Q1. What is multithreading in Python? Why is it used? Name the module used.

Multithreading is a technique where **multiple threads execute concurrently within a single process**. Each thread performs a separate task but shares the same memory space.

Why multithreading is used:

- To improve **responsiveness** of applications
- To perform **multiple tasks simultaneously**
- To efficiently handle **I/O-bound operations** (file handling, networking)
- To reduce **execution time** for concurrent tasks

Module used:

- **`threading` module**

Q2. Why is the `threading` module used?

Explain the use of the following functions:

The `threading` module is used to:

- Create and manage threads
- Synchronize threads
- Control execution flow of threads

1. `activeCount()`

Returns the **number of active threads** currently running.

```
import threading
print(threading.activeCount())
```

---

2. `currentThread()`

Returns the **current thread object**.

```
import threading
print(threading.currentThread().getName())
```

---

3. `enumerate()`

Returns a **list of all active thread objects**.

```
import threading
print(threading.enumerate())
```

Q3. Explain the following thread methods

1. `run()`

- Contains the code that the thread executes
- Automatically called when `start()` is used

```
def run(self):
    print("Thread running")
```

---

2. `start()`

- Starts the thread execution

- Internally calls `run()`

```
t.start()
```

---

3. `join()`

- Makes the calling thread wait until the thread completes execution

```
t.join()
```

---

4. `isAlive()` / `is_alive()`

- Returns `True` if the thread is still running, else `False`

```
print(t.is_alive())
```

Q4. Python program to create two threads

- Thread 1 → prints squares
- Thread 2 → prints cubes

```
import threading

def print_squares():
    squares = [i*i for i in range(1, 6)]
    print("Squares:", squares)

def print_cubes():
    cubes = [i*i*i for i in range(1, 6)]
    print("Cubes:", cubes)
```

```
t1 = threading.Thread(target=print_squares)
t2 = threading.Thread(target=print_cubes)

t1.start()
t2.start()

t1.join()
t2.join()

print("Main thread finished")
```

Q5. Advantages and Disadvantages of Multithreading

Advantages:

1. Better CPU utilization
2. Faster execution for I/O-bound tasks
3. Improved application responsiveness
4. Shared memory allows easy data sharing

Disadvantages:

1. Complex debugging
2. Risk of deadlocks
3. Race conditions
4. Limited CPU-bound performance due to GIL (Global Interpreter Lock)

Q6. Explain Deadlocks and Race Conditions

Deadlock

A **deadlock** occurs when two or more threads wait indefinitely for resources held by each other.

Example scenario:

- Thread A holds Resource 1 and waits for Resource 2
- Thread B holds Resource 2 and waits for Resource 1

➡ Program freezes.

Race Condition

A **race condition** occurs when multiple threads access shared data **simultaneously**, and the final result depends on execution order.

Example:

```
counter = 0

def increment():
    global counter
    counter += 1
```

If two threads execute `increment()` at the same time, the result may be incorrect.