

## Synchronization

```
In [1]: import threading
import time
```

```
In [2]: num_threads = 5
iterations_in_one_thread = 100
```

```
In [3]: counter = 0    # just a variable
```

```
In [4]: def f():
    global counter

    for i in range(iterations_in_one_thread):
        v = counter
        time.sleep(0.000000000000001) # Do some stuff
        v += 1
        counter = v
```

```
In [5]: def run_experiment():
    global counter
    counter = 0
    threads = []
    for i in range(num_threads):
        t = threading.Thread(target=f)
        threads.append(t)
        t.start()

    for i in threads:
        i.join()

    print("Calculated value: %d" % counter)
    print("Expected value:  %d" % (num_threads * iterations_in_one_thread))
```

```
In [6]: run_experiment()

Calculated value: 100
Expected value:  500
```

## Expectation versus Reality

This is what we expect will happen.

```
In [7]: counter = 0

v1 = counter
v1 += 1
counter = v1

v2 = counter
v2 += 1
counter = v2

print(counter)
```

2

This is what really happens.

Beyond Basic Programming - Intermediate Python

recluze.net/learn

```
In [9]: counter = 0

# --- thread 1 start
v1 = counter
v1 += 1
# ---- break

# --- thread 2 start
v2 = counter
v2 = 100
counter = v2
# ---- thread 2 done

# ---- back to thread 1
counter = v1
# ---- thread 1 done

print(counter)
```

1

## Synchronization

What we need is some way of ensuring that **critical sections** are executed all in one go.

```
In [10]: lock = threading.Lock()
```

```
In [11]: def f():
          global counter

          for i in range(iterations_in_one_thread):
              lock.acquire()  # begin critical part
              v = counter
              time.sleep(0.000000000000001)  # Do some stuff
              v += 1
              counter = v
              lock.release()  # end critical part
```

```
In [12]: run_experiment()
```

```
Calculated value: 500
Expected value:   500
```

But we will surely forget to release the lock ... just as we would forget to close open files. Well, context managers to the rescue!

```
In [ ]: def f():
          global counter, lock

          for i in range(iterations_in_one_thread):
              with lock:  # begin critical context
                  v = counter
                  time.sleep(0.000000000000001)  # Do some stuff
                  v += 1
                  counter = v
```

```
In [ ]: run_experiment()
```

You do have to figure out which part you **don't** want breaks in!

It's much more common to figure out which parts you **do** want the thread suspended at!

