Day-3: List and Tuple 23/06/2018

Workshop on "Introduction to Python"

Department Of Computer Science Engineering

SKFGI, Mankundu

1.Python Data Structure

A data Structure is a group of data element that are put together under one name. Data Structure defines a particular way of sorting and organizing data in a computer so that it can be used efficiently.

There are 3 different data structure in the Python programming language:

1.1. Sequence: It is most basic data structure in python. In the sequence data structure each element has a specific index .this index value is start from 0and its automatically incremented for the next element in the sequence. There are several types of sequences in Python, the following three are the most important.

1.1.1Lists are the most versatile sequence type. The elements of a list can be any object, and lists are **mutable** - they can be changed. Elements can be reassigned or removed, and new elements can be inserted. Allows duplicate members.

1.1.2Tuples are like lists, but they are **immutable** - they can't be changed. Allows duplicate members.

1.1.3 Strings are a special type of sequence that can only store characters, and they have a special notation. String is also immutable

1.2. Set is a collection which is unordered and unindexed. No duplicate members.

1.3. Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

2. Python List

Lists are one of the most powerful data strucure in Python. They are just like the arrays declared in other languages. But the most powerful thing is that

list need not be always homogeneous. A single list can contain strings, integers, as well as objects. Lists can also be used for implementing stacks and queues. Lists are mutable, i.e., they can be altered once declared.

2.1How to create a list?

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

Example of list declarations:

		Code	Output
1.	Empty list	my_list = [] print(my_list)	[]
2.	List of integers	my_list = [100, 200, 300] print(my_list)	[100, 200, 300]
3.	List with mixed data types	my_list = [1, "Mother", 3.414] print(my_list)	[1, 'Mother', 3.414]
4.	Nested list	my_list = ["Morning", [8, 4, 6], ['a']] print(my_list)	['Morning', [8, 4, 6], ['a']]

2.2List Index is used to access elements from a list

We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

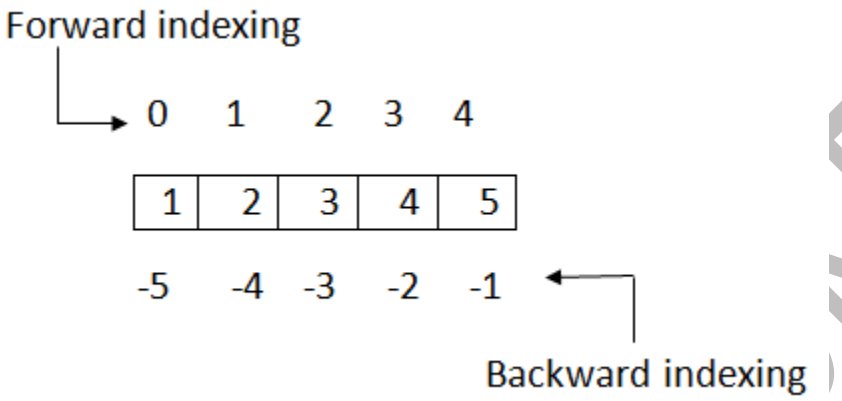
Example:

Code	Output
aList = [123, 'xyz', 'zara', 'abc'];	Index for xyz : 1
print("Index for xyz : ", aList.index('xyz'))	Index for zara : 2

print("Index for zara : ", aList.index('zara'))

2.3Negative indexing

Negative index is used in python to index starting from the last element of the list, tuple or any other container class which supports indexing. -1 refers to the last index, -2 refers to the second last index and so on.



Code **Output**

For lists: 5

```
array = [0,1,2,3,4,5]
print(array[-1])
```

```
array = [0,1,2,3,4,5]          2
print(array[-4])
```

2.4Slice lists in Python

Similar to string, list can also be sliced. We can access a range of items in a list by using the slicing operator (colon) and square brackets[] are used to slice along with the index. Syntax :

Seq=List[start:end:step]

Code **Output**

values = [100, 200, 300, 400, 500] [200, 300]

Get elements from second index to third index.

slice = values[1:3]

print(slice)

values = [100, 200, 300, 400, 500] [100, 300, 500]

Get elements from 0^{th},2^{nd}, 4^{th} index because step value is 2 .

