# Module 3: Collections, functions and Modules

**1> What is the list?**

**Ans.** A list in Python is a versatile data structure that can hold an ordered collection of items of various data types. Lists are mutable, meaning their elements can be changed after creation.

-> **Reversing a List:**

You can reverse a list using slicing or the **reverse()** method.

Example:

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

reversed_list = my_list[::-1]  # Using slicing

print(reversed_list)


# Using reverse() method

my_list.reverse()

print(my_list)
```

**2> How will you remove the last object from a list? Suppose list1 is [2, 33, 222, 14, and 25], what is list1 [-1]?**

**Ans. ==> Removing the Last Object from a List:**

To remove the last object from a list, you can use the **pop()** method without passing an index.

```
list1 = [2, 33, 222, 14, 25]

list1.pop () # Removes the last element

print(list1[-1])
```

**==> Accessing -1 Index in a List:**

**list1[-1]** will access the last element of the list **list1**. In the provided case (**[2, 33, 222, 14, 25]**), **list1[-1]** would yield **25**.

**3> Differentiate between append () and extend () methods?**

**Ans. Difference between append () and extend () methods:**

**-> append ()** method: Adds a single element to the end of the list.
Example: **my_list.append(6)**

**-> extend ()** method: Adds elements from an iterable (list, tuple, etc.) to the end of the list.

Example: **my_list.extend([7, 8, 9])**

**5> How will you compare the two lists?**

**Ans. ==> Comparing Two Lists:**

You can compare lists using the **==** operator.

Example:

```
list1 = [1, 2, 3]

list2 = [1, 2, 3]

if list1 == list2:

    print ("Lists are equal")

else:

    print ("Lists are not equal")
```

**18> What is tuple? Difference between list and tuple.**

**Ans.** A tuple is like a list but is immutable, meaning once it's created, its elements cannot be changed.

-> Lists use square brackets [], while tuples use parentheses ().

**31> How will you create a dictionary using tuples in python?**

**Ans.** In Python, you can create a dictionary using tuples by utilizing the tuples as keys in the dictionary. Tuples can be used as dictionary keys because they are immutable, unlike lists which cannot be used

as keys due to their mutability. Here's an example of creating a dictionary using tuples:

```
# Creating a dictionary using tuples as keys

        my_dict = {('a', 'b'): 10, ('c', 'd'): 20}


    # Accessing values using tuple keys

        print (my_dict[('a', 'b')]) # Output: 10

        print (my_dict[('c', 'd')]) # Output: 20


    # Adding a new tuple key-value pair

        my_dict[('e', 'f')] = 30


    # Displaying the updated dictionary

        print(my_dict)
```

-> In this example:

- The **my_dict** dictionary is created using tuples **('a', 'b')** and **('c', 'd')** as keys with corresponding values **10** and **20**, respectively.
- Values are accessed using tuple keys (**'a', 'b')** and **('c', 'd')**).

- A new tuple key-value pair **('e', 'f'): 30** is added to the dictionary.

-> You can create a dictionary in Python using tuples as keys in a similar manner by specifying the tuples and their associated values within the curly braces **{}**.

**35> How Do You Traverse Through a Dictionary Object in Python?**

**Ans. Traversing Through a Dictionary:**

**Using for Loop:**

-> You can iterate through a dictionary using a **for** loop in different ways:

**Iterating through Keys:**

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

for key in my_dict:

    print(key) # Prints keys

# Alternatively:

for key in my_dict.keys()

print(key) # Prints keys
```

**Iterating through Values:**

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

for value in my_dict.values():
```

```
    print(value) # Prints values
```

**Iterating through Key-Value Pairs:**

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
for key, value in my_dict():
   print (f"Key: {key}, Value: {value}") # Prints key-value pairs
```

**36> How Do You Check the Presence of a Key in A Dictionary?**

**Ans. Checking the Presence of a Key in a Dictionary:**

**Using in Operator:**

-> You can check if a key exists in a dictionary using the **in** operator:

