1. Accessing List

1. Creating a List

A **list** in Python is a collection of items enclosed in square brackets `[]`, separated by commas.
Example:

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

2. Accessing Elements – Indexing

Positive indexing: Counting starts from `0` (left to right).

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

print(fruits[2])  # cherry
```

Negative indexing: Counting starts from `-1` (right to left).

```
print(fruits[-1]) # mango

print(fruits[-2]) # cherry
```

3. Slicing a List (Accessing a Range)

Slicing uses the format:

```
list[start:end]
```

`start` → index to start from (inclusive)

`end` → index to stop before (exclusive)

```
print(fruits[1:3])   # ['banana', 'cherry']
print(fruits[:2])    # ['apple', 'banana']  (start from beginning)
print(fruits[2:])    # ['cherry', 'mango']  (go till end)
print(fruits[-3:-1]) # ['banana', 'cherry']
```

2. List Operations

Common List Operations

1. Concatenation – Joining two or more lists together using the `+` operator.
2. Repetition – Repeating the elements of a list multiple times using the `*` operator.
3. Membership – Checking whether a specific value exists in the list using the `in` or `not in` operators.

Common List Methods

- append() – Adds a single element at the end of the list.
- insert() – Inserts an element at a specific position in the list.
- remove() – Removes the first occurrence of a specified element from the list.
- pop() – Removes and returns an element from the list. By default, it removes the last element, but a specific index can be provided.

3. Working with Lists

Iterating Over a List Using Loops

Looping through each element of a list can be done using `for` loops or `while` loops to perform actions on each item.

Sorting and Reversing a List

- **sort()** – Sorts the list in place (modifies the original list).
- **sorted()** – Returns a new sorted list without changing the original list.
- **reverse()** – Reverses the order of elements in the list in place.

Basic List Manipulations

- **Addition** – Adding new elements using `append()`, `insert()`, or list concatenation (`+`).
- **Deletion** – Removing elements using `remove()`, `pop()`, or `del`.
- **Updating** – Changing the value of an element by assigning a new value to a specific index.
- **Slicing** – Accessing or modifying a portion of the list using the slice notation `[start:end]`.

4. Tuple

Introduction to Tuples and Immutability

A **tuple** is an ordered collection of elements, similar to a list, but it is **immutable**, meaning its elements cannot be changed, added, or removed after creation. Tuples are defined using parentheses `()` instead of square brackets `[]`.

Creating and Accessing Elements in a Tuple

- **Creating** – Tuples can be created by placing elements inside parentheses, separated by commas.

- **Accessing** – Elements can be accessed using positive or negative indexing, and ranges can be accessed using slicing.

Basic Operations with Tuples

- **Concatenation** – Joining two or more tuples together using the `+` operator.

- **Repetition** – Repeating the elements of a tuple multiple times using the `*` operator.

- **Membership** – Checking whether a specific value exists in the tuple using the `in` or `not in` operators.

5. Accessing Tuples

Accessing Tuple Elements Using Positive and Negative Indexing

- **Positive indexing** – Indexing starts from `0` for the first element and increases by 1 from left to right.
- **Negative indexing** – Indexing starts from `-1` for the last element and decreases by 1 from right to left.

Slicing a Tuple to Access Ranges of Elements

Slicing uses the notation `[start:end]` where:

- **start** → index position to begin (inclusive)
- **end** → index position to stop before (exclusive)
- Omitting **start** begins from the first element, and omitting **end** continues to the last element.

6. Dictionaries

Introduction to Dictionaries: Key-Value Pairs

A **dictionary** is an unordered, mutable collection of data in Python, where each item is stored as a **key-value pair**. Keys must be unique and immutable, while values can be of any data type. Dictionaries are defined using curly braces `{}`.

Accessing, Adding, Updating, and Deleting Dictionary Elements

- **Accessing** – Retrieve values by specifying their key inside square brackets `[]` or using the `get()` method.
- **Adding** – Add a new key-value pair by assigning a value to a new key.
- **Updating** – Modify the value of an existing key by reassigning it.
- **Deleting** – Remove a key-value pair using the `del` statement or the `pop()` method.

Dictionary Methods

- **keys()** – Returns a view object containing all keys in the dictionary.
- **values()** – Returns a view object containing all values in the dictionary.
- **items()** – Returns a view object containing all key-value pairs as tuples.

7. Working with Dictionaries

Iterating Over a Dictionary Using Loops

- Loop through **keys** directly by iterating over the dictionary.
- Loop through **values** using `values()`.
- Loop through **key-value pairs** using `items()`.

Merging Two Lists into a Dictionary

- Use a loop to pair each element from the first list (keys) with the corresponding element from the second list (values).
- Use the `zip()` function to combine both lists into key-value pairs and then convert them into a dictionary.

Counting Occurrences of Characters in a String Using Dictionaries

- Create an empty dictionary.
- Loop through each character in the string.
- If the character is already a key, increment its count; otherwise, add it to the dictionary with count `1`.

8. Functions

Defining Functions in Python

A function is a reusable block of code that performs a specific task. Functions are defined using the `def` keyword, followed by the function name, parentheses (with optional parameters), and a colon. The function body is indented.

Different Types of Functions

1. **Without Parameters, Without Return Value** – Performs a task without taking inputs and does not return a value.
2. **Without Parameters, With Return Value** – Performs a task without inputs but returns a result.
3. **With Parameters, Without Return Value** – Takes inputs but does not return a value.

4. **With Parameters, With Return Value** – Takes inputs and returns a result.

Anonymous Functions (Lambda Functions)

A **lambda function** is a small, one-line anonymous function defined using the `lambda` keyword. It can take any number of arguments but has only one expression, which is automatically returned. Lambda functions are often used for short, temporary operations.

9. Modules

Introduction to Python Modules and Importing Modules

A **module** in Python is a file containing Python code (functions, classes, or variables) that can be reused in other programs. Modules help in organizing code into separate files for better readability and maintainability.
Modules can be imported using the `import` statement or specific items can be imported using `from ... import ...`.

Standard Library Modules: math, random

- **math** – Provides mathematical functions like square root, power, trigonometry, logarithms, and constants such as π.
- **random** – Provides functions to generate random numbers, choose random elements, shuffle sequences, and more.

Creating Custom Modules

- Write Python functions, classes, or variables in a `.py` file.
- Save the file with a module name (e.g., `mymodule.py`).
- Import it into another Python file using `import mymodule` or `from mymodule import function_name`.