

Programming Refresher



Threading



Learning Objectives

By the end of this lesson, you will be able to:

- 👁 Implement threading in Python to perform concurrent tasks
- 👁 Apply the acquire and release methods of the lock object to manage thread execution
- 👁 Prevent concurrent access to shared resources using synchronization techniques in Python to ensure data integrity



Business Scenario

ABC is a social media start-up developing a platform for users to share photos and videos.

The platform will run several background processes. To increase processing algorithm performance and optimize computing capacity, the company plans to use multithreading. This approach will also reduce processing costs.

Therefore, ABC will explore threading, multithreading, their life cycle, and thread synchronization.

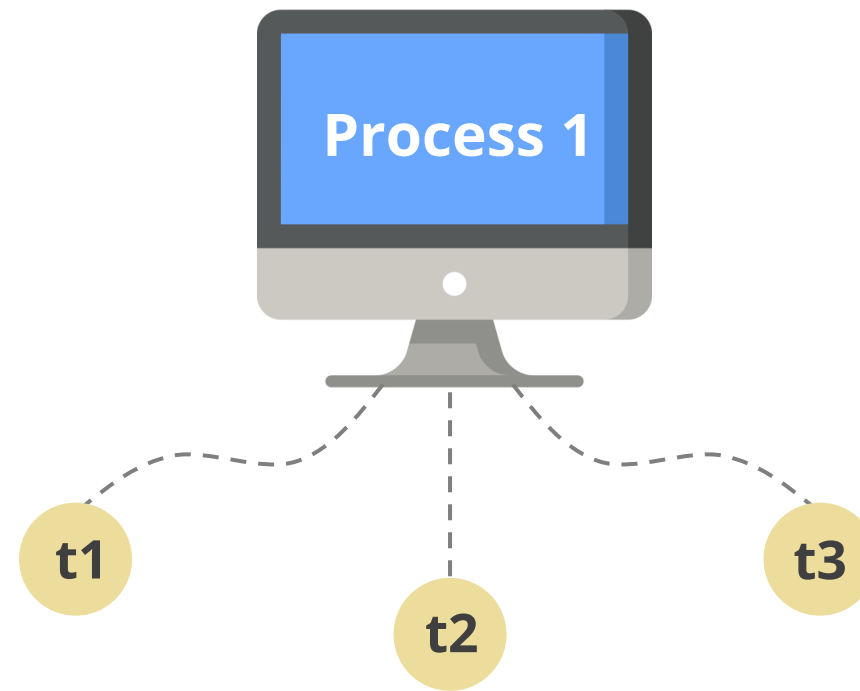




Introduction to Threading

Threading

A thread is the smallest unit of execution with an independent set of instructions.



A process can have multiple threads in it.

Threading

Threads are lightweight and independent processes with the following features:



They have less memory overhead compared to processes.

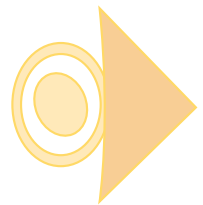


They run within the same process and share runtime resources, such as memory.



A thread has a beginning, an execution path, and a result.

Threading



The thread's current state is stored in an instruction pointer, determining the next execution step and sequence.



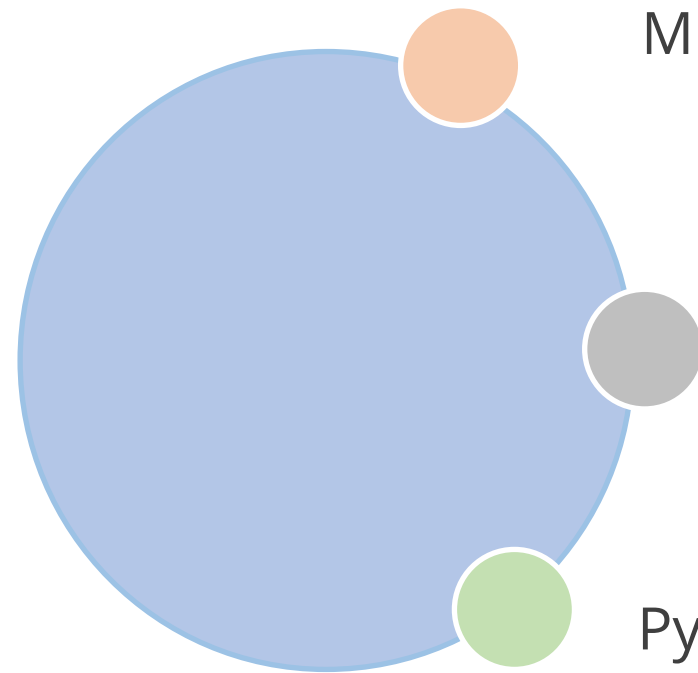
Threading uses idle processes to simulate parallel programming.



