Module – 2 : Python – Collections, Functions and Modules

1) Understanding how to create and access elements in a list.

- **List** ----> ordered, mutable(changeable), allow duplicate values, represent in **[] square** bracket.
  Ex :

```
my_list = []          # empty list

my_list = ['vivek', 'rajkot', 'python', 24, 'java', 3.14, True]
          # multiple datatype list

print(my_list)
```

- **Accessing list :**
  Using index number…
  Index starts from 0 for the first element.
  Negative indexing starts from -1 for the last element.
  Ex :

```
print(my_list[0])
print(my_list[1:4])
print(my_list[:5])
print(my_list[2:])
print(my_list[1:8:2])
print(my_list[-1])          # Last element
```

- **Loop :**
  Ex :

```
for i in my_list:
    print(my_list)
```

2) Indexing in lists (positive and negative indexing).

- **Positive Indexing :**
  Indexing starts from **0** for the first element.
  Increases by **1** for each next element.
  Ex :
  my_list = ["apple", "banana", "cherry", "mango"]

```
print(my_list[0])
print(my_list[1])
print(my_list[2])
print(my_list[3])
```

- **Negative Indexing :**
  Indexing starts from **-1** for the last element.
  Decreases by **1** as you move left.
  Ex :
  my_list = [“apple”, “banana”, “cherry”, “mango”]

```
print(my_list[-1])
print(my_list[-2])
print(my_list[-3])
print(my_list[-4])
```

3) Slicing a list: accessing a range of elements.

- **Slicing a list** means accessing a **range of elements** using the slice notation:
  list[start:end:step]
  start → index where the slice begins (included).
  end → index where the slice stops (excluded).
  step → how many items to skip (default is 1).
- **Ex** : **my_list = [10, 20, 30, 40, 50, 60, 70]**

```
print(my_list[1:5])      # [20, 30, 40, 50]
print(my_list[:3])       # [10, 20, 30]
print(my_list[2:])       # [30, 40, 50, 60, 70]
print(my_list[::2])      # [10, 30, 50, 70]
print(my_list[::-1])     # [70, 60, 50, 40, 30, 20, 10]
print(my_list[2:7:2])    # [30, 50, 70]
```

4) Common list operations: concatenation, repetition, membership.

- **Concatenation :** Joins two or more lists into one.
  Ex :

```
list1 = [1, 2, 3]
list2 = [4, 5]
con_list = list1 + list2
print(con_list)                  # [1, 2, 3, 4, 5]
```

- **Repetition :** Repeats the elements of a list multiple times.

  Ex : **list = [“A”, “B”, “C”]**
  ans = list * 3
  print(ans) **# [‘A’, ‘B’, ‘C’, ‘A’, ‘B’, ‘C’, ‘A’, ‘B’, ‘C’]**

- **Membership :** Checks if an element exists in a list.

  Ex : **fruits = [“apple”, “banana”, “cherry”, “mango”]**

  print(“apple” in fruits) # True
  print(“mango” in fruits) # False
  print(“mango” not in fruits) # True

5) Understanding list methods like append(), insert(), remove(), pop().

- **append() :** Adds an element to the end of the list.

  Ex : **fruits = [“apple”, “banana”]**
  fruits.append(“cherry”)
  print(fruits) **# [‘apple’, ‘banana’, ‘cherry’]**

- **insert() :** Insert an item at a specific index.

  Ex : **fruits = [“apple”, “banana”]**
  fruits.insert(1, “cherry”)
  print(fruits) **# [‘apple’, ‘cherry’, ‘banana’]**

- **remove() :** Remove the first occurrence of a value.

  Ex : **fruits = [“apple”, “banana”, “cherry”]**
  fruits.remove(“cherry”)
  print(fruits) **# [‘apple’, ‘banana’]**

- **pop() :** Remove and return an item from the list.

  Ex : **fruits = [“apple”, “banana”, “cherry”, “orange”]**
  fruits.pop(1)
  # fruits.pop() **-------- Remove last index element**
  # fruits.pop(12) **-------- Give Index_Error**
  print(fruits) **# [‘apple’, ‘cherry’, ‘orange’]**

6) Iterating over a list using loops.

- fruits = [“apple”, “banana”, “cherry”, “orange”]

  - for fruit in fruits:
    print(fruit)

  - for i in range(len(fruits)):
    print(f“{i} -----> {fruits[i]}”)

  - i = 0
    while i < len(fruits):
    print(fruits[i])
    i += 1

  - for index, fruit in enumerate(fruits):
    print(index, fruit)

7) Sorting and reversing a list using sort(), sorted(), and reverse().

