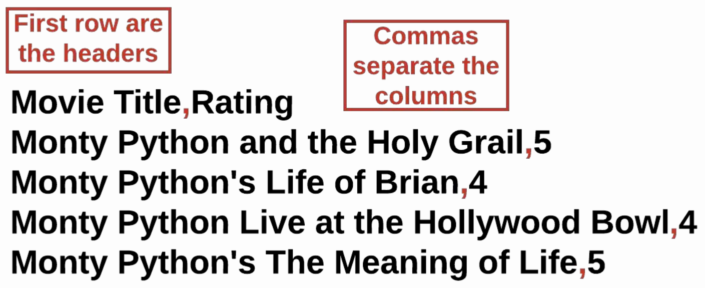
## Learning Objectives - CSV

- **Explain the “syntax” of a CSV file**
- **Print the contents of CSV file in a human-readable way**
- **Unpack for loop variables based on a CSV file**
- **Read and write a CSV file with different delimiters**
- **Write to a CSV file with `writerow` and `writerows`**

# CSV Files

## CSV Files

Python can work with files besides just text files. Comma Separated Value (CSV) files are an example of a commonly used file format for storing data. CSV files are similar to a spreadsheet in that data is stored in rows and columns. Each row of data is on its own line in the file, and commas are used to indicate a new column. Here is an example of a CSV file.


First row are the headers
Commas separate the columns

```
Movie Title,Rating
Monty Python and the Holy Grail,5
Monty Python's Life of Brian,4
Monty Python Live at the Hollywood Bowl,4
Monty Python's The Meaning of Life,5
```

Monty Python CSV

In order to read a CSV file, Python needs to import the `csv` module. If you are importing a module, always start your code with the import statements. The CSV file will be opened much like a text file, but Python needs to run the file through a CSV reader.

```
import csv

with open("student_folder/csv/monty_python_movies.csv", "r") as
        input_file:
    reader = csv.reader(input_file)
    for row in reader:
        print(row)
```

challenge

## What happens if you:

- Remove the `import csv` line of code?
- Use the CSV filename as the parameter of `reader`:

```
import csv

with open("student_folder/csv/monty_python_movies.csv", "r")
        as input_file:
    reader = csv.reader("monty_python_movies.csv")
    for row in reader:
        for data in row:
            print(data)
```

## Next

The first row of a CSV file is helpful because the header values provide context for the data. However, the first row is not useful if you want to know how many rows of data, or calculate the avg value, etc. The `next` command allows Python to skip the first row before looping through the CSV file.

```
import csv

with open("student_folder/csv/home_runs.csv", "r") as
        input_file:
    reader = csv.reader(input_file)
    next(reader) #skip the header row
    for row in reader:
        print(row)
```

challenge

## What happens if you:

- Remove the line `next(reader)`?
- Have two lines of `next(reader)`?

# Printing CSV Data

## Printing CSV Data

Previously, printing the contents of the CSV file would return lists of strings. This is not visually pleasing way of presenting the data. In the code below, `row` is a list. So you can reference each element with an index. This keeps Python from printing square brackets, quotation marks, etc. Instead, it prints just the contents of the CSV file.

```
import csv

with open("student_folder/csv/home_runs.csv", "r") as
        input_file:
    reader = csv.reader(input_file)
    for row in reader:
        print(row[0], row[1], row[2])
```

challenge

### What happens if you:

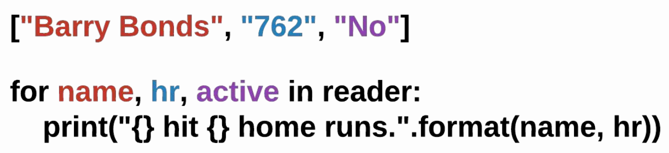
- Add a space between each column of data: `print(row[0], " ", row[1], " ", row[2])`
- Add a tab between each column of data: `print(row[0], "\t", row[1], "\t", row[2])`
- Change the print statement to: `print("{:<14} \t {:<9} \t {:^12}".format(row[0], row[1], row[2]))`

