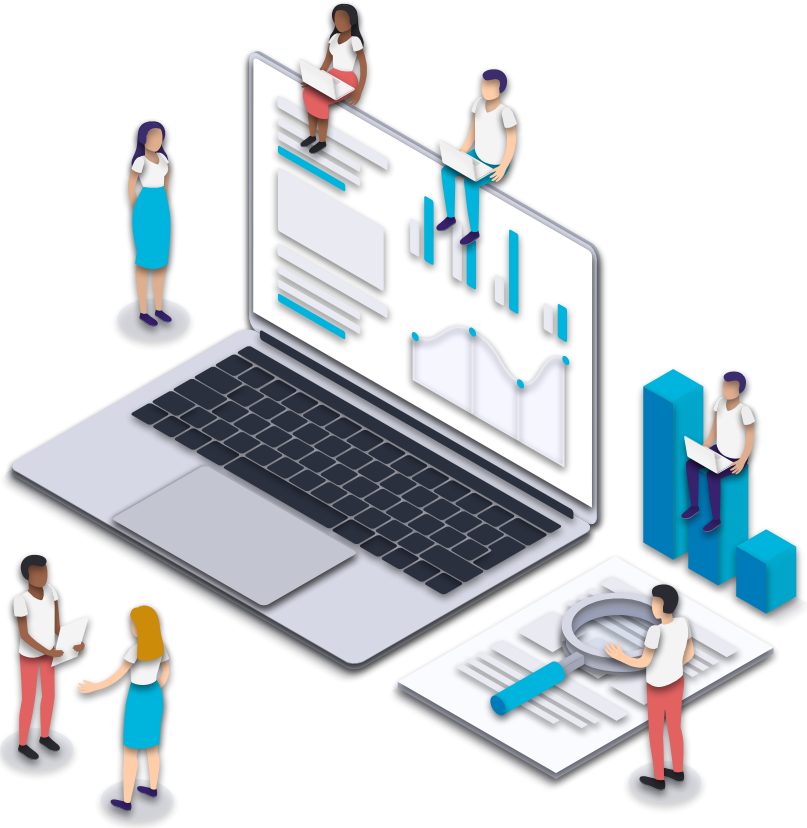
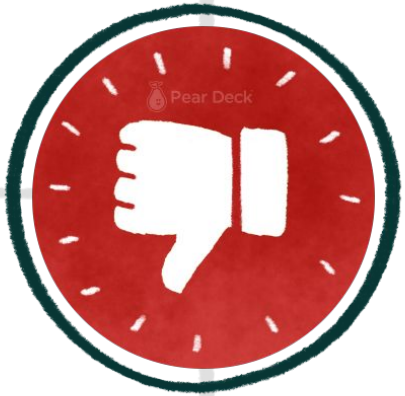


Writing Files

Table of Contents

1 Writing to File with `.write()` Method

How was the pre-class content of the “*writing to files*”?



Students, drag the icon!

Explain the difference between modes "a" and "w"

Students, write your response!

Writing to File with `.write()` Method (review)

- As we mentioned earlier, we can **overwrite** a text to a file using `'w'` mode, which means that every time we use `'w'`, the content of the file is deleted and new content is written. If there isn't any file then it will be created automatically.

Tips:

- Note that data in the other type that we intend to write to the file must be converted to **string type** before the writing process.

Writing to File with `.write()` Method (review)

- Let's create and write string data to a file. We're going to use `.write()` method for writing :

```
1  with open("dummy_file.txt", 'w', encoding="utf-8") as file:
2  # we create and open the file
3
4      file.write('This is the first line of my text file')
5      # writes str data into file
6
7  with open("dummy_file.txt", 'r', encoding="utf-8") as file:
8      print(file.read())  # reads the content of the 'dummy_file'
```

What is the output? Try to figure out in your mind...

