# Theoretical Questions

## 1.What is the difference between interpreted and compiled languages ?

Difference between Interpreted and Compiled Languages:

1. **Execution** – Compiled languages translate the entire code into machine code before execution, while interpreted languages translate and execute code line by line.
2. **Speed** – Compiled languages are generally faster, whereas interpreted languages are slower due to real-time translation.
3. **Portability** – Interpreted languages are more portable across platforms, compiled languages need recompilation for each platform.

Examples – Compiled: C, C++ | Interpreted: Python, JavaScript.

## 2. What is exception handling in Python ?

**Exception handling** in Python is a mechanism to manage runtime errors by using 'try', 'except', 'else', and 'finally' blocks, allowing the program to handle unexpected situations gracefully without crashing, and ensuring proper cleanup or alternative actions when errors occur.

## 3. What is the purpose of the finally block in exception handling ?

The 'finally' block in Python exception handling is used to define code that will run regardless of whether an exception occurs or not, typically for cleanup tasks like closing files, releasing resources, or resetting states.

## 4. What is logging in Python ?

**Logging** in Python is the process of recording events, errors, and informational messages during a program’s execution using the built-in logging module, which helps in debugging, monitoring, and maintaining applications without interrupting their flow.

## 5. What is the significance of the '**del**' method in Python ?

The `__del__` method in Python is a **destructor** that is automatically called when an object is about to be destroyed, typically used to release resources or perform cleanup tasks before the object is removed from memory.

## 6. What is the difference between import and from ... import in Python ?

- `import` – Imports the entire module, and you access its functions or variables using the module name as a prefix (e.g., `math.sqrt()`).
- `from ... import` – Imports specific functions, classes, or variables from a module, allowing you to use them directly without the module prefix (e.g., `sqrt()` instead of `math.sqrt()`).

## 7. How can you handle multiple exceptions in Python ?

You can handle multiple exceptions in Python by:

1. **Using multiple `except` blocks** – One for each exception type.
2. **Catching multiple exceptions in a single block** – By grouping them in parentheses, e.g., `except (TypeError, ValueError):`.

## 8. What is the purpose of the with statement when handling files in Python ?

The `with` **statement** in Python is used for **automatic resource management** when working with files, ensuring that the file is properly closed after its block of code is executed, even if an exception occurs.

## 9. What is the difference between multithreading and multiprocessing ?

- **Multithreading** – Runs multiple threads within the same process, sharing the same memory space; best for I/O-bound tasks but limited by Python's GIL for CPU-bound tasks.
- **Multiprocessing** – Runs multiple processes with separate memory spaces, allowing true parallelism and better performance for CPU-bound tasks.

## 10. What are the advantages of using logging in a program ?

**Advantages of using logging in a program:**

1. Provides a record of events for debugging and analysis.
2. Allows different log levels (INFO, WARNING, ERROR, etc.) for better control.
3. Helps monitor program execution without interrupting it.
4. Can store logs in files for future reference.

## 11. What is memory management in Python ?

**Memory management in Python** is the process of allocating and releasing memory for objects automatically using a built-in **garbage collector** and a **private heap space**, ensuring efficient use of memory and cleanup of unused objects.

## 12. What are the basic steps involved in exception handling in Python ?

The basic steps in exception handling in Python are:

1. **Wrap risky code in a `try` block** to monitor for errors.
2. **Use `except` block(s)** to catch and handle specific or general exceptions.
3. *(Optional)* **Use `else` block** to run code if no exceptions occur.
4. *(Optional)* **Use `finally` block** for cleanup tasks that must run regardless of errors.

## 13. Why is memory management important in Python ?

**Memory management is important in Python** because it ensures efficient use of system resources, prevents memory leaks, improves program performance, and maintains stability by automatically cleaning up unused objects through garbage collection.

## 14. What is the role of try and except in exception handling ?

In Python exception handling, the **`try` block** contains code that may raise an error, while the **`except` block** catches and handles the error, preventing the program from crashing and allowing alternative actions to be taken.

## 15. How does Python's garbage collection system work ?

Python's garbage collection system works by automatically reclaiming memory from objects that are no longer referenced, primarily using **reference counting** and a **cyclic garbage collector** to detect and clean up unused objects involved in reference cycles.

## 16. What is the purpose of the else block in exception handling ?

The `else` **block** in Python exception handling is used to define code that runs **only if no exceptions occur** in the `try` block, keeping normal execution code separate from error-handling code.

## 17. What are the common logging levels in Python ?

The common logging levels in Python are:

1. **DEBUG** – Detailed diagnostic information for debugging.
2. **INFO** – General information about program execution.
3. **WARNING** – Indication of potential problems.
4. **ERROR** – Serious issues that prevent part of the program from running.
5. **CRITICAL** – Severe errors that may cause the program to stop.

## 18. What is the difference between os.fork() and multiprocessing in Python ?

- `os.fork()` – Creates a child process by duplicating the current process (Unix/Linux only), offering low-level control but requiring manual handling of communication and synchronization.
- `multiprocessing` – A high-level Python module that works cross-platform, providing an easy API to create and manage separate processes with built-in communication and synchronization tools.

## 19. What is the importance of closing a file in Python ?

Closing a file in Python is important because it frees up system resources, ensures that all buffered data is written to disk, and prevents file corruption or data loss.

## 20. What is the difference between file.read() and file.readline() in Python ?

- `file.read()` – Reads the entire file (or a specified number of bytes) into a single string.
- `file.readline()` – Reads only one line from the file at a time, including the newline character.

## 21. What is the logging module in Python used for ?

The `logging` **module** in Python is used to record messages about a program's execution—such as debug info, warnings, and errors—allowing developers to monitor, troubleshoot, and analyze the program without interrupting its flow.

## 22. What is the os module in Python used for in file handling ?

The `os` **module** in Python is used for interacting with the operating system, enabling file handling tasks like creating, deleting, renaming, and navigating directories, as well as retrieving file metadata.

## 23. What are the challenges associated with memory management in Python ?

Challenges in Python memory management include handling **reference cycles**, avoiding **memory leaks** from lingering references, managing memory for large datasets efficiently, and minimizing overhead from Python's dynamic object allocation.

## 24. How do you raise an exception manually in Python ?

You can raise an exception manually in Python using the `raise` keyword followed by an exception class, e.g.,

```
raise ValueError("Invalid input")
```

This stops normal execution and triggers the specified exception.

## 25. Why is it important to use multithreading in certain applications ?

Multithreading is important in certain applications because it allows multiple tasks to run concurrently within the same process, improving responsiveness and performance in **I/O-bound** operations like network requests, file handling, or user interface updates.