MODULE – 2

What is List? How will you reverse a list?

Lists are a built-in data type in Python and one of the most powerful data structures. Lists are mutable, meaning they are changeable and dynamic – you can update, delete, and add new list items to the list at any time throughout the life of the program.

You can think of them as containers for storing multiple (typically related) collections of items under the same variable name. To create a list in Python, you need to: Give the list a name.

- Use the assignment operator, =.
- Include 0 or more list items inside square brackets, [ ] – and make sure to separate each list item with a comma.
- List items can be homogeneous, meaning they are of the same type.

For example, you can create a list of only numbers or a list of only strings (or text).

```
names = ["Johny", "Lenny", "Jimmy", "Timmy"]
# print list contents to the console
print(names)
# output
# ['Johny', 'Lenny', 'Jimmy', 'Timmy']
```

- List items can also be heterogeneous, meaning they can all be of different data types.

```
# a list containing strings, integers, and floats (or numbers with a decimal point)
my_information = ["John", "Doe", 34, "London", 1.76]
print(my_information)
# output
# ['John', 'Doe', 34, 'London', 1.76]
```

1. Reverse Items in a List Using the .reverse() Method

- The .reverse() method in Python is a built-in method that reverses the list in place. Reversing the list in place means that the method modifies and changes

the original list. It doesn't create a new list which is a copy of the original but with the list items in reverse order.

- This method is helpful when you are not concerned about preserving the order of the original list. The general syntax of the .reverse() method looks something like the following:

```
list_name.reverse()
```

- The .reverse() method doesn't accept any arguments and doesn't have a return value – it only updates the existing list.

```
#original list
my_numbers = [1, 2, 3, 4, 5]
# reverse the order of items in my_numbers
my_numbers.reverse()
# print contents of my_numbers to the console
print(my_numbers)
# output
# [5, 4, 3, 2, 1]
```

- In the example above, the order of the list items in the original list is reversed, and the original list is modified and updated.

2. Reverse Items in a List Using the reversed() Function

- This function is helpful when you want to access the individual list elements in reverse order. The general syntax for the reversed() function looks something similar to this:

```
reversed(list_name)
```

- The reversed() built-in function in Python returns a reversed iterator – an iterable object which is then used to retrieve and go through all the list elements in reverse order.
- Returning an iterable object means that the function returns the list items in reverse order. It doesn't reverse the list in place. This means that it doesn't modify the original list, nor does it create a new list which is a copy of the original one but with the list items in reverse order. You can pass the result of

the reversed() operation as an argument to the list() function, which will convert the iterator to a list, and store the final result in a variable:

```
my_numbers = [1, 2, 3, 4, 5]
my_numbers_reversed = list(reversed(my_numbers))
 # original list
print(my_numbers)
# new list, which is a copy of the original one with list items in reverse order
print(my_numbers_reversed)

# output
# [1, 2, 3, 4, 5]
# [5, 4, 3, 2, 1]
```

How will you remove last object from a list?

Suppose list1 is [2, 33, 222, 14, and 25], what is list1 [-1]?

In Python, there are several ways to delete the last element from the list. One way is to use the pop() method. This method removes the last element of a list by default, or you can specify the index of the element you want to remove.

For example:

```
List1 = [2,33,222,14,25]

List1.pop() # removes the last element (5)

Print(List1)
```

Output:

```
[2,33,222,14]
```

Another way to delete the last element of a list is to use negative indexing. In Python, you can use negative indexing to access elements from the end of a list. For example, the last element of a list can be accessed using the index -1, the second-to-last element can be accessed using -2, and so on.The del statement is used to delete a specific element from a list, and when used in combination with negative indexing. To delete the last element of a list, you can use the following code:

For example:

```
List1 = [2,33,222,14,25]

del List1[-1]

print(List1)
```

Output:

```
[2,33,222,14]
```

You can also use slicing to remove the last element from a list. The following code removes the last element from a list by creating a new list that excludes the last element:

For example:

```
List1 = [2,33,222,14,25]

List1 = List1[:-1]

print(List1)
```

Output:

```
[2,33,222,14]
```

You can also use the remove() method, which removes the first occurrence of a specified value in a list.

For example:

```
List1 = [2,33,222,14,25]

List1.remove(5)

print(List1)
```

Output:

```
[2,33,222,14]
```

Differentiate between append () and extend () methods?

Extend and Append are two Python list methods used to add elements to a list. Although they appear similar, they have different functionalities and use cases.

