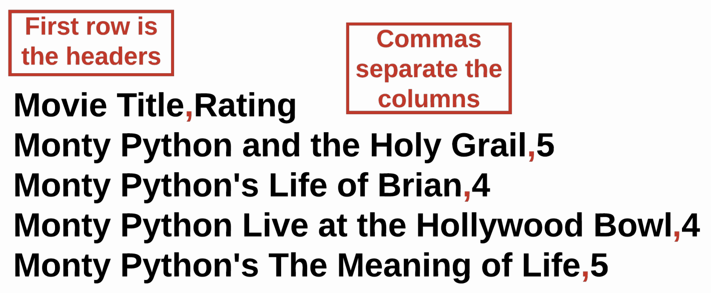
Learning Objectives - CSV

- **Define CSV**
- **Read data from a CSV file**
- **Iterate over a CSV file**
- **Print data from a CSV file with formatting**
- **Parse a CSV file with a different delimiter**
- **Write data to a CSV file**

CSV Files

CSV Files

Java 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. We are going to use the Opencsv package to work with CSV files.



```
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
```

Month Python CSV

In order to read a CSV file, Java needs to import the `opencsv` package and the `apache.commons` package in addition to the `java.io` package. The CSV file will be opened much like a text file; open the file in a `FileReader` object, but wrap it in a `CSVReader` object as opposed to a `BufferedReader` object.

```
//add code below this line
String path = "studentFolder/csv/montyPythonMovies.csv";
try {
  CSVReader reader = new CSVReader(new FileReader(path));

  reader.close();
} catch (Exception e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a CSV");
}
//add code above this line
```

▼ **What happened to the IO exceptions?**

In previous examples in this module, we were catching an `IOException`.

When using the `CSVReader` class, you need to also catch a `CsvValidationException`. Instead of writing two different `catch` statements, we are going to write a generic `catch` statement that will work for any exception in the code.

Using an enhanced loop, Java will read each row of the CSV file. Each row is an array of strings. To see contents of each row, use an enhanced loop to iterate over `row`. We want the information in `row` to remain on the same line, so use `print` instead of `println`, and add a space between each element. Once the enhanced loop has finished running, use `println` to go to the next line.

```
//add code below this line
String path = "studentFolder/csv/montyPythonMovies.csv";
try {
  CSVReader reader = new CSVReader(new FileReader(path));
  for (String[] row : reader) {
    for (String item : row) {
      System.out.print(item + " ");
    }
    System.out.println();
  }
  reader.close();
} catch (Exception e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a CSV");
}
//add code above this line
```

Skipping the Header

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 `skip` method tells the `CSVReader` object how many rows to skip in the CSV file.

```java
//add code below this line
String path = "studentFolder/csv/homeRuns.csv";
try {
  CSVReader reader = new CSVReader(new FileReader(path));
  reader.skip(1);
  for (String[] row : reader) {
    for (String item : row) {
      System.out.print(item + " ");
    }
    System.out.println();
  }
  reader.close();
} catch (Exception e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a CSV");
}
//add code above this line
```

challenge

Try this variation:

- Change the number of lines to skip to 3: `reader.skip(3);`
- Comment out the line with the `skip` method: `//reader.skip(3);`

Printing CSV Data

Printing CSV Data

Iterating over the CSV file and printing each line does not produce visually pleasing output. The code below produces three columns of data, but there is no consistency in the spacing between columns. In particular, the third column nowhere near aligned with the header `Active Player`.

```java
    //add code below this line
    String path = "studentFolder/csv/homeRuns.csv";
    try {
      CSVReader reader = new CSVReader(new FileReader(path));
      for (String[] row : reader) {
        for (String item : row) {
          System.out.print(item + "  ");
        }
        System.out.println();
      }
      reader.close();
    } catch (Exception e) {
      System.out.println(e);
    } finally {
      System.out.println("Finished reading a CSV");
    }
    //add code above this line
```

With the string `format` method, you can introduce consistent spacing printing a string. Start with `String.format("%s", item)`. That tells Java that the value stored in `item` is to be printed as a string. To add padding, insert 15 between the “%” and the “s” so it looks like `String.format("%17s", item)`. Now each string will have a fixed width of 15 spaces. The string “No”, it will have 13 blank spaces (15 - the length of the string). The output is better than before.

```
//add code below this line
String path = "studentFolder/csv/homeRuns.csv";
try {
  CSVReader reader = new CSVReader(new FileReader(path));
  for (String[] row : reader) {
    for (String item : row) {
      System.out.print(String.format("%15s", item));
    }
    System.out.println();
  }
  reader.close();
} catch (Exception e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a CSV");
}
//add code above this line
```

challenge

Try this variation:

