 3, 4, 5, 3, 4, 5)
```

Membership operator

In [169…
```
t1 = (91,24,78)
45 in t1
```

Out[169]:
```
False
```

In [170…
```
45 not in t1
```

Out[170]:
```
True
```

Tuple Methods

count() Returns the number of times a specified value occurs in a tuple

In [107…
```
t1 = (1,2,3,3,4,5,6,1,4,7,1)
t1.count(1)
```

Out[107]:
```
3
```

In [108…
```
t1 = (1,2,3,3,4,5,6,1,4,7,1)
t1.count(3)
```

Out[108]:
```
2
```

In [109…
```
t1 = (1,2,3,3,4,5,6,1,4,7,1)
t1.count(8)
```

Out[109]:
```
0
```

index() Searches the tuple for a specified value and returns the position of where it was found

In [110…
```
t1 = (1,2,3,3,4,5,6,1,4,7,1)
t1.index(8) #gives error if not found
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13240\2880431033.py in <module>
      1 t1 = (1,2,3,3,4,5,6,1,4,7,1)
----> 2 t1.index(8)

ValueError: tuple.index(x): x not in tuple
```

In [111…
```
t1 = (1,2,3,3,4,5,6,1,4,7,1)
```

```
t1.index(1)
```

Out[111]:
```
0
```

In [113…
```
t1 = (1,2,3,3,4,5,6,1,4,7,1)
t1.index(7)
```

Out[113]:
```
9
```

Dictionary -> mutuable

Operations in Dictionary

- and * operators are not supported in dictionary

Membership Operator

In [173…
```
d1 = {1:'a',2:'b',3:'c'}
1 in d1
```

Out[173]:
```
True
```

In [175…
```
1 not in d1
```

Out[175]:
```
False
```

Dictionary Methods

Cannot acess each key-value pair in dictionary through indexing or slicing

clear() Removes all the elements from the dictionary

In [125…
```
d1 = {1:'a',2:'b',3:'c'}
d1
```

Out[125]:
```
{1: 'a', 2: 'b', 3: 'c'}
```

In [126…
```
d1.clear()
d1
```

Out[126]:
```
{}
```

copy() Returns a copy of the dictionary

In [148…
```
d1 = {1:'a',2:'b',3:'c'}
d1
```

Out[148]:
```
{1: 'a', 2: 'b', 3: 'c'}
```

In [129…
```
d2 = d1.copy()
d2
```

Out[129]:
```
{1: 'a', 2: 'b', 3: 'c'}
```

fromkeys() Returns a dictionary with the specified keys and value

In [136…
```
#First argument is key and second argument is value
dict.fromkeys([11,12],['Hello','Python'])
#keys are 11 and 12 and value to each key is list of 2 items
```

Out[136]:
```
{11: ['Hello', 'Python'], 12: ['Hello', 'Python']}
```

In [138…
```
dict.fromkeys([1,2]) #values to the key 1 and 2 are None
```

Out[138]:
```
{1: None, 2: None}
```

get() Returns the value of the specified key

In [139…
```
d1
```

Out[139]:
```
{1: 'a', 2: 'b', 3: 'c'}
```

In [140…
```
d1.get(2)
```

Out[140]:
```
'b'
```

items() Returns a list containing a tuple for each key value pair

In [141…
```
d1.items()
```

Out[141]:
```
dict_items([(1, 'a'), (2, 'b'), (3, 'c')])
```

keys() Returns a list containing the dictionary's keys

In [142…
```
d1.keys()
```

Out[142]:
```
dict_keys([1, 2, 3])
```

values() Returns a list of all the values in the dictionary

In [149…
```
d1.values()
```

Out[149]:
```
dict_values(['a', 'b', 'c'])
```

pop() Removes the element with the specified key

In [150…
```
d1.pop(2) #key as an argument to tell which key has to be deleted
```

Out[150]:
```
'b'
```

In [151…
```
d1
```

Out[151]:
```
{1: 'a', 3: 'c'}
```

popitem() Removes the last inserted key-value pair

In [152…
```
d2
```

Out[152]:
```
{1: 'a', 2: 'b', 3: 'c'}
```

In [153…
```
d2.popitem()
```

Out[153]:
```
(3, 'c')
```

In [154…
```
d2
```
Out[154]:
```
{1: 'a', 2: 'b'}
```

setdefault() Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

In [155…
