Q1. What is multithreading in Python? Why is it used? Name the module used to handle threads in Python. Why is the threading module used? Write the use of the following functions (activeCount, currentThread, enumerate).

Multithreading in Python:
Multithreading is the concurrent execution of multiple threads (smaller units of a process) to improve the program's performance and responsiveness. It allows different parts of a program to run independently.

In [ ]:
```
q.2
```

Q1. What is multithreading in Python? Why is it used? Name the module used to handle threads in Python. Why is the threading module used? Write the use of the following functions (activeCount, currentThread, enumerate).
Multithreading in Python:
Multithreading is the concurrent execution of multiple threads (smaller units of a process) to improve the program's performance and responsiveness. It allows different parts of a program to run independently.

Module for Handling Threads:
The threading module is used to handle threads in Python.

Functions in Threading Module:

activeCount: Returns the number of Thread objects currently alive.
currentThread: Returns the current Thread object corresponding to the caller's thread of control.
enumerate: Returns a list of all Thread objects currently alive.

In [1]:
```
import threading

# activeCount
print(f"Number of active threads: {threading.activeCount()}")

# currentThread
current_thread = threading.currentThread()
print(f"Current Thread: {current_thread.name}")

# enumerate
all_threads = threading.enumerate()
print(f"All Threads: {[thread.name for thread in all_threads]}")
```

```
Number of active threads: 6
Current Thread: MainThread
All Threads: ['MainThread', 'IOPub', 'Heartbeat', 'Control', 'IPythonHistorySavingThread', 'Thread-4']

C:\Users\engin\AppData\Local\Temp\ipykernel_10968\3503778342.py:4: DeprecationWarning: activeCount() is deprecat
ed, use active_count() instead
  print(f"Number of active threads: {threading.activeCount()}")
C:\Users\engin\AppData\Local\Temp\ipykernel_10968\3503778342.py:7: DeprecationWarning: currentThread() is deprec
ated, use current_thread() instead
  current_thread = threading.currentThread()
```

In [ ]:
```
Q3. Explain the following functions (run, start, join, isAlive).
```

In [ ]:
```
Functions in Thread Class:

run: The run method contains the code that will be executed by the thread.
start: The start method starts the thread's activity. It invokes the run method.
join: The join method waits for the thread to complete its execution before moving on.
isAlive: The isAlive method checks if the thread is still executing.
```

Q4. Write a Python program to create two threads. Thread one must print the list of squares, and thread two must print the list of cubes.

In [2]:
```
import threading

def print_squares():
    for i in range(1, 6):
        print(f"Square of {i}: {i*i}")

def print_cubes():
    for i in range(1, 6):
        print(f"Cube of {i}: {i*i*i}")

# Create two threads
thread1 = threading.Thread(target=print_squares)
thread2 = threading.Thread(target=print_cubes)

# Start the threads
thread1.start()
thread2.start()

# Wait for both threads to finish
thread1.join()
thread2.join()
```

```
Square of 1: 1
Square of 2: 4
Square of 3: 9
Square of 4: 16
Square of 5: 25
Cube of 1: 1
Cube of 2: 8
Cube of 3: 27
Cube of 4: 64
Cube of 5: 125
```

In [ ]:
```
Q5. State advantages and disadvantages of multithreading.
```

```
Advantages:

Improved Performance: Multithreading can lead to better utilization of CPU resources and improved program
performance.
Responsiveness: Allows the program to remain responsive even when certain tasks are time-consuming.
Resource Sharing: Threads within the same process can share resources like memory, leading to efficient data
sharing.
Parallel Execution: Tasks can be executed in parallel, enhancing efficiency on multi-core systems.
Concurrency: Enables concurrent execution, allowing multiple parts of the program to progress simultaneously.

Disadvantages:

Complexity: Multithreading introduces complexity in managing shared resources and synchronization.
Difficulty in Debugging: Debugging multithreaded programs can be challenging due to potential race conditions
and deadlocks.
Overhead: There is overhead in creating and managing threads, which may negate the benefits for small tasks.
Potential for Deadlocks: Improper synchronization can lead to deadlocks, where threads are blocked
indefinitely.
GIL (Global Interpreter Lock): In CPython, the GIL allows only one thread to execute Python bytecode at a time,
limiting the benefits of multithreading in CPU-bound tasks.
```

In [ ]:
```
Q6. Explain deadlocks and race conditions.
```

```
Deadlock:
A deadlock occurs when two or more threads are blocked forever, each waiting for the other to release a
resource. It typically involves a circular waiting scenario where each thread is holding a resource that the
other thread needs.

Race Condition:
A race condition occurs when two or more threads access shared data concurrently, and the final outcome depends
on the timing of their execution. It can lead to unpredictable results and errors if proper synchronization
mechanisms are not used
```

In [3]:
```
import threading

resource1 = threading.Lock()
resource2 = threading.Lock()

def thread1_func():
    with resource1:
        with resource2:
            print("Thread 1 is holding both resources.")

def thread2_func():
    with resource2:
        with resource1:
            print("Thread 2 is holding both resources.")

thread1 = threading.Thread(target=thread1_func)
thread2 = threading.Thread(target=thread2_func)

thread1.start()
thread2.start()

thread1.join()
thread2.join()
```

```
Thread 1 is holding both resources.
Thread 2 is holding both resources.
```

In [ ]:
```
Race Condition
```

In [4]:
```
import threading

counter = 0

def increment_counter():
    global counter
    for _ in range(1000000):
        counter += 1

thread1 = threading.Thread(target=increment_counter)
thread2 = threading.Thread(target=increment_counter)

thread1.start()
thread2.start()

thread1.join()
thread2.join()

print(f"Final Counter Value: {counter}")
```

```
Final Counter Value: 2000000
```

In [ ]:
```

```