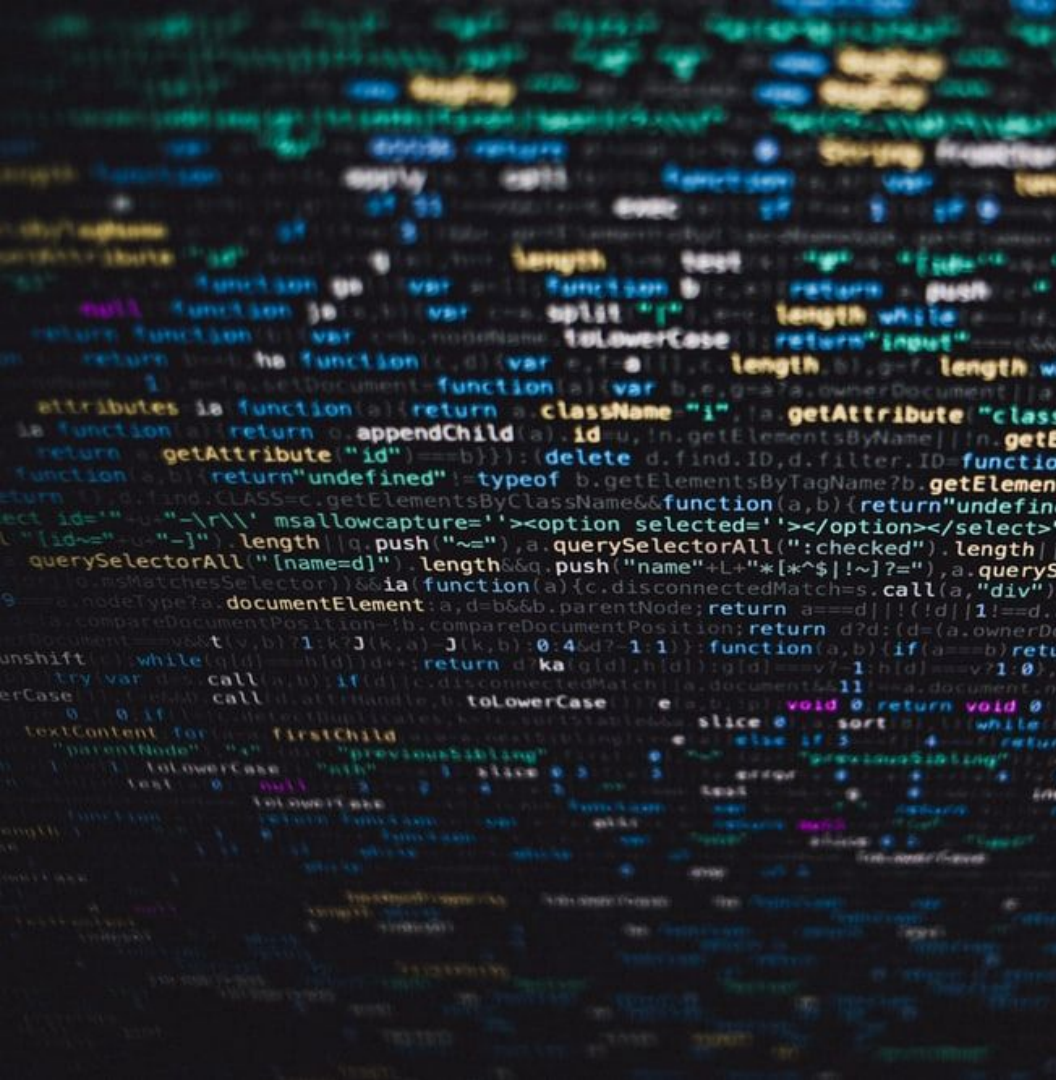
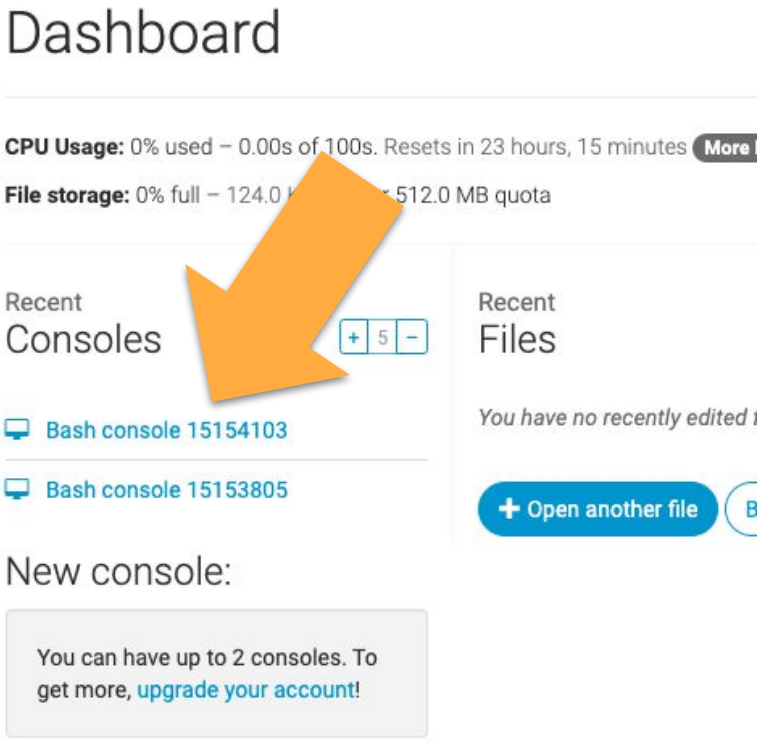


