Week 4 - Assignment 2B

CHAPTER

VI

Warming up exercises

The rest of this week's assignment focuses on how to write **recursive functions**. Use the following questions to help you identify the structure of your recursive functions.

In this chapter:

Exercise 12.
Write a function factorial(n: int) which takes an integer n and computes the factorial of that number.

Exercise 13.
Modify the function in Exercise 12 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 odd numbers from 1 to n. Your function should be recursive.

Asking the following questions can help you identify the structure of a **recursive function:**

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

Exercise 14.
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.

Chapters 2 and 3 of the The Recursive Book of Recursion have the solution and explanation for some of those exercises. Compare it with your solution once you are done.

Remark. Use of Generative AI.
Remember that using generative AI to solve the exercises is not allowed. If you make use of tools like chatGPT for personal use, please avoid doing that during the workgroups. If you have questions, the TAs are there to you understanding the content.

Recursive Problems 1

In this assignment, you will revisit some array exercises and practice the use of recursion.

A function is **recursive** when it has a call to itself in the function definition..

Download the template `recursion_problems_1.py` in Brightspace and implement the following recursive functions. See below the description of each function.

- `sum_even_rec(n: int)`: write a **recursive function** that receive a positive integer n and returns the sum of all the even elements from 0 to n.

  Tip! The code below implements a similar function sum_even(n: int) that does the same without recursion. Try to modify that code to use recursion.

  – What would be the base case of your recursive function?

  – What would be the recursive case?

```
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 += i
    return s
```

- `array_product_rec(numbers: np.ndarray[int])`: which receives an array of integers as parameter and returns the multiplied product of all the elements in the array. **Use recursion to implement your function.**

  Tip! That function is similar to the array_sum_rec problem showed in the lecture.

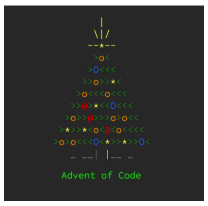
- `concat_rec(words: list[str])`: which receives an array of strings `words` as parameter and returns those strings concatenated together into **a single string** with a single space between all words.

For example `concat_rec(["I", "love", "programming"])` should return `"I love programming"`. **Use recursion to implement your function.**

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

- `half_christmas_tree_rec(height : int):`

With winter approaching, many traditions use evergreen trees as a reminder of spring to come. Let's write a function that helps us print the shape of a Christmas tree in our terminal!



if you like programming puzzles, the Advent of code is an Advent calendar of small programming puzzles for a variety of skill levels that can be solved in any programming language you like. The first puzzle starts December 1st at midnight!

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

> **Example.** `half_christmas_tree_rec(7)` returns a **string**, which should look like this when printed:

```
>>> half_christmas_tree_rec(7).py
*
**
***
****
*****
******
*******
```

Note that the top starts with only one star ('*'). Every line below it should have an increasing number of stars, until the last line, which has exactly the same amount of '*'s as the given `height`. **Implement the function using recursion**.

repeating strings: remember that you can repeat a give string using the multiplier operation. For example:

```python
laughs = 4*'ha'
print(laughs)
```

Output:
```
hahahaha
```

- `find_max_rec(my_list : list[int]):`

The function below returns the maximum value in a list.

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

Modify the function above into the function `find_max_rec` to use recursion.

Note that the function's input and output should be the same (ex: it should still receive a list and return the maximum value in the given list) but the function should necessarily have a call to itself.

Arrays and plotting

The `main()` function from `recursive_problems.py` **plots the values of 3 numpy arrays** corresponding to different mathematical functions/equations:

$$f_1(x) = -x^2 + 2 * x + 5$$

$f_2(x) = 10 * sin(2 * \pi * f * x)$, with frequency f as 2 Hz

$$f_3(x) = f_1 * f_2 * 0.01$$

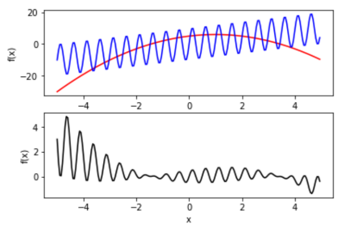
Modify the main function to also plot an 'x' marker at the maximum value of each array.

In addition to that, modify the last subplot (equation 3) to also plot in red all the values of the array that are above two times the mean.

Feel free to use any numpy functions (ex: np.where(), np.max(), np.argmax(), etc..).

Remark. You don't need to use recursion in this problem

Numpy arrays plots:



Submission instructions

Extra documents:
If you have worked together with another student, create a .txt file called `programming_partner.txt` containing the S-number of both of you.

Assignment 2B is part of Assignment 2, which has its deadline after Week 5. Submit your final solutions in Brightspace.

Example. `programming_partner.txt`

```
Recursive Problems 1:
s012345
s123456

```

Both students should submit their work individually on Brightspace.

General submission:
Create a zip file called `Assignment2` containing the files:

- `mini_proj_3.py`, with the following functions:
  - random_array()
  - element_mult()
  - find_max()
  - transpose()
  - is_magic()
  - is_square()
- `recursive_problems_1.py`, with the following functions:
  - sum_even_rec()
  - array_product_rec()
  - concat_rec()
  - half_christmas_tree_rec()
  - find_max_rec()
  - main()
- `programming_partner.txt`

Submit your file to Assignment 2, in Brightspace.