Functions

Solve the following exercises and upload your solutions to Moodle until the specified due date. Make sure to use the *exact filenames* that are specified for each individual exercise. Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are *not allowed* to use any concepts and modules that have not yet been presented in the lecture.

Exercise 1 – Submission: `a4_ex1.py` 20 Points

Write a function `split_list(lst: list, num_sublists: int) -> list` that splits a given list into (mostly) equally-sized sublists as follows:

- `num_sublists` defines how many sublists `lst` should be split into
- The function iteratively fills up all sublists while always keeping a maximum difference in sublist-size of 1
- If `num_sublists` is 0, the original `lst` should be returned.

Use a loop to solve this exercise.

Function call	Result
`split_list([0,1,2,3], 2)`	`[[0,2],[1,3]]`
`split_list([0,1,2,3], 1)`	`[[0,1,2,3]]`
`split_list([0,1,2,3,4,5,6,7], 3)`	`[[0,3,6],[1,4,7],[2,5]]`
`split_list([0,1,2,3,4,5,6,7], 0)`	`[0,1,2,3,4,5,6,7]`
`split_list([0,1,2,3,4], 2)`	`[[0,2,4],[1,3]]`

Table 1: Example function calls and results.

Exercise 2 – Submission: `a4_ex2.py` 15 Points

Write a function `clip(*values, min_=0, max_=1) -> list` that returns a list of clipped values based on arbitrary many input values `*values` (integers or floats), where clipping is defined as follows:

- If a value is smaller than `min_`, append `min_` to the list.
- If a value is bigger than `max_`, append `max_` to the list.
- Otherwise, append the value to the list.

If `*values` is empty, an empty list must be returned.

Function call	Result
`clip()`	`[]`
`clip(1, 2, 0.1, 0)`	`[1, 1, 0.1, 0]`
`clip(-1, 0.5)`	`[0, 0.5]`
`clip(-1, 0.5, min_=-2)`	`[-1, 0.5]`
`clip(-1, 0.5, max_=0.3)`	`[0, 0.3]`
`clip(-1, 0.5, min_=2, max_=3)`	`[2, 2]`

Table 2: Example function calls and results.

Exercise 3 – Submission: `a4_ex3.py` 20 Points

Write a function

```
grade_calculator(
        assignments: list,
        bonus_assignment: int,
        exam: int) -> tuple[bool, int]
```

that computes the grade of a student and whether they passed the course.

- The parameter `assignments` specifies a list of ten integer values which correspond to the points on each assignment. The parameters `bonus_assignment` and `exam` are integer values with respective points.
- The function should check the rules for passing the course and the resulting grade according to the rules outlined in `00_introduction.pdf`, pages 10 and 11.
- It should return a tuple of boolean and integer that tell whether the student passed the course and the grade they receive. The grade should be an integer between 1 and 5.
- Either element in `grade_calculator`, `bonus_assignment` and `exam` can be `None`. In that case, this should be regarded as 0 points for that value.

Function call	Result
`grade_calculator([95,100,39,13,86,71,20,100,83,100], None, 82)`	`(True, 3)`
`grade_calculator([95,100,39,13,86,71,20,100,83,100], 51, 82)`	`(True, 2)`
`grade_calculator([0,100,100,13,100,100,20,100,100,100], 0, 100)`	`(False, 5)`
`grade_calculator([0,100,100,13,100,100,20,100,100,100], 100, 100)`	`(True, 2)`
`grade_calculator([0,100,100,13,100,100,None,100,100,100], 100, 100)`	`(True, 2)`
`grade_calculator([100,100,100,100,100,100,100,100,100,100], 100, 49)`	`(False, 5)`

Table 3: Example function calls and results.

Hints:

- Make sure to return the grade and not the percentage points.
- Check that your solution is able to handle `None` anywhere in the parameters.

Exercise 4 – Submission: `a4_ex4.py` **20 Points**

Write a function `round_(number, ndigits: int)` that rounds a given number (integer or float) to `ndigits` precision. The function works as follows:

- If `ndigits` is `None`, the rounding is performed to 0 decimal places, and an integer is returned.
- In all other cases, a float is returned, and the rounding is performed to `ndigits` precision. You can assume `ndigits` to be ≥ 0.
- You are *not allowed* to use the built-in function `round`.

Function call	Result
`round_(777.777)`	`778`
`round_(777.777, 0)`	`778.0`
`round_(777.777, 1)`	`777.8`
`round_(777.777, 2)`	`777.78`
`round_(777.777, 3)`	`777.777`
`round_(777.777, 4)`	`777.777`

Table 4: Example function calls and results (results might differ slightly because of floating point arithmetic.

Hints:

- As mentioned above, the rounded results might not be exactly precise due to floating point arithmetic. For example, `round_(777.777, 1)` might result in `777.8` or `777.8000000000001` (or comparable numbers), depending on the implementation. Both results are fine.
- The module operator `%` might be useful to solve the task. For example, `some_float % 1` will return the fractional part of some floating point number, or `some_float % 0.1` the fractional part starting at the second decimal place. As mentioned above, the results obtained from the module operation might not be precise because how floating point arithmetic works, which is okay.

Exercise 5 – Submission: `a4_ex5.py` 25 Points

Write a function `sort(elements: list, ascending: bool = True)` that sorts the specified list in-place, i.e., the list is changed directly. The function does not return anything. The parameter `ascending` controls whether the list should be sorted in ascending or in descending order. To implement the sorting functionality, take a look at the insertion sort algorithm (you can find a visual example here (ascending order)):

- Insertion sort separates the list into a sorted part (at the beginning only the first element) and an unsorted part (at the beginning everything else).
- The algorithm takes the element at position $i = 1$, which is the first unsorted element, and inserts it at the correct position of the sorted part.
- Now, the first two elements are sorted and the rest is not.
- i is incremented and the process is repeated.

You are *not allowed* to use the built-in function `sorted` or the list method `some_list.sort()`. You do not need to implement any optimizations (of course, you are free to do so if you want).

Function call	Result
`sort(some_list)`	`some_list` has now the content `[0, 1, 3, 4, 5]`
`sort(some_list, ascending=False)`	`some_list` has now the content `[5, 4, 3, 1, 0]`

Table 5: Example function calls and results, given `some_list = [1, 3, 0, 4, 5]`.