Name:- Limbasiya Vishvash B.

Module 2 : Python – Collections, functions and Modules

Theory:-

Q.1:- Understanding how to create and access elements in a list.

Ans:-

my\_list = []

numbers = [10, 20, 30, 40]

Q.2:- Indexing in lists (positive and negative indexing).

Ans:-

**1. Positive Indexing**

* **Starts from 0** and goes up to len(list) - 1.

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

print(my\_list[0]) # Output: 10 (first element)

print(my\_list[2])

**2. Negative Indexing**

* **Starts from -1** for the **last element**, then -2 for the second-last, and so on.

print(my\_list[-1]) # Output: 50 (last element)

print(my\_list[-3])

Q.3:- Slicing a list: accessing a range of elements.

Ans:-

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

slice1 = my\_list[1:4]

slice2 = my\_list[:3]

slice3 = my\_list[2:]

slice4 = my\_list[0:6:2]

Q.4:- Common list operations: concatenation, repetition, membership.

Ans:-

**1. Concatenation (+)**

list1 = [1, 2, 3]

list2 = [4, 5, 6]

combined = list1 + list2

**2. Repetition (\*)**

my\_list = ["a", "b"]

repeated = my\_list \* 3

**3. Membership (in, not in)**

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

print("banana" in fruits) # True

print("mango" not in fruits)

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

Ans:-

1. **append(item)**

numbers = [1, 2, 3]

numbers.append(4)

**2.insert(index, item)**

numbers = [1, 2, 4]

numbers.insert(2, 3)

**3.remove(item)**

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

fruits.remove("banana")

**4.pop(index)**

numbers = [10, 20, 30, 40]

popped\_item = numbers.pop(1)

Q.6:- Iterating over a list using loops.

Ans:-

1. **Using a for Loop**

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

for fruit in fruits:

print(fruit)

1. **Using a for Loop with range()**

for i in range(len(fruits)):

print(fruits[i])

1. **Using a while Loop**

i = 0

while i < len(fruits):

print(fruits[i])

i += 1

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

Ans:-

**list.sort()**

* Sorts the list **in place** (modifies the original list).

numbers = [3, 1, 4, 2]

numbers.sort()

**sorted(list)**

* Returns a **new list** that is sorted, leaving the original list unchanged.

numbers = [3, 1, 4, 2]

new\_list = sorted(numbers)

**list.reverse()**

* Reverses the elements **in place** (does not sort, just reverses order).

numbers = [1, 2, 3, 4]

numbers.reverse()

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

Ans:-

1.Addition:-

my\_list.append("new\_item")

**insert(index, item)** inserts item at index.

my\_list.insert(2, "inserted\_item")

2.Deletion:-

my\_list.remove("some\_item")

**pop(index)** removes and **returns** the element at index. If no index is provided, it removes the **last element**.

popped\_item = my\_list.pop()

popped\_item = my\_list.pop(1)

3.Updating:-

my\_list[0] = "updated\_value"

**Slice assignment**:

my\_list[1:3] = ["new1", "new2"]

4.Slicing:-

sliced = my\_list[1:4] reversed\_list = my\_list[::-1]

Q.9:- Introduction to tuples, immutability.

Ans:-

: A **tuple** is an **ordered collection** of items, similar to a list.

: However, **tuples are immutable**, meaning their elements **cannot** be changed once created.

Q.10:- Creating and accessing elements in a tuple.

Ans:-

1. **Creating a Tuple**

# Using parentheses

my\_tuple = (10, 20, 30)

# Without parentheses (tuple packing)

another\_tuple = 1, 2, 3

# Single-element tuple (note the trailing comma)

single = (5,)

2. **Accessing Elements**

* **Indexing** works like lists, starting at **0**.

print(my\_tuple[0]) # 10

print(my\_tuple[-1]) # 30 (last element)

3.**Slicing**

* You can slice tuples the same way as lists.

print(my\_tuple[0:2]) # (10, 20)

print(my\_tuple[::-1])

Q.11:- Basic operations with tuples: concatenation, repetition, membership.

Ans:-

1. **Concatenation** (+)

* **Join** two tuples into a single tuple.

t1 = (1, 2)

t2 = (3, 4)

result = t1 + t2 # (1, 2, 3, 4)

2.**Repetition** (\*)

* **Repeat** a tuple multiple times.

t = ("A", "B")

repeated = t \* 3 # ("A", "B", "A", "B", "A", "B")

3.**Membership** (in, not in)

* Check if an element **exists** in a tuple.

t = (1, 2, 3)

print(2 in t) # True

print(5 not in t)

Q.12:- Accessing tuple elements using positive and negative indexing.

Ans:-

**1. Positive Indexing**

* Starts from **0** on the left.

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

print(my\_tuple[0]) # "apple"

print(my\_tuple[2]) # "cherry"

**2. Negative Indexing**

* Starts from **-1** on the right (last element).

print(my\_tuple[-1]) # "cherry"

print(my\_tuple[-2]

Q.13:- Slicing a tuple to access ranges of elements.

Ans:-

my\_tuple = (10, 20, 30, 40, 50)

# From index 1 to 3 (exclusive of 4)

slice1 = my\_tuple[1:4] # (20, 30, 40)

# From start to index 2 (exclusive)

slice2 = my\_tuple[:3] # (10, 20, 30)

# From index 2 to the end

slice3 = my\_tuple[2:] # (30, 40, 50)

# Using a step of 2

slice4 = my\_tuple[0:5:2

Q.14:- Introduction to dictionaries: key-value pairs.

