Files, exceptional handling, logging and memory management

1.What is the difference between interpreted and compiled languages?
→ *Compiled language - Converts the entire source code into machine code before execution. Generally faster because the code is already compiled. Errors are detected at compile-time (before execution).
Produces a separate execute file (.exe, .out, etc.). Less portable-compiled for specific platform.

*Interpreted language - Converts and executes code line by line during runtime. Slower due to real-time interpretation. Errors are detected at runtime (during execution).
Does not produce a separate executable file. More portable- same code can run on any platform with the interpreter.

2.What is exception handling in Python?
→ Exception Handling in Python is a mechanism to gracefully handle errors that occur during the execution of a program, without crashing the entire program.
It is used to prevent program from terminating unexpectedly. Allows developers to manage runtime errors. Improves user experience with friendly error messages.

3.What is the purpose of the finally block in exception handling?
→ The finally block in Python is used to execute code regardless of whether an exception occurred or not in the try block.
*Cleanup Activities- To release external resources (files, network connections, database connections, etc).
*Guaranteed Execution- Code inside finally will always run, whether:
-An exception is raised or not.
-An exception is handled or not.
-A return, break or continue is used inside try or except.

4.What is logging in Python?
→ Logging in Python is the process of tracking events that happen when a program runs. It’s used to record important information, errors, or debugging details, which is helpful for debugging, monitoring,auditing.

5.What is the significance of the __del__ method in Python?
→ The __del__ method in Python is a special method known as a destructor. It is called automatically when an object is about to be destroyed (i.e., when there are no more references to the object).
It cleanup before destruction such as:
*Closing files
*Releasing network or database connections
*Freeing other external resources

6.What is the difference between import and from…import in Python?
→ Both import and from…import are used to bring external modules or specific components into your Python program, but they work a bit differently in terms of usage and namespace handling.
*import statement-use module name as a prefix when accessing its function or variables:

```
        Import math
Ex- result = math.sqrt(25)
```

Adv:keeps, the namespace clean and easy to understand where each function comes from.

*from…import statement- use function directly, without the module name.

```
        from math import sqrt
Ex- result = sqrt(25)
```

Adv:saves typing when using specific functions frequently and make code shorter.

7.How can you handle multiple exceptions in Python?
→ In Python, handling multiple exceptions is essential for writing robust programs that can respond gracefully to various types of runtime errors. Python provides several ways to handle multiple exceptions within a try-except structure.

Each except block is checked in order. If an exception occurs in the try block, the interpreter matches it with the corresponding except block and executes the associated code.

8.What is the purpose of the with statement when handling files in Python?
→ The with statement in Python is used to simplify and safely handle file operations. It ensures that resources like files are properly managed, especially when dealing with reading or writing files.
Main purpose: To automatically open a file, perform operations and close it properly, even if an error occurs during processing.

9.What is the difference between multithreading and multiprocessing?
→ *Multithreading- Running multiple threads within a single process.
Used for I/O bound tasks.
Shared memory between threads.
Communication easier through shared data structures.
If one thread crashes, it may effect others.
*Multiprocessing- Running multiple processes.
Used for CPU bound tasks.
Separate memory space for each process.
Communication is harder, needs inter-process communication.
If one process crashes, others remain safe.

10.What are the advantages of using logging in a program?
→ Logging provides a way to record events, eros and informational messages during program execution. It is far more powerful and flexible than using simple print() statements.
*Track Program execution- Helps you understand what the program is doing at each step.Useful for debugging and monitoring program behavior.

*Error diagnosis- Logs errors and exceptions with full details (like stack traces).. It helps in identifying bugs and failures, especially in production.

*Log to multiple Destinations- Can log messages to: console, files, remote servers, databases.

*Different severity levels- Organizes logs by importance. We can filter messages based on severity level.

*Support concurrent applications- Thread safe and works well in multi threaded or multi process applications.

11.What is memory management in Python?
→ Memory management in Python refers to the process by which Python allocates and deallocates memory to store variables, data structures and objects during program execution.
Python can handle memory management automatically making it easier for developers by managing memory behind the scenes using built in mechanisms.