▼ **What does `{:<14}` mean?**

Printing output that is aligned into columns may not work if you try to use just spaces and tabs to separate the columns. `{:<14}` means left-justify the text and use a width of 14. 14 was chosen because that is the length of the longest names (Alex Rodriguez and Frank Robinson). Names shorter than 14 will “fill” the rest of the width with spaces. `{:>14}` means the text will be right-justified with a width of 14. `{:^14}` means the text will be centered across 14 spaces. Note, the `format` method needs to be used when aligning text.

## Unpacking

In the code above, the variable `row` represents a list of data. The first element is the name of the player, the second element is the number of career home runs, and the third element states they are currently an active player. Python provides a way to take the descriptions of the each element and use it in the for loop.



```
["Barry Bonds", "762", "No"]

for name, hr, active in reader:
    print("{} hit {} home runs.".format(name, hr))
```

Unpacking CSV Info

The for loop has three variables:`name`, `hr`, and `active`. The first variable, `name`, represents the first element in the list, the second variable, `hr`, represents the second element of the list, and the third variable, `active`, represents the third element. This is called unpacking.

```
import csv

with open("student_folder/csv/home_runs.csv", "r") as
        input_file:
    reader = csv.reader(input_file)
    next(reader) #skip the header names
    for name, hr, active in reader:
        print("{} hit {} home runs.".format(name, hr))
```

challenge

## What happens if you:

- Remove `next(reader)` from the program?
- Remove the variable `active`?

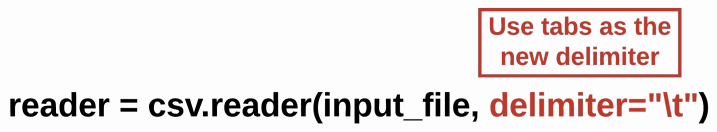
▼ **Unpacking and list length**

Unpacking only works if you know how many elements are in the list. You must have the same number variables in the for loop as there are elements in the list. If you don’t know how long a list is, you can always iterate over the list to access all of the elements.

# Delimiters

## Delimiters

Delimiters are a predefined character that separates one piece of information from another. CSV files use commas as the delimiter by default. However, this makes the file hard to read for humans. It is possible to change the delimiter in Python ( to see an example), but your code must reflect this change.



```
reader = csv.reader(input_file, delimiter="\t")
```

Tab Delimiter

```
import csv

with open("student_folder/csv/data_with_tabs.csv", "r") as
        input_file:
    reader = csv.reader(input_file, delimiter="\t")
    for row in reader:
        print(row)
```

challenge

### What happens if you:

- Change the delimiter to `,`?

▼ **Why did the output change when the delimiter changed?**

There is a slight difference when the delimiter is a tab and when it is a comma. With a tab delimiter, each row is a list of three strings. When the delimiter is a comma, each row is a list with a single string. Python cannot divide the data into the month, high temperature, and low temperature because it cannot find the delimiter. So it returns one, long string.

