

Programming Refresher



Threading



Learning Objectives

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

- 👁 Interpret threading
- 👁 Explain multi-threading
- 👁 Illustrate the steps to create a new thread
- 👁 Summarize the synchronization of threads



Business Scenario

ABC is a social media startup that is developing a platform for users to contribute photos and videos, which will be saved and shared across the platform's network.

Several processes will be running in the background. The firm wants to take advantage of multithreading in order to increase the performance of processing algorithms and make better use of available computing capacity. This will also save the company money on processing costs.

Therefore, the company will be exploring threading, multithreading and its life cycle, and the synchronization of threads.





Introduction to Threading



Discussion

Threading

Duration: 15 minutes

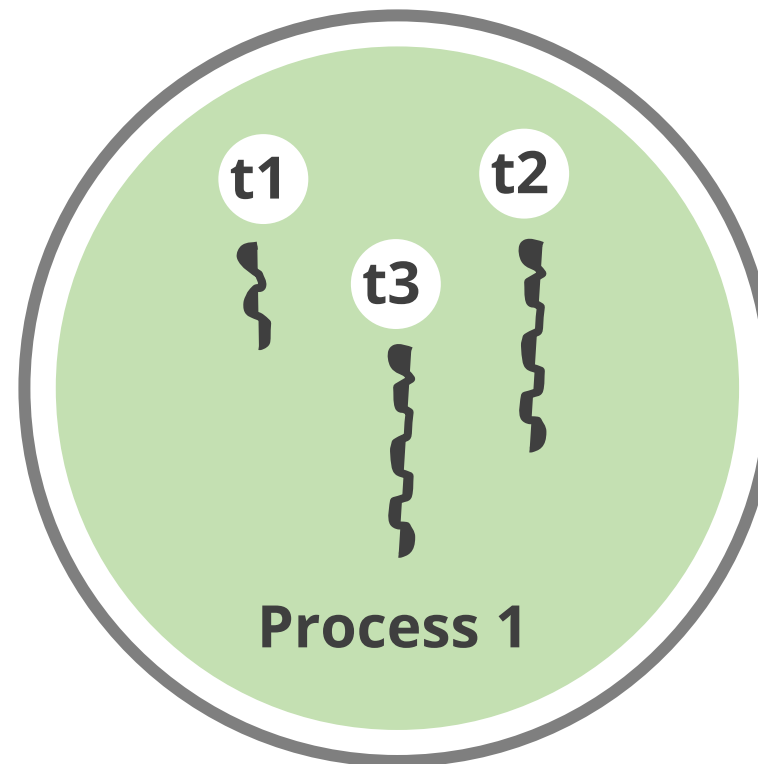
How does threading reduce the runtime resources?

- What is threading?
- Why is threading necessary?



Threading

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



A process can have multiple threads in it as shown in the image.

Threading

Threads are referred to as lightweight and independent processes, which have the following features:



They do not have the same memory overhead as processes.



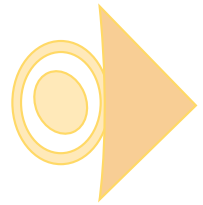
A thread runs within the same process and shares runtime resources (like memory) with the program.



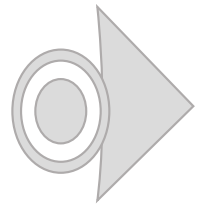
A thread has a beginning, an execution path, and a result.

Threading

Threads ensure an independent execution flow.



The thread's current state is stored in an instruction pointer, which also determines what will be executed next and in what sequence.



To provide the appearance of parallel programming, threading makes use of idle processes.

Threading

Duration: 15 minutes

How does threading reduce the runtime resources?

- What is threading?

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

- Why is threading necessary?

Answer: Threads ensure an independent execution flow. To provide the appearance of parallel programming, threading makes use of idle processes. A thread runs within the same process and shares runtime resources like memory.





