Task 3: Python

1. What is file handling in Python? Name different file modes.

File handling refers to the process of performing operations on a file, such as creating, opening, reading, writing and closing it through a programming interface. It involves managing the data flow between the program and the file system on the storage device, ensuring that data is handled safely and efficiently.

Why do we need File Handling

- To store data permanently, even after the program ends.
- To access external files like .txt, .csv, .json, etc.
- To process large files efficiently without using much memory.
- To automate tasks like reading configs or saving outputs.

➢ **Opening a File**

To open a file, we can use open() function, which requires file-path and mode as arguments.

Syntax:

file = open('filename.txt', 'mode')

filename.txt: name (or path) of the file to be opened.

mode: mode in which you want to open the file (read, write, append, etc.).

➢ **Closing a File**

The file.close() method closes the file and releases the system resources. If the file was opened in write or append mode, closing ensures that all changes are properly saved.

➢ **Reading a File**

Reading a file can be achieved by file.read() which reads the entire content of the file. After reading, it's good practice to close the file to free up system resources.

➢ **Writing a File**

In Python, writing to a file is done using the mode "w". This creates a new file if it doesn't exist, or overwrites the existing file if it does. The write() method is used to add content.

2. What is the difference between read(),readline(), and readlines()?

read()

- Reads entire file content at once

- Returns data as one single string
- Suitable for small files

readline()

- Reads only one line at a time
- Returns a single line as a string
- Used when reading files line by line

readlines()

- Reads all lines at once
- Returns a list of strings, where each element is a line
- Useful when working with multiple lines

3. What is exception hierarchy in Python?

Exception hierarchy in Python refers to the organized structure in which all built-in exceptions are arranged in a parent child relationship. At the top of this hierarchy is the BaseException class, from which all other exceptions are derived. The most commonly used class is Exception, which is a subclass of BaseException and acts as the base for most runtime errors.

This hierarchy allows related exceptions to be grouped together, making error handling more flexible. When a parent exception is caught, all of its child exceptions are also handled automatically. For example, ArithmeticError is a parent class for errors like ZeroDivisionError and OverflowError, while LookupError is the parent for IndexError and KeyError.

4. Difference between try-except and try-except-finally

try-except is used to handle exceptions that may occur during program execution. If an error occurs in the try block, the control is transferred to the except block to handle the exception. If no exception occurs, the except block is skipped.

try-except-finally includes an additional finally block. The code inside the finally block is always executed**,** whether an exception occurs or not. It is mainly used for cleanup operations such as closing files or releasing resources.

5. What is the purpose of the with statement?

The 'with' statement in Python is used for automatic resource management. It ensures that resources like files, database connections, or locks are properly opened and closed. Even if an exception occurs, the with statement guarantees that cleanup code is executed. This makes the code cleaner, safer, and easier to read, and it helps avoid resource leaks.

6. What are iterators in Python?

Iterators in Python are objects that allow you to traverse through elements one at a time**.** An iterator implements two special methods: __iter__() and __next__(). The __iter__() method returns the iterator object itself, and the __next__() method returns the next element in the sequence. When there are no more elements, StopIteration is raised.

Common examples of iterators include lists, tuples, strings, and dictionaries when used in loops. Iterators are memory-efficient because they generate values only when needed**,** rather than storing all values at once.

7. What are Generators and How Are They Different from Functions?

Generators are special types of functions in Python that return values one at a time using the yield keyword instead of return. Unlike normal functions, generators do not store all values in memory; they generate values only when needed, making them memory-efficient. Once a generator yields a value, it pauses execution and resumes from the same point when the next value is requested. Normal functions, on the other hand, execute completely and return a single value at once.

8. What is *args and **kwargs?

*args is used to pass a variable number of positional arguments to a function. It allows a function to accept more arguments than those defined in its parameters.

**kwargs is used to pass a variable number of keyword arguments (arguments with names). It allows functions to handle named arguments in the form of a dictionary.

9. What is Type Casting? Give Examples.

Type casting is the process of converting one data type into another. Python supports both implicit and explicit type casting. Explicit type casting is done using built-in functions.

Examples:

```
int("10")      # Converts string to integer
float(5)       # Converts integer to float
str(25)        # Converts integer to string
list("abc")    # Converts string to list
```

10. What are Built-in Functions in Python? Name Any Five.

Built-in functions are predefined functions provided by Python that can be used without importing any modules. They perform common tasks efficiently and simplify programming.

Examples :

- print()
- input()
- len()
- type()
- sum()