12.What are the basic steps involved in exception handling in python?
→ *try Block-Detect errors: Place risky code inside the try block. Python tries to execute this block.

```
try:
    result = 10 / 0
```

*except Block-Handle errors: If an errors in the try block, Python jumps to the except block.

```
except ZeroDivisionError:
    print(“Cannot divide by zero.”)
```

*else Block-Run if no error: If the try block run without error, the else block (if present) is executed.Good for running code that should only happen if no exception occurs.

```
else:
    print(“Division successful.”)
```

*finally Block-Always executes: The finally block always runs, whether an exception occurred or not. It is used to release resources like files or database connections.

```
finally:
        print(“Execution completed.”)
```

13.Why is memory management important in Python?
→ Memory management is a critical aspect of programming and in Python it ensures that the system’s memory is efficiently used, reused and released as needed. Though Python handles memory automatically, understanding, its importance helps developers write better-performing and more reliable programs.
*Prevents resource wastage.
*Enhances program stability and speed.
*Supports large-scale, complex applications.
*Allows Python programs to run efficiently without manual memory handling.

14.What is role of try and except in exception handling?
→ *try Block-Detect errors
The try block contains the code that might raise an exception. Python executes this block first. If everything in the try block runs correctly, the except block is skipped.

```
try:
     result = 10 / 0
```

*except Block-Handle errors
If an error occurs in the try block, it jumps to the except block. This block contains code to handle the errors gracefully, such as printing a message or taking corrective action. We can specify the type of exception to catch specific errors.

```
except ZeroDivisionError:
          print(“You can’t divide by zero”)
```

Benefits:
*Prevents program from crashing.
*Helps in debugging and user-friendly error messages.
*Makes code more robust and fault- tolerant.

15.How does Python’s garbage collection system work?
→ Python’s garbage collection system is a built-in mechanism used to automatically manage memory by identifying and reclaiming memory occupied by objects that are no longer needed by the program.
*Reference Counting- Every object in Python has a reference count the number of variables or data structures referring to it. When the reference count drops to zero, the object is no longer is use and is immediately deleted.

*Generational Garbage collection- Python uses a generational approach to improve efficiency.

~ Objects are grouped into three generations (0, 1, 2).
~ Younger generations are collected more frequently.
~ Objects that survive multiple collections are moved to an older generation

*Handling Circular Reference- Python provides the gc module to:

~ Enable/disable garbage collection.
~ Manually trigger garbage collection.
~ Monitor performance and debug memory issues.

```
import gc
gc.collect()    #Manually triggers garbage collection
```

16.What is the purpose of the else block in exception handling?
→ In Python, the else block in exception handling is used to define code that should only run if no exception occurs in the try block.
* The else block allows you to separate the code that should run only when everything in the try block is successful.
* It helps in keeping the logic clean and organized, especially when we want to perform certain actions only if no error was raised.

Ex-
```
try:
     number = int(input(“Enter a number:”))
    Except ValueError:
            print(“Invalid input. Please enter a number.”)
     else:
            print(“You entered:”, number)
```

17.What are the common logging levels in Python?
→ Python's logging module provides a set of standard logging levels that indicate the severity or importance of a log message. These levels help developers categorize messages and control what gets logged based on the application's configuration.
*DEBUG- numeric value:10. Detailed information, typically useful for diagnosing problems during development.
*INFO- numeric value:20. General information confirming that things are working as expected.
*WARNING- numeric value:30. An indication that something unexpected happened, or a potential problem in the future. The program still runs.
*ERROR- numeric value:40. A more serious problem that prevents some part of the program from functioning.

18.What is the difference between os.fork() and multiprocessing in Python?
→ Both os.fork() and multiprocessing module in Python are used to create child processes, but they differ significantly in platform support, ease of use and abstraction level.

*os.fork()- low level process creation
A system call that creates a new child process by duplicating the current process.
Availability-Unix/Linux only
Complexity- Low cleveland manual
Use- Suitable when you need full control over process behavior.

```
   Ex-
     import os
     pid = os.fork()
      If pid == 0:
           print("Child process")
     else:
             print("Parent process")
```

*multiprocessing- high level abstraction
A python standard library module that supports spawning processes using an object- oriented API.

