Q1- What is the difference between interpreted and compiled languages

Ans- The main difference between interpreted and compiled languages lies in how the code is translated into machine language so the computer can execute it.

Compiled Languages Definition: In compiled languages, the source code is translated into machine code by a compiler before execution.

Process:

You write the code (e.g., in C, C++)

A compiler converts the entire code into a binary (executable) file.

You run that file.

Interpreted Languages

Definition: Interpreted languages are executed line-by-line by an interpreter at runtime, without prior compilation to machine code.

Process:

You write the code (e.g., in Python, JavaScript)

An interpreter reads and runs it line by line.

Q2- What is exception handling in Python

Ans- Exception handling in Python is a way to manage errors that occur during the execution of a program, without crashing the whole program.

Why Use Exception Handling?

Prevent the program from stopping suddenly

Handle errors gracefully

Display custom error messages

Q3- What is the purpose of the finally block in exception handling

Ans- The finally block is used to write cleanup code that must always run, no matter what happens — whether an exception occurred or not

Purpose of finally To release resources (like closing files, database connections)

To run important final steps after try/except

To ensure reliability of code execution

Q4- What is logging in Python

Ans- Logging in Python means recording messages about your program's execution — like errors, warnings, or other information — into a file or console.

It helps you:

Track what your code is doing

Debug problems

Monitor the program (especially in large or running systems)

Q5- What is the significance of the **del** method in Python

Ans- The **del**() method is a special method in Python known as a destructor.

It is automatically called when an object is about to be destroyed (i.e., when it's no longer needed and is being removed from memory).

Used to clean up resources like:

Closing files

Closing database connections

Releasing memory

What can I help you build?

Q6- What is the difference between import and from ... import in Python

Ans- Both are used to bring in modules or functions into your Python program, but they work differently.

import module: Brings the whole tool box

from module import tool: Brings only the tool you need

Q7- How can you handle multiple exceptions in Python

Ans- In Python, you can handle multiple exceptions using:

1- Multiple except blocks

2- Single except block with a tuple of exceptions

Q8- What is the purpose of the with statement when handling files in Python

Ans- The with statement is used for simpler and safer file handling in Python. It automatically opens and closes the file, even if an error occurs

Why Use with?

Automatically closes the file

Avoids memory leaks or file corruption

Makes code cleaner and shorter

Reduces chances of forgetting to call file.close()

Q9- What is the difference between multithreading and multiprocessing

Ans- Both multithreading and multiprocessing are used for concurrent execution — running multiple tasks at the same time — but they work in different way

Multithreading

Uses multiple threads within a single process.

Threads share the same memory.

Best for I/O-bound tasks (like reading files, web scraping, user input).

Multiprocessing

Uses multiple processes (each has its own memory and Python interpreter).

Best for CPU-bound tasks (like calculations, image processing).

Q10- What are the advantages of using logging in a program

Ans- Using logging instead of just print() provides many benefits, especially in real-world and large-scale applications.

Helps in Debugging

Logs help trace what your program was doing before it failed.

You can easily find the root cause of errors.

Saves Logs to Files

You can store logs in files and check them later — even after the program has stopped running.

Q11- What is memory management in Python

Ans- Memory management in Python means allocating and freeing memory used by variables, data, and programs automatically so your program runs efficiently

Q12- What are the basic steps involved in exception handling in Python

Ans- Python provides a simple and powerful way to handle errors using try-except blocks. Here are the basic steps:

1-Use try Block 2-Add One or More except Blocks 3-Use else Block 4-Use finally Block

Q13- Why is memory management important in Python

Ans-Memory management is very important in Python because it ensures your program:

Runs efficiently

Uses resources wisely

Avoids crashes and slowdowns

Q14- What is the role of try and except in exception handling

Ans-The try and except blocks are core parts of Python’s exception handling system. They help your program catch errors and respond gracefully, instead of crashing.

try Block – Risky Code Area

The try block contains code that might cause an error (exception).

Python will monitor this block for any issues.

except Block – Error Handler

If an error occurs inside the try block, Python jumps to the except block.

It prevents the program from crashing and lets you show a custom error message or take action.

Q15- How does Python's garbage collection system work

Ans-Python’s garbage collection system is designed to automatically free up memory by deleting objects that are no longer needed — so your program runs efficiently without memory leaks.

Q16- F What is the purpose of the else block in exception handling

Ans-The else block in Python exception handling is used to write code that should run only if no exception occurs in the try block

Why Use the else Block?

Keeps your code clean and organized

Separates:

Code that might cause an error (try)

Code that should run only if everything goes well (else)

Prevents accidentally running "success code" inside try, which could hide bugs

Q17- What are the common logging levels in Python

Ans-Python provides 5 standard logging levels, each representing the severity or importance of the message being logged.

