Week 8: Files

Contents

Important

First, `download_week08.py` and run the script in `VSCode`. You may need to confirm the download in your browser.

The script should end with `Successfully completed!`. If you encounter an error, follow the instructions in the error message. If needed, you can proceed with the exercises and seek assistance from a TA later.

In this week, you will learn how to read and write data to files using Python. This is a very important skill to have, as you will often need to load data from files in order to do your analysis. And, of course, you also want to save your results. We will introduce three different types of files: text files, CSV files, and pickle files. You will learn how to load and save all three types of files and what each type of file can be used for.

You will also learn how to visualize your data by making simple plots using the `matplotlib` library.

Introduction: Paths

Exercise 8: Introduction

Only text, but please read it carefully! Understanding paths is important for all of the following exercises.

Before loading and saving files using Python, it is important to understand the concept of "paths". A path defines the location of a file on your computer. There are absolute and relative paths. We will take a look at both now:

An **absolute path** is the full path to a file on your computer. It is the path you would use to find a file if you were to start from the root directory of your computer. The root directory is the *uppermost directory containing all the files on your computer*. It is the place on your computer where you can no longer "go one folder up."

Skip to main content

- Windows: `C:/Users/username/Desktop/02002students/cp/ex06/word_histogram.py`
- Mac: `/Users/username/Desktop/02002students/cp/ex06/word_histogram.py`

Sometimes, these paths can be very long and inconvenient to work with. But we can use **relative** paths.

A **relative path** is a path that is relative to your working directory (where you are executing your files). So in our case, the folder you open in VS Code: the `02002students` folder. This will be our starting point to find the files. If you now want to access this week's exercises, you first go into the folder `cp` and then into `ex08`. There, you will find the folder `files`, where all files you will work with are located. So the relative path to an example file is `cp/ex08/files/example.txt` (this works for all OSes and any computer, as long as you have the same folder structure).

You have also seen many examples of relative paths because that's how we have told you which file to write your code in. For example `cp/ex06/word_histogram.py`

> **Note**
>
> **IMPORTANT**
>
> Make sure you open the folder `02002students` in VS Code, otherwise your relative path will be different. Keep the relative paths in mind for all upcoming exercises. You will need them!
>
> Make sure you only use **relative** paths in all of the exercises!

Exercise 8.1: Reading a text file

Exercise 8.1 ›

In this exercise, you will load a text file (`.txt`) and look at its contents. The small exercises and the following text will help you understand how to read files in different ways and solve the next exercise. There are no tests for this exercise. Just try to get familiar with the functions and find out what works best for you.

First, let's take a look at the functions you will need: To open a file, you can use `open()` and after you are done with the file, you close it again with `close()`. You should always close the file after you are done with it to save it and free up the memory it uses. You can also use the `with` keyword to open a file. This will automatically close the file after you are done with it. This is the preferred way to open files in Python, but it is not as intuitive. (An example of both ways is below in the code box)

For `open()`, there are two parameters you need to specify:

- The path to the file (remember the relative paths introduced in the previous exercise)
- The mode in which you want to open the file, there are four options: `r` (read), `w` (write), `a` (append), and `x` (create).

But we don't just want to open and close files. We want to do something with the file in between. So, let's look at some basic methods to read the content of the file:

- `read()` Reads the whole file and returns it as a string.
- `readline()` Reads one line of the file and returns it as a string.
- `readlines()` Reads the whole file and returns it as a list of strings, where each element corresponds to one line in the file.

Try out all three methods with the following code (just replace `read()` with the other two methods). For `readline()` try printing twice to see what happens:

Skip to main content

```
# open, read and close a file
file = open("cp/ex08/files/names.txt", 'r')
print(file.read())
file.close()

# using the 'with' statement
with open("cp/ex08/files/names.txt", 'r') as file:
    print(file.read())
```

How are they different? What do you notice when using `readlines()`?