- **sort() :** Sorts the list in place (modifies the original list).
  Ex :
  ```
  n = [5, 2, 9, 1]
  n.sort()
  print(n)              # [1, 2, 5, 9]
  n.sort(reverse=True)
  print(n)              # [9, 5, 2, 1]
  ```

- **sorted() :** Returns a new sorted list without changing the original.
  Ex :
  ```
  n = [5, 2, 9, 1]
  sorted_numbers = sorted(n)
  print(sorted_numbers)          # [1, 2, 5, 9]
  print(n)                       # [5, 2, 9, 1]  (unchanged)

  desc_sorted = sorted(n, reverse=True)
  print(desc_sorted)             # [9, 5, 2, 1]
  ```

- **reverse() :** Reverses the list in place (does not sort).
  Ex :
  ```
  n = [5, 2, 9, 1]
  n.reverse()
  print(n)      # [1, 9, 2, 5]  (just reversed order, not sorted)
  ```

8) Basic list manipulations: addition, deletion, updating, and slicing.

- **Addition :**
  fruits = [“apple”, “banana”]
  - fruits.append(“cherry”) # [“apple”, “banana”, “cherry”]
  - fruits.extend([“date”, “elderberry”])
    # [“apple”, “banana”, “cherry”, “date”, “elderberry”]
  - fruits.insert(1, “blueberry”)
    # [“apple”, “blueberry”, “banana”, “cherry”, “date”, “elderberry”]
- **Deletion :**
  fruits = [“apple”, “banana”, “cherry”]
  - fruits.remove(“banana”) # [“apple”]
  - del fruits[0] # [“banana”]
  - removed = fruits.pop(2) # [“apple”, “banana”]
  - fruits.clear() # []
- **Updating :**
  fruits = [“apple”, “banana”, “cherry”]
  - fruits[1] = “blueberry”
    # [“apple”, “blueberry”, “cherry”]
  - fruits[0:2] = [“apricot”, “blackberry”]
    # [“apricot”, “blackberry”, “cherry”]
- **Slicing :**
  n = [0, 1, 2, 3, 4, 5]
  - list = n[1:4] # [1, 2, 3]
  - list = n[:3] # [0, 1, 2]
  - list = n[2:] # [2, 3, 4, 5]
  - list = n[::2] # [0, 2, 4]

9) Introduction to tuples, immutability.

- **Tuple** ---> ordered, unmutable(unchangeble), allow duplicate values, represent in () round bracket.

  Ex : **my_tuple = (“apple”, 3.14, “Python”, 25, True)**

  print(my_tuple)

- **Accessing element :**
  fruits = (“apple”, “banana”, “cherry”)

  print(fruits[0]) # “apple”
  print(fruits[-1]) # “cherry”
  print(fruits[1:3]) # (“banana”, “cherry”)

- **Immutability :**
  The key difference between tuples and lists is immutability: Once a tuple is created, you cannot change, add, or remove its elements.

  n = (1, 2, 3)
  n[0] = 10 # Give error because tuple is immutable.

  New tuple is created by concatenating or slicing. Also mutable objects (like lists) inside a tuple, and modify those inner objects.

  n = (1, [2, 3], 4)
  n[1][0] = 99
  print(n) # (1, [99, 3], 4)

10) Creating and accessing elements in a tuple.

- Creating tuple :

  n = (1, 2, 3, 4)
  mixed = (“apple”, 3.14, “Python”, 25, True)
  nested = (1, (2, 3), 4)

- Accessing tuple :
  fruits = (“apple”, “banana”, “cherry”)

  print(fruits[0]) # “apple”

```
print(fruits[-1])        # “cherry”
print(fruits[1:3])       # (“banana”, “cherry”)
print(fruits[:2])        # (“apple”, “banana”)
print(fruits[::2])       # (“apple”, “cherry”)
```

- ```
  nested = (1, (2, 3), 4)
  print(nested[1][0])      # 2
  ```

11) Basic operations with tuples: concatenation, repetition, membership.

- **Concatenation :** Joins two or more tuple into one.

  Ex :
  ```
  tuple1 = (1, 2, 3)
  tuple2 = (4, 5)
  con_tuple = tuple1 + tuple2
  print(con_tuple)          # (1, 2, 3, 4, 5)
  ```

- **Repetition :** Repeats the elements of a tuple multiple times.