Writing to File with `.write()` Method (review)

- It gives an output what we entered using `.write()` method.

```
1  with open("dummy_file.txt", 'w', encoding="utf-8") as file:
2  # we create and open the file
3
4      file.write('This is the first line of my text file')
5      # writes str data into file
6
7  with open("dummy_file.txt", 'r', encoding="utf-8") as file:
8      print(file.read())  # reads the content of the 'dummy_file'
```

```
1  This is the first line of my text file
2
```

Writing to File with `.write()` Method (review)

- Now let's repeat the process and see what happens. This time the file (**dummy_file**) exists :

```
with open("dummy_file.txt", 'w', encoding="utf-8") as file:
    file.write('This is the new line for my dummy_file')
    # we write new str data into it

with open("dummy_file.txt", 'r', encoding="utf-8") as file:
    print(file.read())  # reads the content of the 'dummy_file'
```

Writing to File with `.write()` Method (review)

- Now let's repeat the process and see what happens. This time the file (**dummy_file**) exists :

```
1  with open("dummy_file.txt", 'w', encoding="utf-8") as file:
2      file.write('This is the new line for my dummy_file')
3      # we write new str data into it
4
5  with open("dummy_file.txt", 'r', encoding="utf-8") as file:
6      print(file.read())  # reads the content of the 'dummy_file'
```

What is the output? Try to figure out in your mind...

Students, write your response!

Writing to File with `.write()` Method (review)

- Now let's repeat the process and see what happens. This time the file (**dummy_file**) exists :

```
with open("dummy_file.txt", 'w', encoding="utf-8") as file:
    file.write('This is the new line for my dummy_file')
    # we write new str data into it

with open("dummy_file.txt", 'r', encoding="utf-8") as file:
    print(file.read())  # reads the content of the 'dummy_file'
```

```
This is the new line for my dummy_file

```

Writing to File with `.write()` Method (review)

- When you write strings to a file using the `.write()` method, the string data is written exactly as it is.
- Whenever you use `.write()` method, each string joined together into one. Therefore you have to put newline characters (`\n`), separators or spaces, etc. manually if you want.
- Consider the following example :

Writing to File with `.write()` Method (review)

- Let's **write** 5 sentences into the **`dummy_file.txt`** file we created before and then **read** the content of that file.

```
1  with open("dummy_file.txt", 'w', encoding="utf-8") as file:
2      file.write('My first sentence')
3      file.write('My second sentence,')
4      file.write('My third sentence\n')
5      file.write('My fourth sentence ')
6      file.write('My last sentence')
7
8  with open("dummy_file.txt", 'r', encoding="utf-8") as file:
9      print(file.read())
```

What is the output? Try to figure out in your mind...

Students, write your response!

Writing to File with `.write()` Method (review)

- The output is as follows :

```
with open("dummy_file.txt", 'w', encoding="utf-8") as file:
    file.write('My first sentence')
    file.write('My second sentence,')
    file.write('My third sentence\n')
    file.write('My fourth sentence ')
    file.write('My last sentence')

with open("dummy_file.txt", 'r', encoding="utf-8") as file:
    print(file.read())
```

```
My first sentenceMy second sentence,My third sentence
My fourth sentence My last sentence

```

Writing to File with `.write()` Method (review)

- **Task :**
  - Now, think of that we have a **`list`** of fruit names.
  - Let's write them to a file named **`fruits.txt`** each on separate lines one after another.
  - Read and display the entire content,
  - Read and display the content in a **`list`** form.

```
1  fruits = ['Banana', 'Orange', 'Apple', 'Strawberry', 'Cherry']
2
```

Writing to File with `.write()` Method (review)

- The code snippet can be as follows :

```
1  fruits = ['Banana', 'Orange', 'Apple', 'Strawberry', 'Cherry']
2
3  with open("fruits.txt", 'w', encoding="utf-8") as file:
4      for basket in fruits:
5          file.write(basket + '\n')  # adds a newline character to each
               string
6
7  with open("fruits.txt", 'r', encoding="utf-8") as file:
8      print(file.read())
9
10 with open("fruits.txt", 'r', encoding="utf-8") as file:
11     print(file.readlines())  # reads and displays entire lines in a list
```

What is the output? Try to figure out in your mind...

Writing to File with `.write()` Method (review)

- Here the output is :

```
Banana
Orange
Apple
Strawberry
Cherry

['Banana\n', 'Orange\n', 'Apple\n', 'Strawberry\n', 'Cherry\n']

```

Writing to File with `.write()` Method

- **Task :**
  - Now, think of that we have a **`list`** of flower names.
  - Let's write them to a file named **`flowers.txt`** each on separate lines one after another and separated by an empty line.

```
1 flowers = ['Jasmine', 'Rose', 'Lily', 'Daisy', 'Tulip']
2
```

Writing to File with `.write()` Method

- The code snippet can be as follows :

```
1  flowers = ['Jasmine', 'Rose', 'Lily', 'Daisy', 'Tulip']
2
3  with open("flowers.txt", 'w', encoding="utf-8") as file:
4      for basket in flowers:
5          file.write(basket + "\n\n")
6
7  with open("flowers.txt", 'r', encoding="utf-8") as file:
8      print(file.read())
9
```

What is the output? Try to figure out in your mind...

