HIGHER-ORDER FUNCTIONS, SELF REFERENCE, AND LAMBDAS

CS 61A

June 30, 2021

1 Higher-Order Functions: Environment Diagrams

Higher-order functions are functions that take in one or more functions as arguments, return a function, or do both. Since environment diagrams are a good tool to keep track of variables, values, and frames, we can use them to illustrate and understand the behavior of HOF's.

1. 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))
```

2. 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)
```

2 Higher-Order Functions: Lambda Expressions

A lambda expression evaluates to a function without binding it to a name. Just like with a def statement, the body of a lambda expression is not evaluated until the function is called. Note that some of the lambda expressions in these problems are also higher-order functions.

1. 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
    """
```

2. What would Python display?

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

3. 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)()
```

3 Higher-Order Functions: Self Reference

We use the term "self reference" to categorize a particular type of HOF in which the function returns itself, either directly or with the help of another function.

1. 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 _____
```