# HOMEWORK 6

I made some remarks throughout the homework. Think about them but don't submit your answers. I am happy to talk about these remarks with you in person though!

## 1. Sorting and applications

Use sorting to solve the following problems.

**Problem 1.** Write a function named three_largest(alist) that takes a list of numbers as input. The function should return a list containing the three largest values from alist, sorted from lowest to highest. If the list contains fewer than three numbers, return the original list sorted from lowest to highest. For example

```
three_largest([8, 3, 1, 9, 12])
```

should return [8, 9, 12].

**Problem 2.** Write a function named find_median(alist) that takes a list of numbers alist as input. The function should return the median value of this list. Recall that the median is defined as follows

- If the list has an odd number of elements: The median is the middle value in the sorted list.
- If the list has an even number of elements: The median is the average of the two middle values.

For example

```
print(find_median([7, 1, 4]))
```

should return 4 and

```
print(find_median([7, 1, 4, 6]))
```

should return $\frac{4+6}{2} = 5$.

**Problem 3.** Write a function named sort_by_price(products) that takes a list of tuples, where each tuple contains a product name and its price (e.g., ("apple", 1.50)). The function should return a new list of these products sorted by their price. For example

```
sort_by_price([("banana", 1.5), ("apple", 2.91), ("orange", 2.25)
    ])
```

should return

```
[("banana", 1.5), ("orange", 2.25), ("apple", 2.91)].
```

To solve this problem, you need to create another function say, get_price(a_product) that takes a product as an input and returns its price. See the practice problem file for some more similar examples.

**Problem 4.** Write a function named sort_names(name_list) that takes a list of names name_list as input. The function should return a new list with these names sorted in alphabetical order, ignoring case sensitivity (recall that, in Python lowercase letters are greater than all uppercase letters.) For example

```
sort_names(["anne", "charlie", "David", "Armin"])
```

should return [”anne”, ”Armin”, ”charlie”, ”David”].

**Problem 5.** Write a function named sort_title(alist) that takes a list of book titles as input. The function should return a new list with these titles sorted by the number of words in each title. For example

```
alist = ["For whom the bell tolls",
         "The old man and the sea",
         "Please look after mom",
         "Siddhartha"]
print(sort_title(alist))
```

should return

```
["Siddhartha", "Please look after mom", "For whom the bell tolls",
    "The old man and the sea"].
```

Coincidentally, Siddhartha is also one of my favorite books. Check it out in your free time!

**Problem 6.** Write a function named choose_best_book(books_and_ratings) that takes a list of tuples, where each tuple contains a book title and its rating. The function should return the name of the book with the highest rating. For simplicity, assume that all the ratings are different. For example

```
books_and_ratings = [
    ("Siddhartha", 4.9),
    ("The Old Man and the Sea", 4.7),
    ("For Whom the Bell Tolls", 4.8),
    ("Please Look After Mom", 4.6)
]
print(choose_best_book(books_and_ratings))
```

should return ”Siddhartha”.

## 2. Recursion and recursive functions

**Problem 7.**

(1) Write a function named recursive_sum(alist) that takes a list of numbers as input and returns the sum of the squares of all elements in the list. For example

```
alist = [1,2, -3]
recursive_sum(alist)
```

should return $1^2 + 2^2 + (-3)^2 = 14$.

(2) Write another function non_recursive_sum(alist) to achieve the same goal.

**Problem 8.**

(1) Write a function name recursive_sum_power_of_two(n) that takes a positive integer $n$ as an input and returns the sum of the first $n$-powers of 2

$$1 + 2 + 2^2 + \ldots + 2^{n-1}.$$

For example

```
print(recursive_sum_power_of_two(4))
```

should return 15 because $2^0 + 2^1 + 2^2 + 2^3 = 15$.

(2) Write another non-recursive function to achieve the same goal.

**Remark 2.1.** Can you predict the general formula for the sum

$$1 + 2 + 2^2 + \ldots + 2^{n-1}.$$

**Problem 9.**

(1) Write a recursive function recursive_count_vowels(s) that takes a string s as input and returns the number of vowels (a, e, i, o, u) in s, ignoring case.

(2) Write a non-recursive function to achieve the same goal.

**Problem 10.**

(1) Write a recursive function recurive_sum_of_powers(n, k) that computes the sum of the first n integers raised to the power of k.

$$1^k + 2^k + \ldots + n^k.$$

For example

```
sum_of_powers(3, 2)
```

should return 14 because $1^2 + 2^2 + 3^2 = 14$.

(2) Write a non-recursive version to achieve the same goal.

(3) Use list comprehension and one of your functions to calculate the ratio

$$\frac{1^k + 2^k + \ldots + n^k}{n^{k+1}},$$

for $k = 3$ and $n \in [10, 10^2, 10^3, 10^4, 10^5, 10^6]$.

**Remark 2.2.** What do you notice about these ratios when $n$ gets bigger? Welcome to calculus!

**Problem 11.** An arithmetic progression is a sequence of numbers such that the difference from any succeeding term to its preceding term remains constant throughout the sequence. The constant difference is called the common difference of that arithmetic progression (often denoted by $d$). For example

$$1, 4, 7, 10, 13, \ldots$$

is an arithmetic progression with $d = 3$. An arithmetic sequence can be defined recursively as follows

$$a_0 = a, a_n = a_{n-1} + d.$$

In the above example, $a = 1, d = 3$.

(1) Write a recursive function arithmetic_sequence(a, d, n) that takes the first team (a), the difference (d), and the index (n) as input and returns the n-th term of the corresponding arithmetic sequence. For example

```
arithmetic_sequence(1, 3, 4)
```

should return 13.

(2) Write a non-recursive function to achieve the same goal. **Hint:** You can create a sequence $[a_0, a_1, \ldots, a_{n-1}]$ using the append method (creating this list is not absolutely necessary but you might find it more intuitive.)

**Problem 12.** The Lucas sequence is defined as follows, $L_0 = 2, L_1 = 1$ and for $n \geq 2$

$$L(n) = L(n-1) + L(n-2).$$

The first few terms of this sequence are

$$2, 1, 3, 4, 7, 11, \ldots$$

(1) Write a recursive function named lucas(n) that returns that $n$-th Lucas number.
(2) Write a non-recursive version to achieve the same goal.

So far, we have learned how recursion works for linear recursion. The same principle works for non-linear recursive relations as well.

**Problem 13.** Let $a_n$ be the sequence defined as follows

$$a(0) = 1, a(n) = a(n-1)^2 + 1.$$

The first few terms of this sequence are

$$1, 2, 5, 26, 677 \ldots$$

(1) Write a recursive function named sequence_a(n) that returns that $n$-th term of this sequence.
(2) Write a non-recursive version to achieve the same goal.

**Problem 14.** Let $a_n$ be the sequence defined as follows: $b_0 = b_1 = 1$ and

$$b_n = \frac{b_{n-1}^2 + 2}{b_{n-2}}.$$

The first few terms of this sequence are

$$1, 1, 3, 11, 41, 153.$$

(1) Write a recursive function named sequence_b(n) that returns that $n$-th term of this sequence.
(2) Write a non-recursive version to achieve the same goal.