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
if 'b' in my_dict:
    print ("Key 'b' exists in the dictionary.")
else:
    print ("Key 'b' does not exist in the dictionary.")
```

**Using get () Method:**

-> The **get ()** method can be used to retrieve the value of a specified key in a dictionary. If the key is not found, it returns a default value (or **None** if not specified):

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

# Checking key presence using get () method

value = my_dict.get('b')

if value is not None:

        print ("Key 'b' exists in the dictionary.")

else:

         print ("Key 'b' does not exist in the dictionary.")
```

-> Both methods (**in** operator and **get ()** method) are commonly used to check the existence of a key in a dictionary in Python. Use the method that suits your specific use case and preference.

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

**Ans. Explanation of zip () method in Python:**

-> The **zip ()** function in Python is used to combine multiple iterable objects (like lists, tuples, etc.) elementwise. It takes iterables as input and returns an iterator of tuples where each tuple contains elements from each iterable at the same index.

**For example:**

```
list1 = ['a', 'b', 'c']

list2 = [1, 2, 3]

result = zip (list1, list2)

for item in result:
```

```
    print(item)
```

**Output:**

```
('a', 1)
    ('b', 2)
    ('c', 3)
```

-> This method is commonly used when you need to iterate through multiple iterables simultaneously or when you want to combine them into a dictionary or perform some operation based on corresponding elements from different iterables.

## 51> How Many Basic Types of Functions Are Available in Python?

**Ans.** There are several types of functions available. They can be broadly categorized into the following basic types:

**1.Built-in functions**: These are functions that are pre-defined in Python and are available for use without the need for importing any modules. Examples include **print()**, **len()**, **max ()**, **min ()**, etc.

**2.User-defined functions**: These are functions created by the user to perform a specific task or set of tasks. They are defined using the **def** keyword in Python.

**3.Anonymous functions (lambda functions)**: These are small, anonymous functions that can have any number of arguments but only one expression. They are defined using the **lambda** keyword.

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

**Ans. Picking a random item from a list or tuple:**

You can use random**. Choice ()** function from the **random** module in Python to pick a random item from a list or tuple.

Example:

```
import random

    my_list = [1, 2, 3, 4, 5]

    random item = random. Choice(my_list)

    print ("Random item from the list:", random item)
```

**53> How can you pick a random item from a range?**

**Ans. Picking a random item from a range:**

To pick a random item from a range in Python, you can convert the range to a list and then use **random. Choice ()** to select a random item from that list.

Example:

```
import random

    my range = range (1, 10)

    random item = random. Choice (list (my range))

    print ("Random item from the range:", random item)
```

**54> How can you get a random number in python?**

**Ans. Getting a random number in Python:**

You can use random**. Random ()** function to get a random floating-point number between 0 and 1. If you need a random integer within a specific range, you can use **random.randint(a, b)** to generate a random integer between **a** and **b** (inclusive).

Examples:

```
import random

    # Random floating-point number between 0 and 1

    random float = random. Random ()

    print ("Random floating-point number:", random float)

    # Random integer between a specific range

    random integer = random.randint(1, 100)

    print ("Random integer between 1 and 100:", random integer)
```

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

**Ans. Setting the starting value in generating random numbers:**

You can set the starting value (seed) for the random number generator using the **random. Seed ()** function from the **random** module. This is useful when you want to generate the same sequence of random numbers every time you run your program, providing a predictable output for debugging or testing purposes.

Example:

```
import random

    # Set a seed value (starting value)

    random. Seed (42)

    # Generate random numbers after setting the seed

    random number = random.randint(1, 100)

    print ("Random number with seed 42:", random number)
```

**56> How will you randomize the items of a list in place?**

**Ans. Randomizing the items of a list in place:**

To shuffle (randomize) the items of a list in place, you can use random**. Shuffle ()** function from the **random** module.

Example:

```
import random
    my_list = [1, 2, 3, 4, 5]
    # Shuffle the list in place
    random. Shuffle (my_list)
    print ("Shuffled list:", my_list)
```