Introduction to Multithreading

Multithreading

Multithreading allows a process to run multiple threads concurrently.



Multiple threads share a single data space with the main thread.

Information sharing is easier with concurrent threads.

Python supports multithreading, enabling multitasking.

Multithreading: Advantages

- ▶ Significantly improves responsiveness
- ▶ Enables tasks to be broken into subtasks and executed concurrently
- ▶ Enhances application performance and rendering

Multithreading: Disadvantages

- ▶ No impact on processor computation speed, potentially reducing performance and causing overhead
- ▶ Careful synchronization required to prevent mutual exclusion
- ▶ Increased risk of deadlock

Threading Module

The **threading** module includes a class, **Thread**, that implements threads.

Class method	Method description
run()	Is the entry point function of a thread
start()	Triggers a thread when run method is called
join([time])	Enables a program to wait for threads to terminate
isAlive()	Determines if a thread is active
getName()	Returns the name of a thread
setName()	Updates the name of a thread



Creating a New Thread

Creating a New Thread

Step 1

Define a Thread subclass.

Step 2

Override the **`__init__`** method to add additional arguments.

Step 3

Override the **`run`** method to implement the required action.

Assisted Practice: Creating a New Thread



Duration: 5 mins

Objective: In this demonstration, you will learn how to create a new thread.

Tasks to perform:

1. Define a thread subclass
2. Override the **__init__** method and add additional arguments
3. Override the run() method

Note

Note: Please refer to the **Reference Material** section to download the **Jupyter Notebook** files for the mentioned topic.



Synchronizing Threads

Synchronizing Threads

Synchronization ensures that only one thread at a time uses a resource when multiple threads try to access and modify a shared resource.

Two threads try to access the shared resources.



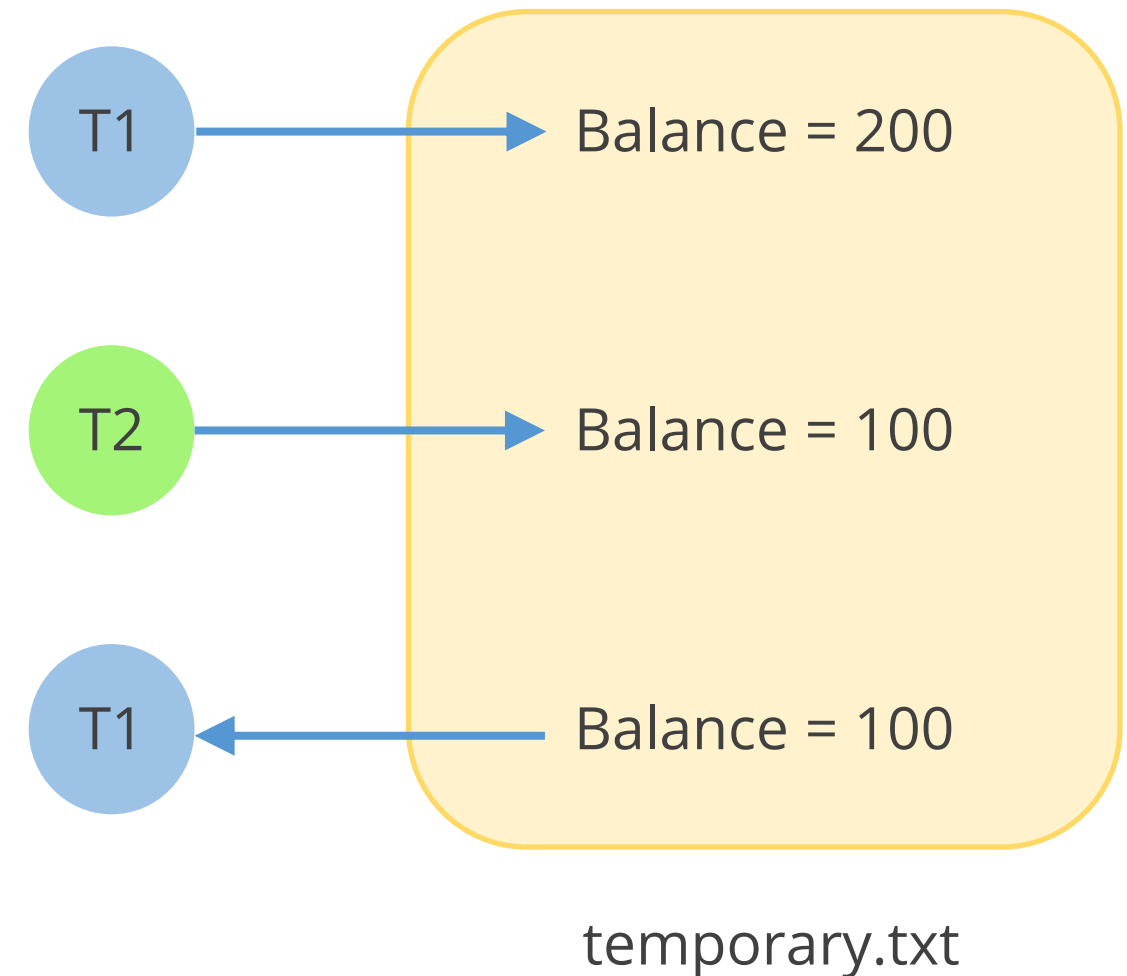
Improper synchronization can lead to a faulty program.