Introduction to Multithreading



Discussion

Threading

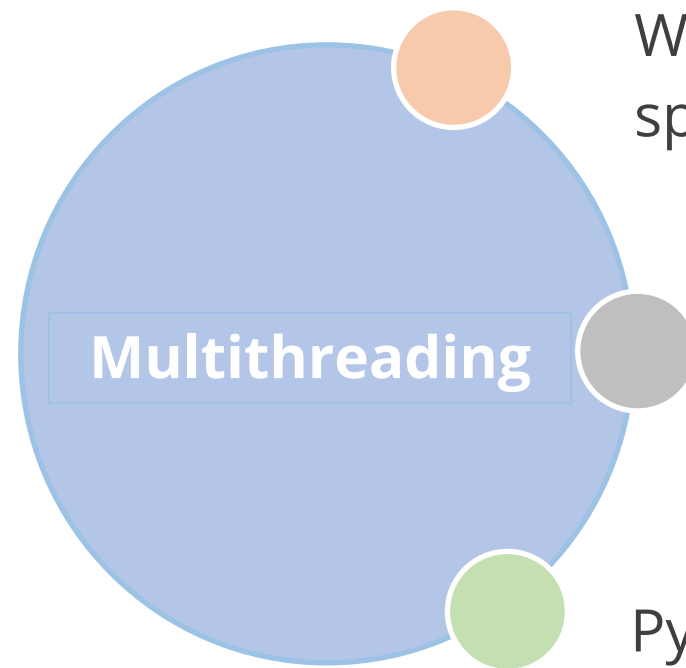
Duration: 15 minutes



- How to run multiple threads simultaneously?
- Is it possible to determine if a thread is dead or alive?

Multithreading

Multithreading is the ability of a process to run many threads concurrently.



When multiple threads are operating concurrently, they share a single data space with the main thread.

This makes it easier for information to be shared.

Python programming makes use of multiple threads and enables multitasking.

Multithreading: Advantages

The advantages of multithreading are as follows:



Improves responsiveness significantly



Allows a task to be broken into many subtasks and executed concurrently



Improves application performance and rendering

Multithreading: Disadvantages

The disadvantages of multithreading are as follows:

- There would be no impact on the computation speed of a processor. It could reduce performance and result in overhead.
- It is important to synchronize carefully to prevent mutual exclusion.
- This raises the possibility of a deadlock.

Threading Module

The *threading* module includes a class *Thread* which is used to implement threads.

Class method	Method description
run()	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

Threading

Duration: 15 minutes



- How to run multiple threads simultaneously?

Answer: Multiple threads can be run simultaneously using the multithreading process. Multithreading is the ability of a process to run many threads concurrently. When multiple threads are operating, they share a single data space with the main thread. This makes it easier for information to be shared.

- Is it possible to determine if a thread is dead or alive?

Answer: Yes, the **isAlive()** function is used to determine if a thread is active or dead.



Creating a New Thread

Creating a New Thread

The steps to create a new thread are as follows.

Step 1

Define a Thread subclass.

Step 2

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

Step 3

Implement the required action by overriding the run function

Assisted Practice: Creating a new thread



Duration: 5 mins

Objective: In this demonstration, we 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



Synchronizing Threads



Discussion

Synchronizing Threads

Duration: 15 minutes

- Why is the synchronization of threads needed?
- How is synchronization of threads done in Python?



Synchronizing Threads

Synchronization is done to make sure that only one thread at a time is using a resource when multiple threads are trying to access and modify a shared resource.

Here, two threads are trying to access the shared resources:



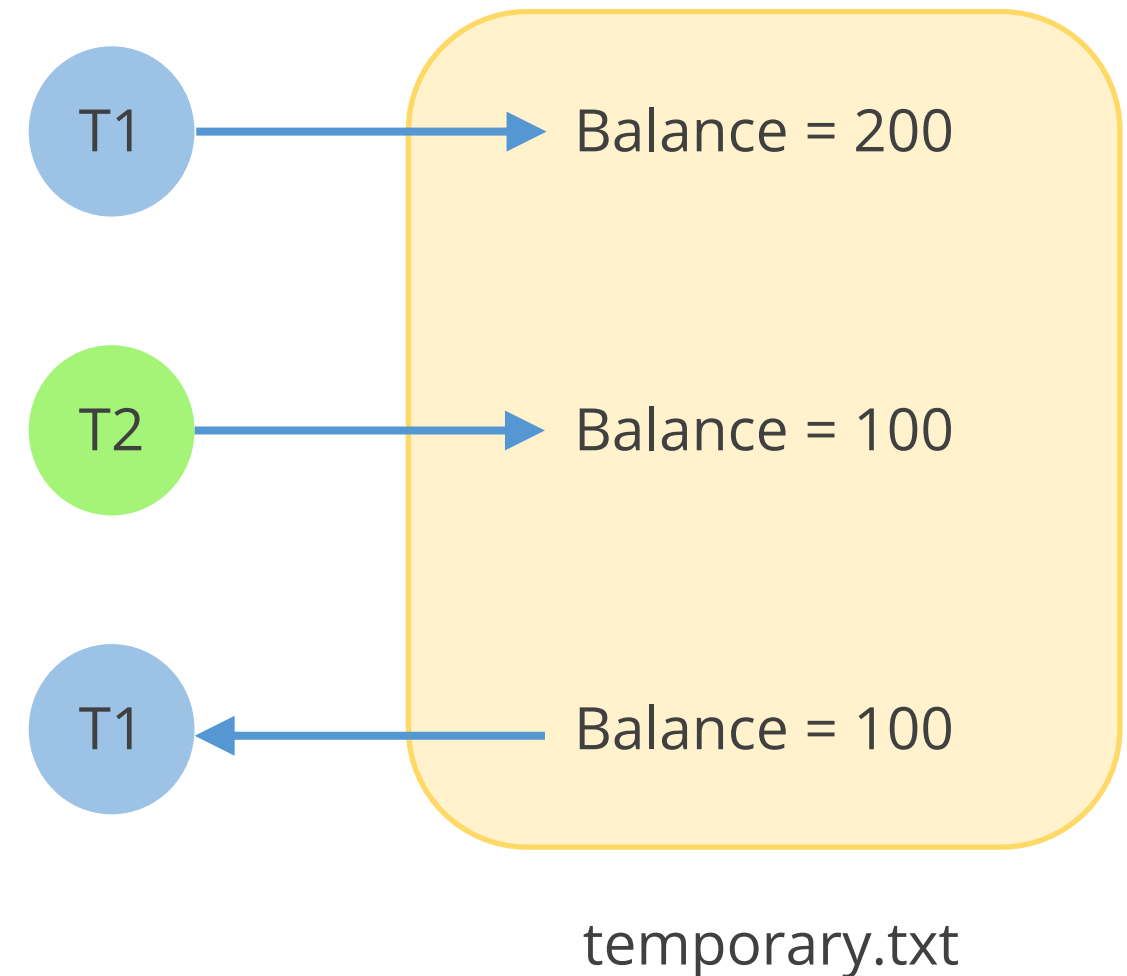
Improper synchronization might lead to a faulty program.

Synchronizing Threads: Example

Consider two different threads T1 and T2 and *temporary.txt* is a shared resource