Extend vs Append Python List Methods

- In Python, there are two ways to add elements to a list: extend() and append(). However, these two methods serve quite different functions. In append() we add a single element to the end of a list. In extend() we add multiple elements to a list.
- The supplied element is added as a single item at the end of the initial list by the append() method. When an Iterable is supplied as a parameter, the extend() method adds each element from the Iterable to the list's end individually. It alters the initial list.

What is Append in Python?

Python's append() function inserts a single element into an existing list. The element will be added to the end of the old list rather than being returned to a new list. Adds its argument as a single element to the end of a list. The length of the list increases by one.

Syntax: list.append(item)

What is extend() in Python?

The Python's List extend() Iterates over its argument and adding each element to the list and extending the list. The length of the list increases by a number of elements in its argument.

Syntax: list.extend(iterable)

Difference Between append() and extend() in Python

Basis for Comparison	Append()	Extend()
Purpose	To add a single entry to the end of a list, use the append() function.	To add additional elements or an iterable to the end of a list, use the extend() function.
Input	accepts only one input element.	accepts as input an iterable (such as a list or tuple).
Operation	The append() function adds the full input to the list as a single item.	extend() adds each item to the list independently after iterating through each one in the input.
Efficiency	Since append() only executes one operation, it is typically quicker and more effective than extend().	When adding elements from numerous iterables or with huge inputs, extend() could take longer.
Time complexity	Append has constant time complexity i.e.,O(1)	Extend has a time complexity of O(k). Where k is the length of the list which need to be added.

How will you compare two lists?

1. Using list.sort() and == operator

The list.sort() method sorts the two lists and the == operator compares the two lists item by item which means they have equal data items at equal positions. This checks if the list contains equal data item values but it does not take into account the order of elements in the list. This means that the list [1,2,3] will be equal to the list [2,1,3] according to this method of comparison.

Example:

```
def compareList(l1,l2):
l1.sort()
l2.sort()
if(l1==l2)
        return "Equal"
else:
        return "Non equal"
l1=[1,2,3]
l2=[2,1,3]
print("First comparison",compareList(l1,l2))
l3=[1,2,3]
l4=[1,2,4]
print("Second comparison",compareList(l3,l4))
```

Output:

```
First comparison Equal
Second comparison Non equal
```

2. Using collections.Counter()

This method tests for the equality of the lists by comparing frequency of each element in first list with the second list. This method also does not take into account the order of the elements of the list.

Example:

```
import collections
def compareList(l1,l2):
```

```
        if(collections.Counter(l1)==collections.Counter(l2)):
                return "Equal"
        else:
                return "Non equal"
l1=[1,2,3]
l2=[2,1,3]
print("First comparison",compareList(l1,l2))
l3=[1,2,3]
l4=[1,2,4]
print("Second comparison",compareList(l3,l4))
```

Output:

```
First comparison Non equal
Second comparison Equal
```

3. Using == operator

This is a modification of the first method. In this method, the lists are compared without sorting and thus, this method takes into account the order of the data items in the lists.

Example:

```
def compareList(l1,l2):
        if(l1==l2):
                return "Equal"
        else:
                return "Non equal"
l1=[1,2,3]
l2=[2,1,3]
print("First comparison",compareList(l1,l2))
l3=[1,2,3]
l4=[1,2,3]
print("Second comparison",compareList(l3,l4))
```

Output:

```
First comparison Non equal
Second comparison Equal
```

What is tuple? Difference between list and tuple.

List and Tuple in Python are the classes of Python Data Structures. The list is dynamic, whereas the tuple has static characteristics. This means that lists can be modified whereas tuples cannot be modified, the tuple is faster than the list because of static in nature. Lists are denoted by the square brackets but tuples are denoted as parenthesis.

List in Python

Lists are one of the most flexible and powerful containers in Python. It is similar to an array in other languages like Java.

The list has the following features -

- You can use Python lists to store data of multiple types simultaneously.
- Lists help preserve data sequences and further process those sequences in other ways.
- Lists are dynamic.
- Lists are mutable.
- Lists are ordered.
- An index is used to traverse a list.

Lists help store multiple items and then iterate over them using a loop. Because lists are dynamic, you can easily add or remove items anytime.

Tuple in Python

Tuples are also a sequence data type containing elements of different data types.

It comes in handy when storing a collection of items, especially if you want those items to be unchanging.

A python tuple has the following features -

- Tuples are used to store heterogeneous and homogeneous data.
- Tuples are immutable in nature.
- Tuples are ordered
- An index is used to traverse a tuple.
- Tuples are similar to lists. It also preserves the data sequence.