The `\n` is a special character that represents a line break (when you press `enter` in a text). It is actually present in all three outputs. It is not printed, but it is there. You can see it when you use `print(repr(file.read()))` to print the raw text. There are different ways to remove the `\n`. Here are some of them:

- Use `read()` and `splitlines()` to split the string into a list of strings, where each element corresponds to one line in the file without `\n`.
- Use `readline()` and `strip()` to remove the `\n` from each line.
- Remove the last character from each line by indexing the string (e.g., `line[:-1]`)

Try these options as well and see how the output differs from before.

You can also read a text file line by line using a for loop. This is the preferred way to read a file line by line, as it is more memory efficient. This is also a great tool if you want to do something with each line in the file, e.g., count the number of lines or words. Try out the following code (also try `line.strip()` instead of `line` to see the difference):

```
with open("cp/ex08/files/names.txt", 'r') as file:
    for line in file:
       print(line)
```

Exercise 8.2: Loading a text file to a string and a list

Exercise 8.2

Now that you know all the basic methods to read a file, let's try to load a text file into a list and into a string without the special character `\n`. Complete the function `load_txt2list()` and `load_txt2str()` in the file `cp/ex08/loading.py` such that it can load any text file and return it as a list or a string. You can use any methods you just practiced to do this. Also, recall some of the list and string methods you learned in the previous weeks:

- You can append to lists using `.append()`
- You can concatenate strings using `+` or turn lists into strings using `delimiter.join(list)`

For `load_txt2str()` you should add `" "` after each line (when concatenating a string); otherwise the lines will be concatenated without a space in between. For `delimiter.join(list)` set the delimiter to `" "`.

Remember to remove the `\n` from each line when you append it to the list or string.

Tip

Test your implementation to see if you can load all names into one list or string. There should be no `\n` in the output:

```
lst = load_txt2list("cp/ex08/files/names.txt")
```

Skip to main content

```
txt = load_txt2str("cp/ex08/files/names.txt")
print(txt)
```

cp.ex08.loading.load_txt2list(*path*)

Load text file content into a list of strings.

This function takes a file path as input, reads the content of the text file located at the specified path, and returns a list of strings. Each string in the list represents a line from the text file.

Parameters:

path (`str`) – A string representing the path to the text file.

Return type:

`list`

Returns:

A list of strings containing the lines of the text file.

cp.ex08.loading.load_txt2str(*path*)

Load text from a file.

This function reads the contents of a file and returns it as a string without linebreaks.

Parameters:

path (`str`) – A string representing the path of the file to be loaded.

Return type:

`str`

Returns:

The text content of the file as a string.

Exercise 8.3: Saving a text file

Exercise 8.3

Now, let's edit and save a file. There is two ways to do this:

- Open the file in write mode (`w`) and write the whole file at once using `write()`. **This will overwrite the file if it already exists.**
- Open the file in append mode (`a`) and write line by line using `write()`. With this method, you can add lines to the end of the file without overwriting it.

The write mode is useful if you want to create a new file. The append mode is useful if you want to add lines to an existing file. In both cases, the file is automatically saved when you close it. The `with` statement automatically closes the file after you are done with it.

Let's start with saving some text to a new file:

```
with open("cp/ex08/files/new_file.txt", 'w') as file:
    file.write("Hello World!")
```

Now open the file and see if it worked. The file should be in the folder `cp/ex08/files`. If you open it, you should see the text "Hello World!". What happens if you open the same file to write something?

