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

**1.Accessing List**

**Theory:**

**Understanding how to create and access elements in a list.**

A **list** in Python is a collection of items enclosed in square brackets [].

Accessing elements by index:

**print(fruits[0]) # Output: apple**

**print(fruits[2]) # Output: cherry**

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

## ****Indexing in Lists****

### ****Positive Indexing****

Indexes start from 0 and go up from left to right.

| **Index** | **0** | **1** | **2** | **3** |
| --- | --- | --- | --- | --- |
| Value | apple | banana | cherry | mango |

### ****Negative Indexing****

Indexes start from -1 from right to left.

| **Index** | **-4** | **-3** | **-2** | **-1** |
| --- | --- | --- | --- | --- |
| Value | apple | banana | cherry | mango |

**Slicing a list: accessing a range of elements.**

## ****Slicing a List****

Use slicing to access a **range of elements** with the syntax:

**list[start:stop]**

Includes the start index

Excludes the stop index

### Example:

print(fruits[1:3]) # Output: ['banana', 'cherry']

### Slice from the beginning:

print(fruits[:2]) # Output: ['apple', 'banana']

### Slice till the end:

print(fruits[2:]) # Output: ['cherry', 'mango']

### Full list:

print(fruits[:]) # Output: ['apple', 'banana', 'cherry', 'mango']

**Lab:**

**Write a Python program to create a list with elements of multiple data types (integers, strings, floats, etc.).**

# Creating a list with elements of various data types

mixed\_list = [42, "hello", 3.14, True, None, [1, 2, 3], {"key": "value"}]

# Display the list

print("Mixed Data Type List:")

print(mixed\_list)

**Write a Python program to access elements at different index positions.**

# List with mixed data types

data = [100, "Python", 45.67, False, "GPT"]

# Accessing elements using positive indexing

print("Element at index 0:", data[0]) # 100

print("Element at index 2:", data[2]) # 45.67

# Accessing elements using negative indexing

print("Element at index -1 (last):", data[-1]) # GPT

print("Element at index -3:", data[-3]) # 45.67

**Practical Examples:**

1. **Write a Python program to create a list of multiple data type elements.**

# Creating a list with different types of data

multi\_type\_list = [25, "Arjun", 3.14, True, None, [1, 2], {'lang': 'Python'}]

# Display the list

print("List with multiple data types:")

print(multi\_type\_list)

**2. Write a Python program to find the length of a list using the len() function.**

# Reusing the same list from above

multi\_type\_list = [25, "Arjun", 3.14, True, None, [1, 2], {'lang': 'Python'}]

# Finding and printing the length of the list

length = len(multi\_type\_list)

print("Length of the list:", length)

**2. List Operations**

**Theory:**

**Common list operations: concatenation, repetition, membership. And Understanding list methods like append(), insert(), remove(), pop().**

| **Operation / Method** | **Example** | **Description** |
| --- | --- | --- |
| Concatenation | list1 + list2 | Joins two lists |
| Repetition | list \* 3 | Repeats list elements |
| Membership | 'a' in list | Checks if element exists |
| append(x) | list.append(5) | Adds at the end |
| insert(i, x) | list.insert(1, 10) | Inserts at index |
| remove(x) | list.remove('apple') | Removes first matching element |
| pop(i) | list.pop(0) | Removes element at index (or last) |

**Lab:**

Write a Python program to add elements to a list using insert() and append().

# Initial list

colors = ['red', 'blue']

# Using append() to add an element at the end

colors.append('green')

# Using insert() to add an element at a specific index

colors.insert(1, 'yellow') # insert 'yellow' at index 1

print("List after append() and insert():")

print(colors)

Write a Python program to remove elements from a list using pop() and remove().

# Initial list

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

# Using remove() to delete the first occurrence of 'banana'

fruits.remove('banana')

# Using pop() to remove the element at index 1

removed\_item = fruits.pop(1)

print("List after remove() and pop():")

print(fruits)

print("Item removed using pop():", removed\_item)

**Practical Examples:**

1. Write a Python program to update a list using insert() and append().

# Initial list

numbers = [10, 20, 30]

# Using append() to add a new element at the end

numbers.append(40)

# Using insert() to add an element at a specific position (e.g., index 1)

numbers.insert(1, 15)

print("Updated List after append() and insert():")

print(numbers)

1. Write a Python program to remove elements from a list using pop() and remove().

# Initial list

cities = ['Delhi', 'Mumbai', 'Chennai', 'Kolkata', 'Mumbai']

# Using remove() to delete the first occurrence of 'Mumbai'

cities.remove('Mumbai')

# Using pop() to remove the element at index 2

removed\_city = cities.pop(2)

print("List after remove() and pop():")

print(cities)

print("City removed using pop():", removed\_city)

**3. Working with Lists**

**Theory:**

**Iterating over a list using loops.**

## ****1.Iterate Over a List Using Loops****

You can use for or while loops to iterate through list elements.

