Lab 3: Midterm Review

> Adapted from cs61a of UC Berkeley.

Starter Files

Get your starter file by cloning the repository: https://github.com/JacyCui/sicp-lab03.git

```
git clone https://github.com/JacyCui/sicp-lab03.git
```

`lab03.zip` is the starter file you need, you might need to unzip the file to get the skeleton code.

```
unzip lab03.zip
```

`README.md` is the handout for this homework. `solution` is a probrab solution of the lab. However, I might not give my solution exactly when the lab is posted. You need to finish the task on your own first. If any problem occurs, please make use of the comment section.

Suggested Questions

Control

Q1: Unique Digits

Write a function that returns the number of unique digits in a positive integer.

> **Hints:** You can use `//` and `%` to separate a positive integer into its one's digit and the rest of its digits.
>
> You may find it helpful to first define a function `has_digit(n, k)`, which determines whether a number `n` has digit `k`.

```
def unique_digits(n):
    """Return the number of unique digits in positive integer n.

    >>> unique_digits(8675309) # All are unique
    7
    >>> unique_digits(1313131) # 1 and 3
    2
    >>> unique_digits(13173131) # 1, 3, and 7
    3
    >>> unique_digits(10000) # 0 and 1
    2
```

```
    >>> unique_digits(101) # 0 and 1
    2
    >>> unique_digits(10) # 0 and 1
    2
    """
    "*** YOUR CODE HERE ***"

def has_digit(n, k):
    """Returns whether K is a digit in N.
    >>> has_digit(10, 1)
    True
    >>> has_digit(12, 7)
    False
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q unique_digits --local
```

Q2: Ordered Digits

Implement the function `ordered_digits`, which takes as input a positive integer and returns `True` if its digits, read left to right, are in non-decreasing order, and `False` otherwise. For example, the digits of 5, 11, 127, 1357 are ordered, but not those of 21 or 1375.

```
def ordered_digits(x):
    """Return True if the (base 10) digits of X>0 are in non-decreasing
    order, and False otherwise.

    >>> ordered_digits(5)
    True
    >>> ordered_digits(11)
    True
    >>> ordered_digits(127)
    True
    >>> ordered_digits(1357)
    True
    >>> ordered_digits(21)
    False
    >>> result = ordered_digits(1375) # Return, don't print
    >>> result
    False

    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q ordered_digits --local
```

Q3: K Runner

An *increasing run* of an integer is a sequence of consecutive digits in which each digit is larger than the last. For example, the number 123444345 has four increasing runs: 1234, 4, 4 and 345. Each run can be indexed **from the end** of the number, starting with index 0. In the example, the 0th run is 345, the first run is 4, the second run is 4 and the third run is 1234.

Implement `get_k_run_starter`, which takes in integers `n` and `k` and returns the 0th digit of the `k`th increasing run within `n`. The 0th digit is the leftmost number in the run. You may assume that there are at least `k+1` increasing runs in `n`.

```
def get_k_run_starter(n, k):
    """
    >>> get_k_run_starter(123444345, 0) # example from description
    3
    >>> get_k_run_starter(123444345, 1)
    4
    >>> get_k_run_starter(123444345, 2)
    4
    >>> get_k_run_starter(123444345, 3)
    1
    >>> get_k_run_starter(123412341234, 1)
    1
    >>> get_k_run_starter(1234234534564567, 0)
    4
    >>> get_k_run_starter(1234234534564567, 1)
    3
    >>> get_k_run_starter(1234234534564567, 2)
    2
    """
    i = 0
    final = None
    while _____________________________:
        while _____________________________:
            _____________________________
        final = _____________________________
        i = _____________________________
        n = _____________________________
    return final
```

Use Ok to test your code:

```
python3 ok -q get_k_run_starter --local
```

Higher Order Functions

These are some utility function definitions you may see being used as part of the doctests for the following problems.

```
from operator import add, mul

square = lambda x: x * x

identity = lambda x: x

triple = lambda x: 3 * x

increment = lambda x: x + 1
```

