1. Difference Between Interpreted and Compiled Languages

- **Interpreted Languages**: Code is executed line by line by an interpreter at runtime (e.g., Python, JavaScript). Slower execution but easier debugging.
- **Compiled Languages**: Code is converted into machine code by a compiler before execution (e.g., C, C++). Faster execution but requires compilation before running.

2. Exception Handling in Python

- A mechanism to handle runtime errors, ensuring the program doesn't crash unexpectedly. It uses `try`, `except`, `finally`, and `else` blocks.

3. Purpose of the finally Block in Exception Handling

- The `finally` block ensures that certain cleanup operations (e.g., closing files, releasing resources) are executed, regardless of whether an exception occurs.

4. Logging in Python

- The `logging` module allows tracking events in a program, such as errors and debugging messages. It helps diagnose issues without using print statements.

5. Significance of the `__del__` Method in Python

- The `__del__` method is a destructor that is called when an object is deleted or goes out of scope, used for cleanup tasks.

6. Difference Between `import` and `from ... import` in Python

- `import module`: Imports the entire module. Example: `import math` (use `math.sqrt(4)`).
- `from module import function`: Imports a specific function. Example: `from math import sqrt` (use `sqrt(4)`).

7. Handling Multiple Exceptions in Python

Use multiple `except` blocks or a single `except` block with a tuple of exceptions. Example:
python
Copy code

```
try:
    x = 1 / 0
except (ZeroDivisionError, ValueError) as e:
    print(f"Error: {e}")
```

8. Purpose of the `with` Statement in File Handling

Ensures proper resource management by automatically closing files after use. Example:
python
Copy code

```
with open("file.txt", "r") as f:
    data = f.read()
```

-

9. Difference Between Multithreading and Multiprocessing

- **Multithreading**: Multiple threads run within the same process, sharing memory. Suitable for I/O-bound tasks.
- **Multiprocessing**: Multiple processes run independently, each with its own memory space. Suitable for CPU-bound tasks.

10. Advantages of Using Logging in a Program

- Provides a systematic way to track errors and events.
- Helps debug applications without modifying code extensively.
- Allows setting different logging levels for better control.

11. Memory Management in Python

- Python uses automatic memory management with techniques like garbage collection and dynamic memory allocation.

12. Basic Steps in Exception Handling in Python

1. Use `try` to enclose risky code.
2. Use `except` to handle specific exceptions.
3. Optionally use `else` for code that runs if no exceptions occur.
4. Use `finally` for cleanup actions.

13. Why Memory Management is Important in Python

- Prevents memory leaks and optimizes resource usage.
- Ensures efficient handling of large datasets.
- Helps in performance optimization.

14. Role of `try` and `except` in Exception Handling

- `try`: Defines a block where exceptions might occur.
- `except`: Defines how to handle specific exceptions.

15. How Python's Garbage Collection Works

- Uses reference counting and a cyclic garbage collector to remove unreferenced objects automatically.

16. Purpose of the `else` Block in Exception Handling

- Runs code only if no exception occurs in the `try` block.

17. Common Logging Levels in Python

- **DEBUG**: Detailed debugging information.
- **INFO**: General informational messages.
- **WARNING**: Indicates a potential problem.
- **ERROR**: A serious issue that needs attention.
- **CRITICAL**: A severe error that may cause the program to crash.

18. Difference Between `os.fork()` and `multiprocessing` in Python

- `os.fork()`: Creates a child process but is Unix-based.
- `multiprocessing`: Provides a cross-platform way to create multiple processes.

19. Importance of Closing a File in Python

- Prevents data loss and resource leaks by ensuring data is written properly.

20. Difference Between `file.read()` and `file.readline()`

- `file.read()`: Reads the entire file or a specified number of bytes.
- `file.readline()`: Reads only one line at a time.

21. Purpose of the Logging Module in Python

- Used to track events, errors, and debugging information systematically.

22. Purpose of the `os` Module in File Handling

- Provides functions to interact with the file system, such as creating, deleting, and modifying files and directories.

23. Challenges in Python Memory Management

- High memory usage due to dynamic typing.
- Cyclic references that require garbage collection.
- Performance overhead in memory allocation.

24. Raising an Exception Manually in Python

Use the `raise` statement to trigger exceptions deliberately. Example:

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

-

25. Importance of Multithreading in Certain Applications

- Improves performance in I/O-bound applications like web scraping, network requests, and file handling.
- Helps in real-time processing where multiple tasks need to run concurrently.