# HIGHER-ORDER FUNCTIONS AND RECURSION

COMPUTER SCIENCE MENTORS CS 61A

July 3, 2019

## 1 Higher-Order Functions

1. What is the difference between lambda functions and def statements? What is one of the purposes of 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))
```

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 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
    """
```

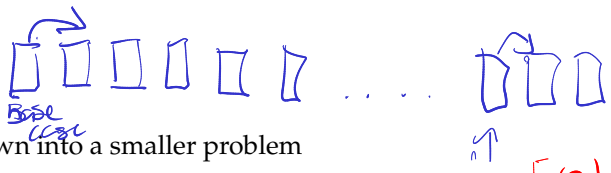
5. What would Python display?

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

CS61A Summer 2019: Richard Roggenkemper and Danelle Nachum, with
Ada Hu, Aini Xu, Caitlin Phan, Christine Luo, Grant Han, Jennifer Huang, Kaavya Shah, Kaitlyn Lee, Kevin Zhu, Kunal Agarwal, Raghav Singh, Rahul Arya, Rahul Devarapalli, Rina Lu

## 2 Recursion

Every Recursive function has three things.




1. One or more base cases
2. One or more ways to break the problem down into a smaller problem
   - E.g. Given a number as input, we need to break it down into a smaller number
3. Solve the smaller problem recursively; from that, form a solution to the original problem

1. What is wrong with the following function? How can we fix it?

```
def factorial(n):
    return n * factorial(n)
```

n-1

```
if n == 1:
    return n
else:
    return n * factorial(n-1)
```

```
if n == 0:
    return 1
```

2. Write a function `is_sorted` that takes in an integer `n` and returns true if the digits of that number are nondecreasing from right to left.

```
def is_sorted(n):
    """
    >>> is_sorted(2)
    True
    >>> is_sorted(22222)
    True
    >>> is_sorted(9876543210)
    True
    >>> is_sorted(98765432100)
    False
    """
```

Steps

What is the base case(s)? (case where the most trivial step is)

What is my recursive call?

`n % 10` ⟷ `n // 10`

`len(str(n)) == 1`

right = n % 10

rest = n // 10

not (rest % 10 ≥ right)

```
if n < 10:
    return True
elif (rest % 10) < right:
    return False
else:
    return is_sorted(rest)
```