slice = values[0:5:2]

print(slice)

2.5 cloning list:

If you want to modify a list and also keep a copy of the original list. Then you should create a separate copy of the list. this process is called cloning.

```
list=[1,2,3,4,5,6]
l2=list
print(l2)
```

Output:

[1, 2, 3, 4, 5, 6]

2.6 The del statement

i)Delete a specific element:

```
a = [-1, 1, 66.25, 333, 333, 1234.5]
  del a[0]
  print(a)
```

[1, 66.25, 333, 333, 1234.5]

ii)Delete more than one elements:

```
  del a[2:4]
  print(a)
```

[1, 66.25, 1234.5]

iii)Delete the entire list

```
  del a[:]
  print(a)
[]
```

2.7Basic List Opeartion:

List behave in the similer way as string when operators like +,* are used.

operation	description	example	output
len()	returns length of the list	l1 = list(("apple", "banana", "cherry")) print(len(l1))	3
concatenation(+)	join two list	list1=[10,20] list2=[30,40] list3=list1+list2 print(list3)	10,20,30,40
repetition	repeat element in the list	l=[1,2,3,4,5,6]*2 print(l)	[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
in	checks if the value is present in the list	list=[1,2,3,4,5,6] print(2 in list)	true
not in	checks if the value is not present in the list	list=[1,2,3,4,5,6] print(10 in list)	false

max()	returns maximum value	l=[10,2,330,4,5,6] print(max(l))	330
min()	returns minimum value	l=[10,2,330,4,5,6] print(min(l))	2
sum()	adds the value in the list that has number	l=[1,2,3,4,5,6] print(sum(l))	21
all()	returns true if all elements of the list are true	l=[0,1,2,3,4,5,6] print(all(l))	false
any()	returns true if any elements of the list is true. if list is empty then return false	l=[0,1,2,3,4,5,6] print(any(l))	true
list()	convert an iterable (tuple, string, set, dictionary)to a list	l=list("hello") print(l)	['h', 'e', 'l', 'l', 'o']
sorted()	returns a new sorted list. the original list is not sorted	l=[6,5,4,3,2,1] print(sorted(l)) print(l)	[1, 2, 3, 4, 5, 6] [6, 5, 4, 3, 2, 1]

2.8 The list Method:

Python has various method to help programmers work efficiently with list . some of these method are summarized

Method	Description	Example	Output
append(obj)	appends an element to the list	l=[6,5,4,3,2,1] l.append(10) print(l)	[6, 5, 4, 3, 2, 1, 10]
remove(obj)	remove or delete obj from the list. gives a valueerror if obj is not present in the list. if multiple copies of obj exists in the list , then the first value is deleted	l=[4,5,4,3,2,4] l.remove(4) print(l)	[5, 4, 3, 2, 4]
count(obj)	count the number of times an element in the list	l=[6,5,4,3,2,4] c=l.count(4) print(c)	2
index(obj)	returns the lowest index of obj in the list. gives a valueerror if obj is not present in the list	l=[4,5,4,3,2,4] c=l.index(4) print(c)	0
insert(index,obj)	insert obj at the specified index in the list	l=[4,5,4,3,2,4] l.insert(0,10) print(l)	[10, 4, 5, 4, 3, 2, 4]
reverse()	reverse the element in the list	l=[6,5,4,3,2,1] l.reverse() print(l)	[1, 2, 3, 4, 5, 6]
sort()	sort the elements in the list. the original list is sorted	l=[6,5,4,3,2,1] l.sort() print(l)	[1, 2, 3, 4, 5, 6]
extended(list)	adds the elements in the list to the end of another list. it is	l=[6,5,4,3,2,1] l1=[10,9,8,7]	[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

	similar to + or += operation	l1.extend(l) print(l1)	
pop([index])	removes the element at the specified index from the list. index is an optional parameter . if no index is specified , then removes the last obj in the list	l=[6,5,4,3,2,1] l.pop(4) print(l) l.pop() print(l)	[6, 5, 4, 3, 1] [6, 5, 4, 3]

Note: insert(),remove(), sort() method s only modify the listand do not return any values

