HIGHER-ORDER ENVIRONMENTS, CURRYING, AND INTRODUCTORY RECURSION

COMPUTER SCIENCE MENTORS 61A

February 10 –February 14, 2025

1 Higher-Order Functions in Environment Diagrams cont.

1. Draw the environment diagram that results from running the code below.

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

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

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

```
def a(y):
    d = 1
    b = lambda x: y(x)
    e = lambda x: x(3)
    return e(b)
d = 5
a(lambda x: 4 - x + d)
```

There are three steps to writing a recursive function:

1. Create base case(s)

2. Reduce your problem to a smaller subproblem and call your function recursively to solve the smaller subproblem(s)

3. Use the subproblems' solutions as pieces to construct a larger problem's solution (This can happen in many layers!)

Real World Analogy for Recursion

Imagine that you're in line for boba, but the line is really long, so you want to know what position you're in.

You decide to ask the person in front of you how many people are in front of them. That way, you can take their response and add 1 to it to find your place. Now, the person in front of you is faced with the same problem that you were trying to solve, with one less person in front of them than you. They decide to take the same approach that you did by asking the person in front of them. This continues until the very first person in line is asked. At this point, the person at the front knows that there are 0 people in front of them, so they can tell the person behind them that there are 0 people in front. Now, the second person can figure out that there is 1 person in front of them, and can relay that back to the person behind them, and so on, until the answer reaches you.

Looking at this example, we see that we have broken down the problem of "how many people are there in front of me?" to 1 + "how many people are there in front of the person in front of me"? This problem will terminate with the person at the front of the line (with 0 people in front of them). Putting this into more formal terms, we are breaking down the problem into a **recurrence relationship**, and the termination case (when the question gets to the very first person in line) is called the **base case**.

As a program goes through recursion, it doesn't formally solve the problem until it reaches the base case, in which case it works its way up from the base case to the original input to construct your final answer. Just like your question as you stand in line, the program:

1. goes out (down the call stack), person by person (call by call) **(recursive case)**

2. until the person at the very front is reached **(base case)**

3. when the answers get added onto and eventually passed back to you, solving the problem. **(constructing the final solution with solutions to subparts)**

3. Here is a Python function that computes the nth Fibonnacci number. Identify the three parts of this recursive program.

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

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

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

5. Complete the definition for `num_digits`, which takes in a number n and returns the number of digits it has.

```python
def num_digits(n):
    """Takes in an positive integer and returns the number of
    digits.

    >>> num_digits(0)
    1
    >>> num_digits(1)
    1
    >>> num_digits(7)
    1
    >>> num_digits(1093)
    4
    """
```