Synchronizing Threads: Example

Consider two different threads, T1 and T2, with *temporary.txt* as a shared resource.

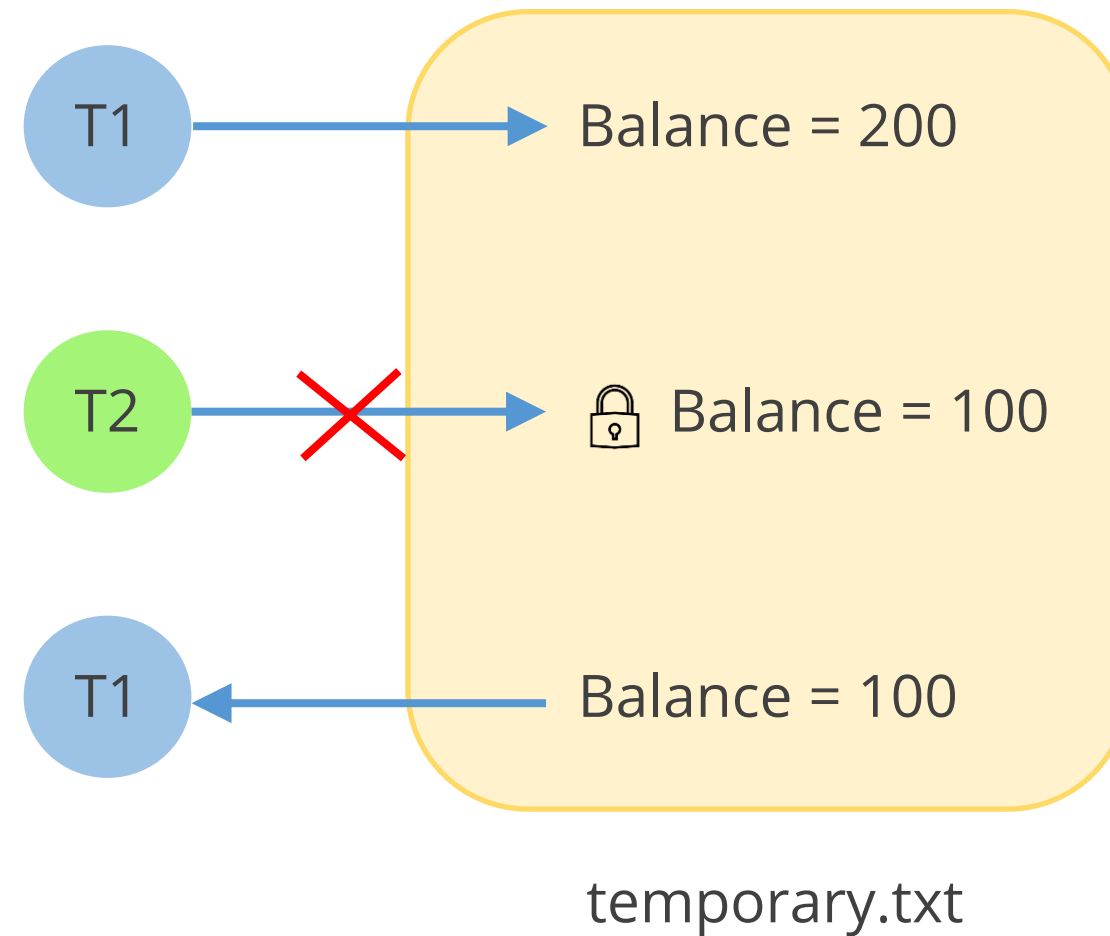
Example

- When T1 starts execution, it saves values in a file called *temporary.txt* to use for computing a result after T1 finishes.
- Before T1 completes, T2 starts and modifies the values that T1 stored in *temporary.txt*.
- Consequently, T1 produces an incorrect result.



Synchronizing Threads: Lock

If synchronization is performed in the example, once thread T1 begins using the file *temporary.txt*, it will be locked. No other thread can access or modify it until thread T1 finishes.



Synchronization in Python

The threading module has built-in locking functionality to synchronize threads.

Locking is essential to manage access to shared resources and prevent data corruption or loss.

The acquire (blocking) method of a lock object allows threads to synchronize by obtaining the lock.

The lock object can create a new lock.

Synchronization in Python: `acquire()`

The lock object has two methods: `acquire()` and `release()`.

`acquire([blocking])`

- The optional blocking parameter specifies whether the thread waits for the lock to be acquired.
- If blocking is set to 0, the thread returns instantly with a value of 0 if the lock can't be obtained and a value of 1 if it can be obtained.
- If blocking is set to 1, the thread blocks and waits for the lock to be released.

Synchronization in Python: release()


The *lock* object has two methods: `acquire()` and `release()`.

`release()`


- The `release()` function releases the lock when it is no longer necessary.
- When the lock is in the locked state, the **release** method unlocks it.
- If multiple threads await the lock, release allows only one thread to proceed.

Use of Synchronization in Python

To prevent concurrent access to an object, you must use a *lock* object.



Python's built-in data structures, such as lists and dictionaries, are thread-safe because they use atomic byte codes.

