1. Accessing Lists

Creating a List

In Python, a list is an ordered collection of elements placed inside square brackets `[]` and separated by commas.
Example: a list named `colors` might store values like `"red"`, `"blue"`, `"green"`, `"yellow"`.

Accessing Elements – Indexing

- **Positive Indexing** – counting begins from 0 (from left to right). For instance, index 0 gives `"red"`, and index 2 gives `"green"`.
- **Negative Indexing** – counting begins from -1 (from right to left). For instance, index -1 gives `"yellow"`, and index -2 gives `"green"`.

Slicing a List

Slicing is written in the form `list[start:end]` where the start index is included, but the end index is excluded.

- If the start is left empty, slicing begins from the first element.
- If the end is left empty, slicing continues till the last element.
  For example, from `colors`, slice `[1:3]` gives `"blue", "green"`, slice `[:2]` gives `"red", "blue"`, and slice `[2:]` gives `"green", "yellow"`.

2. List Operations

Common Operations

- **Concatenation** – joining multiple lists with +.
- **Repetition** – repeating elements using `*`.

- **Membership** – checking if an element exists using `in` or `not in`.

Frequently Used Methods

- **append()** – adds an element to the end.
- **insert()** – adds an element at a given index.
- **remove()** – deletes the first occurrence of a specified value.
- **pop()** – deletes an element by index; without index, it removes the last element.

3. Working with Lists

Iterating Over a List

You can go through each element of a list using a **for loop** or a **while loop**.
This allows you to perform specific actions on each item one by one.

Sorting and Reversing a List

- **sort()** → Sorts the list in place, changing the original list.
- **sorted()** → Returns a new sorted list, leaving the original list unchanged.
- **reverse()** → Reverses the order of the list in place.

Basic List Operations

1. Adding Elements

- `append()` → Adds a new element at the end of the list.
- `insert()` → Inserts a new element at a specific index.
- `+` (concatenation) → Joins two lists to create a new one.

2. Removing Elements

- `remove()` → Removes the first matching element by value.
- `pop()` → Removes and returns the element at a specific index.
- `del` → Deletes an element (or a slice) by index.

3. Updating Elements

- Assign a new value to a specific index to update that element.

4. Slicing a List

- Use `[start:end]` notation to access or modify a portion of the list.

4. Tuples

What is a Tuple?
A tuple is a sequence of elements arranged in order, much like a list.
However, unlike lists, tuples are **immutable**—once created, you can't modify, add, or delete their elements.
They are written using **round brackets** `()` instead of square brackets `[]`.

Creating and Accessing Tuples

- **Create** → Place items inside `()` and separate them with commas.
- **Access** → Use indexing (`0`, `1`, etc.) for forward access, negative indexing (`-1`, `-2`, etc.) for reverse access, and slicing (`start:end`) to get a portion of the tuple.

Common Tuple Operations

- **Join Tuples** → Use the `+` operator to merge two or more tuples.
- **Repeat Elements** → Use `*` to repeat the tuple's contents multiple times.
- **Check Existence** → Use `in` or `not in` to verify whether a value is present.

5. Accessing Tuples

Indexing in Tuples

- **Positive Indexing** → Starts from `0` for the first element and increases by `1` moving left to right.
- **Negative Indexing** → Starts from `-1` for the last element and decreases by `1` moving right to left.

Slicing Tuples
You can extract a portion of a tuple using the `[start:end]` format:

- **start** → Index where the slice begins (included in the result).
- **end** → Index where the slice stops (excluded from the result).
- If **start** is left blank → slicing begins from the first element.
- If **end** is left blank → slicing continues to the last element.

6. Dictionaries

A dictionary is an unordered, mutable collection that stores data as **key–value pairs** within curly braces `{}`.
Keys must be unique and immutable, while values can be of any data type.

Accessing values is done by specifying the key inside square brackets or by using the `get()` method.
New key–value pairs can be added simply by assigning a value to a new key, and existing values can be updated by reassigning them.
Keys can be deleted using `del` or `pop()`.

7. Working with Dictionaries

Dictionaries can be looped over to get keys, values, or both keys and values using the `items()` method.
Two lists can be merged into a dictionary by combining them with the `zip()` function and converting the result into a dictionary.
A dictionary can also be used to count the occurrence of characters in a string by storing each character as a key and its count as the value.

8. Functions

A function is a reusable set of instructions created using the `def` keyword followed by a function name, optional parameters in parentheses, and a colon.

Types of Functions:

- No parameters, no return value.
- No parameters, with return value.
- With parameters, no return value.
- With parameters, with return value.

A **lambda function** is a small, one-line, anonymous function defined with the `lambda` keyword, containing only one expression whose value is returned automatically.

9. Modules

A module is a file containing Python definitions, functions, or variables that can be reused in other programs.
Modules can be imported using `import` or by importing specific components with `from ... import ...`.

Examples from the standard library include:

- **math** – provides mathematical functions and constants.
- **random** – generates random numbers and makes random selections.

Custom modules can be created by writing Python code in a `.py` file and then importing it into another script.