Writing to File with `.write()` Method

- The output is as follows :

```
1 flowers = ['Jasmine', 'Rose', 'Lily', 'Daisy', 'Tulip']
2
```

Output

```
Jasmine

Rose

Lily

Daisy

Tulip
```

2 Writing to File with `.writelines()` Method

Make connections

How are these two methods connected?

Type your answers.

`.readlines()`

`.writelines()`

Students, write your response!

Writing to File with `.writelines()` Method (review)

- There is another method for writing data to the files. It is `.writelines()` method. Unlike `.write()` method, `.writelines()` takes the iterable sequence of strings and writes them to the file.
- The difference between these two methods is just similar to the logic of difference between `.read()` and `.readlines()`.

Writing to File with `.writelines()` Method (review)

- Let's use the same **`list`** of the fruits again but this time in a little bit different way. We should choose the line separators ourselves without using **`for`** loop. Let's see how we do it :

```
1  fruits = ['Banana\n', 'Orange\n', 'Apple\n', 'Strawberry\n', 'Cherry\n']
2
3  with open("fruits.txt", 'w', encoding="utf-8") as file:
4      file.writelines(fruits)  # takes an iterator for writing
5
6  with open("fruits.txt", 'r', encoding="utf-8") as file:
7      print(file.read())
8
9  with open("fruits.txt", 'r', encoding="utf-8") as file:
10     print(file.readlines())
```

Writing to File with `.writelines()` Method (review)

- The output looks like :

```
Banana
Orange
Apple
Strawberry
Cherry

['Banana\n', 'Orange\n', 'Apple\n', 'Strawberry\n', 'Cherry\n']

```

Writing to File with `.writelines()` Method

- **Task :**
  - Use the same **`list`** of flower names,
  - Modify the **`list`** for use,
  - Overwrite them to the same **`flowers.txt`** file each on separate lines one after another.

```
1 flowers = ['Jasmine', 'Rose', 'Lily', 'Daisy', 'Tulip']
2
```

Writing to File with `.writelines()` Method

- The code snippet and the output can be as follows :

```
1  flowers = ['Jasmine\n', 'Rose\n', 'Lily\n', 'Daisy\n', 'Tulip']
2
3  with open("flowers.txt", 'w', encoding="utf-8") as file:
4      file.writelines(flowers)  # takes "flowers" as an iterator
5
6  with open("flowers.txt", 'r', encoding="utf-8") as file:
7      print(file.read())
8
```

Output

```
Jasmine
Rose
Lily
Daisy
Tulip
```

3

Appending to File with `'a'`

Appending to File with `'a'` (review)

- Unlike the previous writing mode (`'w'`), in most cases when we want to add new content to a file, deleting the existing content is undesirable.
- Therefore, there is a need for another mode that both keeps the existing content of the file and saves the new content to the continuation of the file. **In Python, we meet this need with `'a'` mode** which stands for **append**.

Appending to File with `'a'` (review)

- **Task :**
  - Let's add `'melon'` to our existing **`fruits.txt`** file as the last line,
  - Read and display the entire file content,
  - Read and display the entire file content line by line in a **`list`** form.

Appending to File with `'a'` (review)

- The code snippet and the output are as follows:

```
1  with open("fruits.txt", 'a', encoding="utf-8") as file:
2      file.write('Melon\n')  # adds Melon to the end of the text
3
4  with open("fruits.txt", 'r', encoding="utf-8") as file:
5      print(file.read())
6
7  with open("fruits.txt", 'r', encoding="utf-8") as file:
8      print(file.readlines())
```

```
1  Banana
2  Orange
3  Apple
4  Strawberry
5  Cherry
6  Melon
7
8  ['Banana\n', 'Orange\n', 'Apple\n', 'Strawberry\n', 'Cherry\n', 'Melon\n']
9
```

Appending to File with `'a'`

- **Task :**
  - Let's add `'orchid'` to our existing `flowers.txt` file as the last line,
  - Read and display the entire file content.

