Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL405**	**Course Name:**	**Skills Based Python Programming Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**8**
Title of the Experiment:	**Write a program to implement Threading in python.**
Date of Performance:	**04/03/2025**
Date of Submission:	**11/03/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mr. Raunak Joshi

Signature :

Date:

Aim: Write a program to implement Threading in python.

Theory:

Threading is a technique in programming that allows multiple tasks to run concurrently within a single process. In Python, threading is used to achieve multitasking by running multiple threads (smaller units of a process) simultaneously. This is particularly useful for tasks like I/O operations, network requests, and other operations that may block the main program flow.

Introduction to Threading:

- **Thread:** The smallest unit of a process that can be scheduled for execution.
- **Multithreading:** The concurrent execution of multiple threads within a single program, enabling better utilization of CPU resources.

Python supports multithreading using the threading module, which provides a higher-level interface for working with threads compared to the lower-level _thread module.

Why Use Threading?

1. **Concurrency:** Allows multiple operations to happen simultaneously, enhancing performance.
2. **Responsiveness:** Improves application responsiveness by handling tasks like user inputs or network requests in the background.
3. **Resource Sharing:** Threads share the same memory space, making communication between them easier compared to multiprocessing.

Global Interpreter Lock (GIL) in Python:

- Python's **GIL** allows only one thread to execute Python bytecode at a time, even on multi-core processors.

- This limitation affects CPU-bound tasks but does not significantly impact I/O-bound tasks.
- **CPU-bound tasks** (e.g., complex calculations) may not benefit from threading due to GIL constraints.
- **I/O-bound tasks** (e.g., file handling, network communication) benefit greatly as threads can run while waiting for I/O operations to complete.

Python threading Module:

The threading module provides the following functionalities:

- **Thread Class:** Used to create and manage threads.
- **start()**: Starts the thread's activity.
- **run()**: Contains the code to be executed by the thread.
- **join()**: Waits for the thread to complete execution.
- **is_alive()**: Checks if the thread is still running.

Creating Threads in Python:

There are two ways to create threads:

1. **Using Thread Class Directly:** By passing a function as the target.
2. **Subclassing Thread Class:** By creating a custom class that inherits from Thread and overriding the run() method.

Implementation:

```
import threading
```

In [5]:

```
import time
```

In [6]:

```
def print_numbers():
    for i in range(1, 6):
        print(f"Thread 1 - Number: {i}")
        time.sleep(1)
```

In [7]:

```
def print_letters():
        for letter in 'ABCDE':
            print(f"Thread 2 - Letter: {letter}")
            time.sleep(1)
```

In [8]:

```
    def print_squares():
        for i in range(1, 6):
            print(f"Thread 3 - Square of {i}: {i ** 2}")
            time.sleep(1)
```

In [9]:

```
    # Creating threads
    thread1 = threading.Thread(target=print_numbers)
    thread2 = threading.Thread(target=print_letters)
    thread3 = threading.Thread(target=print_squares)
```

In [ ]:

```
    # Starting threads
    thread1.start()
    thread2.start()
    thread3.start()
```

```
Thread 1 - Number: 1
Thread 2 - Letter: A
Thread 3 - Square of 1: 1


Thread 1 - Number: 2
Thread 2 - Letter: B
Thread 3 - Square of 2: 4
Thread 1 - Number: 3
Thread 2 - Letter: C
Thread 3 - Square of 3: 9
Thread 1 - Number: 4
Thread 3 - Square of 4: 16
Thread 2 - Letter: D
Thread 1 - Number: 5Thread 3 - Square of 5: 25
Thread 2 - Letter: E
```

In [11]:

```
    # Ensuring all threads complete
    thread1.join()
    thread2.join()
    thread3.join()
    print("All threads have finished executing.")
```

```
All threads have finished executing.
```

Conclusion : Threading in Python enables concurrent execution of tasks, improving performance and responsiveness. Although the GIL limits threading for CPU-bound tasks, it is highly effective for I/O-bound operations. By using the threading module, developers can efficiently manage multiple threads and handle synchronization to avoid race conditions.