HIGHER-ORDER FUNCTIONS AND CODE WRITING

COMPUTER SCIENCE MENTORS

September 7 - September 11, 2020

1 Environment Diagrams

1. When do we make a new frame in an environment diagram?

2. Draw the environment diagram that results from running the following code.

```
def swap(x, y):
    x, y = y, x
    return print("Swapped!", x, y)

x, y = 60, 1
a = swap(x, y)
swap(a, y)
```

3. Draw the environment diagram that results from running the following code.

```
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(2))
```

2 Lambda

Lambda functions are essentially functions that you can define in one line. The structure of a lambda function is always `lambda arg0, arg1, ...: expr`. For example, these are functionally equivalent:

```
f = lambda x, y: x + y

def f(x, y):
    return x + y
```

Lambda functions can return anything that a regular function can return, including functions. This means that we can define higher-order lambda functions such as `lambda x: lambda y: x + y`

The above can be interpreted as:

```
def outer(x):
    def inner(y):
        return x + y
    return inner
```

Notice that the outermost lambda function takes in a parameter `x` and returns another lambda function which takes in a parameter `y` and returns `x + y`, which is exactly what the `outer` and `inner` functions do.

One important distinction from regular Python functions is that lambda functions are unnamed, so when we define frames for lambda functions in an environment diagram, we denote them using λ rather than with a function name.

If you want to define and use a lambda function in one line, you can do it like this:
`(lambda x, y: x + y)(3, 4)`

This will pass in `3` and `4` for `x` and `y` respectively and then return `3 + 4`.

1. Fill in the blanks (*without using any numbers in the first blank*) such that the entire expression evaluates to `9`.
   `(lambda x: lambda y: _____)(_____)(lambda z: z*z)()`

CSM 61A Fall 2020: Jason Chang and Jade Singh, with
Catherine Cang, Alina Trinh, Ethan Chiu, Jamie Ip, Kenneth Chi, and
Aidan Tong, Aman Shah, Bridget Cheng, Cindy Lin, Ethan Yeh, Ivan Penev, Ivy Li, James Fletcher, Jemmy Zhou, Jennifer Huang, Jessica Yu, Joshua Baum, Kevin Moy, Kunal Agarwal, Lauren Meier, Matthew Guo, Mihira Patel, Nikhita Anumukonda, Owen Gong, Richard Roggenkemper, Sean Sananikone, Uma Unni

3 Higher-Order Functions

1. Why and where do we use lambda and higher-order functions?

2. Draw the environment diagram that results from running the code.

```
x = 20
def foo(y):
    x = 5
    def bar():
        return lambda y: x - y
    return bar

y = foo(7)
z = y()
print(z(2))
```

CSM 61A Fall 2020: Jason Chang and Jade Singh, with
Catherine Cang, Alina Trinh, Ethan Chiu, Jamie Ip, Kenneth Chi, and
Aidan Tong, Aman Shah, Bridget Cheng, Cindy Lin, Ethan Yeh, Ivan Penev, Ivy Li, James Fletcher, Jemmy Zhou, Jennifer Huang, Jessica Yu, Joshua Baum, Kevin Moy, Kunal Agarwal, Lauren Meier, Matthew Guo, Mihira Patel, Nikhita Anumukonda, Owen Gong, Richard Roggenkemper, Sean Sananikone, Uma Unni

3. Draw the environment diagram that results from running the code.

```
apple = 4
def orange(apple):
    apple = 5
    def plum(x):
        return lambda plum: plum * 2
    return plum

orange(apple)("hiii")(4)
```

4. Write a function, `print_sum`, that takes in a positive integer, `a`, and returns a function that does the following:
   (1) takes in a positive integer, `b`
   (2) prints the sum of all natural numbers from 1 to `a*b`
   (3) returns a higher-order function that, when called, prints the sum of all natural numbers from 1 to `(a+b)*c`, where `c` is another positive integer.

```
def print_sum(a):
    """
    >>> f = print_sum(1)
    >>> g = f(2) # 1*2 => 1 + 2
    3
    >>> h = g(4) # (1+2)*4 => 1 + 2 + ... + 11 + 12
    78
    >>> i = h(5) # (3+4)*5 => 1 + 2 + ... + 34 + 35
    630
    """
    def helper(b):

        i, total = _____________________________

        while _____________________________:

            _____________________________

            _____________________________

        print(_____________________________)

        return _____________________________

    return _____________________________
```

5. Write a higher-order function that passes the following doctests.

   Challenge: Write the function body in one line.

```
def mystery(f, x):
    """
    >>> from operator import add, mul
    >>> a = mystery(add, 3)
    >>> a(4) # add(3, 4)
    7
    >>> a(12)
    15
    >>> b = mystery(mul, 5)
    >>> b(7) # mul(5, 7)
    35
    >>> b(1)
    5
    >>> c = mystery(lambda x, y: x * x + y, 4)
    >>> c(5)
    21
    >>> c(7)
    23
    """
```

6. What would Python display?

```
>>> foo = mystery(lambda a, b: a(b), lambda c: 5 + square(c))
>>> foo(-2)
```