QUESTIONS

1.) What Would Python Print (WWPP)
For the following exercises, write down what Python would print. If an error occurs, just write 'Error', and briefly describe the error.

a.)
```python
>>> name = 'rose'
>>> def my_func():
...     name = 'martha'
...     return None
>>> my_func()
>>> name
______ ?
```

[Solution]:
rose

b.)
```python
>>> name = 'ash'
>>> def abra(age):
...     def kadabra(name):
...         def alakazam(level):
...             nonlocal name
...             name = 'misty'
...             return name
...         return alakazam
...     return kadabra
>>> abra(12)('sleepy')(15)
______ ?
>>> name
______ ?
```

[Solution]:
misty
ash

c.)
```python
>>> ultimate_answer = 42
>>> def ultimate_machine():
...     nonlocal ultimate_answer
...     ultimate_answer = 'nope!
...     return ultimate_answer
>>> ultimate_machine()
______ ?
>>> ultimate_answer
______ ?
```

[Solution]:
Error (can't have a nonlocal variable refer to a global variable)
42
d.)

```python
>>> def f(t=0):
...     def g(t=0):
...         def h():
...             nonlocal t
...             t = t + 1
...             return h, lambda: t
...         h, gt = g()
...         return h, gt, lambda: t
...
>>> h, gt, ft = f()
>>> ft(), gt()

>>> h()
>>> ft(), gt()
```

[Solution]:
(0, 0)
(0, 1)

---

**More Environment Diagrams!**

1.) Draw the environment diagram for each of the following, and write return values where prompted:

a.)

```python
x = 3

def boring(x):
    def why(y):
        x = y
    why(5)
    return x

def interesting(x):
    def because(y):
        nonlocal x
        x = y
    because(5)
    return x

interesting(3)

boring(3)
```
[Solution]:

First:

Then:
b.)

def make_person(name):
    def dispatch(msg):
        nonlocal name
        if msg == 'name':
            return name
        elif msg == 'tom-ify':
            name = 'tom'
        else:
            print("wat")
        return dispatch

>>> phillip = make_person('phillip')
>>> phillip('tom-ify')
>>> phillip('name')

[Solution]:
tom
2.) Recall that, earlier in the semester, we represented numerical sequences as a function of one argument (taking in the index). If we want to process the elements of such a sequence, it'd be nice to have a function that 'remembers' where in the sequence it is in.

For instance, consider the `evens` function:

```python
def evens(n):
    """ Return the n-th even number. """
    return 2 * n
```

I’d like to have a function `generate_evens` that spits out the even numbers, one by one:

```python
>>> generate_evens = make_seq_generator(evens)
>>> for i in range(4):
...    print(generate_evens())
0
2
4
6
```

Write a function `make_seq_generator` that, given a function `fn`, returns a new function that returns the elements of the sequence one by one (like in the above example):

**[Solution]:**

```python
def make_seq_generator(seq_fn):
    cur_idx = 0
    def generator():
        nonlocal cur_idx
        result = seq_fn(cur_idx)
        cur_idx += 1
        return result
    return generator
```
3.) Let’s implement counters, in the dispatch-procedure style!

a.) Write a procedure `make_counter` that returns a function that behaves in the following way:

```python
>>> counter1 = make_counter(4)
>>> counter2 = make_counter(42)
>>> counter1('count')
5
>>> counter1('count')
6
>>> counter2('count')
43
>>> counter2('reset')
0
>>> counter1('count')
7
```

To help jog your memory, here’s the skeleton of `make_counter`:

```python
def make_counter(start_val):
    def dispatch(msg):
        if msg == ...
            ...
        return dispatch

[Solution]:

def make_counter(start):
    value = start
    def dispatch(msg):
        nonlocal value
        if msg == 'count':
            value += 1
            return value
        elif msg == 'reset':
            value = 0
            return value
        else:
            return 'Unknown message'
    return dispatch
```
b.) Modify your answer to (a) to include support for a new message, 'clone', that returns a copy of the current counter. The clone should be independent of the original:

```python
def make_counter(start):
    value = start
    def dispatch(msg):
        nonlocal value
        if msg == 'count':
            value += 1
            return value
        elif msg == 'reset':
            value = 0
            return value
        elif msg == 'clone':
            return make_counter(value)
        else:
            return 'Unknown message'
    return dispatch
```