2.9Stack implementation using List

Stack works on the principle of "Last-in, first-out". Also, the inbuilt functions in Python make the code short and simple. To add an item to the top of the list, i.e., to push an item, we use **append()** function and to pop out an element we use **pop()** function. These functions work quiet efficiently and fast in end operations.

```
# stack using list
stack = ["Amar", "Akbar", "Anthony"]
stack.append("Ram")
stack.append("Iqbal")
print(stack)
print(stack.pop())
print(stack)
print(stack.pop())
print(stack)
```

output:

```
['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']
Iqbal
['Amar', 'Akbar', 'Anthony', 'Ram']
Ram
['Amar', 'Akbar', 'Anthony']
```

2.10Using Lists as Queue

Queue works on the principle of "first-in, first-out". Also, the inbuilt functions in Python make the code short and simple. To add an item to the top of the

list, i.e., to insert an item, we use **append()** function and to delete an element we use **pop(0)** function. These functions work quiet efficiently and fast in end operations.

```
# queue using list
q = ["Amar", "Akbar", "Anthony"]
q.append("Ram")
q.append("Iqbal")
print(q)
print(q.pop(0))
print(q)
print(q.pop(0))
print(q)
```

OUTPUT:

```
['Amar', 'Akbar', 'Anthony', 'Ram', 'Iqbal']
Amar
['Akbar', 'Anthony', 'Ram', 'Iqbal']
Akbar
['Anthony', 'Ram', 'Iqbal']
```

2.11 List Comprehensions

List comprehension is an elegant and concise way to create new list from an existing list in Python. List comprehension consists of an expression followed by for statement inside square brackets.

Syntax:

List=[expression for variable in sequence]

Program1: Here is an example to make a list with each item being increasing power of 2.

```
pow2 = [2 ** x for x in range(10)]

print(pow2)
```

Output: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

Program2. Program to combine and print element of two list using list comprehension

```
print([(x,y) for x in [10,20,30] for y in [40,10,30] if x!=y])
```

Output:

[(10, 40), (10, 30), (20, 40), (20, 10), (20, 30), (30, 40), (30, 10)]

2.9 Nested List:-

Nested list means a list within another list, we have already said that a list has element of different data types which can include even a list.

Program1:

```
list=[1,'a',"abc", [1,2,3,4],8]
i=0
while i <len(list):
    print("list[",i,"]---", list[i])
    i+=1
```

output:

```
list[ 0 ]--- 1
list[ 1 ]--- a
list[ 2 ]--- abc
list[ 3 ]--- [1, 2, 3, 4]
list[ 4 ]--- 8
```

2.10 Matrix using List

A matrix is a two-dimensional data structure. In real-world tasks you often have to store rectangular data table. In python matrix is a nested list

Example

```
# a is 2-D matrix with integers
a = [['Roy',80,75,85,90,95],
     ['John',75,80,75,85,100],
     ['Dave',80,80,80,90,95]]

#b is a nested list but not a matrix because number of column are not equal
b= [['Roy',80,75,85,90,95],
    ['John',75,80,75],
    ['Dave',80,80,80,90,95]]
```

2.10.1Accessing elements of the matrix in python by using list index

```
a = [['Roy',80,75,85,90,95],
     ['John',75,80,75,85,100],
     ['Dave',80,80,80,90,95]]
print(a[0])
print(a[0][1])
print(a[1][2])
```

output

```
['Roy', 80, 75, 85, 90, 95]
80
80
```

2.10.2Accessing elements of the matrix in python by using negative list index

```
a = [['Roy',80,75,85,90,95],
     ['John',75,80,75,85,100],
     ['Dave',80,80,80,90,95]]

print(a[-1])

print(a[-1][-2])

print(a[-2][-3])
```

Output:

```
['Dave', 80, 80, 80, 90, 95]
90
75
```

2.10.3. Add elements of a matrix

In python list are mutable, meaning, their elements can be changed unlike string or tuple.
We can use assignment operator (=) to change an item or a range of items.
Example: Change elements of a matrix in python

```
a = [['Roy',80,75,85,90,95],
     ['John',75,80,75,85,100],
     ['Dave',80,80,80,90,95]]
b=a[0]
print(b)
b[1]=75
print(b)
a[2]=['Sam',82,79,88,97,99]
print(a)
a[0][4]=95
print(a)
```

