RECURSION

COMPUTER SCIENCE MENTORS 61A

September 12–September 16, 2022

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
3. Figure out how to use the smaller subproblem's solution in the larger problem's solution

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. 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 is called the **base case**.

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

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

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

2. Write a function `selective_sum`, which takes in an integer `n` and a predicate function `cond`. `selective_sum` returns the sum of all positive integers up to `n` for which `cond(n)` is true.

```
def selective_sum(n, cond):
    """
    >>> is_odd = lambda x: x % 2 == 1
    >>> selective_sum(5, is_odd) # 5 + 3 + 1 = 9
    9
    >>> bigger_than_10 = lambda x: x > 10
    >>> selective_sum(13, bigger_than_10) # 13 + 12 + 11 = 36
    36
    >>> selective_sum(-1, is_odd) # no positive integers <= 1
    0
    """
    if ______________________________________________:

        return ______________________________________________

    if ______________________________________________:

        return ______________________________________________

    return ______________________________________________
```

3. 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
    """
```

4. Fill in `collapse`, which takes in a non-negative integer `n` and returns the number resulting from removing all digits that are equal to an adjacent digit, i.e. the number has no adjacent digits that are the same.

```
def collapse(n):
    """
    >>> collapse(12234441)
    12341
    >>> collapse(11200000013333)
    12013
    """
    rest, last = n // 10, n % 10

    if ________________________________________:

        ________________________________________

    elif ______________________________________:

        ________________________________________
    else:

        ________________________________________
```

5. The *Mandelbrot sequence starting at* (a, b) is a sequence of points in the plane recursively defined by the following:

   - The first term of the sequence is (a, b).
   - If a term in the sequence is (x, y), then the following term is $(x^2 - y^2 + a, 2xy + b)$.

   For example, the first three terms of the Mandelbrot sequence starting at $(1, -1)$ are as follows:

$$(1, -1)$$
$$(1^2 - (-1)^2 + 1, 2(1)(-1) + -1) = (1, -3)$$
$$(1^2 - (-3)^2 + 1, 2(1)(-3) - 1) = (-7, -7)$$

   Write a higher order function `mandelbrot_seq` that accepts two numbers, `start_x` and `start_y`. `mandelbrot_seq` returns a function that takes two numbers `x` and `y` and returns the next term after `(x, y)` in the Mandelbrot sequence starting at `(start_x, start_y)`.

```
def mandelbrot_seq(start_x, start_y):
    """
    >>> seq = mandelbrot_seq(1, -1)
    >>> seq(1, -1)
    (1, -3)
    >>> seq(1, -3)
    (-7, -7)
    """
    def mandelbrot_next(x, y):

        return ____________________, _______________________

    return _____________________________
```

6. Write a function `in_or_out`, which returns `False` if any of the first `limit` terms of the Mandelbrot sequence starting at `(start_x, start_y)` is a distance of more than 2 away from the point $(0, 0)$, and `True` otherwise.

```
def in_or_out(start_x, start_y, limit):
    """
    >>> in_or_out(1, -1, 1) # (1, -1) dist is sqrt(2) < 2
    True
    >>> in_or_out(1, -1, 3) # (1, -3) dist is sqrt(10) > 2
    False
    >>> in_or_out(100, 100, 0) # no terms to consider
    True
    """
    next_term = ______________________________________________
    def helper(x, y, limit):

        if ______________________________________________:
            return True

        elif ______________________________________________:
            return False
        else:
            next_x, next_y = next_term(x, y)

            return ______________________________________________

    return ______________________________________________
```