Appending to File with `'a'`

- The code snippet and the output are as follows:

```
1  with open("flowers.txt", 'a', encoding="utf-8") as file:
2      file.write("\nOrchid")
3
4  with open("flowers.txt", 'r', encoding="utf-8") as file:
5      print(file.read())
6
```

Since we didn't put a newline char (**\n**) at the end of the **Tulip** in the previous **"w"** operation, now we put **\n** here to place the **Orchid** as the last line.

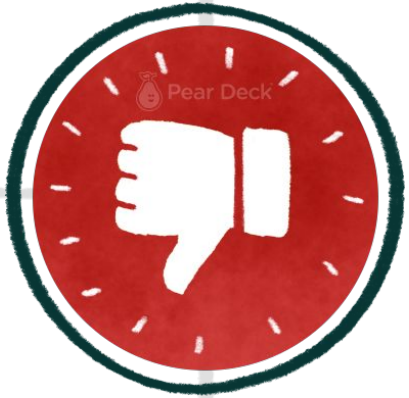
Output

```
Jasmine
Rose
Lily
Daisy
Tulip
Orchid
```

Did you find this lesson interesting and challenging?

Too hard

Just right

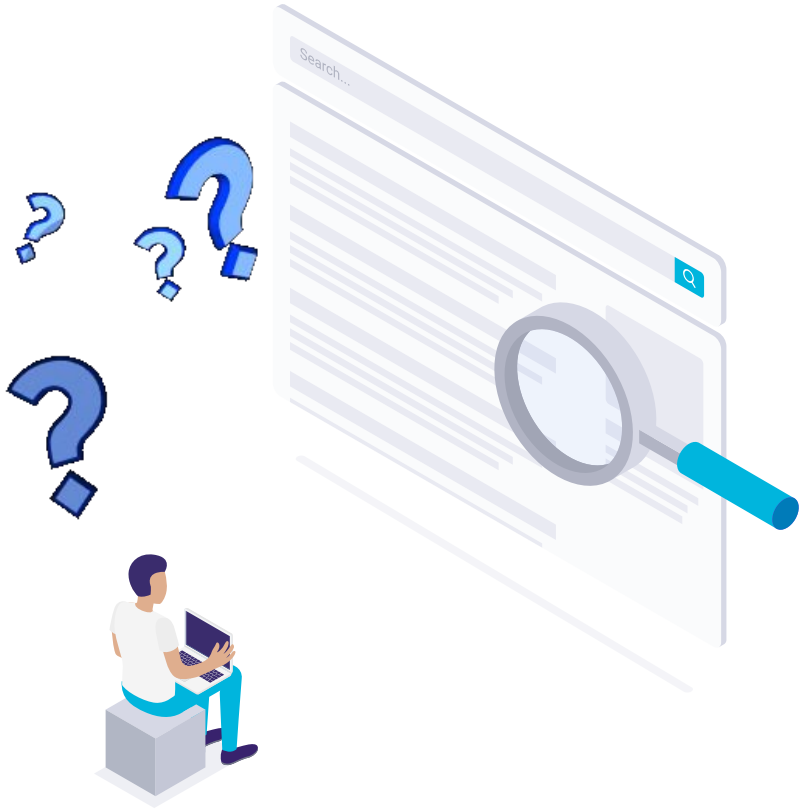


Too easy

Students, drag the icon!

THANKS!

Any questions?

