L13 Tasks

- L13-T1: Recursion: Sum of numbers #1
- L13-T1: Recursion: Sum of numbers #2
- L13-T3: Recursion: Fibonacci numbers
- L13-T4: Complexity issues
- L13-T5: Improved version of Fibonacci
- EXTRA: Comparison of running times

Note: In this week, the tasks are such that you are asked to write functions. For each task, you need also to write some sort of simple main program to demonstrate how your function works.

In this week, there are **no** CodeGrade tasks.

L13-T1: Recursion: Sum of numbers #1 (4 points)

Write a recursive function named `sum_of_list_recursive(numbers)` that calculates the sum of all elements in a list of integers or floats stored in the list `numbers`.

L13-T2: Recursion: Sum of numbers #2 (4 points)

Write a recursive function named `sum_of_digits_recursive(n)` that returns the sum of the digits of a non-negative integer `n`. For example, if `n = 123`, the function should return 6 (=1 + 2 + 3).

Think first about what the **base** case is. Then generate the rule for recursive calls.

L13-T3: Fibonacci numbers (4 points)

In 1202, Fibonacci proposed a puzzle in his book "Liber Abaci" about the growth of a rabbit population under ideal circumstances. The puzzle goes like this:

A man puts a pair of newly born rabbits (one male and one female) in a field. Rabbits take one month to mature before they can mate. One month after mating, the female gives birth to another pair of rabbits (one male and one female). The rabbits never die, and each pair produces a new pair every month from the second month on. The question posed by Fibonacci was: how many pairs of rabbits will there be after one year?

This scenario leads to the Fibonacci sequence, where each number is the sum of the two preceding ones, starting from 0 and 1. The sequence is defined by the recurrence relation:

$$F(n) = F(n-1) + F(n-2)$$

with initial conditions $F(0) = 0$ and $F(1) = 1$.

Write a recursive function `Fibonacci(n)` that that counts the Fibonacci number $F(n)$.

L13-T4: Complexity issues (4 points)

Run your function `Fibonacci(n)` with values n = 30, 35, 40, 45, ... Something strange seems to happen. Explain what the reason for this is.

L13-T5: Improved version of Fibonacci (4 points)

Write an improved version of the function `Fibonacci(n)` so that its running time is linear.

EXTRA: Comparison of running times (6 points)

In Moodle, you can find a file `pseudo_words.txt` that contains 100,000 randomly generated words of length 8. Store that file on your computer.

1. Write a function `read_words(file_name)` which reads the words from the file and returns a list containing those words.
2. Use the `deepcopy()` function from the `copy` module to take three ***deep copies*** of that list so that they do not mix.
3. Compare the running time of the standard Python function `sorted()`, and the functions `quicksort()` and `insertionSort()` introduced in the lecture material.
4. Use the `time` module to compute the running times of the methods. Output the results of your runs according to this example:

```
Time taken by sorted(): X.XXXXXX seconds
Time taken by quicksort(): X.XXXXXX seconds
Time taken by insertionSort(): X.XXXXXX seconds
```