CODE CAMP

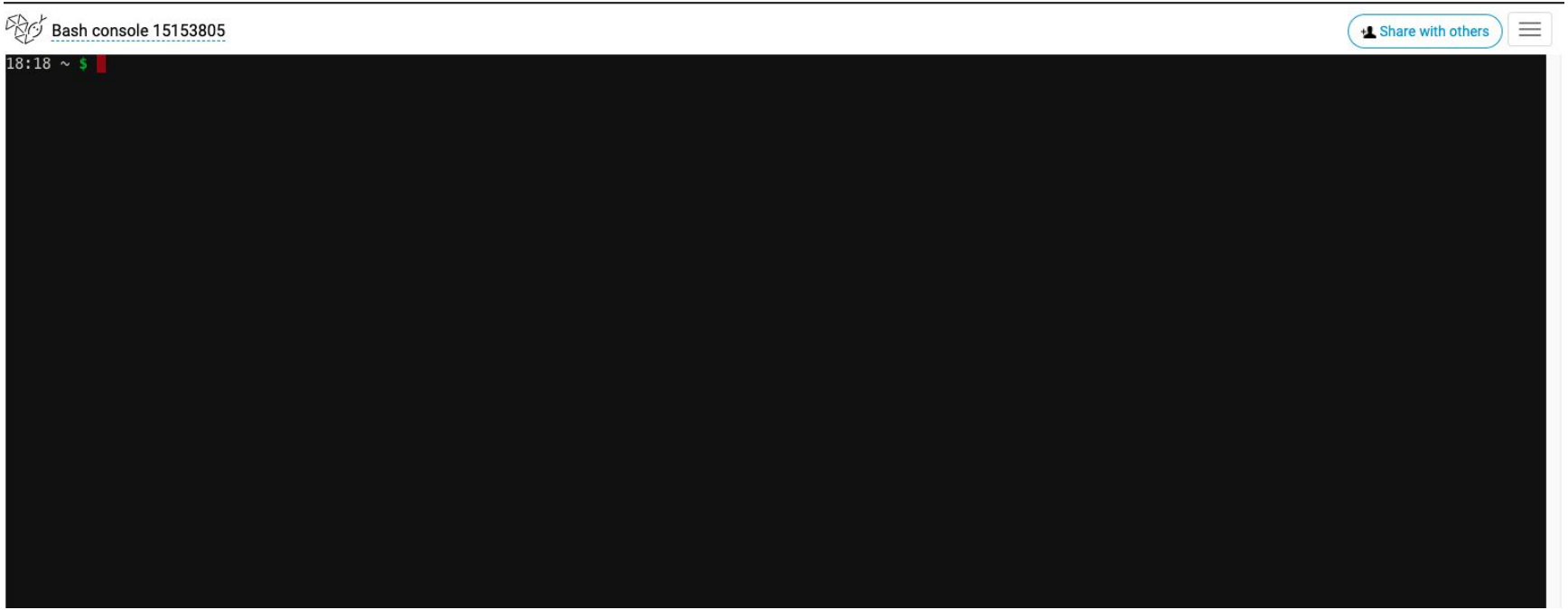
Day 008: File I/O



.001 Log In to Python Anywhere

Visit PythonAnywhere.com and log in to your account. You can click on one of the bash consoles you logged in to previously. They are listed under “Recent Consoles.”

If your command line is cluttered from your previous work you can type the **clear** command to clear the console visual history.



