READING FILES WITH PYTHON

PYTHON

Like we did last week, we can download our data as a CSV file, which is an example of a **flat-file database**, where the data for each column is separated by commas, and each row is on a new line, saved simply as a text file in ASCII or Unicode.

A flat-file database is completely portable, which means that we can open it on nearly any operating system without special software like Google sheets, Microsoft Excel or Apple Numbers.

We'll upload the CSV file to our instance of VS Code by dragging and dropping it:
Then, we'll see the file opened in an editor:

Notice that some rows have multiple genres, and those are surrounded by quotes, like "Crime, Drama", so that the commas *within* our data aren't misinterpreted.

Let's write a new program, favorites.py, to read our CSV file:

```
import csv

with open("favorites.csv", "r") as file:
  reader = csv.reader(file)
  next(reader)
  for row in reader:
    print(row[1])
```

We'll open the file with a reference called file, using the with keyword in Python that will close our file for us.

The csv library has a reader function that will create a reader variable we can use to read in the file as a CSV.

We'll call next to skip the first row, since that's the header row.

Then, we'll use a loop to print the second column in each row, which is the title.

To improve our program, we'll first use a DictReader, dictionary reader, which creates a dictionary from each row, allowing us to access each column by its name. We also don't need to skip the header row in this case, since the DictReader will use it automatically.

```
import csv

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    print(row["title"])
```

Since the first row in our CSV has the names of the columns, it can be used to label each column in our data as well. Now our program will still work, even if the order of the columns are changed.

Now let's try to filter out duplicates in our responses:

```
import csv

titles = [ ]

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    if not row["title"] in titles:
      titles.append(row["title"])

for title in titles:
  print(title)
```

We'll make a new list called titles, and only add each row's title if it's not already in the list. Then, we can print all the titles:

We see that there are still near-duplicates, since Friends and friends are indeed different strings still.

We'll want to change the current title to all uppercase, and remove whitespace around it, before we add it to our list

```
import csv

titles = [ ]

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    title = row["title"].strip().upper()
    if not title in titles:
      titles.append(title)

for title in titles:
  print(title)
```

Now, we've **canonicalized**, or standardized, our data, and our list of titles are much cleaner:

It turns out that Python has another data structure built-in, `set`, which ensures that all the values are unique:

```
import csv

titles = set( )

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    title = row["title"].strip().upper()
    titles.add(title)

for title in titles:
  print(title)
```

Now, we can call `add` on the set, and not have to check ourselves if it's already in the set.

To sort the titles, we can just change our loop to `for title in sorted(titles)`, which will sort our set before we iterate over it:

```
import csv

titles = set( )

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    title = row["title"].strip().upper()
    titles.add(title)

for title in sorted(titles):
  print(title)
```

Counting

We can use a dictionary, instead of a set, to count the number of times we've seen each title, with the keys being the titles and the values being an integer counting the number of times we see each of them:

We have a KeyError, since the title HOW I MET YOUR MOTHER isn't in the dictionary yet.

We'll have to add each title to our dictionary first, and set the initial value to 1:

```
import csv

titles = { }

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    title = row["title"].strip().upper()
    if title in titles:
      titles[title] += 1
    else:
      titles[title] = 1

for title in sorted(titles):
  print(title, titles[title])
```

We can also set the initial value to 0, and then increment it by 1 no matter what

```
import csv

titles = {}

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    title = row["title"].strip().upper()
    if not title in titles:
      titles[title] = 0
    titles[title] += 1

for title in sorted(titles):
  print(title, titles[title])
```

Now, the key will exist in the dictionary, and we can safely refer to its value in the dictionary.

We can sort by the values in the dictionary by changing our loop to:

```
...
def get_value(title):
  return titles[title]

for title in sorted(titles, key=get_value, reverse=True):
  print(title, titles[title])
```

We define a function, f, which just returns the value of a title in the dictionary with titles[title]. The sorted function, in turn, will take in that function as the key to sort the dictionary. And we'll also pass in reverse=True to sort from largest to smallest, instead of smallest to largest.

We can actually define our function in the same line, with this syntax:

```
for title in sorted(titles, key=lambda title: titles[title], reverse=True):
  print(title, titles[title])
```

We can write and pass in a **lambda**, or anonymous function, which has no name but takes in some argument or arguments, and returns a value immediately.

Notice that there are no parentheses or return keyword, but concisely has the same effect as our get_value function earlier

We can also try to count all the occurrences of a specific title:

Now, if our data referred to the same show in different ways, we can try to check if the word "OFFICE" was in the title at all:

```
import csv

counter = 0

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    title = row["title"].strip().upper()
    if "OFFICE" in title:
      counter += 1

print(f"Number of people who like The Office: {counter}")
```

We'll have a simple counter variable, and add one to it

It turns out that a row has a typo, "Thevoffice", so now our count is correct.

We can also use **regular expressions**, a standardized way to represent a pattern that a string must match

For example, we can write a regular expression that matches email addresses:

```
.*@.*\. .*
```

The first period, ., indicates any character. The following asterisk, *, indicates 0 or more times. Then, we want an at sign, @. Then we want 0 or more characters again, .*, and then a literal period in our string, escaped with \. Finally, we want 0 or more characters again with .*

Since we probably want at least 1 character in each segment of an email address, we should change our regular expression to:

```
.+@.+\. .+
```

The plus sign, +, means we are matching for the previous character 1 or more times.

We can restrict the domain of the email to .edu by changing our regular expression to .+@.+\.edu

Languages like Python and JavaScript support regular expressions, which are like a mini-language in themselves, with syntax like:

- . for any character
- .* for 0 or more characters
- .+ for 1 or more characters
- ? for an optional character
- ^ for start of input
- $ for end of input

We can change our program earlier to use re, a Python library for regular expressions:

```
import csv
import re

counter = 0

with open("favorites.csv", "r") as file:
  reader = csv.DictReader(file)

  for row in reader:
    title = row["title"].strip().upper()
    if re.search("OFFICE", title):
      counter += 1

print(f"Number of people who like The Office: {counter}")
```

- The re library has a function, search, to which we can pass a pattern and string to see if there is a match.
- We can change our expression to "^(OFFICE|THE OFFICE)$", which will match either OFFICE or THE OFFICE, but only if they start at the beginning of the string, and stop at the end of the string (i.e., there are no other words before or after).
- We can even change THE OFFICE to THE.OFFICE, allowing any character (like a typo) to be in between those words.

We can also write a program to ask the user for a particular title and report its popularity:

```
import csv

title = input("Title: ").strip().upper()

counter = 0

with open("favorites.csv", "r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        if row["title"].strip().upper() == title:
            counter += 1

print(counter)
```

We ask the user for input, and then open our CSV file. Since we're looking for just one title, we can have one `counter` variable that we increment.

We check for a match after standardizing both the user's input and each row's title