As tuples are immutable, they are faster than the list because they are static.

Syntax Differences

List and tuple act as containers for storing objects. But there is a difference in its use cases and syntax as well.

Lists are surrounded by square brackets [ ] while tuples are surrounded by round brackets ( ).

Creating a list and tuple in python.

```
list_numbers  = [1,2,3,4,5]

tuple_numbers  = (1,2,3,4,5)
```

How will you create a dictionary using tuples in python?

The elements of tuples are enclosed within parentheses and the elements of a dictionary are present in the form of a key-value pair and are enclosed within curly brackets.

1. Using dict()

Here we will create a dictionary from nested tuples and for that we need to pass two values in each tuple one will represent key and the other its corresponding value in the dictionary.

Syntax:
dict((value, key) for key,value in nested_tuple)

Example: Create a dictionary from tuples

```
# one value is age of student
# second value is student name
data = ((24, "bobby"), (21, "ojsawi"))

# convert into dictionary
final = dict((value, key) for key, value in data)

# display
print(final)
```

Output:

```
{'bobby': 24, 'ojsawi': 21}
```

2. Using setdefault()

Here we have used the dictionary method setdefault() to convert the first parameter to key and the second to the value of the dictionary. setdefault(key, def_value) function searches for a key and displays its value, and creates a new key with def_value if the key is not present. Using the append function we just added the values to the dictionary.

Example:

```
def Convert(tup, di):
for a, b in tup:
di.setdefault(a, []).append(b)
return di

# Driver Code
tups = [("akash", 10), ("gaurav", 12), ("anand", 14),
("suraj", 20), ("akhil", 25), ("ashish", 30)]
dictionary = {}
print(Convert(tups, dictionary))
```

Output:

```
{'akash': [10], 'gaurav': [12], 'anand': [14],
'ashish': [30], 'akhil': [25], 'suraj': [20]}
```

How Do You Traverse Through A Dictionary Object In Python?

Traversing a dictionary is visiting each and every step in the dictionary, which is also called iterating the dictionary.
In Python, dictionaries are a useful and commonly used data structure in which we have key-value pairs.
Traversing a dictionary explains how to operate on a dictionary's key-value pairs while traversing through it.

There are multiple ways to traverse the dictionary in Python.

1. Iterate through dictionary keys: keys()

Consider the following dictionary as an example.

```
d = {'key1': 1, 'key2': 2, 'key3': 3}
for k in d.keys():
print(k)
```

Output:

```
key1
key2
key3
```

2. Iterate through dictionary values: values()

```
for v in d.values():
print(v)
```

Output:

```
1
2
3
```

3. Iterate through dictionary items: items()

```
for k, v in d.items():
print(k, v)
```

Output:

```
key1 1
```

key2 2
key3 3

4. Access both key and value without using items()

```
for i in d:
print(i, '->', d[i])
```

Output:

key1->1
key2-> 2
key3-> 3

How Do You Check The Presence Of A Key In A Dictionary?

Using the in operator:

In this method, we use the membership operator in. This operator is used to check if one value is a member of another. It returns a boolean value.
In our case, we use the in operator to check if the key is a member of the dictionary.

Code to check if a key exists in dictionary in python:

```
dict_1 = {“a”: 1, “b”: 2, “c”: 3}
if “a” in dict_1:
print(“Exists”)
else:
print(“Does not exist”)
#Output = “Exists”
```

Similarly, the not in operator can also be used to check if a key does not exist in a dictionary.

However, remember the in operator is case sensitive hence you could either ensure all your keys are in the same case or you could use the upper() or lower() methods respectively.

Using the get() method:

The get() method is a dictionary method that returns the value of the associated key. If the key is not present it returns either a default value (if passed) or it returns None. Using this method we can pass a key and check if a key exists in the python dictionary.

Syntax of get()
dict.get(keyname, value)

Here dict is the name of the dictionary you intent to work with Parameters
Keyname- The keyname of the value to intent to return value - Optional, this value is returned in case the key does not exist

Code to check if the key exists in a dictionary using get()

```
dict_1 = {“a”: 1, “b”: 2, “c”: 3}
if dict_1.get(“a”) is not None:
print(“Exists”)
else:
print(“Does not exist”)
#Output = “Exists”
```

Why Do You Use the Zip () Method in Python?

- The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.
- If the passed iterables have different lengths, the iterable with the least items decides the length of the new iterator.

Syntax:

zip(*iterator1, iterator2, iterator3* ...)