.001 Log In to Python Anywhere

If you've done everything correctly, your screen should appear similar to the image above. At your command prompt, type the command **nano** to open your text editor.

```
content = "Mark,Connecticut\n"
content += "Joan,Florida\n"
content += "Jeff, Utah\n"

print("Write to drive: ")
print(content)

fo = open("mydata.txt", "w")
fo.write(content)
fo.close()
```

.002 File I/O

To facilitate processing data, most program languages have a way to facilitate loading and saving files. Python is no different. In this section, you'll learn basic file I/O (input and output) with Python.

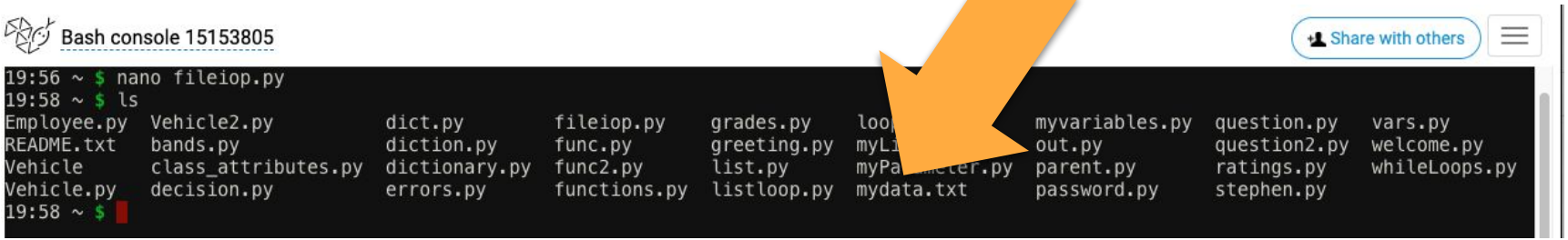
Key in the code on the left and save as **fileio.py**. Run the code and then view the files in your directory with the **ls** (list) console command.

```
content = "Mark,Connecticut\n"
content += "Joan,Florida\n"
content += "Jeff, Utah\n"

print("Write to drive: ")
print(content)

fo = open("mydata.txt", "w")
fo.write(content)
fo.close()
```

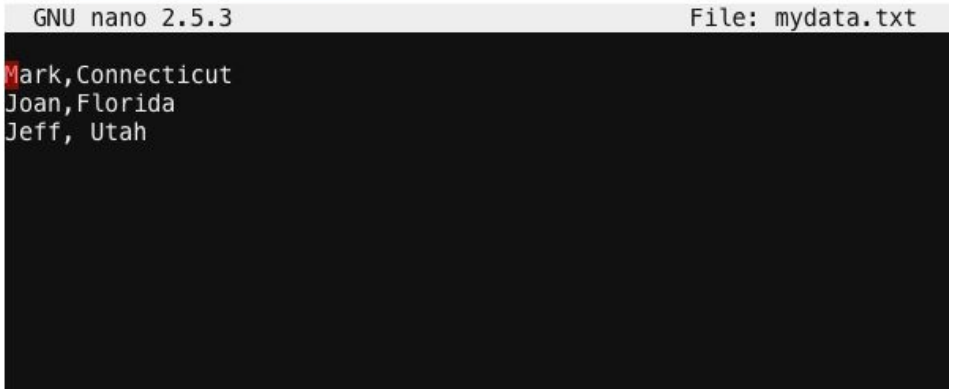
.002 File I/O

You should see the data file created by this program, **mydata.txt**, among the list of files in the directory.

Use the **nano** command and file name to load the data file in to nano to examine the results.



Bash console 15153805




```
content = "Mark,Connecticut\n"
content += "Joan,Florida\n"
content += "Jeff, Utah\n"

print("Write to drive: ")
print(content)

fo = open("mydata.txt", "w")
fo.write(content)
fo.close()
```

.002 File I/O

You'll note that the text file has been saved as expected.

Reference the code below the screenshot of the output. The character entity **\n** is used to create a new line in side the text file. Notice the character entity does not actually appear in the file, once saved. Instead it is processed to create new lines for each entry.

Bash console 15153805

```
 GNU nano 2.5.3                                File: mydata.txt

Mark,Connecticut
Joan,Florida
Jeff, Utah
```

```
content = "Mark,Connecticut\n"
content += "Joan,Florida\n"
content += "Jeff, Utah\n"

print("Write to drive: ")
print(content)

fo = open("mydata.txt", "w")
fo.write(content)
fo.close()
```

.002 File I/O

The following commands actually do the work of saving the the file:

fo = open(“mydata.txt”, “w”)
This line creates the file pointer, **fo**. It also opens a file called **mydata.txt** in **w**rite mode. There are several other modes available for file operations.

fo.write(content)
Writes the value of the variable **content** to the drive using the file pointer created.

fo.close()
Closes the file pointer. Forgetting to close your file pointer can cause a file to be corrupted.