Output:

```
b=['Roy',80,75,85,90,95]
b=['Roy',75,75,85,90,95]
a= [['Roy',75,75,85,90,95],
    ['John',75,80,75,85,100],
    ['Sam',82,79,88,97,99]]
a=[['Roy',75,75,85,95,95],
   ['John',75,80,75,85,100],
   ['Sam',82,79,88,97,99]]
```

Here a is a matrix where we have stored name and marks of the students.
We have stored Roy's marks row in the variable b, by using it's row position from the matrix which is 0 so it becomes a[0].
To change Roy's marks in Science, by directly accessing that position where the marks is stored; as the position of that is a[0][1]. So, in b it will be b[1].
We replaced Dave's row with a new student's marks Sam; we directly accessed the row position which is a[2] of Dave's marks and replace it by Sam's marks .
Roy's marks in Arts was entered wrong, we accessed the position a[0][4] where 0 is the first row and 4 is the fifth column of the matrix in which the data is stored and assigned a new value.

Take input of matrix

```
x=int(input('enter the number of row'))
```

```
y=int(input('enter the column'))
z=[]
z=[[0 for j in range(y)] for i in range(x)]
for i in range(x):
    for j in range(y):
        z[i][j]=int(input('enter the element'))
print(z)
```

Output:

```
enter the number of row2
enter the column2
enter the element1
enter the element2
enter the element3
enter the element4
[[1, 2], [3,4]]
```

Addition of two matrix

```
x=int(input('enter the number of row'))
y=int(input('enter the column'))
z=[]
z=[[0 for j in range(y)] for i in range(x)]
for i in range(x):
    for j in range(y):
        z[i][j]=int(input('enter the element'))
a=[]
a=[[0 for j in range(y)] for i in range(x)]
for i in range(x):
    for j in range(y):
        a[i][j]=int(input('enter the element'))
for i in range(x):
    for j in range(y):
```

```
            c=a[i][j]+z[i][j]
            print(c,end='')
        print()
```

Output:

```
enter the number of row2
enter the column2
enter the element1
enter the element2
enter the element3
enter the element4
enter the element5
enter the element6
enter the element7
enter the element8
 6   8
10   12
```

Transpose of a matrix

```
X = [[12,7], [4 ,5], [3 ,8]]
result = [[0,0,0], [0,0,0]]
for i in range(len(X)):
    for j in range(len(X[0])):
        result[j][i] = X[i][j]
for r in result:
    print(r)
```

Output:

```
[12, 4, 3]
[7, 5, 8]
```

Transpose of a matrix using List Comprehensions

```
X = [[12,7], [4 ,5], [3 ,8]]
```

```
result = [[X[j][i] for j in range(len(X))] for i in range(len(X[0]))]
for r in result:
   print(r)
```

Output:

[12, 4, 3]

[7, 5, 8]

Program:

P1.Write a program that crate a list of number from 1-20 that are either divisible by 2 or divisible by 4

```
x=[]
for i in range(1,21):
    if (i%2==0 or i%4==0):
        x.append(i)
print(x)
```

Output:

[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

P2.WAP to generate a list from 1 to10 then calculate sum of the square of element of the list

```
x=[]
for i in range(1,11):
```

```
        x.append(i)
s=0
for i in x:
    s+=i*i
print(s)
```
Output: 385

P3. WAP to create a list of number in the range 1 to 10.then delete all even number from the list and print the final list

```
x=[]
for i in range(1,11):
        x.append(i)
s=0
for i in x:
    if i%2==0:
        x.remove(i)
print(x)
```
Output: [1, 3, 5, 7, 9]