  Ex :
  ```
  tuple = (“A”, “B”, “C”)
  ans = tuple * 3
  print(ans)          # (‘A’, ‘B’, ‘C’, ‘A’, ‘B’, ‘C’, ‘A’, ‘B’, ‘C’)
  ```

- **Membership :** Checks if an element exists in a tuple.

  Ex :
  ```
  fruits = (“apple”, “banana”, “cherry”, “mango”)

  print(“apple” in fruits)          # True
  print(“mango” in fruits)          # False
  print(“mango” not in fruits)      # True
  ```

12) Accessing tuple elements using positive and negative indexing.

- **Positive index :**
  - Index starts at 0 for the first element.
  - Counts from left to right.

  Ex :

  fruits = (“apple”, “banana”, “cherry”, “mango”)

  ```
  print(fruits[0])        # “apple”
  print(fruits[2])        # “cherry”
  print(fruits[3])        # “mango”
  ```
  ```

- **Negative index :**
  - Index starts at -1 for the last element.
  - Counts from right to left.

  Ex :
  fruits = (“apple”, “banana”, “cherry”, “mango”)

```
print(fruits[-1])        # “mango”
print(fruits[-2])        # “cherry”
print(fruits[-3])        # “banana”
```

13) Slicing a tuple to access ranges of elements.

- **Slicing a tuple** means accessing a **range of elements** using the slice notation:
  tuple(start:end:step)
  start → index where the slice begins (included).
  end → index where the slice stops (excluded).
  step → how many items to skip (default is 1).
- **Ex** : **my_tuple = (10, 20, 30, 40, 50, 60, 70)**

```
print(my_tuple[1:5])        # (20, 30, 40, 50)
print(my_tuple[:3])         # (10, 20, 30)
print(my_tuple[2:])         # (30, 40, 50, 60, 70)
print(my_tuple[::2])        # (10, 30, 50, 70)
print(my_tuple[::-1])       # (70, 60, 50, 40, 30, 20, 10)
print(my_tuple[2:7:2])      # (30, 50, 70)
```

14) Introduction to dictionaries: key-value pairs.

- **Dictionary** **--->** ordered, mutable(changeable), no duplicate value, pair{key:value}, represent in {} curly bracket.

  Ex :

```
student = {
        “id” : 101
        “name” : “Vivek”
        “age” : 21
    }
print(student)
```

- Keys : “id”, “name”, “age”
- Values : 101, “Vivek”, 21

- person = dict(name = “Vivek”, age = 21, city = “Rajkot”)
  print(person)

15) Accessing, adding, updating, and deleting dictionary elements.

- **Accessing element :**

  person = {“name” : “Meet”, “age”: 25, “city”: “Surat”}

  print(person[“name”])
  print(person.get(“age”))
  print(person.get(“id”, “Not Found!”))

- **Adding element :**

  person = {“name” : “Meet”, “age”: 25, “city”: “Surat”}

  person[“email”] = “meet@gmail.com”
  print(person)

- **Updating element :**

  person = {“name” : “Meet”, “age”: 25, “city”: “Surat”}

  person[“age”] = 29
  print(person[“age”])

  person.update({“city” : “Ahemdabaad”, “phone”: 001-4581269})
  print(person)

- **Deleting element :**

  person = {“name” : “Meet”, “age”: 25, “city”: “Surat”}

  del person[“city”]

  person.pop(“city”)

person.popitem() ----- Delete last item

person.clear() ----- clear whole dictionary

16) Dictionary methods like keys(), values(), and items().

- **Keys() :** Returns a view object containing all keys in the dictionary.
  Ex :
  person = {“name” : “Meet”, “age”: 25, “city”: “Surat”}
  - print(person.keys())
  - for key in person.keys():
    print(key)

- **Values() :** Returns a view object containing all values in the dictionary.
  Ex :
  person = {“name” : “Meet”, “age”: 25, “city”: “Surat”}
  - print(person.values())
  - for key in person.values():
    print(key)

- **Items() :** Returns a view object containing all keys and values both in the dictionary.
  Ex :
  person = {“name” : “Meet”, “age”: 25, “city”: “Surat”}
  - print(person.items())
  - for key in person.items():
    print(key)

17) Iterating over a dictionary using loops.

- person = {“name” : “Meet”, “age”: 25, “city”: “Surat”}

- for i in person:
  print(i)

- for i in person.keys():
  print(i)

- for i in person.values():
  print(i)

- for i in person.items():
  print(i)

- for i, j in person.items():
  print(f“{i} ---> {j}”)

18) Merging two lists into a dictionary using loops or zip().

- **Using Loop :**

Ex :