### Using for loop:

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

for fruit in fruits:

print(fruit)

### Using while loop:

i = 0while i < len(fruits):

print(fruits[i])

i += 1

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

## ****Sorting and Reversing a List****

### sort() – Sorts the list in-place (changes the original list)

numbers = [4, 1, 3, 2]

numbers.sort()

print("Sorted list:", numbers)

### Using while loop:

i = 0while i < len(fruits):

print(fruits[i])

i += 1

**Basic list manipulations: addition, deletion, updating, and slicing.**

### Addition (using append() or insert())

my\_list = [10, 20]

my\_list.append(30)

my\_list.insert(1, 15)print("After addition:", my\_list)

### Deletion (using remove(), pop(), or del)

my\_list.remove(15) # removes the value 15

my\_list.pop() # removes the last elementdel my\_list[0] # deletes element at index 0print("After deletion:", my\_list)

### Updating (by index)

my\_list = [100, 200, 300]

my\_list[1] = 250 # updates the value at index 1print("After updating:", my\_list)

**Lab:**

**Write a Python program to iterate over a list using a for loop.**

# List of items

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

# Using for loop to iterate

print("Colors in the list:")

for color in colors:

print(color)

**Write a Python program to sort a list using both sort() and sorted().**

# Original list of numbers

numbers = [5, 2, 9, 1, 7]

# Using sort() (modifies the original list)

numbers.sort()

print("List after using sort():", numbers)

# Resetting list for demonstration of sorted()

numbers = [5, 2, 9, 1, 7]

# Using sorted() (returns a new sorted list)

sorted\_numbers = sorted(numbers)

print("Original list remains unchanged:", numbers)

print("Sorted list using sorted():", sorted\_numbers)

**Practical Examples:**

1. **Write a Python program to iterate through a list and print each element.**

# List of fruits

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

# Iterate using for loop

print("Fruits in the list:")

for fruit in fruits:

print(fruit)

**6) Write a Python program to insert elements into an empty list using a for loop and append().**

# Empty list

numbers = []

# Using for loop to append numbers from 1 to 5

for i in range(1, 6):

numbers.append(i)

print("List after appending elements:")

print(numbers)

**4. Tuple**

**Theory:**

**Introduction to tuples, immutability.**

### What is a Tuple?

A **tuple** is a collection of **ordered** elements.

Tuples are **immutable**, which means **you cannot change**, **add**, or **remove** elements after creation.

**Creating and accessing elements in a tuple.**

### Creating a Tuple:

sample\_tuple = ("red", "green", "blue")

### Accessing Elements (Indexing):

print(sample\_tuple[0]) # Output: redprint(sample\_tuple[2]) # Output: blue

**Basic operations with tuples: concatenation, repetition, membership.**

### Concatenation (+):

t1 = (1, 2, 3)

t2 = (4, 5)

result = t1 + t2print(result) # Output: (1, 2, 3, 4, 5)

### Repetition (\*):

t = ("hello",)print(t \* 3) # Output: ('hello', 'hello', 'hello')

### Membership Test (in):

colors = ("red", "green", "blue")print("green" in colors) # Output: Trueprint("yellow" in colors) # Output: False

**Lab:**

**Write a Python program to create a tuple with multiple data types.**

# Creating a tuple with multiple data types

mixed\_tuple = (25, "hello", 3.14, True, [1, 2, 3])

# Printing the tuple

print("Tuple with multiple data types:")

print(mixed\_tuple)

# Verifying the types of each element

print("\nData types of elements in the tuple:")

for element in mixed\_tuple:

print(f"{element} -> {type(element)}")

**Write a Python program to concatenate two tuples**.

# Define two tuples

tuple1 = (1, 2, 3)

tuple2 = ('a', 'b', 'c')

# Concatenate the tuples

result = tuple1 + tuple2

# Print the result

print("First Tuple:", tuple1)

print("Second Tuple:", tuple2)

print("Concatenated Tuple:", result)

