Python – Collections, functions and Modules

1) Accessing List:

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

Ans-

1. Creating a List

A list in Python is created using square brackets [], and can store multiple values (of any data type).

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

numbers = [10, 20, 30, 40]

mixed = [1, "hello", 3.14, True]
```

2. Accessing Elements

Elements in a list are accessed using **indexing**. Indexing starts from 0.

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

print(numbers[2])  # Output: 30

print(mixed[1])    # Output: hello
```

3. Negative Indexing

Python supports negative indexing. -1 refers to the last element, -2 to the second last, and so on.

```
print(fruits[-1])  # Output: cherry

print(numbers[-2]) # Output: 30
```

4. Accessing with a Loop

You can also loop through a list to access each element.

```
for item in fruits:
   print(item)
```

Q) Indexing in lists (positive and negative indexing)

Ans-

Indexing in Lists

Indexing is the process of accessing individual elements in a list using their position (index).

Positive Indexing

- Starts from 0 and moves forward.
- First element has index 0, second is 1, and so on.

```
items = ['a', 'b', 'c', 'd']


print(items[0])  # Output: a

print(items[1])  # Output: b

print(items[3])  # Output: d
```

Negative Indexing

- Starts from -1 and moves backward.
- -1 is the last element, -2 is second last, etc.

```
items = ['a', 'b', 'c', 'd']


print(items[-1])  # Output: d
print(items[-2])  # Output: c
print(items[-4])  # Output: a
```

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

Ans-

Slicing a List (Short and Simple)

Slicing means getting a part of the list using a range of indexes.

Syntax:

```
list[start:stop]
```

start: where to begin (included)

stop: where to end (not included)

Ex-

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


print(numbers[1:4])  # [20, 30, 40]
print(numbers[:3])   # [10, 20, 30]
print(numbers[2:])   # [30, 40, 50]
```

With step:

```
print(numbers[::2])  # [10, 30, 50]
print(numbers[::-1])  # [50, 40, 30, 20, 10]
```

2. List Operations:

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

Ans-

1. Concatenation (+)

Joins two lists.

```
a = [1, 2]

b = [3, 4]

print(a + b)  # [1, 2, 3, 4]
```

2. **Repetition (*)**

Repeats the list multiple times.

```
a = [1, 2]

print(a * 3)  # [1, 2, 1, 2, 1, 2]
```

3. Membership (in, not in)

Checks if an item exists in the list.

```
a = [10, 20, 30]

print(20 in a)      # True

print(50 not in a)  # True
```

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

Ans –

1. append()

Adds an item at the end of the list.

```
a = [1, 2]
a.append(3)
print(a)  # [1, 2, 3]
```

2. **insert()**

Inserts an item at a specific index.

```
a = [1, 3]
a.insert(1, 2)
print(a)  # [1, 2, 3]
```

3. remove()

Removes the first occurrence of a value.

```
a = [1, 2, 3]
a.remove(2)
print(a)  # [1, 3]
```

4. pop()

Removes and returns an item. If index not given, removes last item.

```
a = [1, 2, 3]
a.pop()
```

```
print(a)  # [1, 2]


a.pop(0)
print(a)  # [2]
```

3. Working with Lists

Q) Iterating over a list using loops.

Ans –

1. Using for loop

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


for fruit in fruits:
    print(fruit)
```

2. **Using for loop with index**

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


for i in range(len(fruits)):
    print(fruits[i])
```

3. **Using while loop**

```
fruits = ['apple', 'banana', 'cherry']
i = 0
```

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

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

Ans –

1. sort()

Sorts the list in place (modifies original list).

```
a = [3, 1, 2]
a.sort()
print(a)  # [1, 2, 3]
```

2. sorted()

Returns a new sorted list (original list remains unchanged).

```
a = [3, 1, 2]
b = sorted(a)
print(b)  # [1, 2, 3]
print(a)  # [3, 1, 2]
```

3. reverse()

Reverses the list in place.

```
a = [1, 2, 3]
a.reverse()
print(a)  # [3, 2, 1]
```

Q) Basic list manipulations: addition, deletion, updating, and slicing

Ans –

1. Addition

- **append()** – adds at end
- **insert()** – adds at specific index

```
a = [1, 2]
a.append(3)      # [1, 2, 3]
a.insert(1, 5)   # [1, 5, 2, 3]
```

2. Deletion

- **remove(value)** – removes first occurrence
- **pop(index)** – removes by index (default: last)
- **del** – deletes by index

```
a = [1, 2, 3, 4]
a.remove(2)    # [1, 3, 4]
a.pop()        # [1, 3]
del a[0]       # [3]
```

3. **Updating**

```
a = [10, 20, 30]
a[1] = 25       # [10, 25, 30]
```

4. **Slicing**

```
a = [10, 20, 30, 40, 50]
```

```
print(a[1:4])    # [20, 30, 40]
```

4. Tuple:

Q) Introduction to tuples, immutability.

Ans –

- **Tuple** is an ordered, fixed collection.
- Created using **()**
  Example: t = (10, 20, 30)
- **Immutable**: You **cannot change**, add, or delete elements after creation.

  Example: t[0] = 5 → Error.

Q) Creating and accessing elements in a tuple.

Ans –

- **Create tuple** using ()
  Example: t = (10, 20, 30)
- **Access elements** using index

```
# Creating a tuple

t = (10, 20, 30, 40)


# Accessing elements

print("First element:", t[0])

print("Last element:", t[-1])

print("Middle elements:", t[1:3])
```

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

Ans –

```
# Tuples
a = (1, 2)
b = (3, 4)

# Concatenation
c = a + b
print("Concatenation:", c)

# Repetition
d = a * 2
print("Repetition:", d)

# Membership
print(2 in a)      # True
print(5 not in a)  # True
```

5. Accessing Tuples:

Q) Accessing tuple elements using positive and negative indexing.

Ans –

