Practice Activity: Recursion

Below is a collection of problems that can be solved recursively using short Python program. Note that in many cases, the problems can also be solved using iteration, built-in Python functions, or even a simple formula. Avoid using these approaches and focus on building a recursive solution instead.

When coding these problems, we suggest that you don't try to solve the problem all at once. Instead, adopt a recursive mindset in which you reduce the original problem to a simpler instance whose answer can then be used to solve the original problem. With this mindset, your function can construct this simpler instance of the problem and then call itself recursively. An important point here is to make sure that the recursive process eventually reduces the problem to an instance that is covered by one of your base cases.

1. Write a function `triangular_sum(num)` that computes the arithmetic sum $0 + 1 + 2 \ldots + (num - 1) + num$. For example, `triangular_sum(3)` should return 6. Note that this sum can be computed directly via a simple arithmetic formula, but use a recursive approach instead. Solution
2. Write a function `number_of_threes(num)` that returns the number of times the digit 3 appears in the decimal representation of the non-negative integer `num`. For example `number_of_threes(34534)` should return 2. Solution
3. Write a function `is_member(my_list, elem)` that returns `True` if `elem` is a member of `my_list` and `False` otherwise. For example, `is_member(['c', 'a', 't'], 'a')` should return `True`. Do not use any of Python's built-in list methods or an operator like `in`. Solution
4. Write a function `remove_x(my_string)` that takes the string `my_string` and deletes all occurrences of the character `'x'` from this string. For example, `remove_x("catxxdogx")` should return `"catdog"`. You should not use Python's built-in string methods. Solution
5. Write a function `insert_x(my_string)` that takes the string `my_string` and adds the character `'x'` between each pair of consecutive characters in the string. For example, `insert_x("catdog")` should return `"cxaxtxdxoxg"`. Solution
6. Write a function `list_reverse(my_list)` that takes a list and returns a new list whose elements appear in reversed order. For example, `list_reverse([2, 3, 1])` should return `[1, 3, 2]`. Do not use the `reverse()` method for lists. Solution
7. **Challenge:** Write a function `gcd(num1, num2)` that takes two non-negative integers and computes the greatest common divisor of `num1` and `num2`. To simplify the problem, you may assume that the greatest common divisor of zero and any non-negative integer is the integer itself. For an extra challenge, your programs should only use subtraction. **Hint:** If you get stuck, try searching for "Euclid's Algorithm". Solution
8. **Challenge:**Write a function `slice(my_list, first, last)` that takes as input a list `my_list` and two non-negative integer indices `first` and `last` satisfying $0 \leq first \leq last \leq n$ where n is the length of `my_list`. `slice` should return the corresponding Python list slice `my_list[first:last]`. For example, `slice(['a', 'b', 'c', 'd', 'e'], 2, 4])` should return `['c', 'd']`.**Important:**

Your solution should not use Python's built-in slice operator `:` anywhere in its implementation. Instead use the method `pop` to remove one element from the input list during each recursive call. (You may mutate the input list to simplify your solution.) Solution

Mark as completed