CLASS - 3

1. File Handling in Python and File Modes

File handling in Python refers to the process of creating, opening, reading, writing, and closing files. It allows programs to store data permanently and retrieve it when required.

Common File Modes:

- **Read (`r`)** – Opens a file for reading
- **Write (`w`)** – Opens a file for writing and overwrites existing content
- **Append (`a`)** – Adds content to the end of a file
- **Read and Write (`r+`)** – Allows both reading and writing
- **Write and Read (`w+`)** – Creates or overwrites a file for both operations
- **Append and Read (`a+`)** – Appends and allows reading

2. Difference Between `read()`, `readline()`, and `readlines()`

- `read()` reads the entire content of a file at once.
- `readline()` reads one line from the file at a time.
- `readlines()` reads all lines and stores them as a list.

Key Difference:
They differ in how much data is read and how it is returned.

3. Exception Hierarchy in Python

The exception hierarchy in Python is a structured classification of built-in exceptions. All exceptions inherit from the base class `BaseException`, with `Exception` being the most commonly used subclass.

Purpose:

- Organizes different error types

- Allows handling multiple related exceptions efficiently
- Helps in writing robust and error-free programs

4.Difference Between `try-except` and `try-except-finally`

- `try-except` handles exceptions and prevents program termination when an error occurs.
- `try-except-finally` ensures that the code inside the `finally` block executes regardless of whether an exception occurs or not.

Use Case:
The `finally` block is commonly used for resource cleanup such as closing files or releasing memory.

5. Purpose of the `with` Statement

The `with` statement is used for resource management. It ensures that resources such as files or database connections are automatically closed after use, even if an error occurs.

Advantages:

- Prevents resource leaks
- Improves code readability
- Eliminates the need for manual cleanup

6. Iterators in Python

Iterators are objects that allow traversal through a collection of elements one at a time. They maintain their state and remember the last accessed element.

Key Features:

- Used for sequential data access
- Memory efficient
- Commonly used in loops

7. Generators and Their Difference from Functions

Generators are special types of functions that produce values one at a time instead of returning them all at once. They pause execution after yielding a value and resume when requested again.

Difference from Normal Functions:

- Functions return a value and terminate
- Generators yield values and preserve execution state
- Generators are more memory efficient

8. `*args` and `**kwargs`

- `*args` allows a function to accept a variable number of positional arguments.
- `**kwargs` allows a function to accept a variable number of keyword arguments.

Purpose:

- Increases function flexibility
- Useful when the number of inputs is unknown

9. Type Casting

Type casting is the process of converting one data type into another. Python supports both implicit and explicit type casting.

Examples (Conceptual):

- Converting a number stored as text into an integer
- Converting an integer into a floating-point number
- Converting numerical values into strings for display

Importance:

- Ensures data compatibility
- Prevents type-related runtime errors

10. Built-in Functions in Python

Built-in functions are predefined functions provided by Python that perform common operations without requiring additional libraries.

Examples (Any Five):

- `print()` – Displays output

- `len()` – Returns length of an object
- `type()` – Returns data type
- `input()` – Accepts user input
- `range()` – Generates a sequence of numbers