RECURSION, TREE RECURSION AND DATA ABSTRACTION

COMPUTER SCIENCE MENTORS CS 61A

February 18, 2019 to February 20, 2019

1 Recursion

1. 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(9087654321)
    False
    """
```

2. (Spring 2015 MT1 Q3C) Implement the `combine` function, which takes a non-negative integer `n`, a two-argument function `f`, and a number `result`. It applies `f` to the first digit of `n` and the result of combining the rest of the digits of `n` by repeatedly applying `f` (see the doctests). If `n` has no digits (because it is zero), `combine` returns `result`.

```
def combine(n, f, result):
    """
    Combine the digits in non-negative integer n using f.

    >>> combine(3, mul, 2) # mul(3, 2)
    6
    >>> combine(43, mul, 2) # mul(4, mul(3, 2))
    24
    >>> combine(6502, add, 3) # add(6, add(5, add(0, add(2, 3)
       )))
    16
    >>> combine(239, pow, 0) # pow(2, pow(3, pow(9, 0))))
    8
    """
    if n == 0:
        return result
    else:
        return combine(_______ , _______ ,

                       _________________________________)
```

2 Tree Recursion

1. Mario needs to jump over a series of Piranha plants, represented as a string of 0's and 1's. Mario only moves forward and can either *step* (move forward one space) or *jump* (move forward two spaces) from each position. How many different ways can Mario traverse a level without stepping or jumping into a Piranha plant? Assume that every level begins with a 1 (where Mario starts) and ends with a 1 (where Mario must end up).
   Hint: Does it matter whether Mario goes from left to right or right to left? Which one is easier to check?

```
def mario_number(level):
    """
    Return the number of ways that Mario can traverse the
    level, where Mario can either hop by one digit or two
    digits each turn. A level is defined as being an integer
    with digits where a 1 is something Mario can step on and
    0 is something Mario cannot step on.
    >>> mario_number(10101)
    1
    >>> mario_number(11101)
    2
    >>> mario_number(100101)
    0
    """
    if ____________:

        __________

    elif ___________________:

        __________

    else:

        ____________________________________________________________
```

2. James wants to print this week's discussion handouts for all the students in CS 61A. However, both printers are broken! The first printer only prints multiples of `n` pages, and the second printer only prints multiples of `m` pages. Help James figure out whether or not it's possible to print exactly `total` number of handouts!

```
def has_sum(total, n, m):
    """
    >>> has_sum(1, 3, 5)
    False
    >>> has_sum(5, 3, 5) # 0 * 3 + 1 * 5 = 5
    True
    >>> has_sum(11, 3, 5) # 2 * 3 + 1 * 5 = 11
    True
    """
    if _____________________________________________________:

        return _____________________________________

    elif _____________________________________________________:

        return _____________________________________

    return _____________________________________________________
```

3. The next day, the printers break down even more! Each time they are used, the first printer prints a random `x` copies $50 \leq x \leq 60$, and the second printer prints a random `y` copies $130 \leq y \leq 140$. James also relaxes his expectations: he's satisfied as long as there's at least `lower` copies so there are enough for everyone, but no more than `upper` copies to prevent waste.

```
def sum_range(lower, upper):
    """
    >>> sum_range(45, 60) # Printer 1 prints within this range
    True
    >>> sum_range(40, 55) # Printer 1 can print a number 56-60
    False
    >>> sum_range(170, 201) # Printer 1 + 2 will print between
        180 and 200 copies total
    True
    """
    def copies(pmin, pmax):
        if ____________________________________________________:

            return ____________________________________________

        elif ____________________________________________________:

            return ____________________________________________

        return ____________________________________________________

    return copies(0, 0)
```

3 Data Abstraction

1. The following is an **Abstract Data Type (ADT)** for elephants. Each elephant keeps track of its name, age, and whether or not it can fly. Given our provided constructor, fill out the selectors:

```
def elephant(name, age, can_fly):
    """
    Takes in a string name, an int age, and a boolean can_fly.
    Constructs an elephant with these attributes.
    >>> dumbo = elephant("Dumbo", 10, True)
    >>> elephant_name(dumbo)
    "Dumbo"
    >>> elephant_age(dumbo)
    10
    >>> elephant_can_fly(dumbo)
    True
    """
    return [name, age, can_fly]
def elephant_name(e):




def elephant_age(e):




def elephant_can_fly(e):
```

2. This function returns the correct result, but there's something wrong about its implementation. How do we fix it?

```
def elephant_roster(elephants):
    """
    Takes in a list of elephants and returns a list of their
       names.
    """
    return [elephant[0] for elephant in elephants]
```

3. Fill out the following constructor for the given selectors.

```
def elephant(name, age, can_fly):
```

```
def elephant_name(e):
    return e[0][0]
def elephant_age(e):
    return e[0][1]
def elephant_can_fly(e):
    return e[1]
```

4. How can we write the fixed `elephant_roster` function for the constructors and selectors in the previous question?

5. **(Optional)** Fill out the following constructor for the given selectors.

```
def elephant(name, age, can_fly):
    """
    >>> chris = elephant("Chris Martin", 38, False)
    >>> elephant_name(chris)
        "Chris Martin"
    >>> elephant_age(chris)
        38
    >>> elephant_can_fly(chris)
        False
    """
    def select(command)
```

```
    return select
def elephant_name(e):
    return e("name")
def elephant_age(e):
    return e("age")
def elephant_can_fly(e):
    return e("can_fly")
```