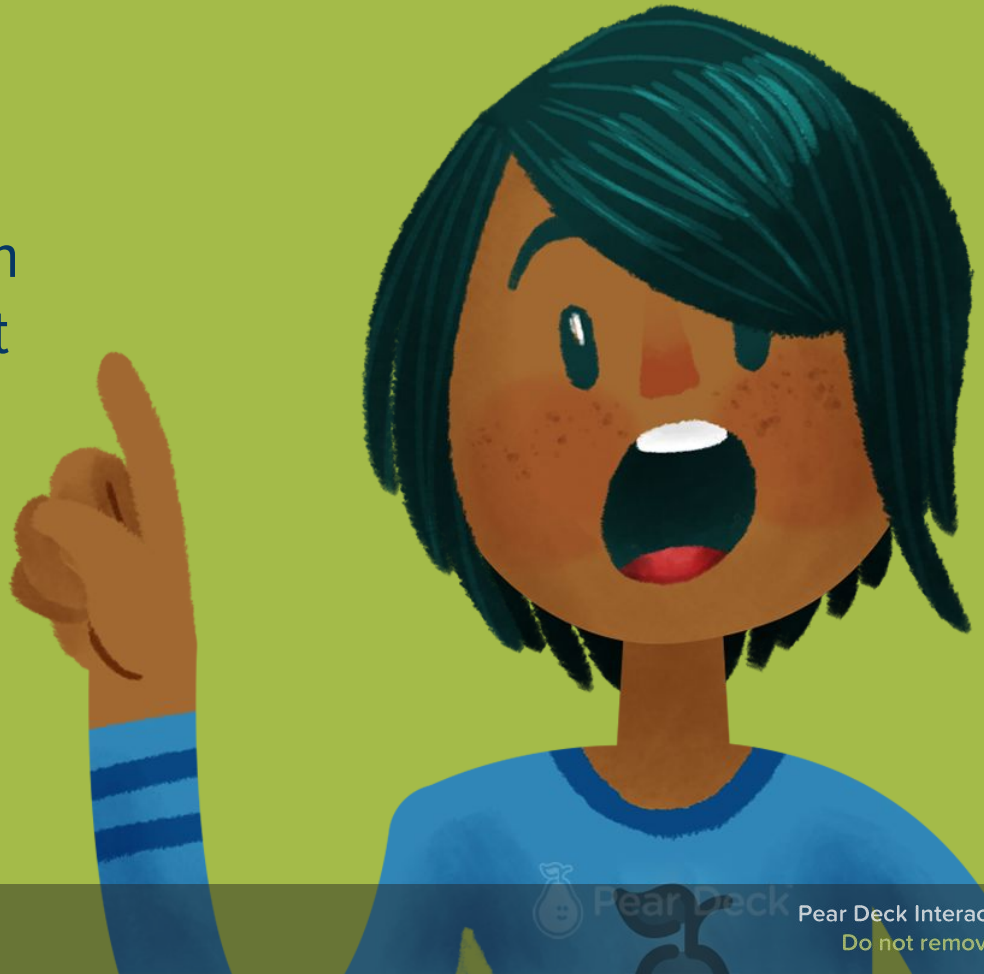


Working with the CSV Files

Table of Contents

File formats...

What did you learn from pre-class content about CSV files?



Students, write your response!

What is a CSV File?

CSV ⇨ **C**omma **S**eparated **V**alues

- Actually, it's a plain text like **.txt** files. Normally, CSV files use a **comma** to separate each specific data value. When you **save** the plain text file that you create by using a simple text editor (such as Notepad in Windows OS) as a **.csv** extension, then you create a CSV file.

What is a CSV File?

- A CSV file is basically a table made up of rows and columns. The structure of a simple CSV file is as follows :

```
column 1,     column 2,     column 3
row1_column1, row1_column2, row1_column3
row2_column1, row2_column2, row2_column3
row3_column1, row3_column2, row3_column3
```

What is a CSV File?

- It represents a table of data.

```
column 1,column 2,column 3
row1_column1, row1_column2, row1_column3
row2_column1, row2_column2, row2_column3
row3_column1, row3_column2, row3_column3
```

column1	column2	column3
row1_column1	row1_column2	row1_column3
row2_column1	row2_column2	row2_column3
row3_column1	row3_column2	row3_column3

What is a CSV File?

- Now, let's create our first CSV file. We can use the previous txt file named **`fruits.txt`**. As usual, save this file with a **.csv** extension into directory where Anconda3/Python installed.

