

## LEARN PYTHON: FILES

### Reading a File

Computers use file systems to store and retrieve data. Each [file](#) is an individual container of related information. If you've ever saved a document, downloaded a song, or even sent an email you've created a file on some computer somewhere. Even **script.py**, the Python program you're editing in the learning environment, is a file.

So, how do we interact with files using Python? We're going to learn how to read and write different kinds of files using code. Let's say we had a file called **real\_cool\_document.txt** with these contents:

**real\_cool\_document.txt**

```
Wowzers!
```

We could read that file like this:

**script.py**

```
with open('real_cool_document.txt') as cool_doc:
    cool_contents = cool_doc.read()
print(cool_contents)
```

This opens a file object called `cool_doc` and creates a new indented block where you can read the contents of the opened file. We then read the contents of the file `cool_doc` using `cool_doc.read()` and save the resulting string into the variable `cool_contents`. Then we print `cool_contents`, which outputs the statement `Wowzers!`.

### Instructions

1.

Use `with` to open the file **welcome.txt**. Save the file object as `text_file`.

Hint

Use this syntax to open a file:

```
with open('filename.txt') as file_object:
    # indented block here
```

2.

Read the contents of `text_file` and save the results in `text_data`.

Hint

Use this syntax on a file object to read the contents of the file into a variable:

```
file_string = file_object.read()
```

3.

Print out `text_data`.

`script.py`

```
with open('welcome.txt') as text_file:
    text_data = text_file.read()
print(text_data)
```

Congratulations on reading your first file at codecademy.com!

---

## Iterating Through Lines

When we read a file, we might want to grab the whole document in a single string, like `.read()` would return. But what if we wanted to store each line in a variable? We can use the `.readlines()` function to read a text file line by line instead of having the whole thing. Suppose we have a file:

**keats\_sonnet.txt**

```
To one who has been long in city pent,
'Tis very sweet to look into the fair
And open face of heaven,—to breathe a prayer
Full in the smile of the blue firmament.
```

`script.py`

```
with open('keats_sonnet.txt') as keats_sonnet:
    for line in keats_sonnet.readlines():
        print(line)
```

The above script creates a temporary file object called `keats_sonnet` that points to the file **keats\_sonnet.txt**. It then iterates over each line in the document and prints the entire file out.

## Instructions

1.

Using a `with` statement, create a file object pointing to the file **how\_many\_lines.txt**. Store that file object in the variable `lines_doc`.

Hint

Remember to open a file using `with` syntax:

```
with open('filename.txt') as file_object:
    # indented block here
```

2.

Iterate through each of the lines in `lines_doc.readlines()` using a `for` loop.

Inside the `for` loop print out each line of **how\_many\_lines.txt**.

---

Hint

You can use the following syntax to print out each line of a file:

```
for line in file_object.readlines():  
    print(line)
```

**script.py**

```
with open('how_many_lines.txt') as lines_doc:  
    for line in lines_doc.readlines():  
        print(line)
```

```
1. How many lines do we write on the daily,  
  
2. Many money, we write many many many  
  
3. How many lines do you write on the daily,  
  
4. Say you say many money, you write many many many
```

---

## Reading a Line

Sometimes you don't want to iterate through a whole file. For that, there's a different file method, `.readline()`, which will only read a single line at a time. If the entire document is read line by line in this way subsequent calls to `.readline()` will not throw an error but will start returning an empty string (`""`). Suppose we had this file:

**millay\_sonnet.txt**

```
I shall forget you presently, my dear,  
So make the most of this, your little day,  
Your little month, your little half a year,  
Ere I forget, or die, or move away,
```

**script.py**

```
with open('millay_sonnet.txt') as sonnet_doc:  
    first_line = sonnet_doc.readline()
```

```
second_line = sonnet_doc.readline()
print(second_line)
```

This script also creates a file object called `sonnet_doc` that points to the file **millay\_sonnet.txt**. It then reads in the first line using `sonnet_doc.readline()` and saves that to the variable `first_line`. It then saves the second line (So make the most of this, your little day,) into the variable `second_line` and then prints it out.

## Instructions

1.

Using a `with` statement, create a file object pointing to the file **just\_the\_first.txt**. Store that file object in the variable `first_line_doc`.

---

Hint

Remember to open a file using `with` syntax:

```
with open('filename.txt') as file_object:
    # indented block here
```

2.