**Practical Examples:**

**7) Write a Python program to convert a list into a tuple.**

# Define a list

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

# Convert list to tuple

my\_tuple = tuple(my\_list)

# Output

print("Original List:", my\_list)

print("Converted Tuple:", my\_tuple)

1. **Write a Python program to create a tuple with multiple data types.**

# Tuple with different data types

mixed\_tuple = (100, "Python", 3.14, False)

# Output

print("Tuple with multiple data types:", mixed\_tuple)

1. **Write a Python program to concatenate two tuples into one.**

# Define two tuples

tuple1 = (1, 2, 3)

tuple2 = (4, 5, 6)

# Concatenate

result = tuple1 + tuple2

# Output

print("Concatenated Tuple:", result)

1. **Write a Python program to access the value of the first index in a tuple.**

# Define a tuple

sample\_tuple = ("apple", "banana", "cherry")

# Access first index value

first\_element = sample\_tuple[0]

# Output

print("Value at the first index:", first\_element)

**5. Accessing Tuples**

**Theory:**

**Accessing tuple elements using positive and negative indexing.**

## ****Accessing Tuple Elements Using Indexing****

### Positive Indexing : ndex starts from 0 (left to right).

### Negative Indexing : Index starts from -1 (right to left).

**Slicing a tuple to access ranges of elements.**

## ****Slicing a Tuple (Accessing a Range of Elements)****

**Syntax**: tuple[start:stop:step]

Includes the start index, excludes the stop index.

**Lab:**

**Write a Python program to access values between index 1 and 5 in a tuple.**

# Define a sample tuple

my\_tuple = ("a", "b", "c", "d", "e", "f", "g")

# Access values from index 1 to 4 (5 is excluded)

sub\_tuple = my\_tuple[1:5]

# Output

print("Values from index 1 to 4:", sub\_tuple)

**Write a Python program to access alternate values between index 1 and 5 in a tuple.**

# Define a sample tuple

my\_tuple = ("a", "b", "c", "d", "e", "f", "g")

# Access every second element from index 1 to 4

alternate\_values = my\_tuple[1:5:2]

# Output

print("Alternate values from index 1 to 4:", alternate\_values)

**Practical Examples:**

**11) Write a Python program to access values between index 1 and 5 in**

**a tuple.**

# Define a tuple

my\_tuple = ('a', 'b', 'c', 'd', 'e', 'f', 'g')

# Access values from index 1 to 4 (5 excluded)

subset = my\_tuple[1:5]

print("Values between index 1 and 5:", subset)

**12)Write a Python program to access the value from the last index in a tuple.6. Dictionaries**

# Define a tuple

my\_tuple = ('apple', 'banana', 'cherry', 'date')

# Access value at last index using negative indexing

last\_value = my\_tuple[-1]

print("Value at the last index:", last\_value)

**Theory:**

**Introduction to dictionaries: key-value pairs.**

## Introduction to Dictionaries

**Dictionary** is an **unordered**, **mutable** collection of **key-value pairs**.

Each **key** is unique and is used to access the corresponding **value**.

my\_dict = {

"name": "Alice",

"age": 25,

"city": "New York"

}

**Accessing, adding, updating, and deleting dictionary elements.**

## Accessing Dictionary Elements

Access value by key:

print(my\_dict["name"]) # Output: Alice

Using .get() (returns None or a default if key doesn’t exist):

print(my\_dict.get("age")) # Output: 25print(my\_dict.get("country", "Not Found")) # Output: Not Found

## Adding and Updating Elements

Add or update a key-value pair:

my\_dict["email"] = "alice@example.com" # Adds new key

my\_dict["age"] = 26 # Updates existing key

## ✅ Deleting Elements

Delete an element using del:

del my\_dict["city"]

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

## Common Dictionary Methods

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| .keys() | Returns all keys as a view object | my\_dict.keys() |
| .values() | Returns all values as a view object | my\_dict.values() |
| .items() | Returns key-value pairs as tuples | my\_dict.items() |
| .clear() | Removes all elements | my\_dict.clear() |
| .update() | Updates dictionary with another dict or pairs | my\_dict.update({"city": "LA"}) |

**Lab:**

**Write a Python program to create a dictionary with 6 key-value pairs.**

# Creating a dictionary with 6 key-value pairs

person\_info = {

"name": "Alice",

"age": 25,

"city": "New York",

"email": "alice@example.com",

"profession": "Engineer",

"is\_active": True

}

# Displaying the dictionary

print("Dictionary with 6 key-value pairs:")

print(person\_info)