```
d1
```
Out[155]:
```
{1: 'a', 3: 'c'}
```

In [157…
```
d1.setdefault(3)
```
Out[157]:
```
'c'
```

In [162…
```
d1.setdefault(4)
```

In [163…
```
d1
```
Out[163]:
```
{1: 'a', 3: 'c', 4: None}
```

update() Updates the dictionary with the specified key-value pairs

In [164…
```
d1
```
Out[164]:
```
{1: 'a', 3: 'c', 4: None}
```

In [166…
```
d1.update({5:'r'})
d1
```
Out[166]:
```
{1: 'a', 3: 'c', 4: None, 5: 'r'}
```

Sets -> mutuable

Indexing and slicing is not possible in sets

Operations in Sets

- and * operators are not supported in sets

- operator is used as difference in sets

In [181…
```
s1 = {1,2,9,3}
s2 = {7,8,9}
s2-s1
```
Out[181]:
```
{7, 8}
```

In [182…
```
s1 -s2
```
Out[182]:
```
{1, 2, 3}
```

Membership Operator

In [183…
```
1 in s1
```

Out[183]:
```
True
```

In [185…
```
8 not in s1
```

Out[185]:
```
True
```

Sets Methods

add() Adds an element to the set

In [187…
```
s1 = {1,2,3}
s1
```

Out[187]:
```
{1, 2, 3}
```

In [188…
```
s1.add(5)
s1
```

Out[188]:
```
{1, 2, 3, 5}
```

clear() Removes all the elements from the set

In [189…
```
s1
```

Out[189]:
```
{1, 2, 3, 5}
```

In [190…
```
s1.clear()
s1 # s1 is all cleared, it's an empty set now
```

Out[190]:
```
set()
```

copy() Returns a copy of the set

In [193…
```
s1={9,6,7}
s1
```

Out[193]:
```
{6, 7, 9}
```

In [194…
```
s2 = s1.copy()
s2
```

Out[194]:
```
{6, 7, 9}
```

difference() Returns a set containing the difference between two or more sets

In [195…
```
s1 = {1,2,3}
s2 = {6,1,2}
s1.difference(s2) #subtract the elements from s1 that are in s2
```

Out[195]:
```
{3}
```

In [197…
```
s2.difference(s1) #subtract the elements from s2 that are in s1
```

Out[197]:
```
{6}
```

difference_update() Removes the items in this set that are also included in another, specified set

In [200…
```
s1 = {1,2,3}
```

```
s2 = {6,1,2}
s1.difference_update(s2) #subtract the elements from s1 that are in s2 and update the se
s1
```

Out[200]:
```
{3}
```

In [201…
```
s1 = {1,2,3}
s2 = {6,1,2}
s2.difference_update(s1) #subtract the elements from s2 that are in s1 and update the se
s2
```

Out[201]:
```
{6}
```

discard() Remove the specified item

In [203…
```
s1 = {1,2,3}
s1.discard(2)
s1
```

Out[203]:
```
{1, 3}
```

In [204…
```
s1 = {1,2,3}
s1.discard(4)
s1
```

Out[204]:
```
{1, 2, 3}
```

intersection() Returns a set, that is the intersection of two or more sets

In [205…
```
s1 = {1,2,3}
s2 = {6,1,2}
s2.intersection(s1)
```

Out[205]:
```
{1, 2}
```

In [206…
```
s1 = {1,2,3}
s2 = {6,1,2}
s1.intersection(s2)
```

Out[206]:
```
{1, 2}
```

intersection_update() Removes the items in this set that are not present in other, specified set(s)

In [208…
```
s1 = {1,2,3}
s2 = {6,1,2}
s2.intersection_update(s1) #Update the result in s2
s2
```

Out[208]:
```
{1, 2}
```

In [209…
```
s1 = {1,2,3}
s2 = {6,1,2}
s1.intersection_update(s2) #Update the result in s1
s2
```

Out[209]:
```
{1, 2, 6}
```

isdisjoint() Returns whether two sets have a intersection or not

In [213…
```
s1 = {3,4}
s2 = {1,2}
```

```
s1.isdisjoint(s2)
```

Out[213]:
```
True
```

In [214…
```
s1 = {3,1,4}
s2 = {1,2}
s1.isdisjoint(s2)
```

Out[214]:
```
False
```

issubset() Returns whether another set contains this set or not

In [216…
```
s1 = {3,1,4}
s2 = {1,2}
s2.issubset(s1)
```

Out[216]:
```
False
```