Save the first line of **just\_the\_first.txt** into the variable `first_line`.

---

Hint

Use the file object method `.readline()` to store a line into the variable `line`.

3.

Print out the variable `first_line`.

`script.py`

```
with open('just_the_first.txt') as first_line_doc:
    first_line = first_line_doc.readline()
    print(first_line)
```

```
You do look, my son, in a moved sort,
```

---

## Writing a File

Reading a file is all well and good, but what if we want to create a file of our own? With Python we can do just that. It turns out that our `open()` function that

we're using to open a file to read needs another argument to open a file to write to.

### script.py

```
with open('generated_file.txt', 'w') as gen_file:  
    gen_file.write("What an incredible file!")
```

Here we pass the argument 'w' to `open()` in order to indicate to open the file in write-mode. The default argument is 'r' and passing 'r' to `open()` opens the file in read-mode as we've been doing.

This code creates a new file in the same folder as *script.py* and gives it the text `What an incredible file!`. It's important to note that if there is already a file called *generated\_file.txt* it will completely overwrite that file, erasing whatever its contents were before.

## Instructions

### 1.

Create a file object for the file **bad\_bands.txt** using the `open()` function with the `w` argument. Assign this object to the temporary variable `bad_bands_doc`.

Hint

Remember to use the `with` statement to open a file and pass the 'w' argument to open it in write mode:

```
with open('file_to_write.txt', 'w') as file_object:  
    # indent
```

### 2.

Use the `bad_bands_doc.write()` method to add the name of a musical group you dislike to the document `bad_bands`.

### script.py

```
with open('bad_bands.txt', 'w') as bad_bands_doc:  
    bad_bands_doc.write("Poison")
```

---

## Appending to a File

So maybe completely deleting and overwriting existing files is something that bothers you. Isn't there a way to just add a line to a file without completely deleting it? Of course there is! Instead of opening the file using the argument 'w' for write-mode, we open it with 'a' for append-mode. If we have a generated file with the following contents:

## generated\_file.txt

```
This was a popular file...
```

Then we can add another line to that file with the following code:

## script.py

```
with open('generated_file.txt', 'a') as gen_file:  
    gen_file.write("\n... and it still is")
```

In the code above we open a file object in the temporary variable `gen_file`. This variable points to the file `generated_file.txt` and, since it's open in append-mode, adds the string `\n... and it still is` to the file. The newline character `\n` moves to the next line before adding the rest of the string. If you were to open the file after running the script it would look like this:

## generated\_file.txt

```
This was a popular file...  
... and it still is
```

Notice that opening the file in append-mode, with `'a'` as an argument to `open()`, means that using the file object's `.write()` method *appends* whatever is passed to the end of the file. If we were to run **script.py** again, this would be what **generated\_file.txt** looks like:

## generated\_file.txt

```
This was a popular file...  
... and it still is  
... and it still is
```

Notice that we've appended `"\n... and it still is"` to the file a second time! This is because in **script.py** we opened **generated\_file.txt** in append-mode.

## Instructions

### 1.

We've got a file, **cool\_dogs.txt**, filled with all the cool dogs we know. Somehow while compiling this list we forgot about one very cool dog. Let's fix that problem by adding him to our **cool\_dogs.txt**.

Open up our file **cool\_dogs.txt** in append-mode and assign it to the file object `cool_dogs_file`.

---

### 2.

Inside your `with` block, add `"Air Buddy\n"` to **cool\_dogs.txt**. Air Buddy is a Golden Retriever that plays basketball, which more than qualifies him for this list. The `\n` character moves to the next line after appending the string.

---

Hint

Use `cool_dogs_file.write()` to add a string to the end of **cool\_dogs.txt**

**script.py**

```
with open('cool_dogs.txt', 'a') as cool_dogs_file:
    cool_dogs_file.write("Air Buddy\n")
```

---

## What's With "with"?

We've been opening these files with this `with` block so far, but it seems a little weird that we can only use our file variable in the indented block. Why is that? The `with` keyword invokes something called a *context manager* for the file that we're calling `open()` on. This context manager takes care of opening the file when we call `open()` and then closing the file after we leave the indented block.