**Write a Python program to access values using dictionary keys.**

# Accessing values using dictionary keys

print("\nAccessing values using keys:")

print("Name:", person\_info["name"])

print("Age:", person\_info["age"])

print("City:", person\_info["city"])

print("Email:", person\_info["email"])

print("Profession:", person\_info["profession"])

print("Active Status:", person\_info["is\_active"])

**Practical Examples:**

1. **Write a Python program to create a dictionary of 6 key-value pairs.**

# Creating a dictionary with 6 key-value pairs

student = {

"name": "John",

"age": 20,

"grade": "A",

"roll\_no": 101,

"email": "john@example.com",

"college": "Sigma University"

}

# Display the dictionary

print("Student Dictionary:")

print(student)

1. **Write a Python program to access values using keys from a dictionary.**

# Accessing values using dictionary keys

print("\nAccessing values from the dictionary:")

print("Name:", student["name"])

print("Age:", student["age"])

print("Grade:", student["grade"])

print("Roll Number:", student["roll\_no"])

print("Email:", student["email"])

print("College:", student["college"])

**7. Working with Dictionaries**

**Theory:**

**Iterating over a dictionary using loops.**

Dictionaries can be iterated using a for loop with the .items(), .keys(), or .values() methods. This allows access to keys and values one by one.

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

Two lists can be combined into a dictionary where one list holds keys and the other holds values. This can be done using the zip() function or a loop.

**Counting occurrences of characters in a string using dictionaries.**

A dictionary can be used to count how many times each character appears in a string. The key is the character, and the value is the count.

**Lab:**

**Write a Python program to update a value in a dictionary.**

# Original dictionary

student = {

"name": "Arjun",

"age": 21,

"course": "B.E IT"

}

# Display original value

print("Original Dictionary:")

print(student)

# Update value

student["age"] = 22 # Updating age from 21 to 22

# Display updated dictionary

print("\nUpdated Dictionary:")

print(student)

**Write a Python program to merge two lists into one dictionary using a loop.**

# Two lists

keys = ["id", "name", "subject"]

values = [101, "Arjun", "Python"]

# Merging using loop

merged\_dict = {}

for i in range(len(keys)):

merged\_dict[keys[i]] = values[i]

# Display merged dictionary

print("\nMerged Dictionary:")

print(merged\_dict)

**Practical Examples:**

1. **Write a Python program to update a value at a particular key in a dictionary.**

# Sample dictionary

student = {

"name": "Arjun",

"age": 20,

"course": "Python"

}

# Update the value of the 'age' key

student["age"] = 21

# Display updated dictionary

print("Updated Dictionary:")

print(student)

**16) Write a Python program to separate keys and values from a dictionary using**

**keys() and values() methods.**

# Sample dictionary

person = {

"name": "Alice",

"city": "New York",

"profession": "Engineer"

}

# Get keys and values separately

keys = list(person.keys())

values = list(person.values())

# Display the results

print("Keys:", keys)

print("Values:", values)

**17) Write a Python program to convert two lists into one dictionary using a for loop.**

# Two lists

keys = ["id", "name", "course"]

values = [1, "Arjun", "Python"]

# Merge into dictionary using a loop

result = {}

for i in range(len(keys)):

result[keys[i]] = values[i]

# Display result

print("Merged Dictionary:")

print(result)

1. **Write a Python program to count how many times each character appears in a string.**

# Input string

text = "programming"

# Dictionary to store character counts

char\_count = {}

for char in text:

char\_count[char] = char\_count.get(char, 0) + 1

# Display character counts

print("Character Frequencies:")

for char, count in char\_count.items():

print(f"{char}: {count}")

**8. Functions**

**Theory:**

**Defining functions in Python.**

**Theory:**  
A function is a reusable block of code that performs a specific task. It is defined using the def keyword.

**Syntax:**

def function\_name(parameters):

# code block

return result

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

#### ****a) Function with No Parameters and No Return Value****

def say\_hello():

print("Hello!")

#### ****b) Function with Parameters and No Return Value****

def greet\_user(name):

print(f"Hello, {name}!")

#### ****c) Function with Parameters and Return Value****

def add(a, b):

return a + b

#### ****d) Function with No Parameters but Returns a Value****

def get\_pi():

return 3.14159

**Anonymous functions (lambda functions).**

Lambda functions are small, one-line anonymous functions used for simple operations. They don’t use def or a name.

