TREE RECURSION

COMPUTER SCIENCE MENTORS 61A

September 19–September 23, 2022

In most recursive problems we've seen so far, the solution function contains only one call to itself. However, some problems will require multiple recursive calls—we call this type of recursion "tree recursion" because the propagation of function frames reminds us of the branches of a tree. Despite the fancy name, these problems are still solved the same way as those requiring a single function call: we define a base case, use a recursive call to solve a smaller subproblem, and then solve the original, larger problem with the solution to our subproblem. The difference? Instead of just using the solution to one subproblem, we may need to use multiple subproblems' solutions to solve our original problem.

Tree recursion will often be needed when solving counting problems (how many ways are there of doing something?) and optimization problems (what is the maximum or minimum number of ways of doing something?), but remember that there are all sorts of problems that may need multiple recursive calls!

1. The *Gibonacci sequence* is a recursively defined sequence of integers; we denote the nth Gibonacci number g_n. The first three terms of the sequence are $g_0 = 0, g_1 = 1, g_2 = 2$. For $n \geq 3$, g_n is defined as the sum of the previous three terms in the sequence.

   Complete the function `gib`, which takes in an integer `n` and returns the nth Gibonacci number, g_n. Also, identify the three parts of recursive function design as they are used in your solution.

```
def gib(n):
    """
    >>> gib(0)
    0
    >>> gib(1)
    1
    >>> gib(2)
    2
    >>> gib(3) # gib(2) + gib(1) + gib(0) = 3
    3
    >>> gib(4) # gib(3) + gib(2) + gib(1) = 6
    6
    """
    if _____:

        return _____

    return _____
```

Vibha Tantry and Jordan Schwartz, with
Gabe Classon, Aditya Balasubramanian, Oliver Yu, Hanze Tan, Gene Pan, Carolyn Wang, Raman Varma, Noor Haq, Angela Zhang, Abigail Brooks, Ted Kim, Aiden Joshua Vehemente, Marie Chorpita, Ashley Chiu, Bill Hu, Omar Yu, Angel Alberto Aldaco, Andrew Park, Chase Clements, Grace Yi, Preshtha Garg, Laksith Venkatesh Prabu, Aditya Murali, Irene Yang, Andy Chen, and Hailey Park

2. Including the original call, how many calls are made to `gib` when you evaluate `gib(5)`?

3. Gabe's Donut shop has an unlimited supply of f different flavors of donuts. Adit wants to buy a box containing d donuts. Complete the skeleton for the function `donut`, which determines the number of possible ways there are for Adit to select his d donuts from the f flavors. You may assume that `d` and `f` are non-negative integers.

   Hint: Does order matter?

```
def donut(d, f):
    """
    >>> donut(12, 1)
    1
    >>> donut(12, 2)
    13
    >>> donut(12, 12)
    1352078
    >>> donut(0, 0)
    1
    """
    if ____________________:

        return ____________________

    if ____________________:

        return ____________________

    return ________________________________________________
```

4. Mario needs to get from one end of a level to the other, but there are deadly Piranha plants in his way! Mario only moves forward and can either *step* (move forward one space) or *jump* (move forward two spaces) from each position. A level is represented as a series of ones and zeros, with zeros denoting the location of Piranha plants. Mario can step on ones but not on zeros. 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):
    """
    >>> mario_number(10101)
    1
    >>> mario_number(11101)
    2
    >>> mario_number(100101)
    0
    """
    if ____________________________:

        ____________________________

    elif ____________________________:

        ____________________________

    else:

        ____________________________________________________________
```

5. In an alternate universe, 61A software is not that good (inconceivable!). Tyler is in charge of assigning students to discussion sections, but sections.cs61a.org only knows how to list sections with either `m` or `n` number of students (the two most popular sizes). Given a `total` number of students, can Tyler create sections and not have any leftover students? Return `True` if he can and `False` otherwise.

```
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
    >>> has_sum(61, 11, 15) # can't express 61 as a * 11 + b * 15
    False
    """
    if _____________________________________________________:

        return True

    elif _____________________________________________________:

        return False

    return _____________________________________________
```

6. Implement the function `make_change`, which takes in a non-negative integer amount in cents `n` and returns the minimum number of coins needed to make change for `n` using 1-cent, 3-cent, and 4-cent coins.

```
def make_change(n):
    """
    >>> make_change(5) # 5 = 4 + 1 (not 3 + 1 + 1)
    2
    >>> make_change(6) # 6 = 3 + 3 (not 4 + 1 + 1)
    2
    """

    if _____________________:
        return 0

    elif _____________________:

        ________________________________________

    elif _____________________:

        ________________________________________
    else:

        ________________________________________
```