These levels help you control what kind of messages get shown or save

Q18- What is the difference between os.fork() and multiprocessing in Python

Ans-Both os.fork() and the multiprocessing module are used to create new processes in Python — but they are very different in usage, portability, and safety.

os.fork()

Comes from Unix/Linux (not available on Windows).

Creates a child process by duplicating the current process.

You get two copies of the program: parent and child.

multiprocessing Module

A high-level built-in Python module.

Works on all platforms (Windows, macOS, Linux).

Allows creation of independent processes with safe memory separation.

Q19- What is the importance of closing a file in Python

Ans- Closing a file in Python is very important to ensure that:

Your data is saved properly

Your system’s resources are freed

You avoid data corruption and memory leaks

Q20- What is the difference between file.read() and file.readline() in Python

Ans- Both file.read() and file.readline() are used to read from a file, but they behave differentl

file.read()

Reads the entire file content (or a specified number of characters).

Returns a single string containing all lines.

file.readline()

Reads only one line at a time.

Returns a string up to the next newline (\n)

Q21- What is the logging module in Python used for

Ans- The logging module in Python is used to record messages about your program's execution — such as errors, warnings, and status updates — to the console or to a file.

Purpose of the logging Module: Tracks events that happen during program execution

1-Helps in debugging and troubleshooting

2-Allows you to log different types of messages based on severity

3-Saves logs to a file for future reference

4-Useful in both small scripts and large applications

Q22- What is the os module in Python used for in file handling

Ans- The os module in Python provides a way to interact with the operating system, especially for file and directory operations like:

1-Creating, deleting, or renaming files/folders

2-Navigating directories

3-Getting file information

Q23- What are the challenges associated with memory management in Python

Ans- Python makes memory management automatic and easy using garbage collection and reference counting, but there are still some challenges developers should be aware of.

1-Circular References 2-Memory Leaks 3-High Memory Usage with Large Data 4-Lack of Manual Control 5-Memory Fragmentation

Q24- How do you raise an exception manually in Python

Ans- In Python, you can manually trigger an exception using the raise keyword. This is useful when you want to stop the program or signal an error on purpose.

You Can Raise Built-in Exceptions Like:

ValueError

TypeError

ZeroDivisionError

FileNotFoundError

Q25- Why is it important to use multithreading in certain applications?

Ans- Multithreading is important when you want your program to perform multiple tasks at the same time, especially when dealing with I/O operations (like file access, user input, or network calls).

Improves Responsiveness

In desktop or GUI applications, multithreading keeps the interface responsive while background tasks (like loading data) continue

Efficient for I/O-Bound Tasks

For tasks that spend a lot of time waiting (e.g., reading files, downloading from the internet), threads help you do other work during the wait.