```
# Tuple
t = (10, 20, 30, 40, 50)

# Positive indexing
print("First element:", t[0])
print("Third element:", t[2])

# Negative indexing
print("Last element:", t[-1])
print("Second last element:", t[-2])
```

Q) Slicing a tuple to access ranges of elements.

Ans –

```
# Tuple
t = (10, 20, 30, 40, 50)

# Slicing
print("Elements from index 1 to 3:", t[1:4])
print("First three elements:", t[:3])
print("Last two elements:", t[-2:])
```

6. Dictionaries:

Q) Introduction to dictionaries: key-value pairs.

Ans –

- A **dictionary** in Python stores data in **key-value pairs**.
- Each key maps to a value:
  key: value
- Keys must be **unique** and **immutable** (like strings, numbers, tuples).
- Values can be of any data type.

```
dictionary = {
    "key1": value1,
    "key2": value2
}
```

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

Ans –

- **Access**: Use the key to get the value.
- **Add**: Assign a new key-value pair.
- **Update**: Change value of an existing key.
- **Delete**: Use del or pop() to remove key-value pairs.

```
# Dictionary
student = {"name": "John", "age": 20}


# Accessing
print("Name:", student["name"])
```

```
# Adding
student["grade"] = "A"

# Updating
student["age"] = 21

# Deleting
del student["grade"]  # or use student.pop("grade")

print("Updated dictionary:", student)
```

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

Ans –

- keys() → Returns all keys in the dictionary.
- values() → Returns all values.
- items() → Returns all key-value pairs as tuples.

```
student = {"name": "John", "age": 20, "grade": "A"}

# Get keys
print("Keys:", student.keys())

# Get values
```

```
print("Values:", student.values())

# Get key-value pairs
print("Items:", student.items())
```

7. Working with Dictionaries:

Q) Iterating over a dictionary using loops.

Ans –

You can use loops to:

- Access **keys** directly
- Use .items() to get **key-value pairs**

```
student = {"name": "John", "age": 20, "grade": "A"}

# Loop through keys
for key in student:
    print(key, "=", student[key])

# Loop through key-value pairs
for key, value in student.items():
    print(key, "->", value)
```

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

Ans –

- Use zip() to pair elements from two lists.
- Or use a loop to assign keys and values manually.

```
keys = ["name", "age", "grade"]
values = ["John", 20, "A"]

# Using zip()
merged_dict = dict(zip(keys, values))
print(merged_dict)
```

Code Using Loop:

```
keys = ["name", "age", "grade"]

values = ["John", 20, "A"]


# Using loop

merged_dict = {}

for i in range(len(keys)):

    merged_dict[keys[i]] = values[i]


print(merged_dict)
```

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

Ans –

- Loop through each character.
- Use a dictionary to store and count each character.

```
text = "hello"
count = {}

for char in text:
    if char in count:
        count[char] += 1
    else:
        count[char] = 1

print(count)
```

8. Functions:

Q) Defining functions in Python.

Ans –

- A function is a block of code that runs when called.
- Defined using the def keyword.
- Can take parameters and return values.

Syntax:

```
def function_name(parameters):
        # code block
        return value
```

```
def greet(name):
    return "Hello, " + name
print(greet("Alice"))
```

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

Ans –

1. Without Parameters & Without Return Value

```
def say_hello():
    print("Hello")


say_hello()
```

2. With Parameters & Without Return Value

```
def greet(name):
    print("Hello", name)


greet("Alice")
```

3. Without Parameters & With Return Value

```
def get_message():
    return "Welcome"


msg = get_message()
```

```
print(msg)
```

4. With Parameters & With Return Value

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


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

Q) Anonymous functions (lambda functions).

Ans –

- A **lambda function** is a small anonymous function.
- Defined using the lambda keyword.
- Can have any number of arguments, but only one expression.

Syntax:

```
lambda arguments: expression


# Lambda function to add two numbers
add = lambda x, y: x + y


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

9. Modules:

Q) Introduction to Python modules and importing modules.

Ans –

- A module is a file with Python code (functions, variables, classes).
- Python has built-in modules (like math, random), and you can create your own.
- Use the import statement to use a module in your program.

Ways to Import Modules:

1. Import whole module

   ```
   import math

   print(math.sqrt(25))
   ```

2. Import specific function

   ```
   from math import sqrt
   print(sqrt(25))
   ```

3. Import with alias

   ```
   import math as m
   print(m.sqrt(25))
   ```

Q) Standard library modules: math, random.

Ans –

Standard Library Modules: math, random

1. math Module

Used for mathematical operations.

Example:

```
import math


print(math.sqrt(16))      # Square root → 4.0
print(math.pow(2, 3))     # 2^3 → 8.0
print(math.factorial(5))  # 5! → 120
print(math.pi)            # Value of pi
```

2. random Module

Used to generate random numbers.

Example:

```
import random


print(random.randint(1, 10))       # Random int between 1 and 10
print(random.choice(['a', 'b', 'c'])) # Random element from list
print(random.random())             # Random float between 0 and 1
```

Q) Creating custom modules.

Ans –

- A **custom module** is simply a Python file (.py) that contains functions, variables, or classes.
- You can **reuse** code by importing this file in other Python scripts.

Steps to Create and Use a Custom Module

1. Create a Python file named mymodule.py:

```
# mymodule.py
def greet(name):
    return f"Hello, {name}!"


pi = 3.141
```

2) Use it in another Python file:

```
# main.py
import mymodule

print(mymodule.greet("Mitesh"))  # Output: Hello, Mitesh!
print(mymodule.pi)               # Output: 3.141
```