Q4: Make Repeater

Implement the function `make_repeater` so that `make_repeater(func, n)(x)` returns `func(func(...func(x)...))`, where `func` is applied `n` times. That is, `make_repeater(func, n)` returns another function that can then be applied to another argument. For example, `make_repeater(square, 3)(42)` evaluates to `square(square(square(42)))`.

```
def make_repeater(func, n):
    """Return the function that computes the nth application of func.

    >>> add_three = make_repeater(increment, 3)
    >>> add_three(5)
    8
    >>> make_repeater(triple, 5)(1) # 3 * 3 * 3 * 3 * 3 * 1
    243
    >>> make_repeater(square, 2)(5) # square(square(5))
    625
    >>> make_repeater(square, 4)(5) # square(square(square(square(5))))
    152587890625
    >>> make_repeater(square, 0)(5) # Yes, it makes sense to apply the function zero times!
    5
    """
    "*** YOUR CODE HERE ***"

def composer(func1, func2):
    """Return a function f, such that f(x) = func1(func2(x))."""
```

```
    def f(x):
        return func1(func2(x))
    return f
```

Use Ok to test your code:

```
python3 ok -q make_repeater --local
```

Q5: Apply Twice

Using `make_repeater` define a function `apply_twice` that takes a function of one argument as an argument and returns a function that applies the original function twice. For example, if `inc` is a function that returns `1` more than its argument, then `double(inc)` should be a function that returns two more:

```
def apply_twice(func):
    """ Return a function that applies func twice.

    func -- a function that takes one argument

    >>> apply_twice(square)(2)
    16
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q apply_twice --local
```

Q6: Doge

Draw the environment diagram for the following code.

```
wow = 6

def much(wow):
    if much == wow:
        such = lambda wow: 5
        def wow():
            return such
        return wow
    such = lambda wow: 4
    return wow()

wow = much(much(much))(wow)
```

You can check out what happens when you run the code block using Python Tutor.

Q7: Environment Diagrams - Challenge

> These questions were originally developed by Albert Wu and are included here for extra practice. I recommend checking your work in PythonTutor after filling in the diagrams for the code below.

Challenge 1

Draw the environment diagram that results from executing the code below.

> Guiding Notes: Pay special attention to the names of the frames!
>
> Multiple assignments in a single line: We will first evaluate the expressions on the right of the assignment, and then assign those values to the expressions on the left of the assignment. For example, if we had `x, y = a, b`, the process of evaluating this would be to first evaluate `a` and `b`, and then assign the value of `a` to `x`, and the value of `b` to `y`.

```
def funny(joke):
    hoax = joke + 1
    return funny(hoax)

def sad(joke):
    hoax = joke - 1
    return hoax + hoax

funny, sad = sad, funny
result = funny(sad(1))
```

You can check out what happens when you run the code block using Python Tutor.

Challenge 2

Draw the environment diagram that results from executing the code below.

```
def double(x):
    return double(x + x)

first = double

def double(y):
    return y + y

result = first(10)
```

You can check out what happens when you run the code block using Python Tutor.

Self Reference

Q8: Protected Secret

Write a function `protected_secret` which takes in a `password`, `secret`, and `num_attempts`.

`protected_secret` should return another function which takes in a password and prints `secret` if the password entered matches the `password` given as an argument to `protected_secret`. Otherwise, the returned function should print "INCORRECT PASSWORD". After `num_attempts` incorrect passwords are used, the secret is locked forever and the function should print "SECRET LOCKED".

For example:

```
>>> my_secret = protected_secret("oski2021", "The view from the top of the Campanile.", 1)
>>> my_secret = my_secret("oski2021")
The view from the top of the Campanile.
>>> my_secret = my_secret("goBears!")
INCORRECT PASSWORD # 0 Attempts left
>>> my_secret = my_secret("zoomUniversity")
SECRET LOCKED
```

See the doctests for a detailed example.

```
def protected_secret(password, secret, num_attempts):
    """
    Returns a function which takes in a password and prints the SECRET if the password entered matches
    the PASSWORD given to protected_secret. Otherwise it prints "INCORRECT PASSWORD". After NUM_ATTEMPTS
    incorrect passwords are entered, the secret is locked and the function should print "SECRET LOCKED".

    >>> my_secret = protected_secret("correcthorsebatterystaple", "I love UCB", 2)
    >>> my_secret = my_secret("hax0r_1") # 2 attempts left
    INCORRECT PASSWORD
    >>> my_secret = my_secret("correcthorsebatterystaple")
    I love UCB
    >>> my_secret = my_secret("hax0r_2") # 1 attempt left
    INCORRECT PASSWORD
    >>> my_secret = my_secret("hax0r_3") # No attempts left
    SECRET LOCKED
    >>> my_secret = my_secret("correcthorsebatterystaple")
    SECRET LOCKED
    """
    def get_secret(password_attempt):
        "*** YOUR CODE HERE ***"
    return get_secret
```

Use Ok to test your code:

```
python3 ok -q protected_secret --local
```

In the end, you can use doctest module to run all your doctest.

```
python3 -m doctest lab03.py
```