**Syntax:**

lambda arguments: expression

**Lab:**

**Write a Python program to create a function that takes a string as input and prints it.**

# Function to print a given string

def print\_message(message):

print("You entered:", message)

# Example usage

user\_input = input("Enter a message: ")

print\_message(user\_input)

**Write a Python program to create a calculator using functions.**

# Define functions for calculator operations

def add(a, b):

return a + b

def subtract(a, b):

return a - b

def multiply(a, b):

return a \* b

def divide(a, b):

if b != 0:

return a / b

else:

return "Error: Division by zero"

# Calculator menu

print("Simple Calculator")

print("1. Add")

print("2. Subtract")

print("3. Multiply")

print("4. Divide")

# Get user choice and numbers

choice = input("Enter choice (1-4): ")

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

# Perform operation

if choice == '1':

print("Result:", add(num1, num2))

elif choice == '2':

print("Result:", subtract(num1, num2))

elif choice == '3':

print("Result:", multiply(num1, num2))

elif choice == '4':

print("Result:", divide(num1, num2))

else:

print("Invalid choice")

**Practical Examples:**

1. **Write a Python program to print a string using a function.**

# Function to print a string

def print\_string():

print("Hello, this is a function printing a string!")

# Call the function

print\_string()

1. **Write a Python program to create a parameterized function that takes two arguments and prints their sum.**

# Function with two parameters

def print\_sum(a, b):

print("The sum is:", a + b)

# Call the function with arguments

print\_sum(10, 5)

1. **Write a Python program to create a lambda function with one expression.**

# Lambda function to square a number

square = lambda x: x \* x

# Call the lambda function

print("Square of 4 is:", square(4))

1. **Write a Python program to create a lambda function with two expressions.**

# Lambda function to add two numbers

add = lambda a, b: a + b

# Call the lambda function

print("Sum is:", add(6, 7)**)**

**9. Modules**

**Theory:**

**Introduction to Python modules and importing modules.**

A module in Python is a file containing Python code (functions, classes, variables) that can be reused in other programs. Modules help in organizing code and promoting reuse.

**Types of Modules:**

**Standard Library Modules** (built-in)

**Third-party Modules** (need installation)

**User-defined (Custom) Modules**

**Importing Modules:**

import module\_name

**You can also import specific parts:**

from module\_name import function\_name

**Standard library modules: math, random.**

#### 1).math Module:

Used for mathematical operations.

#### 2).random Module:

Used to generate random numbers.

**Creating custom modules.**

**Step 1: Create a file named** my\_module.py

# my\_module.py

def greet(name):

print(f"Hello, {name}!")

def add(a, b):

return a + b

**Step 2: Use the custom module in another Python file**

# main.py

import my\_module

my\_module.greet("Arjun")print("Sum is:", my\_module.add(10, 5))

**Lab:**

**Write a Python program to import the math module and use functions like sqrt(), ceil(), floor().**

import math

num = 9.7

# Square root

sqrt\_value = math.sqrt(num)

# Ceiling value (smallest integer greater than or equal to num)

ceil\_value = math.ceil(num)

# Floor value (largest integer less than or equal to num)

floor\_value = math.floor(num)

print(f"Number: {num}")

print(f"Square root: {sqrt\_value}")

print(f"Ceiling value: {ceil\_value}")

print(f"Floor value: {floor\_value}")

**Write a Python program to generate random numbers using the random module.**

import random

# Generate a random float between 0 and 1

random\_float = random.random()

# Generate a random integer between 1 and 100

random\_int = random.randint(1, 100)

# Pick a random element from a list

colors = ["red", "blue", "green", "yellow"]

random\_color = random.choice(colors)

print(f"Random float between 0 and 1: {random\_float}")

print(f"Random integer between 1 and 100: {random\_int}")

print(f"Randomly selected color: {random\_color}")

**Practical Examples:**

1. **Write a Python program to demonstrate the use of functions from the math module.**

import math

num = 16.4

print("Number:", num)

print("Square root:", math.sqrt(num)) # Square root

print("Ceiling value:", math.ceil(num)) # Smallest integer >= num

print("Floor value:", math.floor(num)) # Largest integer <= num

print("Factorial of 5:", math.factorial(5)) # Factorial of 5

print("Value of pi:", math.pi) # Pi constant

1. **Write a Python program to generate random numbers between 1 and 100 using the random module.**

import random

# Generate and print 5 random integers between 1 and 100

print("Random numbers between 1 and 100:")

for \_ in range(5):

print(random.randint(1, **100))**