Multithreading in Python

CSE3011
Presented By
Rishu Kumar Agarwal (21BCE10451)

Threading in Python

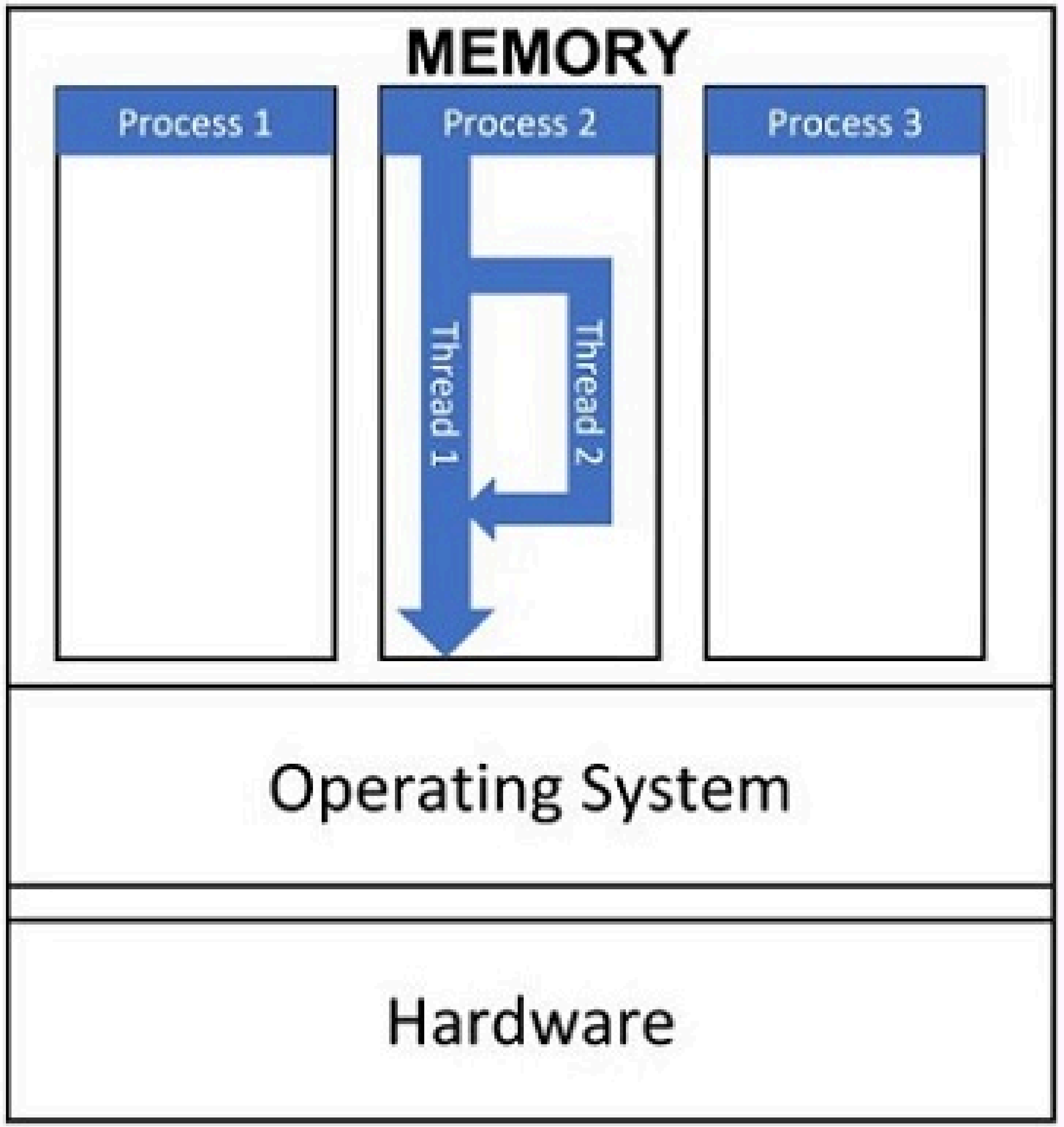
Multithreading refers to the mechanism of dividing the main task in more than one sub-tasks and executing them in an overlapping manner. This makes the execution faster as compared to single thread.

A thread can be thought of as a sub-process in a single program. Threads of a single program share the same memory space allocated to it.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

- Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.

- Threads are independent , if there is an exception in one thread it doesn’t affect remaining threads



Advantages

1. **Improved Responsiveness:** Multithreading allows applications to remain responsive by executing multiple tasks concurrently.
2. **Efficient Resource Utilization**: Threads enable better utilization of CPU resources, especially in I/O-bound scenarios, enhancing overall efficiency.
3. **Simplified Concurrent Programming**: Multithreading provides a straightforward approach to handling concurrent tasks, making code more scalable and manageable.
4. **Enhanced Performance for I/O-bound Tasks:** Multithreading improves performance for tasks involving I/O operations, as threads can perform other tasks while waiting.
5. **Thread-based Parallelism:** While limited by the Global Interpreter Lock (GIL), multithreading still enables parallel execution on multi-core systems, enhancing performance.
6. **Simplified Communication and Coordination**: Threads facilitate easy communication and coordination between concurrent tasks, streamlining development.

