**Module 7) Python – Collections, functions, and Modules**

Accessing List Theory:

**1. Creating and Accessing Elements in a List**

A **list** is a collection of items that are **ordered**, **mutable (changeable)**, and **allow duplicates**.

**Creating a List:**

python

my\_list = [10, 20, 30, 40, 50]

**Accessing Elements (using index):**

python

print(my\_list[0])

print(my\_list[2])

**2. Indexing in Lists**

Each element in a list has an **index** (position number).

**Positive Indexing (from left to right):**

* Starts at **0**
* Example:

python

my\_list = ['a', 'b', 'c', 'd']

print(my\_list[0]) # 'a'

print(my\_list[2]) # 'c'

**Negative Indexing (from right to left):**

* Starts at **-1**
* Example:

python

my\_list = ['a', 'b', 'c', 'd']

print(my\_list[-1]) # 'd'

print(my\_list[-2]) # 'c'

**3. Slicing a List**

Slicing is used to access **a portion (range)** of a list.

**Syntax:**

python

list [ start : end : step]

* **start** = starting index (included)
* **end** = ending index (excluded)
* **step** = jump (optional)

**🔹 Basic Slicing Example:**

python

my\_list = [10, 20, 30, 40, 50, 60]

print(my\_list[1:4])

**🔹 Step Value:**

python

print(my\_list[0:6:2])

**🔹 Negative Slicing:**

python

print(my\_list[-4:-1]

2. List Operations

**1. Common List Operations**

**🔹 Concatenation (+)**

You can join two lists using the + operator.

python

list1 = [1, 2, 3]

list2 = [4, 5]

result = list1 + list2

print(result)

**🔹 Repetition (\*)**

You can repeat a list using the \* operator.

python

list1 = [1, 2]

result = list1 \* 3

print(result)

**🔹 Membership (in, not in)**

Used to check if an item exists in the list.

python

fruits = ['apple', 'banana', 'cherry']

print('banana' in fruits)

print('grape' not in fruits)

**2. Common List Methods**

**🔹 append()**

Adds an element **at the end** of the list.

python

nums = [1, 2]

nums.append(3)

print(nums)

**🔹 insert(index, value)**

Inserts an element at a **specific index**.

python

nums = [1, 3]

nums.insert(1, 2)

print(nums

**🔹 remove(value)**

Removes the **first occurrence** of the value.

python

nums = [1, 2, 3, 2]

nums.remove(2)

print(nums)

**🔹 pop(index)**

Removes and returns the element at the given index.  
If no index is given, removes the **last item**.

python

nums = [10, 20, 30]

nums.pop(1)

print(nums)

3. Working with Lists Theory

Theory:

**1. Iterating Over a List Using Loops**

You can **loop through a list** using a for or while loop.

**🔹 Using for loop:**

python

colors = ['red', 'green', 'blue']

for color in colors:

print(color)

**🔹 Using while loop:**

python

i = 0

while i < len(colors):

print(colors[i])

i += 1

**2. Sorting and Reversing a List**

**🔹 sort() → Sorts the list in place (modifies the original list)**

python

nums = [3, 1, 4, 2]

nums.sort()

print(nums)

**🔹 sorted() → Returns a new sorted list, keeps original unchanged**

python

nums = [3, 1, 4, 2]

sorted\_nums = sorted(nums)

print(sorted\_nums

print(nums)

**🔹 reverse() → Reverses the list in-place**

python

nums = [10, 20, 30]

nums.reverse()

print(nums)

**3. Basic List Manipulations**

**🔹 Addition (Adding Elements)**

* append() – adds to end
* insert() – adds at a specific index

python

fruits = ['apple', 'banana']

fruits.append('cherry')

fruits.insert(1, 'orange')

**🔹 Deletion (Removing Elements)**

* remove() – removes by value
* pop() – removes by index
* del – deletes by index

python

fruits.remove('banana')

fruits.pop(0)

del fruits[1]

**🔹 Updating Elements**

You can update values by assigning to an index:

python

fruits[0] = 'grape'

**🔹 Slicing (Accessing a Range)**

python

numbers = [10, 20, 30, 40, 50]

print(numbers[1:4])

print(numbers[:3])

print(numbers[::2])

4. Tuple Theory:

**1. Introduction to Tuples and Immutability**

**🔹 What is a Tuple?**

A **tuple** is a collection of elements that is:

* **Ordered** (has index positions)
* **Immutable** (cannot be changed after creation)
* **Allows duplicate values**

**2. Creating and Accessing Elements in a Tuple**

**🔹 Create a Tuple:**

python

t1 = (10, 20, 30)

t2 = ("Ruchi", True, 3.14)

python

single = (5)

**🔹 Accessing Tuple Elements:**

Use indexing, like lists:

python

print(t1[0

print(t2[-1])

**3. Basic Operations with Tuples**

**🔹 Concatenation (+)**

Joins two tuples into one:

python

t1 = (1, 2)

t2 = (3, 4)

result = t1 + t2

print(result)

**🔹 Repetition (\*)**

Repeats a tuple:

python

t = (10, 20)

print(t \* 3)

**🔹 Membership (in, not in)**

Checks if an element exists in the tuple:

python

t = (5, 10, 15)

print(10 in t)

print(25 not in t)

5. Accessing Tuples Theory:

**1. Accessing Tuple Elements using Indexing**

Just like lists, **tuples are indexed** collections. You can access elements using **positive** or **negative** indexing.

**🔹 Positive Indexing**

* Starts from **0** and goes left to right.
* Used when position from beginning is known.

