File I/O

can code communities

File Directory

- Your file explorer has all the files stored on your computer, organized into folders (also called directories)
- Each file has a path, which is the hierarchy of folders you need to follow to find the file.
    - In Windows, paths have backwards slashes:
        - `C:\Users\YourUsername\Programming\My_Code.py`
    - In Unix (Mac or Linux), paths have forward slashes:
        - `c/Users/YourUsername/Programming/My_Code.py`

Keep your file explorer organized so you can easily find and access your files!

Extracting Zip Files

- Often when you download files from the internet, a folder will be compressed into a zip file.
- This saves space, making it easier to transport files, but you need to extract the zip file to use it.
- Right-click on a zip file and **extract all**. This lets you choose a new location for your files.

Types of Files

- There are many types of files we can open using Python
    - Just like Python files have a .py extension, each type file has its own extension
    - Some common extensions are .csv, .txt, and .json (JavaScript Object Notation)
- When you open a file, make sure to get the correct extension

Opening Files

- To open a file, use the open() function, which takes 2 parameters: **filename** (required) and **mode** (optional)
    - **Filename** is the name of your file, as a string in quotes.
        - If it's in the same folder, all you need is the name. If it's in a different folder, you need the entire path.
        - Make sure to get the correct extension.
- open() returns a file object that you can use later.

```
f = open("my_file.txt")
```

Closing Files

Make sure to close your files! There are multiple ways to do this.

The first option is to open your file, set it as a variable, then close it later.

```python
file = open("my_file.txt")
# file processing
file.close()
```

However, if your code runs into an error during the file processing, your file won't close.

Closing Files, Option 2

To ensure your file closes properly, even if there are errors in your file processing, you can use a `try-finally` block:

```
file = open("my_file.txt")
try:
       # file processing
finally:
       file.close()
```

Closing Files, Option 3

If you iterate through your file using a `with` statement, your file automatically closes, even if an error occurs during file processing.

```
with open("file.txt") as file:
       # file processing
```

The file automatically closes once the `with` block ends, making this the cleanest and least error-prone way to open/close a file.

File Mode

- The mode is the second, optional parameter to the open function, and allows you to put the file in "read-only" or "write-only" mode
- Most commonly used modes:
  - `'r'` open for reading (default)
  - `'w'` open for writing, which truncates (overwrites) the file

```
f = open("my_file.txt", "r")
```

Open my_file.txt in read-only mode

All Modes

- `'r'` open for reading (default). Starts at the beginning of the file.
- `'r+'` open for reading and writing. Starts at the beginning of the file.
- `'w'` open for writing. Truncate (overwrite) the file. Starts at the beginning of the file.
- `'w+'` open for reading and writing. Create the file if it does not exist, otherwise truncate it. Starts at the beginning of the file.
- `'a'` open for writing, create the file if it doesn't exist, and append to the end of the file instead of overwriting. Subsequent writes will always end up at the current end of the file.
- `'a+'` open for reading and writing, create the file if it doesn't exist, and append to the end of the file instead of overwriting. Subsequent writes will always end up at the current end of the file.

Reading Files

Once you have a file object in read-only mode, we can read the file.

There are three functions to read data from a file:

- `.read()` Reads the entire file into a multi-line string
- `.readline()` Reads one line of the file into a string
- `.readlines()` Reads the entire file into a list, where each element in the list is a string representing one line.

Exercise

Create a program that reads a text file named "example.txt" and outputs the number of lines in the file.

(Assume your text file is in the same directory as your Python file)

Hint: Open the file in read-only mode, and use the .readlines() function.

Exercise - Analyzing Sales Data

You are a data analyst who must analyze a company's sales data to determine which products are selling the best.

The data is stored in a CSV file named "sales_data.csv". The file contains the following columns: Product Name, Date Sold, Units Sold, Price per Unit, Total Sale

Write a Python script that reads the data from the CSV file and calculates the total revenue generated by each product.

- Make sure to ignore the first line, since that's the header.
- You can go through the file line by line, and **split** each line into strings based on **commas** to get the individual column values.
- You can keep track of each product and its total sale in a **dictionary**.
- The only two relevant columns are Product Name and Total Sale.

Writing to Files

Make sure to open your file in a mode that allows writing - the default mode is read-only.

There are two functions to write data to a file:

- `.write(S)` Insert the string `S` in a single line in the file.
- `.writelines(L)` For a list `L` containing strings, insert each string as a new line in the file.

Exercise

Create a program that asks the user for their name and age, then writes this information to a new text file named "user_info.txt".

Write the name on the first line of the file, and the age on the second line of the file.

Exercise - Tracking Employee Data

You work for a company that needs to keep track of employee data, such as name, age, and salary. Write a Python program that prompts the user to input data for each employee, and writes the data to a CSV file named "employee_data.csv".

The CSV file has the following columns: Name, Age, Salary

Until the user inputs "quit", keep prompting the user for employee data, and write it to the CSV file.

- Remember to follow a CSV file format: Each line should be separated by commas and end in a newline.

File Formatting in Windows vs Unix

- Text files are formatted differently depending on whether you use Windows or Unix (Mac/Linux)
    - In Windows, there is `\r\n` at the end of every line (`\r` is carriage return)
    - In Unix, there is just `\n` at the end of every line
- This usually doesn't cause any issues if you're just doing simple file processing, but it makes the exact contents of a text file differ based on what operating system you're using