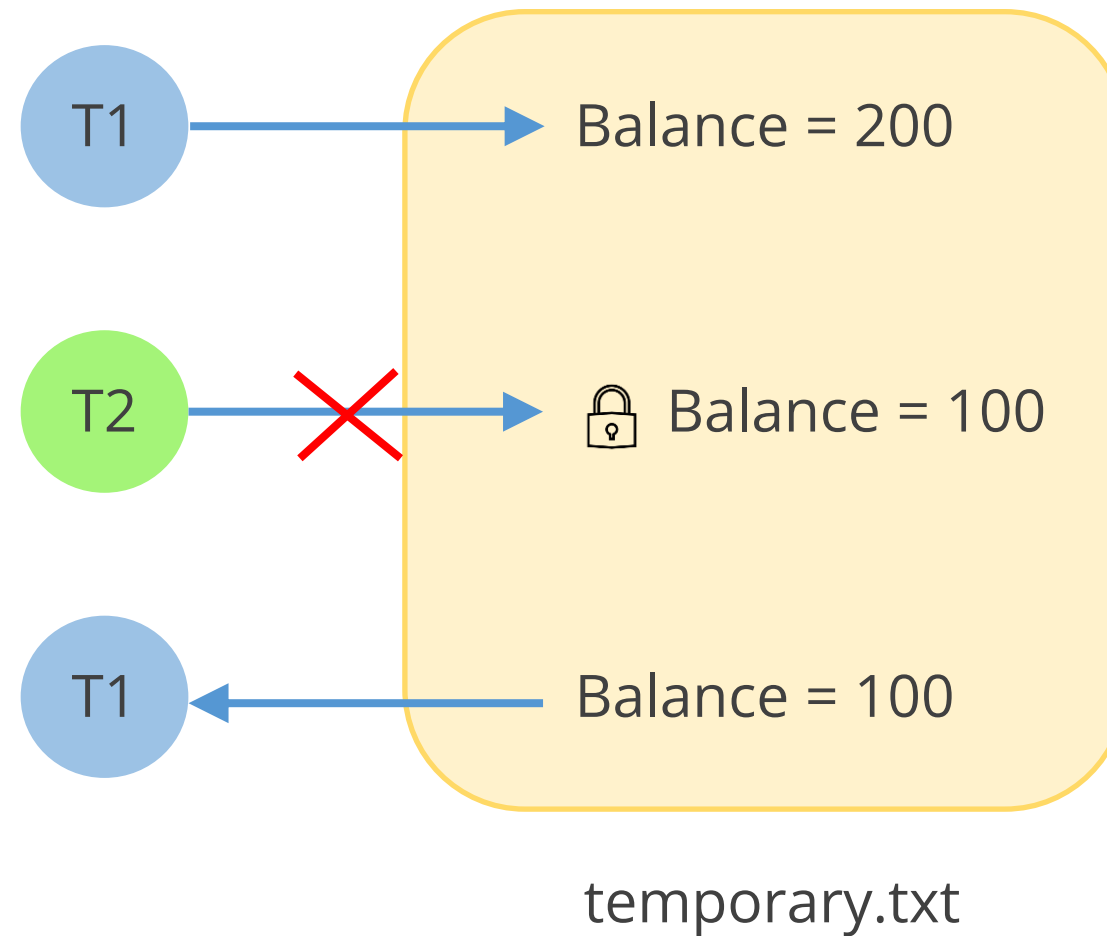
Example

- When T1 begins its execution, it saves certain values in a file called *temporary.txt* that will be used to compute a result when T1 has finished.
- Before T1 returns, T2 begins, and T2 modifies the values that T1 had stored in the file *temporary.txt*.
- T1 will hence produce 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, and no other thread will be able to access or modify it until thread T1 returns.



Synchronization in Python

The *Threading* module has built-in locking functionality that allows you to synchronize threads.

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

The *lock* object *acquire(blocking)* method is used to enable threads to execute synchronously.

The *lock* object can be used to generate 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, then a thread will return instantly with a value of 0 if the lock can't be obtained and a value of 1 if it can.
- If blocking is set to 1, the thread blocks and awaits the release of the lock.

Synchronization in Python: release()


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

release()


- The `release()` function is used to release the lock when it is no longer necessary.
- When the lock is locked, unlock it and then return.
- Allow just one thread to proceed if multiple threads are waiting for the lock.

Use of Synchronization in Python

To prevent concurrent access to an object, we must utilize a *lock* object.



Python's built-in data structures, such as lists and dictionaries, are thread-safe as a result of their use of atomic byte codes.



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

Synchronizing Threads

Duration: 15 minutes

- Why is the synchronization of threads needed?

Answer: Synchronization is done to make sure that only one thread is using a resource at a time when multiple threads are trying to access and modify a shared resource.

- How is synchronization of threads done in Python?

