Exercise 1

In this exercise, you will work with microbiome data from healthy patients and patients with Chron's disease. Which is often associated with disbiosis of the gut microbiota.

1a. Open the files Open the `bacteria_abundances_W7.txt` file. Print the first 2 rows. The columns contain the name of different bacterial species. The numbers in each following row contain the "abundance" of that bacteria in a sample. The first 5 samples correspond to healthy patients the next 5 to sick patients.

1b. Iterate over the lines For each patient, print the value corresponding to the most abundant bacteria.

1c. Create two new files In one file store the information for healthy patients, in the other the information for sick patients. Hint, you need the first row for both files.

1d. Open the files and store the information Open the new files and create two dictionaries, one containing the information on the healthy patients and one for the sick patients. The structure could be the following:

```
bacteria_list = ['Escherichia_coli', 'Faecalibacterium', 'Clostridium',
'Akkermansia', 'Enterococcus', 'Lactobacillus', 'Ruminococcus',
'Prevotella', 'Streptococcus', 'Bacteroides']
healthy_patients = {'healthy1' : [5,30,5,2,5,20,5,8,5,15],
```

and so on..

Alternatively, you are free to come up with your own structure if you want, as long as it functions with the next question.

print the relevant information on screen with a description of what you are showing.

1e. Find other bacteria We know already that morbus chron is associated with increased e.coli values. However, we also suspect that reduction of other bacteria may also play a role. To look into that, make a function that calculates the average for each bacteria in healthy and sick individuals. Can you spot differences?

EXTRA 1f. Calculate standard deviation Can you make a function that calculates the standard deviation? https://en.wikipedia.org/wiki/Standard_deviation Print the same information as above but add the standard deviation to the shown information.

Exercise 2

In this exercise, you will work with historical temperature data from the Deutscher Wetterdienst (DWD). This dataset contains seasonal average air temperatures for various regions in Germany starting from 1881.

2a. Download the Data

- Visit the DWD Data Portal.
- Download the data files for each season (`air_temperature_mean_summer.txt`, `air_temperature_mean_winter.txt`, etc.).
- Save these files in the same folder as your Python exercise script.

2b. Print File Content Preview

Write code that:

- Opens one of the downloaded files (e.g., `air_temperature_mean_summer.txt`).
- Reads and prints the first 1000 characters of the file content to get an overview of its structure.

2c. Extract Location Names

- Extract and print the names of the locations present in the file.
- You can identify location names by reading the first few lines of the file and extracting the header or analyzing the data entries.

2d. Store Data in Dictionaries

- Create four dictionaries for each season: `avg_temp_Summer`, `avg_temp_Winter`, `avg_temp_Spring`, and `avg_temp_Autumn`.

- Use the structure: `{'YEAR': ('location', temperature)}` or `{'YEAR': ['location', temperature]}`
- Print the contents of one of the dictionaries.

2e. Calculate Average Temperature for a Season

- Write a function `average_temperature_season(season_dict)` that calculates the average temperature across all regions for a given season.
- Print the average summer temperature for the years 1881 and 2024.

2f. Calculate Average Temperature for a Year

- Write a function `average_temperature_year(year, *season_dicts)` that calculates the average temperature across all four seasons for a given year.
- Print the average temperature for the year 1881 and for every 10 years after that.

Exercise 3

Here we will build a script that can run wordle. The famous game! The rules are simple: you have to guess a five-letter word in six tries. At each attempt, you receive information that tells you how close (or far) you are from your guess. https://wordly.org/

3a. Select a random word The words we will be working with as a database are the following:

```
words = ['biome', 'bases', 'lyase', 'zymin', 'roots', 'sample', 'local',
'loops']
```

Import the random module and make a function that returns a random word from a list. Print a random word from this list.

3b. Evaluate the guesses Make a function that given two strings (guess and answer) returns a list, corresponding to the status of the answer: 0 - the letter is not present in the answer 1 - the letter is present in the answer but at another location 2 - the letter is present in the answer at this location For example, if answer == 'roots' and the guess == 'boots' the function should return `[0, 2, 2, 2, 2]` if answer == 'roots' and guess == 'rests' the function should return `[2, 0, 1, 2, 2]`. Test the function.

3c. Create the loop The user has up to 6 tries, if the user ever gets the right answer, the game ends and prints a congratulatory message. Select a random word from the list. At each try, read the user input in the variable 'guess'. Provide the variable guess and the answer to the function you built in 3b. Print a message shows how it went. At the sixth try, print the final message.

EXTRA 3d. Make a leaderboard Instead of the word list above, use the list from the worlde_list.csv file. Once per run (not at each iteration), before guessing the word, ask the user to submit its username. If the user guesses the word, he gains X points, where `X = (7-attempt_number)**2` If the user cannot guess in time, their score is -1 At the end of the game, open the wordle_leaderboard.txt and parse the information. If the user is not present, add it to the file with their score: For example:

```
Vtracann,-1,0,1,0%
```

Where Vtracann is the username, -1 is the total points, 0 is the total victories, 1 is the total attempts and 0% is the winrate. If the username is already present in the file, update its score. Sort the list in descending order (based on points) Print the top 5 based on points.