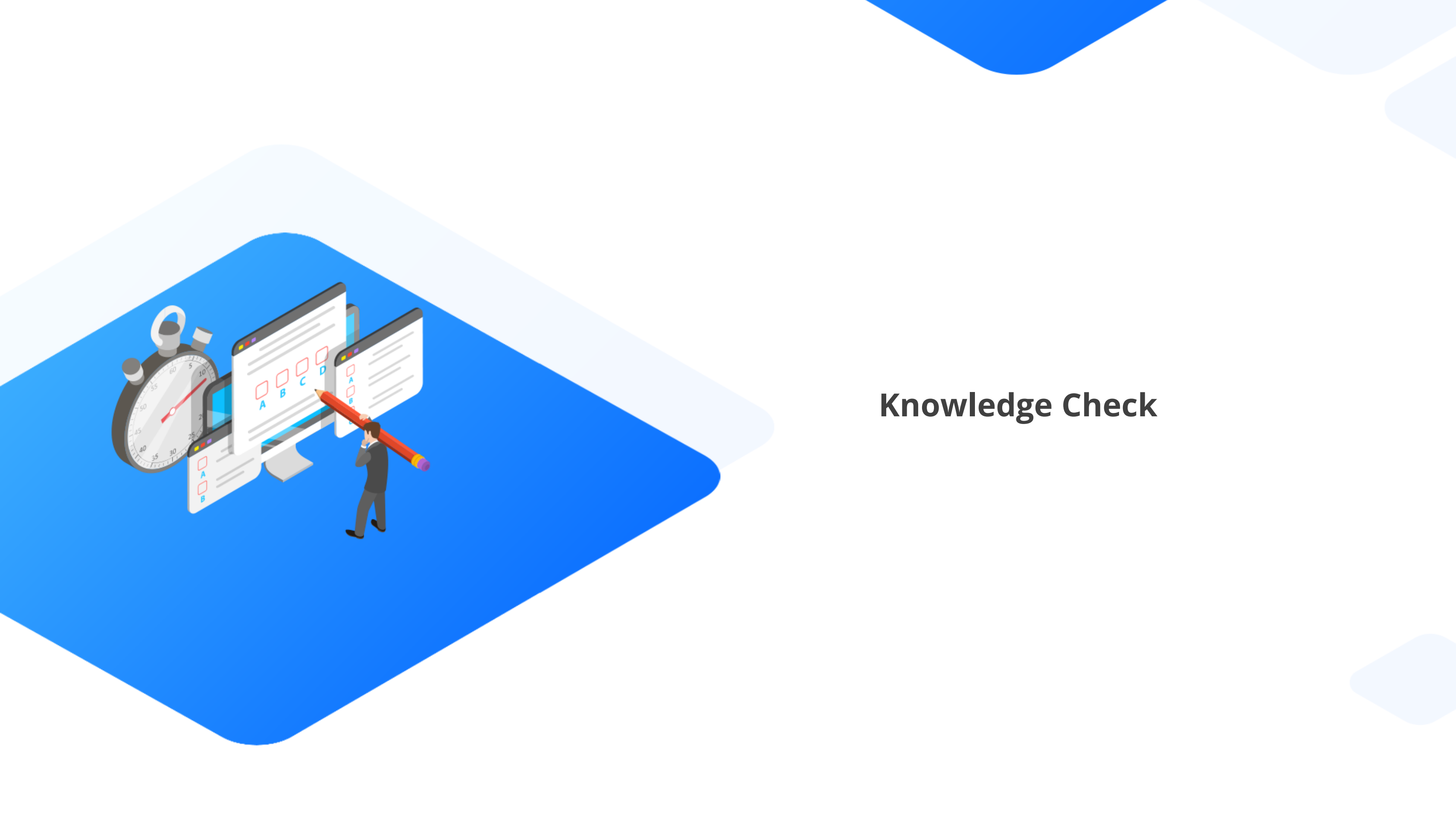


Other Python data structures or fundamental data types, such as integers and floats, do not have this security.

Key Takeaways

- 🕒 Threading is using idle processes to give the appearance of parallel programming.
- 🕒 A thread is the smallest unit of execution with an independent set of instructions.
- 🕒 Synchronization ensures the sharing of resources between multiple threads, which will not cause data loss.
- 🕒 Synchronization is achieved using the lock, acquire, and release methods of the module.





Knowledge Check

Knowledge Check

1

Which Python library supports threads?

- A. thread
- B. threading
- C. Threading
- D. _threading



Knowledge Check

1

Which Python library supports threads?

- A. thread
- B. threading
- C. Threading
- D. _threading

The correct answer is **B**

The threading module includes the class Thread, which supports threads in Python.



Knowledge Check

2

What is the responsibility of the join() method in the Thread class?

- A. The join() method helps threads merge with another thread.
- B. The join() method combines all the threads in a process.
- C. The join() method enables a program to wait for threads to terminate.
- D. The join() method waits for other threads to merge with it.



Knowledge Check

2

What is the responsibility of the join() method in the Thread class?

- A. The join() method helps threads merge with another thread.
- B. The join() method combines all the threads in a process.
- C. The join() method enables a program to wait for threads to terminate.
- D. The join() method waits for other threads to merge with it.



The correct answer is **C**

The join() method in the Thread class enables a program to wait for threads to terminate.

Knowledge Check

3

Which of the following functions releases the lock?

- A. lock()
- B. release()
- C. free()
- D. acquire([1])



Knowledge Check

3

Which of the following functions releases the lock?

- A. lock()
- B. release()
- C. free()
- D. acquire([1])

The correct answer is **B**

The **release()** function releases the lock when it is no longer necessary.





Thank You!