Skip to main content

```
with open("cp/ex08/files/new_file.txt", 'w') as file:
    file.write("This is a new line!")
```

Check the file again to see what happened. The file was overwritten, so the text "Hello World!" is gone. If you want to add a new line to the file, you need to use the append mode:

```
with open("cp/ex08/files/new_file.txt", 'a') as file:
    file.write("This is another new line!")
```

Check the file again to see what happened. Did you notice that the text is not printed in a new line? This is because we did not add a line break to the end of the string. You can do this by adding `\n` to the end of the string:

```
with open("cp/ex08/files/new_file.txt", 'w') as file:
    file.write("This is a new line!\n")
    file.write("This is a new line in a new line!")
```

Check the file again to see what happened. Now, the text is printed in two lines. Also, note that the file was overwritten again, but the two new lines were both added to the file after each other since we didn't close the file in between. So you can write multiple lines to the same file using the write mode, as long as you don't close the file in between.

Now, let's implement two functions that can save a string and a list to a text file. Complete the functions `save_str2txt()` and `save_list2txt()` in the file `cp/ex08/saving.py` such that it can save any string or list to a text file. The functions should take the content you want to save and the path of the file you want to save it to as parameters.

For a string, we just want to save it directly to a text file. For a list, we want to save each element of the list in a new line, so remember to add a line break to the end of each line when saving a list.

Your function should make a new file that you specify in the function call by using its relative path. If the file already exists, it should be overwritten (**Important**: Make sure you don't overwrite any provided files!). You can use the write mode (`w`) for this.

Tip

Try saving the following string and list to a file called `saved_string.txt` and `saved_list.txt` in the folder `cp/ex08/files`. Check both files to see if they look as expected. The `saved_list.txt` should have three lines, while the `saved_string.txt` should only have one line.

```
string = "This is a string that we want to save to a file."
lst = ["This is a list", "that we want to save", "to a file."]
```

cp.ex08.saving.save_str2txt(*content*, *path*)

Save a string content to a text file.

This function takes a string content and a file path as input and saves the content to a text file in the 'files' directory relative to the current script's location.

Parameters:

- **content** (`str`) – A string representing the content to be saved.
- **path** (`str`) – A string representing the path of the file.

Return type:

`None`

Skip to main content

`cp.ex08.saving.save_list2txt(content, path)`

Save a list of strings to a text file.

This function takes a list of strings and a file path as input and saves each item of the list as a separate line in a text file. The file is saved in the 'files' directory relative to the current script's location.

Parameters:

- **content** (`list`) – A list of strings representing the content to be saved.
- **path** (`str`) – A string representing the path of the file.

Return type:

`None`

Exercise 8.4: Counting words and letters in a text file

Exercise 8.4

In this exercise, we want to count the number of words and letters in a text file. You can use your functions from the previous exercises to load the files.

There are several steps to this exercise, so let's break it down:

First, let's think about how we can count the number of words in a string. Implement a simple function `count_words()`, that counts the words in a string. Test it with this example (or other you come up with):

```
>>> string = "This is a string with some words in it and we want to know how many there are."
>>> wordcount = count_words(string)
>>> print('There are ', wordcount, 'words')
There are  18 words
```

Tip

Now, let's think about how we can count the number of words in a file. First, we need to load the file. You can use your function from the previous exercise to do this. You have two options, either loading the text file as a string or as a list. Which one do you think will work better for this exercise?

Tip

Now, let's think about how we can count the number of letters in a string. Try to implement a simple function `count_letters()` the letters in the following string:

```
>>> string = "This is a string with some words in it and we want to know how many letters there are."
>>> lettercount = count_letters(string)
>>> print('There are ', lettercount, 'letters')
There are  68 letters
```

You might find this very easy. It can be done by just taking the length of the string. But what if we only want to count the actual characters and not the spaces? How can you remove the spaces from the string?

Skip to main content

Tip

Now, we can combine all three parts into the function `count_words_letters()` in the file `cp/ex08/counting.py`. This function should take the path of the file as a parameter and return the number of words and letters in the file as a `tuple`. You can use your functions from the previous exercises to load the files. You can also use the functions `count_words()` and `count_letters()` you just implemented to count the words and letters in the file.

