Python: functions

- objectives: problem decomposition, abstraction and complexity handling, reuse
- variable scope and visibility

Details

- modifiable input parameters

Exercises in class

Exe. #22:

Q1: Are the two versions of the elaboration equivalent?

☒ No, they do not perform the same functionality
☐ Yes, they perform the same functionality BUT they are different in terms of performance/effort
☐ Yes, they perform the same functionality and are similar in terms of performance/effort

Exe. #23:

Two strings are *compatible* if they have the same length and contain the same character in the same position, except when there is a space.

Write a Python function that receives two strings as input and returns True if the strings are compatible, False otherwise. The strings contain only lowercase alphabetic characters and spaces. Examples:

```
seq1: "compatible"
seq2: "c mp t ble"
output: True

seq1: "compatible"
seq2: "c mp t bles"
output: False

seq1: "exam les"
seq2: "e amp es"
output: True
```

Version with first the case of equal size strings and computation, then the anomaly of two strings with different sizes.

Version with a `while` loop

Exe. #24:

Write a Python function that receives in input a list and modifies it so that all elements in an odd position if they are odd, are put to 0. Examples:

```
list in input: [1, 3, 5, 3, 4, 5, 5, 1, 0, 4, 7, 8, 10]
list at the end: [1, 0, 5, 0, 4, 0, 5, 0, 0, 4, 7, 8, 10]
```

More effective way, iterating only on odd elements.

Erroneous solution: it does not modify the value in the list

Exe. #25:

Write a Python function that receives a string in input, followed by three characters, `fromc`, `toc` and `replc`. The function computes and returns a new string where all characters included in the interval `[fromc, toc]` boundaries included, are replaced with the third character `replc`.

Considerations on the fact that not only a string is not modifiable, but the visibility of the parameter is limited to the function itself. Even if we assign a new value to the parameter, it will not be visible outside.

Proposed exercises

Proposed #34:

Write a function `validateParentheses` to determine if a given string contains balanced round parentheses. The function returns True if the string is valid, and False otherwise. Note that every opening parenthesis "(" has a corresponding closing parenthesis ")". Examples:

```
input: "((this is(a test)))"
output: True

input: ")another test("
output: False

input: "((this is)(a test)(again()))check"
output: False
```

Proposed #35:

Write a Python function that receives in input a strictly positive integer and computes and returns the list of its _proper divisor_s. Recall a *proper divisor* of a positive integer `n` is any positive divisor of `n` other than `n` itself.

Proposed #36:

Write a Python function that receives in input two strings that have a single character in common. The function finds and returns such a character.

Version 2: uselessly complex and expensive

Proposed #37:

Write a Python function that receives in input a string that represents a password. The password is considered valid if all three conditions are met: + it contains at least a capital letter and a small letter + it contains at least a digit + it does not have two adjacent identical characters

The function returns `True` or `False` based on the password being valid or not.

Proposed #38:

A list of lists describes a map representing the altitude of a territory. Each element of the data structure represents a land area and its value (a float) represents its average altitude in meters. *Definition*: A **maximum landslide risk point** is the point where there is the maximum elevation difference with respect to one of its *adjacent points*. *Adjacent points* are those elements whose coordinates differ by 1 in one or both indices (including diagonal neighbors). Consider as an example the map represented below:

90.63	37.81	31.08	66.97	72.15	14.50	89.89	84.26	84.60	21.83
90.41	65.41	77.28	42.58	36.68	24.04	67.79	89.28	90.34	69.86
24.98	10.92	83.42	5.52	28.58	75.20	83.78	88.33	32.92	80.16
39.19	23.55	17.98	70.28	90.52	90.13	84.78	80.41	74.39	69.38
45.02	85.73	4.82	28.45	91.25	33.40	3.65	75.03	21.20	36.57
55.20	60.92	60.12	73.17	31.20	50.64	63.30	15.98	31.05	37.69
85.36	33.29	2.49	20.15	12.81	9.87	91.63	16.37	85.03	52.39
10.27	30.06	38.12	15.09	58.50	29.37	48.48	62.15	4.40	70.21
98.72	59.60	31.12	58.84	32.77	62.32	9.48	**96.07**	78.30	40.53

The **maximum risk point** is the element in bold because the difference between 96.07 and 4.40 is the maximum in the entire data structure. Write a function `risk` that receives as input the terrain map and calculates and returns the coordinates of the maximum landslide risk position and the value of the maximum elevation difference.

Write a program that calls function `risk` passing a map and visualises the coordinates of the maximum risk point, the maximum difference and the altitude of the point