ASSIGNMENT 2B

December, 2023

Recursive problems

The rest of this week's assignment focuses on how to write recursive functions (Problems 7-11). Use the following questions to help you identify the structure of your recursive functions.

- What is the base case?
- What argument is passed to the recursive function call?
- How does this argument become closer to the base case?

Recursive exercises

Exercise 1

Write a function factorial(n: int) which takes an integer n and computes the factorial of that number. *Chapter 2 of the The Recursive Book of Recursion* has the solution and explanation for this exercise, but try doing it by yourself.

Exercise 2

Modify the function in Exercise 1 to only compute a modified version of the factorial, considering only odd numbers. The function factorial_odd(n: int) should take an integer n and computes the multiplication of all even numbers from 1 to n. Your function should be recursive.

Exercise 3

A palindrome is a word that is the same if you reverse the order of the letters (ex.: 'level'). Write a function is_palindrome(word: str) which takes a word and returns True if the word is a palindrome and False otherwise.

Hint: You can use a counter to navigate over each letter of the word. Look at how we can count every new call of a recursive function in function print_message() (Lecture 4 slides).

Hint 2: Remember you can use negative indexing to access a letter at the end of the work. For example "Christmas"[-2] is "a"

Chapter 3 of the The Recursive Book of Recursion has the solution and explanation for this exercise.

Problem 6 - Individual

The main code from Assignment 2A plots the maximum value of an array in red (either the array with random numbers or the one generated with np.arange()). Modify the main function to find the mean of the array, and plot all the points above the mean in red (instead of just the maximum value). Feel free to use any numpy function. (This is not a problem that you need to use recursion)

Problem 7 - Individual

Look at the code below, which implements the function sum_even(n: int) that takes an integer n as a parameter and returns the sum of all the even numbers from 0 to n.

```python
def sum(n: int) -> int:
    """
    A function calculating the sum of all positive even integers from 0 to n

    Parameters
    ----------
    n: int
        an integer

    Returns
    -------
    s: int
        the sum of every even number from 0 to n
    """
    s = 0
    for i in range(n+1):
        if i%2==0:
            s += n
    return s
```

Implement a similar function sum_even_rec(n: int), which does the same, but using recursion.

Problem 8 - Individual

Similarly to the array_sum_rec problem showed in the lecture, implement the function array_product_rec(numbers: np.ndarray[int]), which takes an array of integers and returns the total multiplied product of them.

Problem 9 - Think-Pair-Share

Implement a function concat_rec(words: list[str]) that takes an array of strings and returns those strings concatenated together into **a single string** with a single space between all words. For example concat_rec(["I", "love", "programming"]) should return "I love programming". Use recursion to implement your function.

*Note that your function should add a space in between strings, but **not** after the last word*

Problem 10 - Think-Pair-Share

With winter approaching, many traditions use evergreen trees as a reminder of the spring to come. With that in mind, let's implement a function that help us print the shape of a christmas tree in our terminal!

The function half_christmas_tree_rec(height : int), should return a **string** with '*' and '\n' that when printed forms a triangular pattern of '*'s that resemble the half of a christmas tree of height height.

For example, half_christmas_tree_rec(7) returns a **string**, which should look like this when printed:

```
*
**
***
****
*****
******
*******
```

Note that the top starts with only one star ('*'). Every line below it should have an increasing number of stars, until the bottom, with the same amount of start in the height argument. Implement the function using recursion.

Problem 11 - Think-Pair-Share

The function below returns the maximum value in a list.

```

def find_max(llist : list[int]) -> int:
    max_value = llist.pop()
    for value in llist:
        if value>max_value:
            max_value = value
    return max_value
```

Re-write the function to use recursion. The function output should be the same (i.e., it should still return the maximum value in a list) but the function code should have a call to itself.

Attention!

- Do not change the function definitions (names or parameter order) in the template file. You can change what a function returns.
- Try to comment your code the most you can and write the documentation of the functions.
- Your functions will be (mostly) automatically tested (similar to project Lovelace exercises). We have provided some tests. (Keep in mind the tests provided do not cover everything that is asked. The functions will be tested for edge cases, so make sure to deal with those cases inside your functions.