**Q1. What is multiprocessing in Python? Why is it useful?**

**Multiprocessing** in Python is a module that allows the creation and management of multiple processes. It enables parallel execution of code, taking advantage of multiple CPU cores to perform tasks simultaneously.

**Usefulness:**

* **Improves Performance**: It allows computationally intensive tasks to be executed concurrently.
* **Efficient Resource Utilization**: By using multiple cores, the program can process data faster.
* **Bypasses GIL**: Python's Global Interpreter Lock (GIL) restricts execution of multiple threads in parallel within the same process. Multiprocessing creates separate processes, each with its own GIL, thus avoiding this limitation.

**Q2. What are the differences between multiprocessing and multithreading?**

| **Feature** | **Multiprocessing** | **Multithreading** |
| --- | --- | --- |
| **Definition** | Involves multiple processes running independently. | Involves multiple threads within a single process. |
| **GIL Bypass** | Each process has its own Python interpreter, bypassing GIL. | Threads share the same Python interpreter, limited by GIL. |
| **Resource Usage** | More resource-intensive as processes do not share memory. | Lightweight as threads share the same memory space. |
| **Use Case** | Suitable for CPU-bound tasks. | Suitable for I/O-bound tasks. |
| **Complexity** | More complex to manage inter-process communication. | Easier to share data between threads. |

**Q3. Python Code to Create a Process Using the Multiprocessing Module**

import multiprocessing

def worker\_function():

print("Process is running")

if \_\_name\_\_ == "\_\_main\_\_":

process = multiprocessing.Process(target=worker\_function)

process.start()

process.join()

**Q4. What is a Multiprocessing Pool in Python? Why is it Used?**

A **multiprocessing pool** is an abstraction provided by the Pool class in the multiprocessing module. It manages a pool of worker processes and can distribute tasks across these workers.

**Why is it used?**

* **Task Distribution**: Efficiently distributes tasks among multiple worker processes.
* **Ease of Use**: Simplifies parallel execution of functions using methods like map and apply.
* **Automatic Worker Management**: Automatically manages the lifecycle of worker processes.

**Q5. Creating a Pool of Worker Processes Using Multiprocessing**

import multiprocessing

def square(n):

return n \* n

if \_\_name\_\_ == "\_\_main\_\_":

with multiprocessing.Pool(processes=4) as pool:

numbers = [1, 2, 3, 4, 5]

results = pool.map(square, numbers)

print(results)

**Q6. Python Program to Create 4 Processes, Each Printing a Different Number**

import multiprocessing

def print\_number(number):

print(f"Process {multiprocessing.current\_process().name} prints: {number}")

if \_\_name\_\_ == "\_\_main\_\_":

numbers = [1, 2, 3, 4]

processes = []

for i, num in enumerate(numbers):

process = multiprocessing.Process(target=print\_number, args=(num,), name=f"Process-{i+1}")

processes.append(process)

process.start()

for process in processes:

process.join()