In [217…
```
s1 = {3,1,2,4}
s2 = {1,2}
s2.issubset(s1)
```

Out[217]:
```
True
```

issuperset() Returns whether this set contains another set or not

In [220…
```
s1 = {3,1,4}
s2 = {1,2}
s1.issuperset(s2)
```

Out[220]:
```
False
```

In [221…
```
s1 = {3,1,2,4}
s2 = {1,2}
s1.issuperset(s2)
```

Out[221]:
```
True
```

pop() Removes an element from the set

In [224…
```
s1 = {3,1,2,4}
s1
```

Out[224]:
```
{1, 2, 3, 4}
```

In [225…
```
s1.pop() #pop doesnot take any argument in sets
```

Out[225]:
```
1
```

In [226…
```
s1
```

Out[226]:
```
{2, 3, 4}
```

remove() Removes the specified element

In [232…
```
s1 = {3,1,2,4}
s1
```

Out[232]:
```
{1, 2, 3, 4}
```

In [233…
```
s1.remove(3)
s1
```

Out[233]:
```
{1, 2, 4}
```

In [229…
```
s1.remove(7) # gives error if it could not find the specified value
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13240\3205015142.py in <module>
----> 1 s1.remove(7)

KeyError: 7
```

symmetric_difference() Returns a set with the symmetric differences of two sets

In [234…
```
s1 = {1,2,3}
s2 = {6,1,2}
s1.symmetric_difference(s2) # gives output of s2-s1 and s1-s2
```

Out[234]:
```
{3, 6}
```

In [235…
```
s1 = {1,2,3}
s2 = {6,1,2}
s2.symmetric_difference(s1) # gives output of s2-s1 and s1-s2
```

Out[235]:
```
{3, 6}
```

symmetric_difference_update() inserts the symmetric differences from this set and another

In [237…
```
s1 = {1,2,3}
s2 = {6,1,2}
s2.symmetric_difference_update(s1) # update the result in s2
s2
```

Out[237]:
```
{3, 6}
```

In [239…
```
s1 = {1,2,3}
s2 = {6,1,2}
s1.symmetric_difference_update(s2) # update the result in s1
s1
```

Out[239]:
```
{3, 6}
```

union() Return a set containing the union of sets

In [241…
```
s1 = {1,2,3}
s2 = {6,1,2,7}
s1.union(s2)
```

Out[241]:
```
{1, 2, 3, 6, 7}
```

update() Update the set with another set, or any other iterable

In [244…
```
s1 = {1,2,3}
s1
```

Out[244]:
```
{1, 2, 3}
```

In [245…
```
s1.update({5,6,7})
s1
```

Out[245]: {1, 2, 3, 5, 6, 7}

Sorted() Function

used to sort any iterable or sequence of items (such as strings,tuples, lists, sets)

and will always return the sorted list

In [246…
```
tup1 = (190,2,31,34)
type(tup1)
```
Out[246]: tuple

In [247…
```
sorted(tup1)
```
Out[247]: [2, 31, 34, 190]

In [248…
```
sorted(tup1,reverse=True) #return in descending order
```
Out[248]: [190, 34, 31, 2]

In [249…
```
s1 = {190,2,31,34}
type(s1)
```
Out[249]: set

In [250…
```
sorted(s1)
```
Out[250]: [2, 31, 34, 190]

In [252…
```
sorted(s1,reverse = True)
```
Out[252]: [190, 34, 31, 2]

In [253…
```
string1 = 'Python'
type(string1)
```
Out[253]: str

In [254…
```
sorted(string1)
```
Out[254]: ['P', 'h', 'n', 'o', 't', 'y']

In [255…
```
sorted(string1,reverse=True)
```
Out[255]: ['y', 't', 'o', 'n', 'h', 'P']

In [261…
```
d1={1:'3',4:'4',3:'p'}
type(d1)
```
Out[261]: dict

In [259…
```
sorted(d1) #returns the sorted keys of dictionary
```
Out[259]: [1, 3, 4]

In [260…
```
sorted(d1,reverse=True) #returns the sorted keys of dictionary in descending order
```

Out[260]:
```
[4, 3, 1]
```

In [262…
```
l=['nb','mu','qa','lo']
#sorting the list on the basis of second element of the list item
sorted(l,key=lambda x: x[1])
```

Out[262]:
```
['qa', 'nb', 'lo', 'mu']
```

In [273…
```
#sorting the list based on the maximum element in each item of the list.