Why is closing the file so complicated? Well, most other aspects of our code deal with things that Python itself controls. All the variables you create: integers, lists, dictionaries — these are all Python objects, and Python knows how to clean them up when it's done with them. Since your files exist *outside* your Python script, we need to tell Python when we're done with them so that it can close the connection to that file. Leaving a file connection open unnecessarily can affect performance or impact other programs on your computer that might be trying to access that file.

The `with` syntax replaces older ways to access files where you need to call `.close()` on the file object manually. We can still open up a file and append to it with the old syntax, as long as we remember to close the file connection afterwards.

```
fun_cities_file = open('fun_cities.txt', 'a')

# We can now append a line to "fun_cities".
fun_cities_file.write("Montréal")

# But we need to remember to close the file
fun_cities_file.close()
```

In the above script we added "Montréal" as a new line in our file **fun\_cities.txt**. However, since we used the older-style syntax, we had to remember to close the file afterwards. Since this is necessarily more verbose

(requires at least one more line of code) without being any more expressive, using `with` is preferred.

## Instructions

### 1.

In **script.py** there's a file object that doesn't get closed correctly. Let's fix it by changing the syntax!

Remove this line:

```
close_this_file = open('fun_file.txt')
```

And change it to use the `with` syntax from our previous exercises.

Remember to indent the rest of the body so that we don't get an `IndentError`.

**script.py**

```
with open("fun_file.txt") as close_this_file:
    setup = close_this_file.readline()
    punchline = close_this_file.readline()

print(setup)
```

```
What did the pirate say when he turned 80?
```

---

## What Is a CSV File?

Text files aren't the only thing that Python can read, but they're the only thing that we don't need any additional parsing library to understand. [CSV](#) files are an example of a text file that impose a structure to their data. CSV stands for *Comma-Separated Values* and CSV files are usually the way that data from spreadsheet software (like Microsoft Excel or Google Sheets) is exported into a portable format. A spreadsheet that looks like the following

Name	Username	Email
Roger Smith	rsmith	<a href="mailto:wigginsryan@yahoo.com">wigginsryan@yahoo.com</a>
Michelle Beck	mlbeck	<a href="mailto:hcosta@hotmail.com">hcosta@hotmail.com</a>
Ashley Barker	a_bark_x	<a href="mailto:a_bark_x@turner.com">a_bark_x@turner.com</a>



Name	Username	Email
Lynn Gonzales	goodmanjames	<a href="mailto:lynniegonz@hotmail.com">lynniegonz@hotmail.com</a>
Jennifer Chase	chasej	<a href="mailto:jchase@ramirez.com">jchase@ramirez.com</a>
Charles Hoover	choover	<a href="mailto:choover89@yahoo.com">choover89@yahoo.com</a>
Adrian Evans	adevans	<a href="mailto:adevans98@yahoo.com">adevans98@yahoo.com</a>
Susan Walter	susan82	<a href="mailto:swilliams@yahoo.com">swilliams@yahoo.com</a>
Stephanie King	stephanieking	<a href="mailto:sking@morris-tyler.com">sking@morris-tyler.com</a>
Erika Miller	jessica32	<a href="mailto:ejmiller79@yahoo.com">ejmiller79@yahoo.com</a>

In a CSV file that same exact data would be rendered like this:

### users.csv

```
Name,Username,Email
Roger Smith,rsmith,wigginsryan@yahoo.com
Michelle Beck,mlbeck,hcosta@hotmail.com
Ashley Barker,a_bark_x,a_bark_x@turner.com
Lynn Gonzales,goodmanjames,lynniegonz@hotmail.com
Jennifer Chase,chasej,jchase@ramirez.com
Charles Hoover,choover,choover89@yahoo.com
Adrian Evans,adevans,adevans98@yahoo.com
Susan Walter,susan82,swilliams@yahoo.com
Stephanie King,stephanieking,sking@morris-tyler.com
Erika Miller,jessica32,ejmiller79@yahoo.com
```

Notice that the first row of the CSV file doesn't actually represent any data, just the labels of the data that's present in the rest of the file. The rest of the rows of the file are the same as the rows in the spreadsheet software, just instead of being separated into different cells they're separated by... well I suppose it's fair to say they're separated by commas.

### Instructions

#### 1.

CSV files are just plain text files!

Open `logger.csv` using our standard `with` syntax, saving the file object in the temporary variable `log_csv_file`.

#### 2.

Print out the contents of `logger.csv` by calling `.read()` on the file. Notice that it is parsed as a string.

