Practical No 4

Aim: Multi-threading and Fibonacci Generation

4.1: Implement multi-threading to generate and print Fibonacci sequences

```
import threading

def fibonacci(n, name):
    a, b = 0, 1
    print(f"{name} generating {n} Fibonacci numbers:")
    for i in range(n):
        print(f"{name}: {a}")
        a, b = b, a + b

# Create threads for two Fibonacci sequences
thread1 = threading.Thread(target=fibonacci, args=(5, "Thread-1"))
thread2 = threading.Thread(target=fibonacci, args=(7, "Thread-2"))

thread1.start()
thread2.start()

thread1.join()
thread2.join()

print("All Fibonacci threads finished.")
```

```
Thread-1 generating 5 Fibonacci numbers:
Thread-1: 0
Thread-1: 1
Thread-2 generating 7 Fibonacci numbers:
Thread-1: 1
Thread-2: 0
Thread-1: 2
Thread-2: 1
Thread-1: 3
Thread-2: 1
Thread-2: 2
Thread-2: 3
Thread-2: 5
Thread-2: 8
All Fibonacci threads finished.
```

4.2: Thread safety and synchronization when accessing shared variables

```
import threading

lock = threading.Lock()
sharedSum = 0

def addFibonacciSum(n):
    global sharedSum
    a, b = 0, 1
    for _ in range(n):
        with lock:  # Ensure only one thread updates at a time
            sharedSum += a
```

```
        a, b = b, a + b

threads = []

# Create and start 3 threads
for i in range(3):
    t = threading.Thread(target=addFibonacciSum, args=(5,))
    threads.append(t)
    t.start()

# Wait for all threads to finish
for t in threads:
    t.join()

print("Total sum of Fibonacci numbers (shared):", sharedSum)
```

```
Total sum of Fibonacci numbers (shared): 21
```

4.3: Thread pooling and task delegation using ThreadPool Executor

```
from concurrent.futures import ThreadPoolExecutor

def fibonacciList(n):
    seq = []
    a, b = 0, 1
    for _ in range(n):
        seq.append(a)
        a, b = b, a + b
    return seq

# Thread pool with 3 workers
with ThreadPoolExecutor(max_workers=3) as executor:
    results = list(executor.map(fibonacciList, [5, 7, 10]))

# Display results
for i, seq in enumerate(results, 1):
    print(f"Task {i} result: {seq}")
```

```
Task 1 result: [0, 1, 1, 2, 3]
Task 2 result: [0, 1, 1, 2, 3, 5, 8]
Task 3 result: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```