Bash console 15153805

```
20:23 ~ $ python directory.py
Enter a full name: Mark Lassoff
Enter a phone number: 203-292-0689
Enter an email address: mark@frameworktv.com


Press enter to add a new entry or XXX to stop
Enter a full name: Joan Capuzziello
Enter a phone number: 561-555-1212
Enter an email address: joan@florida.com


Press enter to add a new entry or XXX to stop
Enter a full name: Brett Lassoff
Enter a phone number: 860-855-1111
Enter an email address: brett@gmail.com


Press enter to add a new entry or XXX to stopXXX
Saving Directory:
Mark Lassoff,203-292-0689,mark@frameworktv.com
Joan Capuzziello,561-555-1212,joan@florida.com
Brett Lassoff,860-855-1111,brett@gmail.com

20:24 ~ $
```

.003 Creating a CSV

A **CSV**, **c**omma **s**eparated **v**alues file, is a file formatted with commas and new line characters to represent individual records.

CSV file content might look like the following:

```
Mark Lassoff,203-292-0689,mark@frameworktv.com
Joan Capuzziello,561-555-1212,joan@florida.com
Brett Lassoff,860-855-1111,brett@gmail.com
```

Don’t forget that that each file is terminated with a **\n** character.

Examine the console output from executing a program that saved this csv file to the file folder.

Bash console 15153805

```
20:23 ~ $ python directory.py
Enter a full name: Mark Lassoff
Enter a phone number: 203-292-0689
Enter an email address: mark@frameworktv.com


Press enter to add a new entry or XXX to stop
Enter a full name: Joan Capuzziello
Enter a phone number: 561-555-1212
Enter an email address: joan@florida.com


Press enter to add a new entry or XXX to stop
Enter a full name: Brett Lassoff
Enter a phone number: 860-855-1111
Enter an email address: brett@gmail.com


Press enter to add a new entry or XXX to stopXXX
Saving Directory:
Mark Lassoff,203-292-0689,mark@frameworktv.com
Joan Capuzziello,561-555-1212,joan@florida.com
Brett Lassoff,860-855-1111,brett@gmail.com

20:24 ~ $
```

.003 Activity 001

Write a program that allows the user to input a number of directory entries identically to what you see on the left. When the user no longer wants to input directory entries, they should enter XXX at the end of the entry.

The program should out the directory to the screen and then save the file as **directory.csv.**

See if you can complete the activity without looking at the solution code. However, should you need it, the solution code is available at: https://social.frameworktv.com/directory-solution

```
fo = open("directory.csv", "r+")
str = fo.read()
fo.close()
print(str)
```

.004 Reading Files

Key in the code at the right which will read the file you just created from the drive.

Save and test the file and you should see whatever data you stored in **directory.csv** appear on the console screen.

This time when the file pointer is created it is created in read/write mode with **r+**. In this mode you may both read from a file and write from a file.

Save this file as **read_directory.py**.

```
fo = open("directory.csv", "r+")
str = fo.read()
fo.close()
print(str)
```

.004 Reading Files

The way this code was written the contents of the **CSV** are dumped into the variable **str** and then output to the console.

This is inadequate for most programs that use CSV data.

Fortunately Python has a csv module that will help with this.

```
import csv

with open('directory.csv') as csv file:
   csv reader = csv.reader(csv_file, delimiter=',')
   line count = 0
   for row in csv reader:
           name = row[0]
           phone = row[1]
           email = row[2]
           line count += 1
          print(name + "\t\t" + phone + "\t\t" + email)
```

.004 Reading Files

Modify **read_directory.py** so that the it appears as above. Note the **import** statement is used to access the csv module. The **with...as** statement opens the file pointer to the csv file and indicates is type as **csv_file**. The **csv.reader** object declares the file to processed and the delimiter, which with csv files should be a comma.

The **for...in** loop iterates through the file line-by-line with the line contents being pushed in to an array that includes **row[0]**, **row[1]**, and **row[2]**, representing each column of data. The **\t** character entity inserts tabs in the output.

.005 Activity 002

Download the csv file at https://social.frameworktv.com/bands-csv. (You might use the raw button on the GIST console to more easily download the raw data.

This files represents a number of bands and my rating of each band on a 10 scale with 10 being best.

Write a Python program that first loads and displays each band and rating. Then allow the user to modify the rating of any band on the list, add bands and corresponding ratings to the list, or delete and band and its corresponding rating from the list. For this type of program it is probably best to display a short command menu after the existing bands and ratings in the CSV are displayed.

Test your program and make sure all features discussed are included.