```
keys = [“a”, “b”, “c”]
values = [1, 2, 3]

my_dict = {}

for i in range(len(keys)):
    my_dict[keys[i]] = values[i]

print(my_dict)
```

- **Using Zip() :**

Ex :

```
keys = [“a”, “b”, “c”]
values = [1, 2, 3]

my_dict = {}

my_dict = dict(zip(keys, values))

print(my_dict)
```

19) Counting occurrences of characters in a string using dictionaries.

- my_str = “Hello! My name is Vivek and I am learning python language.”

```
count = {}

for i in my_str:
  if i in count:
      count[i] += 1
  else:
      count[i] = 1

print(count)
```

20) Defining functions in Python.

- A function is a block of reusable code.
- Defined using the def keyword.
- Function with parameters :
  Ex:

```
def greet(name):
    print(f“Hello, {name}!”)

greet(“Vivek”)
```

- Function with return value :
  Ex :

```
def addition(a, b):
    return a + b

result = addition(5, 3)
print(result)
```

- Function with default value :
  Ex :

```
def greet(name = “Guest”):
    print(f"Hello, {name}!")

greet()                 # uses default
greet(“Vivek”)          # overrides default
```

21) Different types of functions: with/without parameters, with/without return values.

- **Without parameters & without return value :**

```
def greet():
   print("Hello, World!")

greet()
```

- **Without parameters & with return value :**

```
def get_pi():
   return 3.14

value = get_pi()
print(value)
```

- **With parameters & without return value :**

```
def person(name):
   print(f"Hello, {name}!")

person("Vivek")
```

- **With parameters & with return value :**

```
def add_numbers(a, b):
   return a + b

result = add_numbers(5, 3)
print(result)
```

22) Anonymous functions (lambda functions).

- Anonymous functions in Python, also known as lambda functions.
- Functions are small, unnamed functions defined with the lambda keyword.

lambda arguments: expression

- lambda is the keyword.
- arguments are the inputs (like function parameters).
- expression is a single expression whose result will be returned.

- **Ex** :

```
add = lambda x, y: x + y
print(add(5, 3))                    # 8
```

23) Introduction to Python modules and importing modules.

- A module in Python is simply a file that contains Python code (functions, classes, or variables).
- It helps organize large programs into smaller, manageable pieces.
- Modules allow code reusability.
- You can also create your own custom modules (mymodule.py).
- Types of modules :
  Built-in modules --- Already available in Python (like, math, os, random).
  User-defined modules --- Files created by you (like, calculator.py).
  External modules --- Installed using pip (like, numpy, pandas).
- **Ex** :

```
import math
print(math.sqrt(16))                # 4.0
```

- All function access from module like, **from math import ***
- Creating and Importing a Custom Module,
- mymodule.py

```
def greet(name):
    return f"Hello, {name}!"
```

- Another py file…

```
import mymodule

print(mymodule.greet("Vivek"))
# Hello, Vivek!
```

24) Standard library modules: math, random.

- Math Module :
  - The math module provides mathematical functions and constants.

Ex :

```
import math

print(math.sqrt(49))
print(math.pow(6,2))
print(math.factorial(4))
```

```
print(math.floor(12.45))
print(math.log(2))
print(math.ceil(4.75))
print(math.pi)

angle = math.radian(60)
print(math.sin(angle))
```

- Random Module :
  - The random module is used to generate random numbers.

Ex :

```
import random

print(random.random())
# Random float between 0.0 and 1.0.

print(random.randint(a, b))
# Random integer between a and b (inclusive).

print(random.uniform(a, b))
# Random float between a and b.

print(random.choice(seq))
# Randomly picks one item from a sequence (like list/tuple).
```

25) Creating custom modules.

- **Module :** A module is simply a Python file (.py) that contains functions, variables, or classes you want to reuse.

- **mymodule.py**

```
def add(a,b):
    return a+b

def production(a,b):
    return a*b

def stu_data(id, name):
    print("Id : ",id)
    print("Name : ",name)
```

- **main.py**

```
from mymodule import *

print("Add :",add(12,8))
print("Production :",production(5,4))
print("Student Data :",stu_data(101, 'Vivek'))
```

- **mymodule.py** ---> our custom module.
- **main.py** ---> main program where we use it.