More About Locks
Remember that we can make a lock with

```python
L = Lock()
```
and we can pass L to a function. If the function has code that we don't want multiple processes to be executing at the same time we can prevent that with

```python
def func(lock):
    <safe code>
    lock.acquire()
    <dangerous code>
    lock.release()
    <more safe code>
```
What is the best version of the following code?

A

```python
def func(r, lock):
    lock.acquire()
    for i in range(10):
        r.value = r.value + 1
    lock.release()
```

B

```python
def func(r, lock):
    for i in range(10):
        lock.acquire()
        r.value = r.value + 1
        lock.release()
```

C

```python
def func(r, lock):
    lock.acquire()
    for i in range(10):
        r.value = r.value + 1
```

D: Multiple options are equally good.
Will this guarantee the integrity of r?

```python
def func(r):
    lock = Lock()
    for i in range(10):
        lock.acquire()
        r.value = r.value + 1
    lock.release()
```

Why or why not?
What happens if we run this code with several processes, where lock is unused anywhere else in the program?

```python
def func(r, lock):
    for i in range(10):
        lock.acquire()
        r.value = r.value+1
```

A. Everything will be okay  
B. It will cause an error  
C. It will freeze up.  
D. I don't know
What happens if we run this code with just one process, where lock is unused anywhere else in the program?

def func(r, lock):
    for i in range(10):
        lock.acquire()
        r.value = r.value + 1

A. Everything will be okay  
B. It will cause an error  
C. It will freeze up.  
D. I don't know
Moral: Always be careful to release any lock you acquire.
Deadlock is a state of a program in which several processes are each blocking the others from getting what they need to continue. This causes the program to freeze. If you aren't careful you can get into a deadlock if you are using multiple locks.
Which of these could cause deadlock?

A

```python
def func1(lock1, lock2)
    lock1.acquire()
    lock2.acquire()
    <some code>
    lock2.release()
    lock1.release()

def func2(lock1, lock2)
    lock2.acquire()
    lock1.acquire()
    <some code>
    lock1.release()
    lock2.release()
```

B

```python
def func1(lock1, lock2)
    lock1.acquire()
    lock2.acquire()
    <some code>
    lock1.release()
    lock2.release()

def func2(lock1, lock2)
    lock2.acquire()
    lock1.acquire()
    <some code>
    lock2.release()
    lock1.release()
```

C

```python
def func1(lock1, lock2)
    lock1.acquire()
    <some code>
    lock1.release()

def func2(lock1, lock2)
    lock2.acquire()
    lock1.acquire()
    <more code>
    lock2.release()
    lock1.release()
```

D: Several cause deadlock