CS 61A Summer 2017

Higher-Order Functions

Mentoring 3: June 28, 2017

1 Higher-Order Functions

1.1 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))
```

1.2 What's the difference here?

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

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

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

1.4 Consider the following method.

```
def make_adder(x):
    def adder(n):
        return x + n
    return adder

make_adder(4)(5)
```

(a) What is the operator of the above expression?

(b) What are the operands?

(c) Draw the expression tree.

1.5 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
    """
```

1.6 What would Python display?

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

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

```
def dream1(f):
    kick = lambda x: mind()
    def dream2(secret):
        mind = f(secret)
        kick(2)
    return dream2

inception = lambda secret: lambda: secret
real = dream1(inception)(42)
```

1.8 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)()
```

1.9 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)
```