`script.py`

```
with open('logger.csv') as log_csv_file:
    var = log_csv_file.read()

print(var)
```

```
Name, Age, ID
Richard
Andrews, 43, 0de2ecf31df2386377b1d2dc4fae8b16fad05ad0
Hailey
Sellers, 24, 3d9b8a95458c1df2687191e8146a97ca4afb28aa
Jessica Pace, 39, a5daa63ef893cb86bc8df1110cc9a5f8e1d0c563
Jasmine
Escobar, 42, 9844e403841ec84b9a2fb3caf9d2a1c9ee042d31
Karen Kelly, 26, 8338f76ac0e9a76d73d57790f1d9843b185b5428
Patricia
Christensen, 70, 23099bb630c1c64989458393045f08de3bac0eb9
Jessica
Hansen, 24, a8c035ccd099ef909a46e0d96b76c0f132c9c562
Raymond
Adams, 53, a051901830ff6c2095524ef92b1541eef9f8c64d
Stephanie
Morrow, 53, 3bad04a5fc0a7ec33735ae45535f354887988f35
Timothy
Ramos, 34, b4930920b5256c4e592541297e43a556c8fe33a8
```

---

### Reading Different Types of CSV Files

I need to level with you, I've been lying to you for the past two exercises. Well, kind of. We've been acting like CSV files are Comma-Separated Values files. It's true that CSV stands for that, but it's also true that other ways of separating values are valid CSV files these days.

People used to call Tab-Separated Values files TSV files, but as other separators grew in popularity everyone realized that creating a new `[a-z]sv` file format for every value-separating character used is not sustainable.

So we call all files with a list of different values a CSV file and then use different *delimiters* (like a comma or tab) to indicate where the different values start and stop.

Let's say we had an address book. Since addresses usually use commas in them, we'll need to use a different delimiter for our information. Since none of our data has semicolons (;) in them, we can use those.

### addresses.csv

```
Name;Address;Telephone
Donna Smith;126 Orr Corner Suite 857\nEast Michael, LA 54411;906-918-6560
Aaron Osborn;6965 Miller Station Suite 485\nNorth Michelle, KS 64364;815.039.3661x42816
Jennifer Barnett;8749 Alicia Vista Apt. 288\nLake Victoriaberg, TN 51094;397-796-4842x451
Joshua Bryan;20116 Stephanie Stravenue\nWhitneytown, IA 87358;(380)074-6173
Andrea Jones;558 Melissa Keys Apt. 588\nNorth Teresahaven, WA 63411;+57(8)7795396386
Victor Williams;725 Gloria Views Suite 628\nEast Scott, IN 38095;768.708.3411x954
```

Notice the `\n` character, this is the escape sequence for a new line. The possibility of a new line escaped by a `\n` character in our data is why we pass the `newline=''` keyword argument to the `open()` function.

Also notice that many of these addresses have commas in them! This is okay, we'll still be able to read it. If we wanted to, say, print out all the addresses in this CSV file we could do the following:

```
import csv

with open('addresses.csv', newline='') as addresses_csv:
    address_reader = csv.DictReader(addresses_csv, delimiter=';')
    for row in address_reader:
        print(row['Address'])
```

Notice that when we call `csv.DictReader` we pass in the `delimiter` parameter, which is the string that's used to delineate separate fields in the CSV. We then iterate through the CSV and print out each of the addresses.

## Instructions

1.

Import the `csv` module.

---

2.

Open up the file **books.csv** in the variable `books_csv`.

---

3.

Create a `DictReader` instance that uses the `@` symbol as a delimiter to read `books_csv`. Save the result in a variable called `books_reader`.

---

4.

Create a list called `isbn_list`, iterate through `books_reader` to get the ISBN number of every book in the CSV file. Use the `['ISBN']` key for the dictionary objects passed to it.

