CS61A Notes – Week 14 – Concurrency Solutions

Questions:

```
>>> def make_withdraw(balance):
        def withdraw(amount):
            nonlocal balance
            if amount > balance:
                print('Insufficient funds')
            else:
                balance = balance - amount
                print(balance)
        return withdraw

>>> w = make_withdraw(10)
```

Question 1)
What are all the possible pairs of printed statements that could arise from executing the following 2 lines? Assume that the 2 function calls are executed in parallel.

```
>>> w(8)
>>> w(7)
```

[2, “Insufficient...”] [3, “Insufficient...”] [2,3] [3,2] [2,2] [3,3] [2,-5] [3,-5]

Question 2)
Suppose that Steven, Aki, and Eric decide to pool some money together:

```
>>> balance = 100
```

Now suppose: Steven deposits $10, Aki withdraws $20, and Eric withdraws half of the money in the account by executing the following commands:

Steven: `balance = balance + 10`
Aki: `balance = balance - 20`
Eric: `balance = balance - (balance / 2)`

a. List all the different possible values for `balance` after these three transactions have been completed, assuming that the banking system forces the three processes to run sequentially in some order.

Note that the original code was incorrect for Aki’s code. It should read balance = balance – 20
Correct answers are: 35, 40, 45, 50

b. What are some other values that could be produced if the system allows the processes to be interleaved?

There are a lot of answers to this one. Three answers are 110, 80, and 50.

Protecting shared state

Locks

Question 1)

Protect the critical section of the `make_withdraw` function by acquiring and releasing the lock.

```
    >>> from threading import Lock

    >>> def make_withdraw(balance):

            balance_lock = Lock()

            def withdraw(amount):

                nonlocal balance

               balance_lock.acquire()

                if amount > balance:

                    print("Insufficient funds")

                else:

                    balance = balance - amount

                    print(balance)

                balance_lock.release()
```

Question 2)

What are the possible pairs of printed values if the following code is now run?

```
   >>> w = make_withdraw(10)
    >>> w(8) #these 2 lines are executed in parallel
    >>> w(7) #these 2 lines are executed in parallel
```

[2, “Insufficient...”] [3, “Insufficient...”]

Deadlock

Deadlock is a situation that occurs when two or more processes are stuck, waiting for each other to finish.

Fill in the following code such that if `compute()` and `anti_compute()` are run in parallel, then deadlock might occur.

```
>>> x_lock = Lock()
>>> y_lock = Lock()
>>> x = 1
>>> y = 0
>>> def compute():
       _____x_lock.acquire()______________
       _____y_lock.acquire()______________
        y = x + y
        x = x * x
       _____x_lock.release()______________
       _____y_lock.release()______________

>>> def anti_compute():
       _____y_lock.acquire()______________
       _____x_lock.acquire()______________
        y = y - x
        x = sqrt(x)
       _____y_lock.release()______________
       _____x_lock.release()______________
```