Example:

If one tuple contains more items, these items are ignored:

```
a = ("John", "Charles", "Mike")
b = ("Jenny", "Christy", "Monica", "Vicky")

x = zip(a, b)

print(tuple(x))
```

Output:

(('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))

How Many Basic Types Of Functions Are Available In Python?

Functions are the basic building block of any Python program, defined as the organized block of reusable code, which can be called whenever required.

A function is used to carry out a specific task. The function might require multiple inputs. When the task is done executing, the function can or can not return one or more values. There are two types of functions in python:

- User-Defined Functions - these types of functions are defined by the user to perform any specific task
- Built-in Functions - These are pre-defined functions in python.

Built-in Functions

Built-in functions are already defined in python. A user has to remember the name and parameters of a particular function. Since these functions are pre-defined, there is no need to define them again. Some of the widely used built-in functions are given below:

Example:

```
l = [2, 5, 19, 7, 43]

print("Length of string is",len(l))

print("Maximum number in list is ",max(l))

print("Type is",type(l))
```

Output:

```
Length of string is 5

Maximum number in list is  43

Type is <class 'list'>
```

In the above program, we created a list and stored a few numbers in it, then called various in-built functions on the list and displayed the output.

Some of the widely used python built-in functions are:

Function	Description
len()	Returns the length of a python object
abs()	Returns the absolute value of a number
max()	Returns the largest item in a python iterable
min()	Returns the largest item in a python iterable
sum()	Sum() in Python returns the sum of all the items in an iterator
type()	The type() in Python returns the type of a python object
help()	Executes the python built-in interactive help console
input()	Allows the user to give input
format()	Formats a specified value
bool()	Returns the boolean value of an object

User-Defined Functions

These functions are defined by a programmer to perform any specific task or to reduce the complexity of big problems and use that function according to their need.

Example:

```
def sub(x, y):
    return x-y
print(sub(5,2))
```

Output:

3

How can you pick a random item from a list or tuple?

random.choice() method

The random.choice() function is a built-in Python function that can be used to randomly select an item from a list. It takes a list as an argument and returns a randomly chosen element from that list. The function returns a single item, not a list or other collection. If the list is empty, the function will raise an IndexError.

```
import random
l = [0, 1, 2, 3, 4]
print(random.choice(l))
# 1
```

Tuples and strings are also handled similarly to lists. When a string is provided, one character is returned.

```
print(random.choice(('xxx', 'yyy', 'zzz')))
# yyy
print(random.choice('abcde'))
# b
```

An error is raised if the list, tuple, or string is empty.

```
# print(random.choice([]))

# IndexError: Cannot choose from an empty sequence
```

How can you pick a random item from a range?

Using random.randrange() function

- The random.randrange() method is used to generate a random number in a given range, we can specify the range to be 0 to the length of the list and get the index, and then the corresponding value.

Example:

```
# importing random module
Import random
print('Random Numbers from 1 to 50 are:')

# getting a random number from 1 to 50
x = random.randrange(1,50)
print(x)

# getting an alternative(odd number) random number from 1 to 50
y = random.randrange(1,50,2)
print(y)
```

Output:

Random Numbers from 1 to 50 are:

28

9

Using random.randint()

- The random.randint() is used to generate the random number, also this can be used to generate any number in a range, and then using that number, we can find the value at the corresponding index, just like the above-mentioned technique. But it differs by the fact that it requires 2 mandatory arguments for range.

Example:

```
# importing random module
Import random

# Giving a lower limit value of the range
lowerlimit = 1

# Giving a higher limit value of the range
higherlimit = 50

print('Random Number from', lowerlimit,'and', higherlimit,'is:')

# getting a random number from 1 to 50
x = random.randint(1,50)

# printing the generated random number
print(x)
```

Output:

Random Number from 1 and 50 is:

18

How can you get a random number in python?

- To generate random number in Python, randint() function is used. This function is defined in random module.
- The random.randint() is used to generate the random number, also this can be used to generate any number in a range, and then using that number, we can find the value at the corresponding index, just like the above-mentioned technique. But it differs by the fact that it requires 2 mandatory arguments for range.

Example:

```
# importing random module
Import random

# Giving a lower limit value of the range
lowerlimit = 1

# Giving a higher limit value of the range
higherlimit = 50

print('Random Number from', lowerlimit,'and', higherlimit,'is:')

# getting a random number from 1 to 50
x = random.randint(1,50)

# printing the generated random number
print(x)
```

Output:

Random Number from 1 and 50 is:

18

How will you set the starting value in generating random numbers?