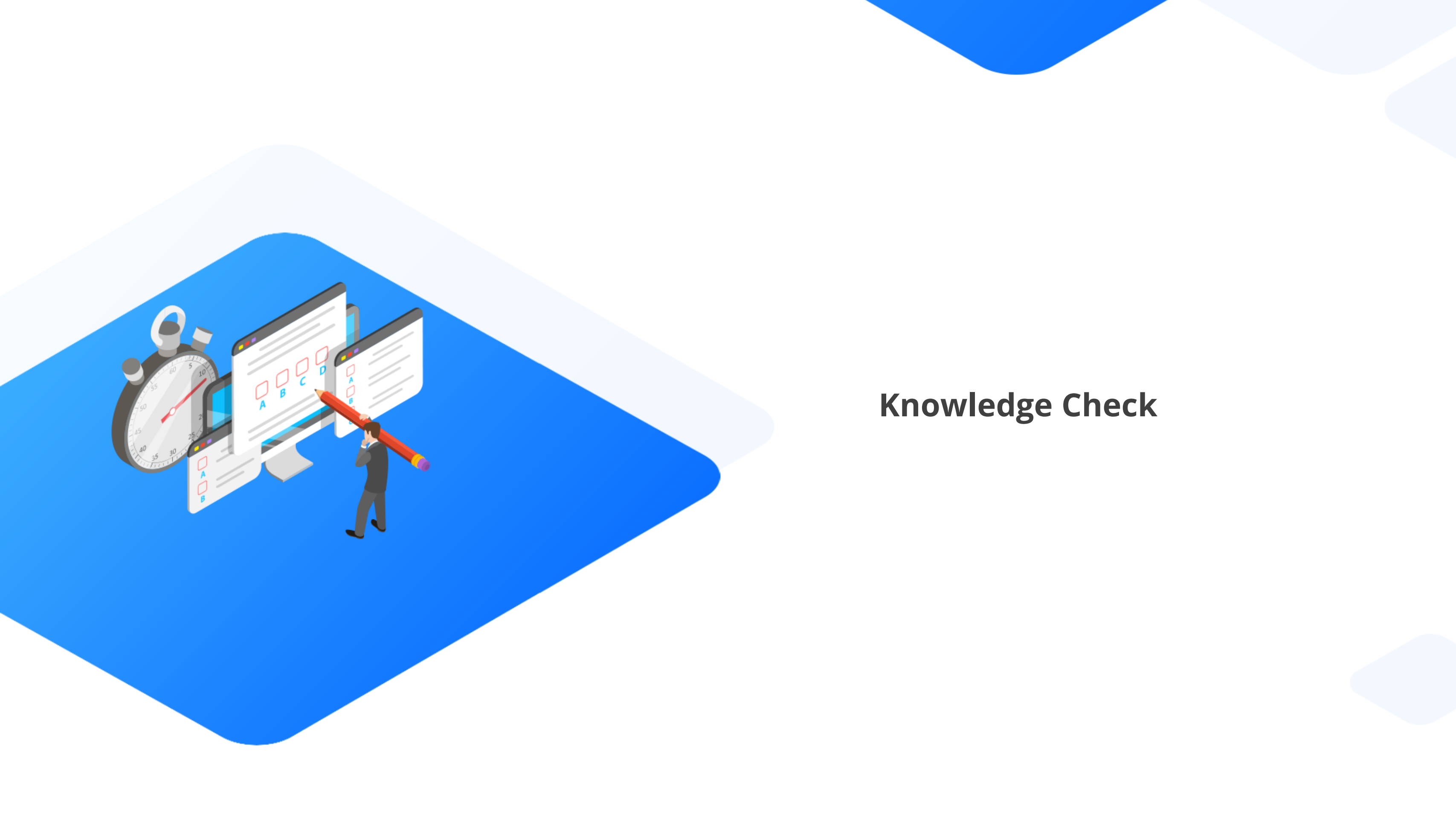
Answer: The Threading module has built-in locking functionality that allows for the synchronization of threads.



Key Takeaways

- Threading is making use of idle processes to give the appearance of parallel programming.
- A thread is the smallest unit of execution with an independent set of instructions
- Python 3 provides a threading module to implement threads
- Synchronization is used to ensure the sharing of resources between multiple threads, which will not cause loss of data.
- Synchronization is achieved using 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 join() method in Thread class?

- A. The join() method helps threads to 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.



Knowledge Check

2

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

- A. The join() method helps threads to 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.



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 is used to unlock the lock?

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



Knowledge Check

3

Which of the following functions is used to unlock the lock?

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

The correct answer is **B**

The `release()` function is used to release the lock when it is no longer necessary.





Thank You