Optional: Try implementing the function without loading the whole file into a string. You can do this by reading the file line by line and counting the words and letters in each line. This is more memory efficient but also a bit more complicated to implement. Use a variable (e.g. `counter_words` and `counter_letters`) to keep track of the number of words/letters and add the number of words/letters in each line to the counter.

Try your function by testing it on the file `cp/ex08/files/hamlet.txt` and printing the number of words and letters in the file. The output should be:

```
>>> path = "cp/ex08/files/hamlet.txt"
>>> words, letters = count_words_letters(path)
>>> print('Number of words:', words, '\nNumber of letters:', letters)
Number of words: 645
Number of letters: 3062
```

cp.ex08.counting.count_words(*string*)

Count the number of words in a string.

Parameters:

string – A string.

Return type:

`int`

Returns:

The number of words in the string.

cp.ex08.counting.count_letters(*string*)

Count the number of letters in a string.

Parameters:

string – A string.

Return type:

`int`

Returns:

The number of letters in the string without spaces.

cp.ex08.counting.count_words_letters(*path*)

Count the number of words and letters in a text file.

This function takes a file path as input, reads the content of the text file located at the specified path, and returns the number of words and letters in the text file as a tuple.

Parameters:

path (`str`) – A string representing the path to the text file.

Return type:

`tuple`

Skip to main content

A tuple containing the number of words and letters in the text file.

Exercise 8.5: Reading and saving a csv file

Exercise 8.5

Another popular file type to save results is a *comma separated values* file (or *CSV* file). Normally, CSV files use a comma to separate each specific data value. Opening the file is the same as a .txt file (use the `open` function), but reading works a bit differently. For reading, we use the `csv` module and its reader.

This is an example of reading a .csv file:

```
import csv
with open("cp/ex08/files/ex.csv", 'r') as file:
    csvreader = csv.reader(file)
    for row in csvreader:
        print(row)
```

Did you notice that the output is a list of strings? If you have numbers in the file, you need to convert them to `int` or `float` for numerical operations. If your file only contains numbers, you can also use `csv.reader(file, quoting=csv.QUOTE_NONNUMERIC)` to automatically convert all numbers to `float`, but this does *not* work if you have strings in your file.

You can also read a csv file into a dictionary, where the first row is assumed to be the keys.

```
import csv

with open("cp/ex08/files/ex.csv", 'r') as file:
    csvreader = csv.DictReader(file)
    for row in csvreader:
        print(row)
```

To save a csv file, you open the file in `'w'` mode and use the `csv.writer`, and `writerow()` to add rows. The file is automatically saved when closing it (or leaving the `with` loop). Again, make sure **not** to overwrite given files.

Implement a function `save_csv()` in the file `cp/ex08/save_csv.py` that takes a list of lists and a path as input and saves the list to a csv file.

Try this example of saving some information in a csv file to test your function:

```
lst = [["Name", "Age", "Height"], ["Alice", 20, 1.70], ["Bob", 25, 1.80], ["Charlie", 30, 1.90]]
save_csv(lst, "cp/ex08/files/ex5.csv")
```

Check the file to see if it looks as expected. It should contain four rows and three columns.

Note

There are also many other ways to read and save CSV files, e.g., using the `pandas` library and lots more options in the `csv` library. These will not be introduced here but could be relevant to use later in your programming life.

`cp.ex08.save_csv.save_csv(lst, path)`

Save a list of lists to a csv file.

Skip to main content

Parameters:

- **lst** (`list`) – a list of lists
- **path** (`str`) – the path to the file

Exercise 8.6: How common is your last name?

Exercise 8.6

In this exercise, you are given a CSV file with all last names in Denmark and how often they occur (`cp/ex08/files/efternavne.csv`). We want to know the percentage of a certain last name in Denmark.

To start solving this problem, first, load the file and look at a few rows to see how your data looks.