**Example:**

python

t = (10, 20, 30, 40, 50)

print(t[0])

print(t[2])

**🔹 Negative Indexing**

* Starts from **-1** and goes right to left.
* Useful when accessing from the end.

**Example:**

python

print(t[-1])

print(t[-3])

**2. Slicing a Tuple**

Tuple slicing is used to **access a group (range)** of elements.

**🔹 Syntax:**

python

tuple[start : end : step]

* start – Starting index (inclusive)
* end – Ending index (exclusive)
* step – (optional) Jump or gap

**🔹 Slicing Examples:**

python

t = (0, 10, 20, 30, 40, 50, 60)

print(t[1:5])

print(t[:4])

print(t[::2])

print(t[-4:-1])

6. Dictionaries Theory:

**1. Introduction to Dictionaries**

A **dictionary** is a collection of **key-value pairs**, where:

* Each key is **unique**
* Values can be of any data type
* Defined using **curly braces {}**

**🔹 Dictionary Syntax:**

python

student = {

"name": "Ruchi",

"age": 20,

"grade": "A"

}

* "name", "age", "grade" → keys
* "Ruchi", 20, "A" → values

**2. Accessing, Adding, Updating, and Deleting Dictionary Elements**

**🔹 Accessing Elements**

Use the key inside square brackets [] or the get() method:

python

print(student["name"])

print(student.get("age"))

**🔹 Adding Elements**

Simply assign a new key-value pair:

python

student["city"] = "Ahmedabad"

**🔹 Updating Elements**

Update by assigning a new value to an existing key:

python

student["grade"] = "A+"

**🔹 Deleting Elements**

* Use del to remove a key-value pair:

python

del student["age"]

* Use pop() to remove and return a value:

python

student.pop("grade")

**3. Dictionary Methods**

**🔹 keys(): Returns all the keys**

python

print(student.keys())

**🔹 values(): Returns all the values**

python

print(student.values())

**🔹 items(): Returns all key-value pairs as tuples**

python

print(student.items())

**1. Iterating Over a Dictionary Using Loops**

You can loop through:

* **Keys**
* **Values**
* **Key-value pairs**

**🔹 Looping through keys:**

python

student = {"name": "Ruchi", "age": 20}

for key in student:

print(key, student[key])

**🔹 Looping through .items():**

python

for key, value in student.items():

print(f"{key} : {value}")

**🔹 Looping through .keys() or .values():**

python

CopyEdit

for key in student.keys():

print(key)

for val in student.values():

print(val)

**2. Merging Two Lists into a Dictionary**

You can merge using:

**🔹 Method 1: zip() function**

python

keys = ['a', 'b', 'c']

values = [1, 2, 3]

merged = dict(zip(keys, values))

print(merged)

**🔹 Method 2: Using a for loop**

python

keys = ['x', 'y', 'z']

values = [10, 20, 30]

merged = {}

for i in range(len(keys)):

merged[keys[i]] = values[i]

print(merged)

**3. Counting Occurrences of Characters in a String Using a Dictionary**

You can use a dictionary to **count how many times each character appears** in a string.

**🔹 Example:**

python

text = "banana"

count = {}

for char in text:

if char in count:

count[char] += 1

else:

count[char] = 1

print(count)

8. Functions

Theory:

**1. Defining Functions in Python**

A **function** is a block of reusable code that performs a specific task.

**🔹 Why use functions?**

* Code reusability
* Better structure
* Easy debugging

**🔹 Defining a Function**

python

def greet():

print("Hello, world!")

**🔹 Calling the Function**

python

greet()

**2. Types of Functions**

**a) Without Parameters, Without Return**

python

def welcome():

print("Welcome!")

welcome()

**b) With Parameters, Without Return**

python

def greet(name):

print("Hello", name)

greet("Ruchi")

**c) With Parameters, With Return**

python

def add(x, y):

return x + y

result = add(5, 3)

print(result)

**d) Without Parameters, With Return**

python

def get\_number():

return 42

print(get\_number)

**3. Anonymous Functions (Lambda Functions)**

A **lambda** function is a **short, one-line** anonymous function with no name.

**🔹 Syntax:**

python

lambda arguments : expression

**🔹 Example – Square of a number:**

python

square = lambda x: x \* x

print (square (5))

**🔹 Example – Add two numbers:**

python

add = lambda a, b: a + b

print (add (3, 4))

9. Modules

Theory:

**1. Introduction to Python Modules**

A **module** is a file containing Python code (functions, variables, classes) that you can **import and reuse** in other programs.

**🔹 Why Use Modules?**

* Organize code in multiple files
* Reuse code across projects
* Use built-in or custom functionality

**🔹 Importing Modules**

You can import a module using:

python

import module\_name

Or import specific functions:

python

from module\_name import function\_name

**🔹 Example:**

python

import math

print(math.sqrt(16))

**2. Standard Library Modules**

Python comes with many **built-in modules** like:

**math – Mathematical operations**

python

import math

print(math.sqrt(25))

print(math.ceil(4.2))

print(math.floor(4.8))

**random – Generate random numbers**

python

import random

print(random.randint(1, 10))

print(random.choice([10, 20, 30]))

**3. Creating Custom Modules**

You can create your own module by saving functions in a .py file.

**🔹 Example: Create a file called mymodule.py**

python

# mymodule.py

def welcome(name):

print("Welcome,", name)

**🔹 Use it in another file:**

python

import mymodule

mymodule.welcome("Ruchi")