- Python number method seed() sets the integer starting value used in generating random numbers. Call this function before calling any other random module function.
- The Python random seed() method takes two parameters in which the first parameter (default value = None) is an optional parameter that is used to provide the seed value for the random number generator. The second parameter (default value = 2) is required which is used to tell Python's random generator how the conversion of parameters into an integer value is to be performed. The method returns None, it only fixes the generation of the random values.

Uses of random.seed()

- This is used in the generation of a pseudo-random encryption key. Encryption keys are an important part of computer security. These are the kind of secret keys which used to protect data from unauthorized access over the internet.
- It makes optimization of codes easy where random numbers are used for testing. The output of the code sometime depends on input. So the use of random numbers for testing algorithms can be complex. Also seed function is used to generate same random numbers again and again and simplifies algorithm testing process.

Example:

```
# importing random module
import random

random.seed(3)

# print a random number between 1 and 1000.
print(random.randint(1, 1000))

# If seed function is not used
# Gives totally unpredictable responses.
print(random.randint(1, 1000))
```

Output: 244
607

How will you randomizes the items of a list in place?

In Python, there are several ways to shuffle a list. Here are various Python ways for shuffling lists.

- Using random.shuffle() function
- Using random.sample() function
- Using Fisher–Yates shuffle Algorithm
- Using random.randint() and pop() function

Using random.shuffle() function

- The shuffle() method in the random module is used to shuffle a list. It takes a sequence, such as a list, and reorganizes the order of the items.
- This shuffle() method changes the original list, it does not return a new list. The ordering of lists is lost in this process which is the only drawback of this method.

Syntax

random.shuffle(sequence, function)

- sequence - any sequence like a list, tuple, etc.
- function(optional)- a function's name that returns a value between 0.0 and 1.0. The function random() will be used if it is not specified.

Example:

```
import random
test_list = [1, 4, 5, 6, 3]
print("The original list is : " + str(test_list))

# using random.shuffle() to shuffle a list
random.shuffle(test_list)

print("The shuffled list is : " + str(test_list))
```

Output:

```
The original list is : [1, 4, 5, 6, 3]
The shuffled list is : [5, 1, 3, 4, 6]
```

Using random.sample() function

- The random.sample() method in python returns a new shuffled list. The original list remains unchanged.
- The random.sample() method returns a list containing a randomly selected number of elements from a sequence.

Syntax

random.sample(sequence, k)

- sequence - any sequence like a list, tuple, etc
- k - number of elements to return

Example:

```
import random
test_list = [1, 4, 5, 6, 3]
print("The original list is : " + str(test_list))

# using random.sample()to shuffle a list
res = random.sample(test_list, len(test_list))
print("The shuffled list is : " + str(res))
```

Output:

```
The original list is : [1, 4, 5, 6, 3]
The shuffled list is : [4, 3, 1, 6, 5]
```

Using Fisher-Yates shuffle Algorithm

- This is a well-known algorithm in Python that is used to shuffle a sequence of numbers.

Fisher-Yates shuffle Algorithm

- The Fisher-Yates shuffle Algorithm has a time complexity of O(n). The assumption is that we are given the function rand(), which generates a random number in O(1) time. The idea is to begin with the last element and swap it with a randomly chosen element from the entire array (including the last). Consider

the array from 0 to n-2 (with the size reduced by one) and repeat the process until we reach the first element.

Example:

```
import random

test_list = [1, 4, 5, 6, 3]
print("The original list is : " + str(test_list))

# using Fisher–Yates shuffle Algorithm to shuffle a list

for i in range(len(test_list)-1, 0, -1):

    # Pick a random index from 0 to i
    j = random.randint(0, i + 1)

    # Swap arr[i] with the element at random index
    test_list[i], test_list[j] = test_list[j], test_list[i]

print("The shuffled list is : " + str(test_list))
```

Output:

```
The original list is : [1, 4, 5, 6, 3]
The shuffled list is : [3, 4, 5, 6, 1]
```

Using random.randint() and pop() function

random.randint() − Returns a random number within the specified range

Example:

The following program returns the shuffled list using random.randint() and pop() function –

```
import random
arr = [1, 2, 3, 4, 5, 6]
print("Original List: ", arr)
n = len(arr)
```

```
for i in range(n):
    j = random.randint(0, n-1)
    element = arr.pop(j)
    arr.append(element)

print("Shuffled List: ", arr)
```

Output:

```
Original List:  [1, 2, 3, 4, 5, 6]
Shuffled List:  [1, 5, 2, 6, 3, 4]
```