### script.py

```
import csv

with open('books.csv', newline='') as books_csv:
    books_reader = csv.DictReader(books_csv, delimiter='@')
    isbn_list = []
    for row in books_reader:
        isbn_list.append(row['ISBN'])
    print(isbn_list)
```

```
['978-0-12-995015-8', '978-1-78110-100-1', '978-0-315-25137-3', '978-0-388-70665-7', '978-1-75098-721-6', '978-1-06-483628-6', '978-0-7419-8114-1', '978-1-4457-0480-7', '978-0-657-61030-2', '978-1-5039-7539-2']
```

### Writing a CSV File

Naturally if we have the ability to read different CSV files we might want to be able to programmatically create CSV files that save output and data that someone could load into their spreadsheet software. Let's say we have a big list of data that we want to save into a CSV file. We could do the following:

```
big_list = [{'name': 'Fredrick Stein', 'userid': 6712359021, 'is_admin': False}, {'name': 'Wiltmore Denis', 'userid': 2525942, 'is_admin': False}, {'name': 'Greely Plonk', 'userid': 15890235, 'is_admin': False}, {'name': 'Dendris Stulo', 'userid': 572189563, 'is_admin': True}]

import csv

with open('output.csv', 'w') as output_csv:
    fields = ['name', 'userid', 'is_admin']
    output_writer = csv.DictWriter(output_csv, fieldnames=fields)

    output_writer.writeheader()
    for item in big_list:
        output_writer.writerow(item)
```

In our code above we had a set of dictionaries with the same keys for each, a prime candidate for a CSV. We import the `csv` library, and then open a new CSV file in write-mode by passing the `'w'` argument to the `open()` function.

We then define the fields we're going to be using into a variable called `fields`. We then instantiate our CSV writer object and pass two arguments. The first is `output_csv`, the file handler object. The second is our list of fields `fields` which we pass to the keyword parameter `fieldnames`.

Now that we've instantiated our CSV file writer, we can start adding lines to the file itself! First we want the headers, so we call `.writeheader()` on the writer object. This writes all the fields passed to `fieldnames` as the first row in our file. Then we iterate through our `big_list` of data. Each `item` in `big_list` is a dictionary with each field in `fields` as the keys. We call `output_writer.writerow()` with the `item` dictionaries which writes each line to the CSV file.

## Instructions

1.

We have a list in the workspace `access_log` which is a list of dictionaries we want to write out to a CSV file.

Let's start by importing the `csv` module.

---

2.

Open up the file **logger.csv** in the temporary variable `logger_csv`. Don't forget to open the file in write-mode.

---

3.

Create a `csv.DictWriter` instance called `log_writer`. Pass `logger_csv` as the first argument and then `fields` as a keyword argument to the keyword `fieldnames`.

---

4.

Write the header to `log_writer` using the `.writeheader()` method.

---

5.

Iterate through the `access_log` list and add each element to the CSV using `log_writer.writerow()`.

## script.py

```
access_log = [{'time': '08:39:37', 'limit': 844404, 'address': '1.227.124.181'},
{'time': '13:13:35', 'limit': 543871, 'address': '198.51.139.193'}, {'time': '19:
40:45', 'limit': 3021, 'address': '172.1.254.208'}, {'time': '18:57:16', 'limit':
 67031769, 'address': '172.58.247.219'}, {'time': '21:17:13', 'limit': 9083, 'add
ress': '124.144.20.113'}, {'time': '23:34:17', 'limit': 65913, 'address': '203.23
6.149.220'}, {'time': '13:58:05', 'limit': 1541474, 'address': '192.52.206.76'},
{'time': '10:52:00', 'limit': 11465607, 'address': '104.47.149.93'}, {'time': '14
:56:12', 'limit': 109, 'address': '192.31.185.7'}, {'time': '18:56:35', 'limit':
6207, 'address': '2.228.164.197'}]
fields = ['time', 'address', 'limit']

import csv

with open('logger.csv', 'w') as logger_csv:
    #fields = ['name', 'userid', 'is_admin']
    log_writer = csv.DictWriter(logger_csv, fieldnames=fields)

    log_writer.writeheader()
    for item in access_log:
        log_writer.writerow(item)
```

---

## Reading a JSON File

CSV isn't the only file format that Python has a built-in library for. We can also use Python's file tools to read and write [JSON](#). JSON, an abbreviation of JavaScript Object Notation, is a file format inspired by the programming language JavaScript. The name, like CSV is a bit of a misnomer — some JSON is not valid JavaScript (and plenty of JavaScript is not valid JSON).