Next, think about how to get the percentage of one name and how to implement each step. There are three steps:

1. You need the total amount of all names.
2. You need the occurrences of the name we are interested in.
3. The final percentage is *occurrence of name / total occurrences of all names * 100%*

Tip

Implement the function `name_occurence()` , which takes the file path and a name as input. You can assume that the name exists in the file. The function should give a percentage as `float` as output. Add your implementation to `cp/ex08/name_occurence.py` to test it.

Try this example to see if your functions work as expected (you can also try your own last name):

```
>>> path = 'cp/ex08/files/efternavne.csv'
>>> name = 'Andersen'
>>> percentage = name_occurence(path, name)
>>> print(percentage, '% of people in Denmark have the last name', name)
3.446267008547812 % of people in Denmark have the last name Andersen
```

cp.ex08.name_occurence.name_occurence(*path*, *name*)

Compute the occurrence in percent of a given last name.

Parameters:

- **path** (`str`) – Path to file with last names and occurrences
- **name** (`str`) – name of interest for incidence.

Return type:

`float`

Returns:

percentage of occurrence of name.

Exercise 8.7: Plotting data

Exercise 8.7

Skip to main content

Now that you know how to load and save data, we can start to analyze it. A very good first step is to look at your data, an for this, plots are very convenient. In this exercise, you will learn how to plot data using the library `matplotlib`.

Note: for this exercise, you don't need to implement any functions, and there are no tests, but plotting your data is a very useful skill and can be fun! So try it out!

For plotting, we will use the module `pyplot` from the library `matplotlib`. This is how it is usually imported and how you can make a simple plot:

```
import matplotlib.pyplot as plt
plt.plot(data)
plt.show()
```

Load the data in `weather_uk_2012.csv` and plot data (make sure to convert it to `float` before plotting). Look up the documentation of the plotting function here and play around with all the options (colors, lines, markers, ...) to make a nice-looking plot. You can also try adding a legend, axis labels, a title, etc.

Tip

Optional Load the data in `weather_uk_100years.csv`, which has data for the last 100 years (measured every ten years). Plot the rows as separate lines and label them according to the first entry in each row, which is the year (the year should be excluded when plotting the data).

Tip

If you made a really nice plot you want to keep, you can save it as a *.png* file using `plt.savefig("filename.png")`.

Exercise 8.8: Guess the language of a text

In this exercise, we want to guess the language of a text. We will do this by counting the frequency of letters in the text and comparing it to the frequency of letters in a text of a specific language.

First, we need to load the text file and count the frequency of letters in the text. For this, you have to count how often each letter appears in the text and how many letters there are in total. The function should return the percentage of the letter appearing.

For counting all letters, remember to remove spaces. You should also remove `.`, `,`, `?` and `!` from the text, as we don't want to count them as letters (you can do this by replacing them with an empty string).

Implement the function `frequency_letter()` in the file `cp/ex08/language_guess.py` such that it can count the frequency of a letter in a text.

Tip

Test your function for the letters `a` and `e`. The output should be:

```
>>> path = 'cp/ex08/files/hamlet.txt'
>>> frequency_a = frequency_letter(path, 'a')
>>> frequency_e = frequency_letter(path, 'e')
>>> print('The letter a appears', frequency_a, '% of the time in the text')
The letter a appears 7.635632953784327 % of the time in the text
>>> print('The letter e appears', frequency_e, '% of the time in the text')
```

Skip to main content

To now guess the language of the text, you can compare what your function returns to the frequency of letters in a text of a specific language. Here are the frequencies of the letters `a` and `e` in English, German, and French:

- English: `a`: 8.2%, `e`: 12.7%
- German: `a`: 6.5%, `e`: 16.3%
- French: `a`: 7.6%, `e`: 14.7%

Implement the function `language_guess()` in the file `cp/ex08/language_guess.py` such that it can guess the language of a text. Choose intervals of +/- 1% for each letter (i.e., for English, the frequency of a should be 7.2-9.2%). To classify the text as a language, both letters should be in the correct interval. If you can guess a language, your function should return the language as a string (`English`, `German`, or `French`). If you can't guess the language, your function should return `Unknown`.

cp.ex08.language_guess.frequency_letter(*filename*, *letter*)

Calculate the frequency of a letter in a text file.