P4: WAP to search an element and print index. If the element exists at multiple location in the list ,then print all the indices

```
y=int(input('enter the number of lelemnt'))
x=[]
for i in range(y):
    x.append(int(input('enter the value')))
print(x)
s=int(input('enter the element which you want to search'))
c=0
z=[]
for i in range(y):
    if x[i]==s:
        z.append(i)
        c=c+1
if c>0:
    print('item found',c,' times and index list is ',z)
else:
    print('item not found')
```
Output: enter the number of lelemnt10
enter the value1

```
enter the value2
enter the value1
enter the value3
enter the value1
enter the value4
enter the value1
enter the value5
enter the value1
enter the value6
[1, 2, 1, 3, 1, 4, 1, 5, 1, 6]
enter the element which you want to search1
item found 5  times and index list is  [0, 2, 4, 6, 8]
```

p5: WAP to remove all duplicate from a list

```
duplicate=[1, 2, 1, 3, 1, 4, 1, 5, 1, 6]
final_list = []
for num in duplicate:
    if num not in final_list:
        final_list.append(num)
print(final_list)
```

Output: [1, 2, 3, 4, 5, 6]

3.Python – Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.

3.1Difference between List and Tupple

List	Tupple
List is **mutable** , it can't be used as a key in a dictionary, whereas a tuple can be used.	Tuples are **immutable** . This means that you cannot change the values in a tuple once you have created it.
The literal syntax of lists is shown by square brackets **[]** .	The literal **syntax** of tuples is shown by parentheses()
Lists are for **variable length**	Tuples are for **fixed length** .
lists show **order**	Tuples show **structure**
Lists have **O(1)** append, insert and delete performance.	Tuples have **O(N)** append, insert, and delete performance
Lists are homogeneous sequences.	Tuples are heterogeneous data structures (i.e., their entries have different meanings),

3.2.Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

3.2.1Return the item in position 1

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

```
banana
```

3.2.2You cannot change values in a tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple[1] = "blackcurrant" # test changeability
```

Error:

```
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    thistuple[1] = "blackcurrant"
TypeError: 'tuple' object does not support item assignment
```

Output:-

```
print(thistuple)
```

```
('apple', 'banana', 'cherry')
```

3.2.3Updating Tuples

Tuples are immutable, which means you cannot update or change the values of tuple elements. You are able to take portions of the existing tuples to create new tuples as the following example demonstrates –

```
#!/usr/bin/python3
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')
# Following action is not valid for tuples
# tup1[0] = 100;
```

```
# So let's create a new tuple as follows
tup3 = tup1 + tup2
print (tup3)
```

Output: (12, 34.56, 'abc', 'xyz')

3.2.4 Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement.

```
tup = ('physics', 'chemistry', 1997, 2000);
print (tup)
del tup;
print ("After deleting tup : ")
print (tup)
```

Output:

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
  File "C:/Users/PC/AppData/Local/Programs/Python/Python36-32/t3.py",
line 8, in <module>
    print (tup)
NameError: name 'tup' is not defined
```

3.2.5 Immutable Tuples

```
tuple1 = (0, 1, 2, 3)
tuple1[0] = 4
```

Error:- Traceback (most recent call last):

```
  File "<pyshell#19>", line 1, in <module>
    tuple1[0] = 4
```

TypeError: 'tuple' object does not support item assignment

3.3.Basic Tuple Opeartion:

Tuple behave in the similer way as string when operators like +,* are used.

operation	description	example	output
len()	returns length of the tuple	t=(1,2,3,4,5) print(len(t))	53
concatenation(+)	join two tuple	t1=(10,20) t2=(30,40) t3=list1+list2 print(list3)	10,20,30,40
repetition	repeat element in the tuple	l=(1,2,3,4,5,6)*2 print(l)	(1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6)
in	checks if the value is present in the tuple	t=(1,2,3,4,5,6) print(2 in t)	true
not in	checks if the value is not present in the tuple	t=(1,2,3,4,5,6) print(20 in t)	false
max()	returns maximum value	l=(10,2,330,4,5,6) print(max(l))	330
min()	returns minimum value	l=(10,2,330,4,5,6)	2

		print(min(l))	
sum()	adds the value in the tuple that has number	l=(1,2,3,4,5,6) print(sum(l))	21
all()	returns true if all elements of the tuple are true	l=(0,1,2,3,4,5,6) print(all(l))	false
any()	returns true if any elements of the list tuple is true. if tuple is empty then return false	l=(0,1,2,3,4,5,6) print(any(l))	true
tuple()	convert an iterable (list, string, set, dictionary)to a list	l=tuple("hello") print(l)	('h', 'e', 'l', 'l', 'o')
sorted()	returns a new sorted tuple. the original tupleis not sorted	l=(6,5,4,3,2,1) print(sorted(l)) print(l)	[1, 2, 3, 4, 5, 6] [6, 5, 4, 3, 2, 1]