JSON's format is endearingly similar to Python dictionary syntax, and so JSON files might be easy to read from a Python developer standpoint. Nonetheless, Python comes with a `json` package that will help us parse JSON files into actual Python dictionaries. Suppose we have a JSON file like the following:

### purchase\_14781239.json

```
{
  'user': 'ellen_greg',
  'action': 'purchase',
```

```
}
    'item_id': '14781239',
}
```

We would be able to read that in as a Python dictionary with the following code:

### json\_reader.py

```
import json

with open('purchase_14781239.json') as purchase_json:
    purchase_data = json.load(purchase_json)

print(purchase_data['user'])
# Prints 'ellen_greg'
```

First we import the `json` package. We opened the file using our trusty `open()` command. Since we're opening it in read-mode we just need to pass the file name. We save the file in the temporary variable `purchase_json`.

We continue by parsing `purchase_json` using `json.load()`, creating a Python dictionary out of the file. Saving the results into `purchase_data` means we can interact with it. We print out one of the values of the JSON file by keying into the `purchase_data` object.

## Instructions

### 1.

Let's read a JSON file! Start by importing the `json` module.

---

Hint

Import the JSON library with the following command:

```
import json
```

### 2.

Open up the file **message.json**, saving the file object to the variable `message_json`.

Open the file in read-mode, without passing any additional arguments to `open()`.

---

Hint

Remember the syntax for opening a file:

```
with open('file.json') as file_json:
    pass
```

3.

Pass the JSON file object as an argument to `json.load()` and save the resulting Python dictionary as `message`.

---

4.

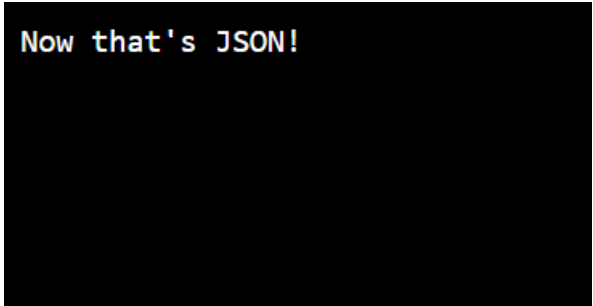
Print out `message['text']`.

### script.py

```
import json

with open('message.json') as message_json:
    message = json.load(message_json)

print(message['text'])
# Prints 'ellen_greg'
```



Now that's JSON!

---

### Writing a JSON File

Naturally we can use the `json` library to translate Python objects to JSON as well. This is especially useful in instances where you're using a Python library to serve web pages, you would also be able to serve JSON. Let's say we had a Python dictionary we wanted to save as a JSON file:

```
turn_to_json = {
    'eventId': 674189,
    'dateTime': '2015-02-12T09:23:17.511Z',
    'chocolate': 'Semi-sweet Dark',
    'isTomatoAFruit': True
}
```

We'd be able to create a JSON file with that information by doing the following:



```
import json

with open('output.json', 'w') as json_file:
    json.dump(turn_to_json, json_file)
```

We import the `json` module, open up a write-mode file under the variable `json_file`, and then use the `json.dump()` method to write to the file. `json.dump()` takes two arguments: first the data object, then the file object you want to save.

## Instructions

### 1.

In your workspace, we've put a dictionary called `data_payload`. We want to save this to a file called **data.json**.

Let's start by importing the `json` library.

---

### 2.

Open a new file object in the variable `data_json`. The filename should be `'data.json'` and the file should be opened in write-mode.

---

Hint

Remember to open a file in write-mode with the following syntax:

```
with open('file.json', 'w') as file_json:
    pass
```

### 3.

Call `json.dump()` with `data_payload` and `data_json` to convert our data to JSON and then save it to the file **data.json**.

---

Hint

Using `json.dump()` with the file object as a second argument writes the resulting JSON to the file:

```
payload = {'message': 'OK'}
with open('file.json', 'w') as file_json:
    json.dump(payload, file_json)
```

## script.py

```
data_payload = [
    {'interesting message': 'What is JSON? A web application\'s little pile of secrets.'},
    {'follow up': 'But enough talk!'}
]
```

```
import json

with open('data.json', 'w') as data_json:
    json.dump(data_payload, data_json)
```

---

## Review

Now you know all about files! You were able to:

- Open up file objects using `open()` and `with`.
- Read a file's full contents using Python's `.read()` method.
- Read a file line-by-line using `.readline()` and `.readlines()`
- Create new files by opening them in write-mode.
- Append to a file non-destructively by opening a file in append-mode.
- Apply all of the above to different types of data-carrying files including CSV and JSON!

You have all the skills necessary to read, write, and update files programmatically, a very useful skill in the Python universe!