Ans:-

person = {"name": "Alice", "age": 25}

print(person["name"])

Q.15:- Accessing, adding, updating, and deleting dictionary elements.

Ans:-

1. **Accessing**

* Use **dict[key]** to retrieve the value.

person = {"name": "Alice", "age": 25}

print(person["name"]) # "Alice"

2. **Adding**

* Assign a **new key** with a **value**.

person["city"] = "New York"

3. **Updating**

* Assign a **new value** to an **existing key**.

person["age"] = 26

4. **Deleting**

* Use **del dict[key]** or **dict.pop(key)**.

del person["city"]

# or

removed\_value = person.pop("age")

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

Ans:-

1.**keys()**

* Returns a **view** of all the **keys** in the dictionary.

my\_dict = {"name": "Alice", "age": 25}

print(my\_dict.keys()) # dict\_keys(['name', 'age'])

2.**values()**

* Returns a **view** of all the **values** in the dictionary.

print(my\_dict.values()) # dict\_values(['Alice', 25])

3.**items()**

* Returns a **view** of all the **(key, value) pairs** as tuples.

print(my\_dict.items())

Q.17:- Iterating over a dictionary using loops.

Ans:-

1. **By Keys**

my\_dict = {"name": "Alice", "age": 25}

for key in my\_dict:

print(key, my\_dict[key])

1. **By Items (Key-Value Pairs)**

for key, value in my\_dict.items():

print(key, value)

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

Ans:-

1.**Using zip() and dict()**

keys = ["name", "age", "city"]

values = ["Alice", 25, "New York"]

merged\_dict = dict(zip(keys, values))

print(merged\_dict) # {'name': 'Alice', 'age': 25, 'city': 'New York'}

2. **Using a for Loop**

keys = ["name", "age", "city"]

values = ["Alice", 25, "New York"]

merged\_dict = {}

for i in range(len(keys)):

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

print(merged\_dict)

Q.19:- Counting occurrences of characters in a string using dictionaries.

Ans:-

text = "hello world"

char\_count = {}

for char in text:

if char in char\_count:

char\_count[char] += 1

else:

char\_count[char] = 1

print(char\_count)

Q.20:- Defining functions in Python.

Ans:-

def function\_name(parameters):

"""Docstring (optional): Describes the function."""

# Function body

return value # (optional)

**Example:**

def add\_numbers(a, b):

return a + b

result = add\_numbers(5, 3)

print(result)

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

Ans:-

1. **Without Parameters, Without Return Value**  
   👉 Performs a task but doesn't take input or return anything.

def greet():

print("Hello!")

greet()

1. **Without Parameters, With Return Value**

def get\_number():

return 10

print(get\_number())

1. **With Parameters, Without Return Value**

def greet(name):

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

greet("Alice")

1. **With Parameters, With Return Value**

def add(a, b):

return a + b

print(add(5, 3))

Q.22:- Anonymous functions (lambda functions).

Ans:-

**Syntax:**

lambda arguments: expression

**Example Usage:**

1. **Basic Lambda Function (Addition)**

add = lambda a, b: a + b

print(add(3, 5)) # Output: 8

1. **Lambda Inside map()**

numbers = [1, 2, 3, 4]

squared = list(map(lambda x: x\*\*2, numbers))

print(squared) # Output: [1, 4, 9, 16]

1. **Lambda Inside filter()**

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

even\_nums = list(filter(lambda x: x % 2 == 0, nums))

print(even\_nums) # Output: [2, 4]

1. **Lambda Inside sorted()**

names = ["Alice", "Bob", "Charlie"]

sorted\_names = sorted(names, key=lambda x: len(x))

print(sorted\_names)

Q.23:- Introduction to Python modules and importing modules.

Ans:-

**1. Importing a Module**

Use the import keyword to import a module.

**Example: Importing the math module**

import math

print(math.sqrt(16)) # Output: 4.0

**2. Importing Specific Functions**

You can import specific functions using from module import function.

from math import sqrt

print(sqrt(25)) # Output: 5.0

**3. Importing with an Alias**

Use as to give a module an alias.

import numpy as np

print(np.pi) # Output: 3.141592653589793

**4. Importing All Functions (Not Recommended)**

from math import \*

print(factorial(5)) # Output: 120

**5. Creating a Custom Module**

Save the following as my\_module.py:

def greet(name):

return f"Hello, {name}!"

Then, import and use it:

import my\_module

print(my\_module.greet("Alice"))

Q.24:- Standard library modules: math, random.

Ans:-

**1. math Module (Mathematical Functions)**

To use, first import:

import math

**Common Functions:**

* math.sqrt(x): Square root

print(math.sqrt(16)) # Output: 4.0

* math.factorial(x): Factorial

print(math.factorial(5)) # Output: 120

* math.pow(x, y): Exponentiation (x^y)

print(math.pow(2, 3)) # Output: 8.0

* math.pi: Returns the value of π

print(math.pi)

**2. random Module (Generating Random Numbers)**

To use, first import:

import random

**Common Functions:**

* random.randint(a, b): Random integer between a and b

print(random.randint(1, 10)) # Random number between 1 and 10

* random.random(): Random float between 0.0 and 1.0

print(random.random()) # Output: 0.3456 (example)

* random.choice(sequence): Randomly selects an item from a list

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

print(random.choice(colors)) # Output: 'blue' (example)

* random.shuffle(sequence): Shuffles a list randomly

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

random.shuffle(numbers)

print(numbers)