3.4 The tuple Method:

Python has various method to help programmers work efficiently with tuple . some of these method are summarized

Method	Description	Example	Output
count(obj)	count the number of times an	l=(6,5,4,3,2,4)	2

	element in the list	c=l.count(4) print(c)	
index(obj)	returns the lowest index of obj in the list. gives a valueerror if obj is not present in the list	l=(4,5,4,3,2,4) c=l.index(4) print(c)	0

3.4 Nested Tuples

Python allows us to define a tuple inside another tuple

```
l=((1,2,3),('hi',4,5),('hello',6,7))
for i in range(len(l)):
    print(l[i])
```

Output:- (1, 2, 3)

('hi', 4, 5)

('hello', 6, 7)

3.5 Tuples in a loop

```
 l=(6,5,4,3,2,1)
for i in range(len(l)):
    print(l[i])
```

Output:

3.6. Indexing, Slicing

```
 L = ('spam', 'Spam', 'SPAM!')
```

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

4.Set:A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).
However, the set itself is mutable. We can add or remove items from it.
Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

How to create a set?

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```
Output: {1, 2, 3}
{'Hello', 1.0, (1, 2, 3)}

Note1 : set do not have duplicates
```
my_set = {1,2,3,4,3,2}
print(my_set)
my_set = set([1,2,3,2])
print(my_set)
```
Output:
{1, 2, 3, 4}
{1, 2, 3}
Note2: Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set () function without any argument.
```
a = {}
print(type(a))
a = set()
print(type(a))
```
Output:

<class 'dict'>
<class 'set'>

How to change a set in Python?

Sets are mutable. But since they are unordered, indexing have no meaning.

We cannot access or change an element of set using indexing or slicing. Set does not support it.

We can add single element using the add() method and multiple elements using the update()method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
my_set = {1,3}
print(my_set)
my_set.add(2)
print(my_set)
my_set.update([2,3,4])
print(my_set)
my_set.update([4,5], {1,6,8})
print(my_set)
```

Output:

```
{1, 3}
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6, 8}
```

How to remove elements from a set?

A particular item can be removed from set using methods, discard() and remove().

The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