Disadvantages

1. **Deadlock:** Multithreading introduces the risk of deadlock, a situation where two or more threads are waiting for each other to release resources, resulting in a halt in execution.
2. **Complex Synchronization:** Threads require careful synchronization to avoid issues like race conditions and deadlocks, adding complexity to code.
3. **Global Interpreter Lock (GIL)**: Python's GIL limits true parallelism, as only one thread can execute Python bytecode at a time, impacting performance, especially for CPU-bound tasks.
4. **Increased Memory Overhead:** Each thread consumes memory for its stack, leading to increased memory consumption with a large number of threads.
5. **Difficulty in Debugging:** Debugging multithreaded applications can be challenging, as issues related to synchronization are not always easy to reproduce or detect.

Threading class methods:

1. **run() :** You can create threads by subclassing the Thread class and overriding the run() method with the code you want to execute in the new thread.

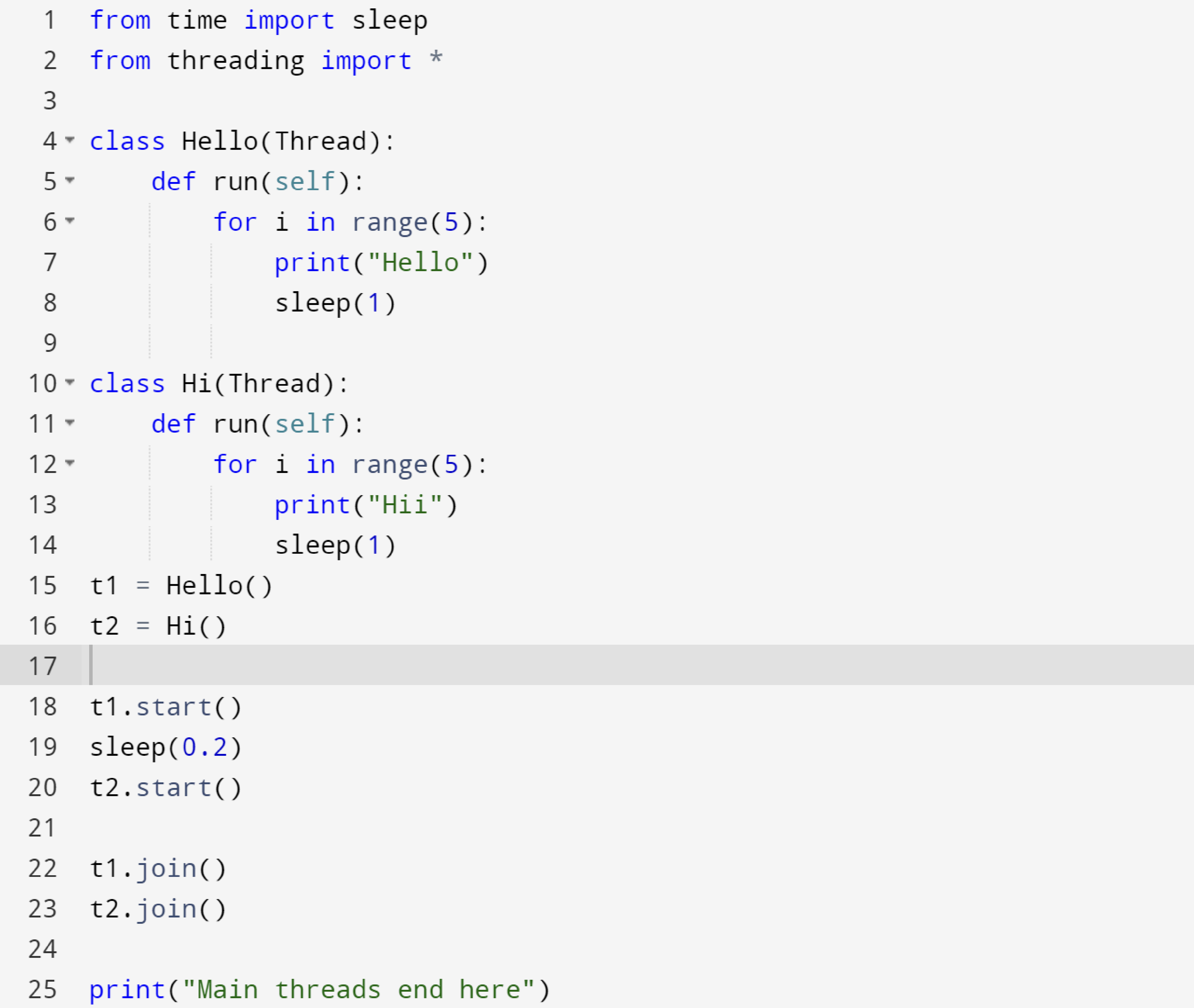
2. start() : to start a thread, you call the start() method. This begins the execution of the thread, and the code inside the run() method or the target function is executed concurrently.

