INTRODUCTION TO PROGRAMMING, AUTUMN 2025, ONLINE EXAM 4

STARTED: 12:30 | ENDS: 16:30 | 0:00

This exam has ended. See your points below.

Exercise 1

Complete this in exercise template `exercise1.py`

Write a program, which asks user to type in strings one by one. When the user inputs an empty string, program outputs number of strings typed in, the length of the longest string and the most common character.

Sample execution of the program, where the words after "Type in a string: " are inputs from the user:

```
Type in a string: car
Type in a string: cat
Type in a string: dog
Type in a string: llama
Type in a string:
Total number of strings: 4
The length of the longest string: 5
The most common character in strings: a
```

The most common character in the sample execution of the program is `'a'` because it occurs four times in strings given as input. You can assume that there is only one suitable candidate for the most common character and that the user inputs at least one string.

Exercise 2

Complete this in exercise template `exercise2.py`

The exercise is to program a function `find_allowed`, which finds all words that contain at least the desired number of the following letters: "aeiouy".

The function has the following properties:

- The function is given 2 arguments.
- The first argument is a **list of strings.**
- The second argument is an **integer**, which is the minimum number of the searched letters.
- The program returns a **new list**, which contains all the words that have at least the requested number of desired letters.
- The program does not modify the original list.

Example of using the function:

```
wordlist = ["apple", "banana", "cherry", "orange", "peach", "pineapple"]
minimum = 3
```

```
result = find_allowed(wordlist, minimum)
print(result)
```

The output is:

```
['banana', 'orange', 'pineapple']
```

Exercise 3

Complete this in exercise template `exercise3.py`

Create a function called `read_points()` that reads the statistics of ice hockey teams from the file `statistics.txt`.

The file is read from the same folder as `exercise3.py`

Lines in the file represent the statistics of an ice hockey team, formatted as:

[team]:[wins]-[losses]-[ties]

For example, the string: "Kumpula Intelligence:0-8-1" describes a team called `Kumpula Intelligence` with 0 wins, 8 losses and 1 tie.

For example, a line in the file: "Kumpula Intelligence:0-8-1" describes a team called Kumpula `Intelligence` with 0 wins, 8 losses, and 1 tie.

The function `read_points()` calculates the total points from each line as follows: three points for a win, one point for a tie, and no points for a loss. Finally, the function returns a list containing the results in the form (team_name, points), that is, a list of tuples.

If the number of wins, losses, or ties found on a line cannot be read as integers, the function raises a ValueError with the message "Invalid format in the file: [file line]", where [file line] is the line from which the error was found.

You can assume that the file format is otherwise correct, meaning it has the appropriate number of colons and dashes, and that the integers are positive.

You can copy the following lines and add them exactly as they are to your test file `statistics .txt` for testing your program:

```
Heba hawks:5-6-1
Brewsters:3-12-10
Sleepers:0-0-0
KBC:6-2-1
Navy jerries:7-0-1
Loosisters:8-5-0
```

When the file `statistics.txt` consists of the lines above, the program returns the following list:

```
[('Heba hawks', 16),
('Brewsters', 19),
('Sleepers', 0),
('KBC', 19),
('Navy jerries', 22),
('Loosisters', 24)]
```

You may want to add or modify some lines that contain incorrect data to test your program's exception handling mechanism. Below are examples of lines that include deliberate errors:

```
KBC:AAA-1-ll
Loosisters:7-4.5-1
```

These example lines intentionally contain incorrect information (such as letters instead of numbers) to allow you to verify that your program identifies invalid inputs and handles them correctly with ValueError exceptions.

That is, the execution of the program ends in the first faulty line found. For example, the error that ends the execution of the program could be the following:

```
ValueError: Invalid format in the file: KBC:AAA-1-ll
```