```
my_set = {1, 3, 4, 5, 6}
print(my_set)
my_set.discard(4)
print(my_set)
my_set.remove(6)
print(my_set)
```

my_set.discard(2)
print(my_set)

Output:

{1, 3, 4, 5, 6}
{1, 3, 5, 6}
{1, 3, 5}
{1, 3, 5}

Note: We can also remove all items from a set using clear().
Example: my_set.clear()

Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

Operation	Description	Example	Output
Union	Union of A and B is a set of all elements from both sets. Union is performed using | operator. Same can be accomplished using the method union().	A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A | B) print(A.union(B))	{1, 2, 3, 4, 5, 6, 7, 8}
Intersection	Intersection of A and B is a set of elements that are common in both sets. Intersection is performed using & operator. Same can be accomplished using the method intersection().	A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A &B) print(A.intersection(B))	{4,5}
Difference	Difference of A and B (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of element in B but not in A. Difference is performed using - operator. Same can be accomplished using the method difference().	A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A -B) print(A.difference(B))	{1, 2, 3}
Symmetric Difference	Symmetric Difference of A and B is a set of elements in both A and B except those that are	A = {1, 2, 3, 4, 5} B = {4, 5, 6, 7, 8} print(A ^ B)	{1, 2, 3, 6, 7, 8}

	common in both. Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric_difference().	print(A.symmetric_difference(B))	

Different Python Set Methods

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with set objects.

Method	Description	Example	Output

clear()	Remove all elements form a set	vowels = {'a', 'e', 'i', 'o', 'u'}
		print('Vowels (before clear):', vowels)
		# clearing vowels
		vowels.clear()
		print('Vowels (after clear):', vowels)
			Vowels (before clear): {'e', 'i', 'o', 'u', 'a'}
			Vowels (after clear): set()

copy()	Return a shallow copy of a set. A set can be copied using = operator	numbers = {1, 2, 3, 4} new_numbers = numbers.copy() new_numbers.add('5') print('numbers: ', numbers) print('new_numbers: ', new_numbers)	numbers: {1, 2, 3, 4} new_numbers: {1, 2, 3, 4, '5'}
difference_update()	Remove all elements of another set from this set	A = {'a', 'c', 'g', 'd'} B = {'c', 'f', 'g'}	A = {'a', 'd'} B = {'g', 'f',

	When you run the code, result will be None A will be equal to A-B B will be unchanged	result = A.difference_update(B) print('A = ', A) print('B = ', B) print('result = ', result)	'c'} result = None
intersection_update()	Update the set with the intersection of itself and another When you run the code, result will be None A will be equal to the intersection of A, B B remains unchanged	A = {1, 2, 3, 4} B = {2, 3, 4, 5} result = A.intersection_update(B) print('result =', result) print('A =', A) print('B =', B)	result = None A = {2, 3, 4} B = {2, 3, 4, 5}
isdisjoint()	Return True if two sets have a null intersection	A = {1, 2, 3, 4} B = {5, 6, 7} C = {4, 5, 6} print(A.isdisjoint(B)) print(A.isdisjoint(C))	True False
issubset()	Return True if another set contains this set	A = {1, 2, 3} B = {1, 2, 3, 4, 5} C = {1, 2, 4, 5} print(A.issubset(B)) print(B.issubset(A))	True False

issuperset()	Return True if this set contains another set	A = {1, 2, 3, 4, 5} B = {1, 2, 3} C = {1, 2, 3} print(A.issuperset(B)) print(B.issuperset(A))	True False
pop()	Remove and return an arbitary set element. Raise KeyError if the set is empty	A ={'a', 'b', 'c', 'd'} print('Return Value is', A.pop()) print('A = ', A)	c
symmetric_difference_update()	Update a set with the symmetric difference of itself and another	A = {'a', 'c', 'd'} B = {'c', 'd', 'e' } result = A.symmetric_difference_update(B) print('A = ', A) print('B = ', B) print('result = ', result)	A = {'a', 'e'} B = {'c', 'e', 'd'} result = None

Set Membership Test

We can test if an item exists in a set or not, using the keyword in.

```
my_set = set("apple")
print('a' in my_set)
print('p' not in my_set)
```

Output:

True

False

Built-in Functions with Set

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with set to perform different tasks.

Function	Description
all()	Return `True` if all elements of the set are true (or if the set is empty). pyList = {1,2,3,4,5,0} print(all(pyList))
any()	Return `True` if any element of the set is true. If the set is empty, return `False`. pyList = {1,2,3,4,5,0} print(any(pyList))
len()	Return the length (the number of items) in the set. pyList = {'e', 'a', 'u', 'o', 'i'} print(len(pyList))
max()	Return the largest item in the set. num = {3, 2, 8, 5, 10, 6} print('Minimum is:', max(num))
min()	Return the smallest item in the set. num = {3, 2, 8, 5, 10, 6} print('Minimum is:', min(num))
sorted()	Return a new sorted list from elements in the set(does not sort the set itself). pyList = {'e', 'a', 'u', 'o', 'i'} print(sorted(pyList))
sum()	Retrun the sum of all elements in the set. numbers = {2.5, 3, 4, -5} numbersSum = sum(numbers) print(numbersSum)

Python Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets.

Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable and can be used as keys to a dictionary.

Frozensets can be created using the function `frozenset()`.

This datatype supports methods like `copy()`, `difference()`, `intersection()`, `isdisjoint()`, `issubset()`, `issuperset()`, `symmetric_difference()` and `union()`. Being immutable it does not have method that add or remove elements.

```
A = frozenset([1, 2, 3, 4])
B = frozenset([3, 4, 5, 6])
>>> A.isdisjoint(B)
False
>>> A.difference(B)
frozenset({1, 2})
>>> A | B
frozenset({1, 2, 3, 4, 5, 6})
>>> A.add(3)
...
AttributeError: 'frozenset' object has no attribute 'add'
```