sorted(l,key=max) #passing the max fuction in key
#here the max function will be applied to each item of the list
```

Out[273]:
```
['nb', 'lo', 'qa', 'mu']
```

zip() function

Zip the 2 iterables with the same index

In [264…
```
names = ['Raj','Sid','Geeta','Parth']
marks = [78, 45, 100, 89]
list(zip(names,marks))
```

Out[264]:
```
[('Raj', 78), ('Sid', 45), ('Geeta', 100), ('Parth', 89)]
```

enumerate() function

gives the index and value both for every item of iterable

In [266…
```
list(enumerate(names))
```

Out[266]:
```
[(0, 'Raj'), (1, 'Sid'), (2, 'Geeta'), (3, 'Parth')]
```

In [268…
```
list(enumerate('Programming'))
```

Out[268]:
```
[(0, 'P'),
 (1, 'r'),
 (2, 'o'),
 (3, 'g'),
 (4, 'r'),
 (5, 'a'),
 (6, 'm'),
 (7, 'm'),
 (8, 'i'),
 (9, 'n'),
 (10, 'g')]
```

map() function

takes 2 argument function and iterable or sequence

used to apply any particular function to each item of iterable

In [269…
```
l1 = ['1','2','3','4']
# The type of every element in l1 is string
type(l1[0])
```

Out[269]:
```
str
```

In [270…
```
# Need to convert the every element of the list to int
list(map(int,l1))
```

Out[270]:
```
[1, 2, 3, 4]
```

filter() function

takes 2 argument function that returns true or false and iterable or sequence

used to apply any particular function to each item of iterable

and filter out the elements that returns false while applying the function

In [271…
```
list1 = ['1','23','ss1', '45','45#','56@@', 'awe']
list1
```

Out[271]:
```
['1', '23', 'ss1', '45', '45#', '56@@', 'awe']
```

Only keep the items from the above lists that has only digits and remove the all the other items

In [276…
```
list(filter(str.isdigit,list1))
```

Out[276]:
```
['1', '23', '45']
```

reduce() function

takes 2 argument function and iterable or sequence

used to apply any particular function to each item of iterable

and reduce the sequence to single value

To get the multiplication of the items in the lists

In [279…
```
list2 = [1,2,3,5,7]
list2
```

Out[279]:
```
[1, 2, 3, 5, 7]
```

In [280…
```
from functools import reduce
reduce(lambda x,y : x*y, list2)
```

Out[280]:
```
210
```

User Defined functions

The functions that are created or defined by the user based on its needs Syntax: def function_name(arguments): body return

In [281…
```
#Example function to add 2 numbers
def add(a,b):
    return a+b
```

In [282…
```
add(2,3)
```

Out[282]:
```
5
```

Recurssion

Function that call itself

In [284…
```
# Recursive function for finding a factorial of a number
def fact(n):
    if n == 1:
        return 1
    return n * fact(n-1)
```

In [285…
```
fact(5)
```

Out[285]:
```
120
```

Lambda Expression

Lambda is an anonymous function syntax : variable_name = lambda arguments: return value

In [286…
```
# Creating a Lambda function to find whether a number is even or odd
x = lambda arg: 'Even' if arg%2 == 0 else 'Odd'
x(2)
```

Out[286]:
```
'Even'
```

In [287…
```
x(3)
```

Out[287]:
```
'Odd'
```