# Writing to CSV Files

## Writerow

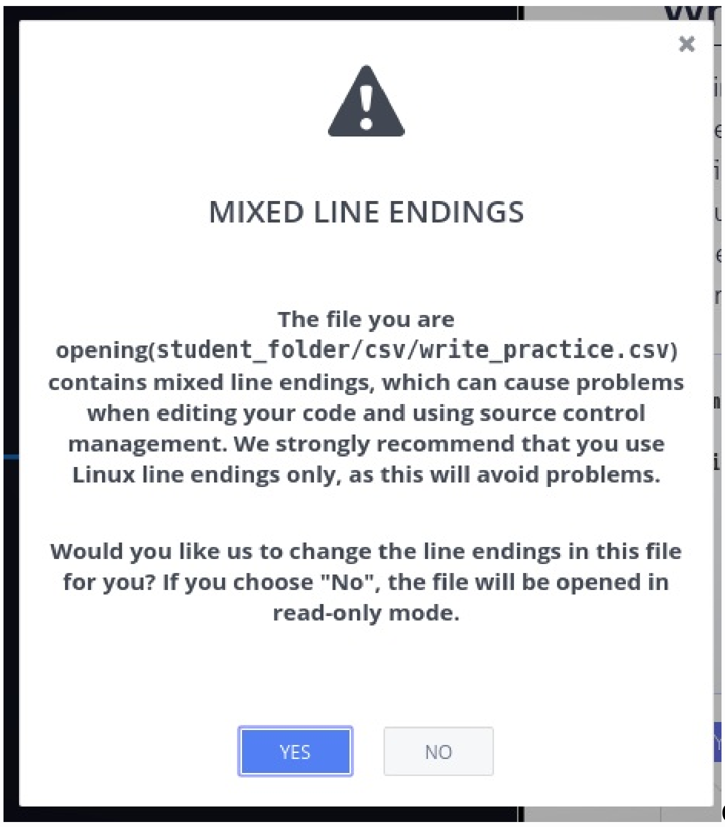
Writing to a CSV file is similar to writing to a text file. Open the file and set the mode to write (`"w"`). If the file does not exist, Python will create it. Send content to the file with the method `writerow`. Like reading a CSV, you need to import the `csv` module. Instead of using a `reader`, you need to use a `writter` to write information to the file. When you read information from a CSV file, it is a list of strings. So information written to a CSV file should also be a list of strings.

```
import csv

with open("student_folder/csv/write_practice.csv", "w") as
        output_file:
    writer = csv.writer(output_file, lineterminator="\n")
    writer.writerow(["Greeting", "Language"])
    writer.writerow(["Hello", "English"])
    writer.writerow(["Bonjour", "French"])
    writer.writerow(["Hola", "Spanish"])
    writer.writerow(["Namaste", "Hindi"])
```

▼ **What does `lineterminator` mean?**

The `csv` writer ends each line with the escape characters . The combination of two different escape characters causes the following warning.



MIXED LINE ENDINGS

The file you are opening(`student_folder/csv/write_practice.csv`) contains mixed line endings, which can cause problems when editing your code and using source control management. We strongly recommend that you use Linux line endings only, as this will avoid problems.

Would you like us to change the line endings in this file for you? If you choose "No", the file will be opened in read-only mode.

Changing the `lineterminator` to  will remove this warning because there will only be one escape character at the end of each line.

challenge

### What happens if you:

- Add a different delimiter: `csv.writer(output_file, delimiter="\t")`?
- Remove the last two lines of code and run it again?
- Change the mode to append `"a"` and run the program again?

## Writerows

The `writerow` method writes only one row of information to a CSV file. The `writerows` method can write several rows of information to a CSV file. `writerows` takes either a list of strings (a single row of information) or a list of lists of strings (many rows of information).

**Single Row - A List of Strings**

```
["Artist", "Album", "Copies"]
```

**Many Rows - A List of Lists of Strings**

```
[
  ["Artist",          "Album",                        "Copies"],
  ["Michael Jackson", "Thriller",                     "47 million"],
  ["Eagles",          "Their Greatest Hits 1971-1975", "38 million"],
  ["Eagles",          "Hotel California",             "26 million"]
]
```

Writerows

```
import csv

with open("student_folder/csv/write_practice.csv", "w") as
        output_file:
    writer = csv.writer(output_file, lineterminator="\n")
    writer.writerows([
      ["Artist", "Album", "Copies"],
      ["Michael Jackson", "Thriller", "47 million"],
      ["Eagles", "Their Greatest Hits 1971-1975", "38 million"],
      ["Eagles", "Hotel California", "26 million"]
    ])
```

challenge

## What happens if you:

- Change `writerows` to `writerow` and open the file?
- Change the mode to append `"a"` and run the program again?

# Formative Assessment 1

## Formative Assessment 2