- Change the string formatting to: `String.format("%-15s", item)`

Consistent spacing is better than no spacing. However, not every column needs the same width. Some are wider than others. To give unique spacing to each column, remove the enhanced loop that iterates over the `row` array. The array has three elements. The first (player’s name) needs a width of 17 spaces, the second element (home runs) needs a width of 11 spaces, and the third element (active player) needs 13 spaces of padding. This creates more user-friendly output.

```java
//add code below this line
String path = "studentFolder/csv/homeRuns.csv";
try {
  CSVReader reader = new CSVReader(new FileReader(path));
  for (String[] row : reader) {
    String column1 = String.format("%-17s", row[0]);
    String column2 = String.format("%-11s", row[1]);
    String column3 = String.format("%-13s", row[2]);
    System.out.println(column1 + column2 + column3);
  }
  reader.close();
} catch (Exception e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a CSV");
}
//add code above this line
```

challenge

Try this variation:

- Add two dashed lines to the output:

```java
for (String[] row : reader) {
  System.out.println("--------------------------------
--------------");
  String column1 = String.format("|%-17s", row[0]);
  String column2 = String.format("|%-11s|", row[1]);
  String column3 = String.format("%-13s|", row[2]);
  System.out.println(column1 + column2 + column3);
}
System.out.println("----------------------------------
------------");
```

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 Java to a tab (click here to see an example), but your code must reflect this change. We are no longer going to use the OpenCSV package. Instead, we are going to manually parse the file with a `BufferedReader` object.

Use the `ready` method from the lesson on iterating over a file. Create the string `line` which represents each line of the CSV file. Within `line`, there are tabs (`"\t"`). We want to take the long string and convert it to an array of smaller strings based on the position of the tab. These smaller strings are called tokens. The `split` method takes a string and returns an array of strings split on a given character. Create the array `tokens` which is the string `line` split on the value of the variable `delimiter`. Finally, iterate over `tokens` and print each token with some formatting.

```java
//add code below this line
String path = "studentFolder/csv/dataWithTabs.csv";
String delimiter = "\t";
try {
  BufferedReader reader = new BufferedReader(new
    FileReader(path));
  while(reader.ready()) {
    String line = reader.readLine();
    String[] tokens = line.split(delimiter);
    for(String token : tokens) {
      System.out.print(String.format("%-10s", token));
    }
    System.out.println();
  }
  reader.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished reading a CSV");
}
//add code above this line
```

challenge

What happens if you:

- Change the delimiter from `'\t'` 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 an array of three strings. When the delimiter is a comma, each row is an array with a single string. Java 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

CSVWriter

If a `CSVReader` object is used to read a CSV file, then a `CSVWriter` object is used to write to a CSV file. Create a `CSVWriter` object. Create an array of strings to represent each row of the CSV file. Remember, there should be a header row that describes each column of data. Use the `writeNext` method to write the array of strings to the file. Click on the link to open the file after running the code.

```
//add code below this line
String path = "studentFolder/csv/writePractice.csv";
try {
  CSVWriter writer = new CSVWriter(new FileWriter(path));

  String[] header = {"Greeting", "Language"};
  String[] row1 = {"Hello", "English"};
  String[] row2 = {"Bonjour", "French"};
  String[] row3 = {"Hola", "Spanish"};
  String[] row4 = {"Namaste", "Hindi"};

  writer.writeNext(header);
  writer.writeNext(row1);
  writer.writeNext(row2);
  writer.writeNext(row3);
  writer.writeNext(row4);

  writer.close();
} catch (IOException e) {
  System.out.println(e);
} finally {
  System.out.println("Finished writing to a CSV");
}
//add code above this line
```

Open the File

challenge

Try these variations:

- Comment out the last two lines when writing data to the file:

```java
writer.writeNext(header);
writer.writeNext(row1);
writer.writeNext(row2);
// writer.writeNext(row3);
// writer.writeNext(row4);
```

- “Turn on” the ability to append to a file by adding `true` to the `FileWriter` object:

```java
CSVWriter writer = new CSVWriter(new FileWriter(path,
  true));
```

Open the File