







Select lesson text to explore the Explain with AI Premium Offering

> )

# Lock

Python's Lock is the equivalent of a Mutex and this lesson looks at its various uses.

### Lock

Lock is the most basic or primitive synchronization construct available in Python. It offers two methods: acquire() and release(). A Lock object can only be in two states: locked or unlocked. A Lock object can only be unlocked by a thread that locked it in the first place.

A Lock object is equivalent of a mutex that you read about in operating systems theory.

### **Acquire**

Whenever a Lock object is created it is initialized with the unlocked state. Any thread can invoke acquire() on the lock object to lock it. Advanced readers should note that acquire() can only be invoked by a single thread at any point because the GIL ensures that only one thread is being executed by the interpreter. This is in contrast to other programming languages with more robust threading models where multiple threads could be executing on different cores and theoretically attempt to acquire a lock at exactly the same time.

If a Lock object is already acquired/locked and a thread attempts to acquire() it, the thread will be blocked till the Lock object is released. If the caller doesn't want to be blocked indefinitely, a floating point timeout value can be passed in to the acquire() method. The method returns true if the lock is successfully acquired and false if not.

### Release

The release() method will change the state of the Lock object to unlocked and give a chance to other waiting threads to acquire the lock. If multiple threads are already blocked on the acquire call then only one arbitrarily chosen (varies across implementations) thread is allowed to acquire the Lock object and proceed.

## Example

?

An example is presented below where two threads share a list and one of the threads tries to modify the list while the other attempts to read it. Using a lock object we make sure that the two threads share the list in a thread-safe manner.

```
sharedState = [1, 2, 3]
my_lock = Lock()
```

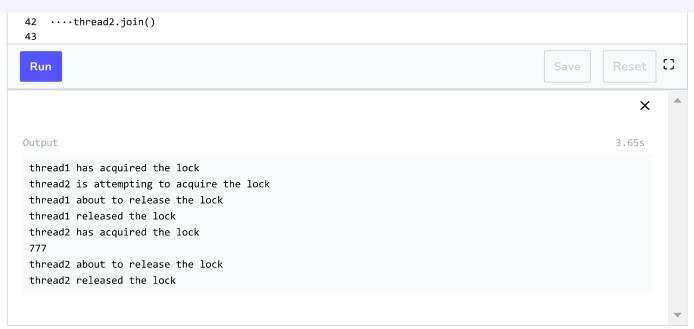
#### Select lesson text to explore the Explain with AI Premium Offering

```
my lock.acquire()
    print("{0} has acquired the lock".format(current_thread().getName()))
    time.sleep(3)
    sharedState[0] = 777
    print("{0} about to release the lock".format(current_thread().getName()))
    my lock.release()
    print("{0} released the lock".format(current_thread().getName()))
def thread2_operations():
    print("{0} is attempting to acquire the lock".format(current_thread().getName()))
    my_lock.acquire()
    print("{0} has acquired the lock".format(current_thread().getName()))
    print(sharedState[0])
    print("{0} about to release the lock".format(current_thread().getName()))
    my_lock.release()
    print("{0} released the lock".format(current_thread().getName()))
if __name__ == "__main__":
    # create and run the two threads
    thread1 = Thread(target=thread1_operations, name="thread1")
    thread1.start()
    thread2 = Thread(target=thread2_operations, name="thread2")
    thread2.start()
    # wait for the two threads to complete
    thread1.join()
    thread2.join()
```

```
13
14
        time.sleep(3)
        sharedState[0] = 777
15
16
        print("{0} about to release the lock".format(current_thread().getName()))
17
18
        my_lock.release()
        print("{0} released the lock".format(current_thread().getName()))
19
20
21
22
   def thread2 operations():
        print("{0} is attempting to acquire the lock".format(current_thread().getName()))
23
24
        my_lock.acquire()
25
        print("{0} has acquired the lock".format(current_thread().getName()))
26
27
        print(sharedState[0])
        print("{0} about to release the lock".format(current_thread().getName()))
28
29
        my_lock.release()
        print("{0} released the lock".format(current_thread().getName()))
30
31
32 if·__name__·==·"__main__":
                                                                                                                6
33 ····#·create·and·run·the·two·threads
34 ····thread1·=·Thread(target=thread1_operations, ·name="thread1")
35
   ....thread1.start()
```

```
36
37 ····thread2·=·Thread(target=thread2_operations,·name="thread2")
38 ····thread2.start()
```

Select lesson text to explore the Explain with AI Premium Offering



If you examine the output from the above snippet, you'll notice that thread 2 blocks on acquire method and waits for thread 1 to release it before it can successfully acquire it.

# **Deadlock Example**

Consider the example below, where two threads are instantiated and each tries to invoke release() on the lock acquired by the other thread, resulting in a deadlock.

```
from · threading import *
   import time
 2
 3
 4
 5
   def thread_one(lock1, lock2):
 6
        lock1.acquire()
 7
        time.sleep(1)
 8
        lock2.release()
 9
10
11 def thread_two(lock1, lock2):
12
       lock2.acquire()
        time.sleep(1)
13
        lock1.release()
14
15
16
17 if name == " main ":
18
       lock1 = Lock()
19
       lock2 = Lock()
20
21
       t1 = Thread(target=thread_one, args=(lock1, lock2))
        t2 = Thread(target=thread_one, args=(lock1, lock2))
                                                                                                               6
22
23
24
       t1.start()
        +2 -+--+/1
```