`fruits.csv`

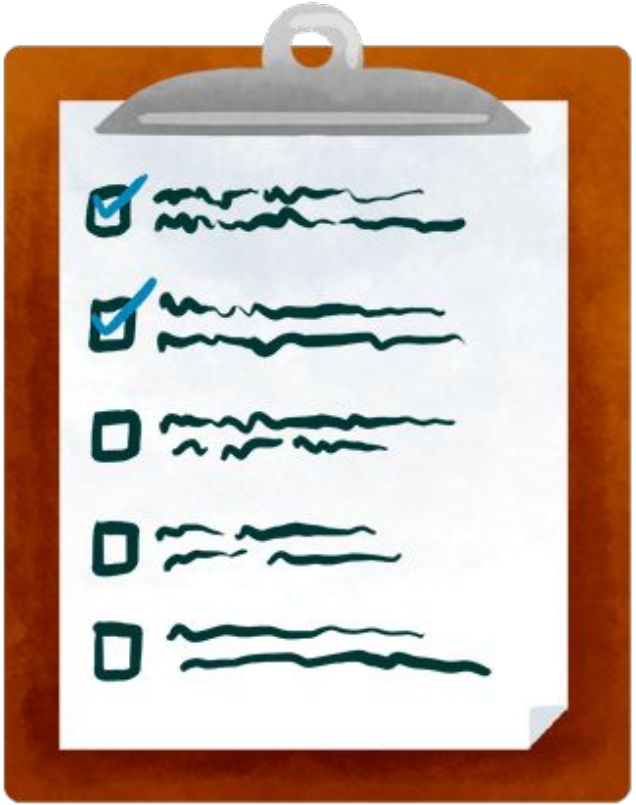
```
no,fruit,amount
1,Banana,4 lb
2,Orange,5 lb
3,Apple,2 lb
4,Strawberry,6 lb
5,Cherry,3 lb
```

2

Reading Methods of the CSV Files

Reading Methods of the CSV Files

1. You can use the ordinary way of reading using `open()` function and `.read()` method just you learned in the previous lesson.
2. You can use `csv` Module.
3. You can use the **Pandas** library for which created data analysis purposes. It is highly recommended for big data analysis.



CSV Files reading methods

*List the **reading methods** of **CSV Files** stated in the previous slide.*

Students, write your response!

3

Ordinary Reading Method of CSV Files

Ordinary Reading Method of CSV Files (review)

- Let's read the **fruits.csv** file we created.

```
1 with open("fruits.csv", 'r', encoding="utf-8") as file:
2     print(file.read())
```

What is the output? Try to figure out in your mind...

Ordinary Reading Method of CSV Files (review)

- Let's read the **fruits.csv** file we created in the previous lesson.

```
with open("fruits.csv", 'r', encoding="utf-8") as file:
    print(file.read())
```

```
no,fruit,amount
1,Banana,4 lb
2,Orange,5 lb
3,Apple,2 lb
4,Strawberry,6 lb
5,Cherry,3 lb

```

Ordinary Reading Method of CSV Files

- **Task :**
  - Create a **csv** file named **people.csv** consisting of **5** member of your family or friends,
  - The file should include **4 columns** (row number, first name, last name, ages) and **5 rows** (excluding the caption).
  - Read and display the content of **people.csv** file in an ordinary method.

Ordinary Reading Method of CSV Files

- **The file sample :**

people.csv

```
row num,first name,last name,ages
1,John,Doe,22
2,lara,bold,12
3,Adam,Smith,46
4,Henry,smithson,40
5,Mosely,Britt,51
```

Ordinary Reading Method of CSV Files

- The code snippet and the output are as follows :

```
1 with open("people.csv", "r", encoding="utf-8") as file:
2     print(file.read())
3
```

output

```
row num,first name,last name,ages
1,John,Doe,22
2,lara,bold,12
3,Adam,Smith,46
4,Henry,smithson,40
5,Mosely,Britt,51
```

4

Reading the CSV Files with `csv` Module

Reading the CSV Files with `csv` Module (review)

- Since the content of the CSV files is displayed in a more readable and regular format with this method, this method is more preferred than the previous one.
- **First**, we should load `csv` **module**. **Second**, we will use `csv.reader()` function in a `for` loop to read the CSV file.

Reading the CSV Files with `csv` Module (review)

- Examine the following example carefully :

```
1  import csv  # loads csv module
2
3  with open("fruits.csv", 'r', newline = '', encoding = 'utf-8') as file:
4      csv_rows = csv.reader(file)  # reader() function takes each
5                                   # row (lines) into a list
6      for row in csv_rows:
7          print(row)
```

Reading the CSV Files with `csv` Module (review)

- Examine the following example carefully :

```
1  import csv  # loads csv module
2
3  with open("fruits.csv", 'r', newline = '', encoding = 'utf-8') as file:
4      csv_rows = csv.reader(file)  # reader() function takes each
5                                   # row (lines) into a list
6      for row in csv_rows:
7          print(row)
```

```
1  ['no', 'fruit', 'amount']
2  ['1', 'Banana', '4 lb']
3  ['2', 'Orange', '5 lb']
4  ['3', 'Apple', '2 lb']
5  ['4', 'Strawberry', '6 lb']
6  ['5', 'Cherry', '3 lb']
7
```

Reading the CSV Files with `csv` Module

Tips:

- If `newline=''` is not specified, newlines embedded inside quoted fields will not be interpreted correctly, and on platforms that use `\r\n` linendings on write an extra `\r` will be added.
- It is best practice and always safe to specify `newline=''`, since the `csv` module does its own newline handling.