3. join() : The join() method is used to wait for a thread to complete its execution before proceeding further. This is useful when you want to synchronize the execution of threads and ensure that one thread finishes before another begins

4. isAlive() : The Thread class provides methods like is_alive() to check if a thread is currently executing and is_alive() to check if a thread has finished execution.

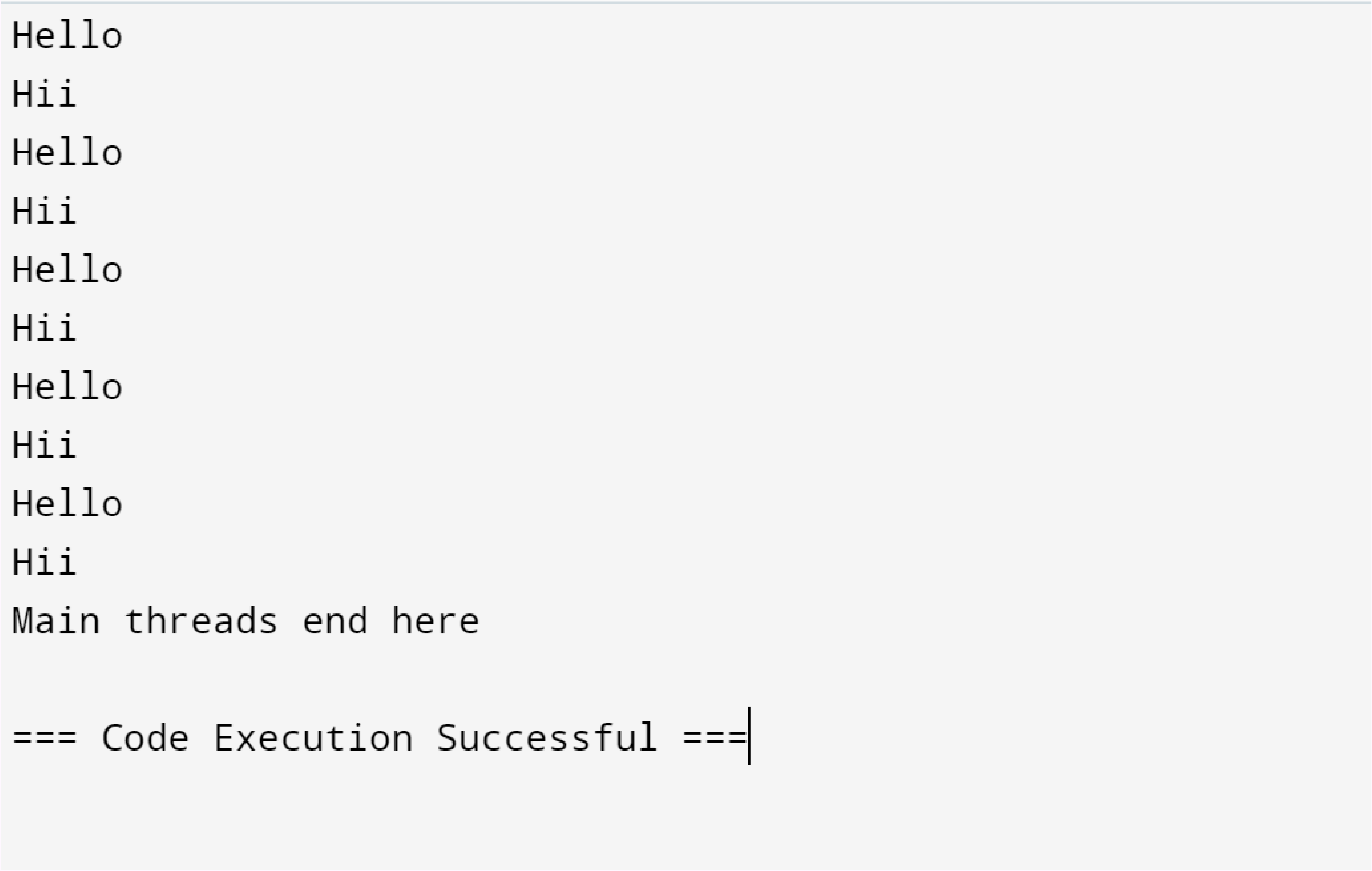
5. getName(): the getName() method returns the name of a thread

6. setName(): the setName() method sets the name of a thread



```python
from time import sleep
from threading import *

class Hello(Thread):
    def run(self):
        for i in range(5):
            print("Hello")
            sleep(1)

class Hi(Thread):
    def run(self):
        for i in range(5):
            print("Hii")
            sleep(1)
t1 = Hello()
t2 = Hi()

t1.start()
sleep(0.2)
t2.start()

t1.join()
t2.join()

print("Main threads end here")
```

Output



```
Hello
Hii
Hello
Hii
Hello
Hii
Hello
Hii
Hello
Hii
Main threads end here

=== Code Execution Successful ===
```

Threading methods

some additional methods –

- **threading.activeCount()** – Returns the number of thread objects that are active.
- **threading.currentThread() –** Returns the number of thread objects in the caller's thread control
- **threading.enumerate() –** Returns a list of all thread objects that are currently active.

Thank You