Practical Questions

```
#Q1- How can you open a file for writing in Python and write a string to it

#Ans-
file = open("example.txt", "w")


file.write("Hello, this is a test string!")


file.close()




#Q2- Write a Python program to read the contents of a file and print each line

#Ans-
with open("example.txt", "r") as file:

    for line in file:
        print(line, end='')



#Q3- How would you handle a case where the file doesn't exist while trying to open it for reading

#Ans-
try:
    with open("example.txt", "r") as file:
        for line in file:
            print(line, end='')
except FileNotFoundError:
    print("The file was not found.")



#Q4- Write a Python script that reads from one file and writes its content to another file

#Ans-
with open("source.txt", "r") as source_file:
    content = source_file.read()

with open("destination.txt", "w") as destination_file:
    destination_file.write(content)




#Q5- How would you catch and handle division by zero error in Python

#Ans-
try:
    result = 10 / 0
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")



#Q6- Write a Python program that logs an error message to a log file when a division by zero exception occurs

#Ans-

import logging

logging.basicConfig(filename="error.log", level=logging.ERROR)

try:
    result = 10 / 0
except ZeroDivisionError as e:
    logging.error("Division by zero error occurred: %s", e)




#Q7- How do you log information at different levels (INFO, ERROR, WARNING) in Python using the logging module
```

```
#Ans-
import logging

logging.basicConfig(filename="app.log", level=logging.DEBUG)

logging.info("This is an info message.")
logging.warning("This is a warning message.")
logging.error("This is an error message.")



#Q8- Write a program to handle a file opening error using exception handling

#Ans-
try:
    file = open("nonexistent_file.txt", "r")
    content = file.read()
    file.close()
except FileNotFoundError:
    print("Error: The file could not be found.")



#Q9-  How can you read a file line by line and store its content in a list in Python

#Ans-

#1- Using readlines()

with open("example.txt", "r") as file:
    lines = file.readlines()

print(lines)

#2- Using a loop

lines = []
with open("example.txt", "r") as file:
    for line in file:
        lines.append(line)

print(lines)



#Q10- How can you append data to an existing file in Python

#Ans-
with open("example.txt", "a") as file:
    file.write("This line will be added at the end.\n")



#Q11- Write a Python program that uses a try-except block to handle an error when attempting to access a
#dictionary key that doesn't exist

#Ans-
my_dict = {"name": "Alice", "age": 25}

try:
    value = my_dict["city"]
    print("City:", value)
except KeyError:
    print("Error: The key 'city' does not exist in the dictionary.")



#Q12-Write a program that demonstrates using multiple except blocks to handle different types of exceptions

#Ans-

try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print("Result is:", result)
    my_list = [1, 2, 3]
    print("Fifth element:", my_list[4])
except ValueError:
    print("Error: Invalid input. Please enter a number.")
```

```
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except IndexError:
    print("Error: List index out of range.")
```

```
#Q13- How would you check if a file exists before attempting to read it in Python

#Ans-
import os

if os.path.exists("example.txt"):
    with open("example.txt", "r") as file:
        content = file.read()
        print(content)
else:
    print("The file does not exist.")
```

```
#Q14- Write a program that uses the logging module to log both informational and error messages

#Ans-
import logging

logging.basicConfig(filename="app.log", level=logging.INFO)

logging.info("Program started")

try:
    number = int(input("Enter a number: "))
    result = 10 / number
    logging.info("Division successful: 10 / %d = %f", number, result)
except ZeroDivisionError:
    logging.error("Division by zero attempted.")
except ValueError:
    logging.error("Invalid input:
```

```
#Q15- Write a Python program that prints the content of a file and handles the case when the file is empty

#Ans-

try:
    with open("example.txt", "r") as file:
        content = file.read()
        if content:
            print("File Content:\n", content)
        else:
            print("The file is empty.")
except FileNotFoundError:
    print("Error: The file does not exist.")
```

```
#Q16-  Demonstrate how to use memory profiling to check the memory usage of a small program

#Ans-
pip install memory-profiler
```

```
from memory_profiler import profile

@profile
def create_list():
    data = [i for i in range(1000000)]
    return data

if __name__ == "__main__":
    create_list()
```

```
#Q17-Write a Python program to create and write a list of numbers to a file, one number per line

#Ans-

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

with open("numbers.txt", "w") as file:
    for number in numbers:
        file.write(str(number) + "\n")
```

```
#Q18- How would you implement a basic logging setup that logs to a file with rotation after 1MB

#Ans-

import logging
from logging.handlers import RotatingFileHandler

logger = logging.getLogger("MyLogger")
logger.setLevel(logging.INFO)

handler = RotatingFileHandler("app.log", maxBytes=1_000_000, backupCount=3)
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)

logger.addHandler(handler)

# Example logging
logger.info("This is an informational message.")
logger.error("This is an error message.")
```

```
#Q19- Write a program that handles both IndexError and KeyError using a try-except block

#Ans-

my_list = [10, 20, 30]
my_dict = {"a": 1, "b": 2}

try:
    print("List item:", my_list[5])       # This will raise IndexError
    print("Dictionary value:", my_dict["z"])  # This will raise KeyError
except IndexError:
    print("Error: List index is out of range.")
except KeyError:
    print("Error: Dictionary key not found.")
```

```
#Q20- How would you open a file and read its contents using a context manager in Python

#Ans-
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

```
#Q21- Write a Python program that reads a file and prints the number of occurrences of a specific word

#Ans-

word_to_count = "python"

with open("example.txt", "r") as file:
    content = file.read().lower()
    count = content.count(word_to_count.lower())

print(f"The word '{word_to_count}' occurred {count} times.")
```

```
#Q22-How can you check if a file is empty before attempting to read its contents
```

```
#Ans-

import os

file_path = "example.txt"

if os.path.getsize(file_path) == 0:
    print("The file is empty.")
else:
    with open(file_path, "r") as file:
        content = file.read()
        print(content)



#Q23- Write a Python program that writes to a log file when an error occurs during file handling

#Ans-

import logging

logging.basicConfig(filename="file_errors.log", level=logging.ERROR, format='%(asctime)s - %(levelname)s - %(message)s')

try:
    with open("nonexistent_file.txt", "r") as file:
        content = file.read()
        print(content)
except FileNotFoundError as e:
    logging.error("File not found: %s", e)
except IOError as e:
    logging.error("IO error occurred: %s", e)
```