Reading the CSV Files with `csv` Module (review)

- The default value of the `delimiter` parameter of the `csv.reader()` function is `","`. Thus, we can display each comma-separated element as separate **strings**.
- At this time let's use the `delimiter=","` parameter and see the same result as the previous one.

```
csv.reader(file, delimiter = ",")
```

Reading the CSV Files with `csv` Module (review)

- We get the same output as the previous one :

```
1  import csv
2
3  with open("fruits.csv", 'r', newline = '', encoding = 'utf-8') as file:
4      csv_rows = csv.reader(file, delimiter= ",")  # gives the same output as the
           previous one
5
6      for row in csv_rows:
7          print(row)
```

```
1  ['no', 'fruit', 'amount']
2  ['1', 'Banana', '4 lb']
3  ['2', 'Orange', '5 lb']
4  ['3', 'Apple', '2 lb']
5  ['4', 'Strawberry', '6 lb']
6  ['5', 'Cherry', '3 lb']
7
```

Reading the CSV Files with `csv` Module

- **Task :**
  - Considering the **people.csv,** read and display the content of **it** using `csv` module.

Reading the CSV Files with `csv` Module

- The reading code and the output are as follows :

```
1  import csv
2
3  with open("people.csv", 'r', encoding = 'utf-8') as file:
4      csv_rows = csv.reader(file)
5
6      for row in csv_rows:
7          print(row)
8
```

output

```
['row num', 'first name', 'last name', 'ages']
['1', 'isabella', 'bold', '22']
['2', 'lara', 'bold', '12']
['3', 'solomon', 'bold', '46']
['4', 'adam', 'smithson', '40']
['5', 'mose', 'smithson', '51']
```

Reading the CSV Files with `csv` Module (review)

- We can also display each row in the **`list`** as a single **`string`**.
- If we determine a character that is not included in the **CSV file** and allocate this character as the value of **`delimiter`**, we can print all lines of the CSV file as **`list`**s as a single **`string`** element.
- For the **`delimiter`** parameter, let's pick **`":"`** which is not in the CSV file.

Reading the CSV Files with `csv` Module

- Let's take a look at this pre-class example :

```
1  import csv
2
3  with open("fruits.csv", 'r', newline = '', encoding = 'utf-8') as file:
4      csv_rows = csv.reader(file, delimiter = ':')  # we specified a char ":" that is
           not used
5                                                    # in the csv file as a value of
                                                 delimiter
6      for row in csv_rows:
7          print(row)
```

What is the output? Try to figure out in your mind...

Students, write your response!

Reading the CSV Files with `csv` Module

- Let's take a look at this pre-class example :

```
1  import csv
2
3  with open("fruits.csv", 'r', newline = '', encoding = 'utf-8') as file:
4      csv_rows = csv.reader(file, delimiter = ':')  # we specified a char ":" that is
           not used
5                                                    # in the csv file as a value of
                                                 delimiter
6      for row in csv_rows:
7          print(row)
```

```
1  ['no,fruit,amount']
2  ['1,Banana,4 lb']
3  ['2,Orange,5 lb']
4  ['3,Apple,2 lb']
5  ['4,Strawberry,6 lb']
6  ['5,Cherry,3 lb']
7
```

we get each row as a single string in a `list`

Reading the CSV Files with `csv` Module

- **Task :**
  - Considering the **people.csv,** read and display the content of **it** using `csv` module.
  - At this time display each row **as a single string** in a `list`.

Reading the CSV Files with `csv` Module

- The reading code and the output are as follows :

```
1  import csv
2
3  with open("people.csv", 'r', encoding = 'utf-8') as file:
4      csv_rows = csv.reader(file, delimiter = '-')  # it doesn't contain "-" char
5
6      for row in csv_rows:
7          print(row)
8
```

output

```
['row num,first name,last name,ages']
['1,isabella,bold,22']
['2,lara,bold,12']
['3,solomon,bold,46']
['4,adam,smithson,40']
['5,mose,smithson,51']
```

THANKS!

Any questions?