Availability-Cross platform.
Use- High level and user-friendly. Handles inter-process communication (IPC) using pipes, queues, etc
Ex-
```
 from multiprocessing import Process

def task():
     print(“Child process”)

p = Process(target=task)
p.start()
p.join()
```

19.What is the importance of closing a files in Python?
→ Closing a file in Python is a crucial step in file handling. When you’re done working with a file, it’s important to close it properly using the .close() method or the with statement to ensure resources are freed and data is safely written.
*Frees System Resources- Every open files uses system memory and file descriptors. Closing the file releases these resources so they can be used elsewhere.
*Ensures Data is Written to Disk- In write or append modes, Python buffers data before writing it to disk. The .close() method flushes the buffer, ensuring all data is saved. Not closing the file might result in data loss or corruption.
*Prevents File Locking Issues- Some operating systems lock files while they are open. If a file is not closed, it may prevent other programs or processes from accessing it.
*Improves Program Stability- Leaving files open for too long can lead to “Too many open files” error. Properly closing files keeps program clean and efficient.
*Good Programming Practice- Closing file is a part of responsible resource management. It reflects well-written professional code.

20.What is the difference between file.read() and file.readline() in Python?
→ Both file.read() and file.readline() are used to read contents from a file, but they differ in how much data they read at a time and how they process it.

*file.read()- The entire content of the file.
A single string containing all characters from the file.
Use: Useful when you want to load the whole file at once.
Drawback: can be memory-intensive for large files.
Ex-

```
    with open(“example.txt”, “r”) as file:
            content = file.read()
             print(content)       #Prints entire file
```

*file.readline()- One line at a time.
A string containing a single line.
Use: Ideal for processing large files line by line.
Efficient: Uses less memory since it reads only one line.
Ex-

```
     with open(“example.txt”, “r”) as file:
            line = file.readline()
           print(line)       #Prints only the first line
```

21.What is the logging module in python is used for?
→ *Track and Record Events- Logs key events like function calls, errors, or system activities.
*Debugging- Helps developers find and fix bugs by tracking program flow and variables.
*Error Reporting- Captures error messages without stopping the program.
*Monitoring Applications- Keeps a log of what the application is doing in real time or over time.
*Persistent Record Keeping- Can store logs in files, databases or remote servers for future analysis.

22.What is the os module in Python used for in file handling?
→ It is a built-in standard library that provides functions for interacting, with the operating system.
*Working with File Paths-
os.path.join()- Combines paths correctly across operating systems.
os.path.exists()- Checks if a file or directory exists.

os.path.abspath()- Returns the absolute path of a file.

*File and Directory management-
os.remove(“file.txt”)- Deletes a file.
os.rename(“old.txt”, “new.txt”)- Renames a file or directory.
os.mkdir(“folder”)- Creates a new directory.
os.rmdir(“remove”)- Removes an empty directory.
os.listdir(“path”)- Lists all files and folders in a directory.

*Navigating the file system-
os.getcwd()- Returns the current working directory.
os.chdir(“new_folder”)- Changes the current working directory.

*Permissions and Metadata-
os.stat(“file.txt”)- Returns information like file size, creation/modification time, etc.
os.chmod()- Changes the file permissions.

23.What are the challenges associated with memory management in Python?
→*Memory Leaks-Even though Python has garbage collection, memory leaks can occur if references to unused objects are not properly removed.

*Circular References-Python uses reference counting for memory management. If two object refer to each other, they may not be deleted automatically, leading to memory not being freed.

*Unpredictable Garbage Collection Timing- Garbage collection does not run on a fixed schedule. This can cause spikes in memory usage, especially in long-running applications.

*High memory usage for simple objects- Python objects have additional overhead compared to low-level languages like c.

*Difficulty in Tracking memory usage- Python does not always provide clear visibility into how much memory is being used, making it hard to optimize. Tools like tracemalloc, gc, or third-party modules are needed.

24.How do you raise an exception manually in Python?
→ *Raising a ValueError

Ex-

```
age = -5
if age < 0:
   raise ValueError (“Age cannot be negative”)
```

*Raising a custom Exception

Ex-

```
  class MyCustomError(Exception):
            pass
raise MyCustomError(“This is a custom exception”)
```

25.Why is it important to use multithreading in certain applications?
→ *Improves Application Responsiveness- In GUI applications, it ensures the interface remains responsive while performing background task.
*Enables Concurrent Execution- Allows tasks like reading a file, processing and data and waiting for user input to happen at the same time, instead of waiting for each to complete sequentially.
*Efficient use of idle time- In I/O bound tasks, CPU can switch to another thread while one thread waits for data to be read/written.
*Better Resource Utilization- Makes better use of multi-core processors, especially when combined with multiprocessing or when using Python implementations that bypass the GIL like Jython or IronPython.
*Useful for Background Task- Ideal for tasks such as logging, monitoring, or periodic updates that need to run in the background without disturbing the main program.