This function takes a text file and a letter as input and calculates the frequency of the letter in the text file in percent.

Parameters:

- **filename** (`str`) – A string representing the name of the text file.
- **letter** (`str`) – A string representing the letter to be searched for.

Return type:

`float`

Returns:

A float representing the frequency of the letter in the text file in percent.

cp.ex08.language_guess.language_guess(*filename*)

Guess the language of a text file.

This function takes a text file as input and guesses the language of the text file based on the frequency of the letters 'a' and 'e' in the text file.

Parameters:

filename (`str`) – A string representing the name of the text file.

Return type:

`str`

Returns:

A string representing the guessed language.

Exercise 8.9: Bacteria growth experiment

> **Note**
>
> If you downloaded the download script before 11:30 on 26.05.2023 you are missing some of the experiement files. Please download and run the script again.

Imagine you want to know how long it takes for a certain bacteria to grow to a certain threshold. You can do this by measuring the bacteria growth over time. To be sure that you get the right result, you should do this experiment several times and average the results. In this exercise, you get the results of 160 experiments, where the bacteria growth was measured every 10 minutes. The data of each experiment is in the folder `cp/ex08/files/experiments` as a CSV file with the

Skip to main content

A great first step is to look at your data by plotting it. You can plot all experiments by looping through all files:

```
import matplotlib.pyplot as plt
import csv

for i in range(160):
    with open(f'cp/ex08/files/experiments/experiment_{i:03d}.csv', 'r') as f:
       reader = csv.reader(f, quoting=csv.QUOTE_NONNUMERIC)
       row = next(reader)
       plt.plot(row)
plt.show()
```

For each experiment, you want to know the timestep at which a certain threshold was reached (exceeded). Implement a function `growth_threshold_reached()` in the file `cp/ex08/growth_experiment.py` that calculates the average time point at which the threshold was reached (exceeded) for all experiments. *If the experiment does not reach the threshold, you should assume that the threshold is reached at time point 10.* The function should take the path to the folder with the experiments and the threshold as parameters. You can assume that the files are named `experiment_000.csv` to `experiment_159.csv`.

Tip

Test your function for a threshold of 8.5. The output should be:

```
>>> path = 'cp/ex08/files/experiments'
>>> threshold = 8.5
>>> average_timepoint = growth_threshold_reached(path, threshold)
>>> print('The average timepoint at which the threshold was reached is', average_timepoint)
The average timepoint at which the threshold was reached is 9.29375
```

cp.ex08.growth_experiment.growth_threshold_reached(*path*, *threshold*)

Return the time point at which the growth threshold is reached.

Parameters:

- **path** (`str`) – The path to the folder of the data files.
- **threshold** (`float`) – The threshold value.

Return type:

`float`

Returns:

The average time point.

Exercise 8.10: *Optional* Saving a pickle file

There are many other ways to load and save data that we do not have time to cover here. One that is often used is using pickle files.

Pickle files are binary files that can be used to save any Python object. They are very useful if you want to save a large object (e.g., a list of lists, dictionaries, ...) and load it again later. You can read more about pickle files here.

Try this example of how to save a list of lists to a pickle file and read it again. You can also use the `with` statement to open the file, but you need to use `'wb'` (write binary) and `'rb'` (read binary) instead of `'w'` and `'r'`.

```
import pickle
lst = [[1, 2, 3], [4, 5, 6], ['some_string']]
with open('lst.pkl', 'wb') as f:
```

Skip to main content

```
    lst = pickle.load(f)
print(lst)
```

As you can see, it has the exact same format as before. Try saving a dictionary to a pickle file and loading it again to practice pickle files a bit more.