Revised: 1/5/2020

File I/O

Michael C. Hackett

Assistant Professor, Computer Science

Community College of Philadelphia

Lecture Topics

- File Input/Output
  - Writing data to text files
  - Reading data from text files
  - Appending data to text files

- Basic Text Processing
  - String Tokenization
  - Trimming Strings
  - CSV Files

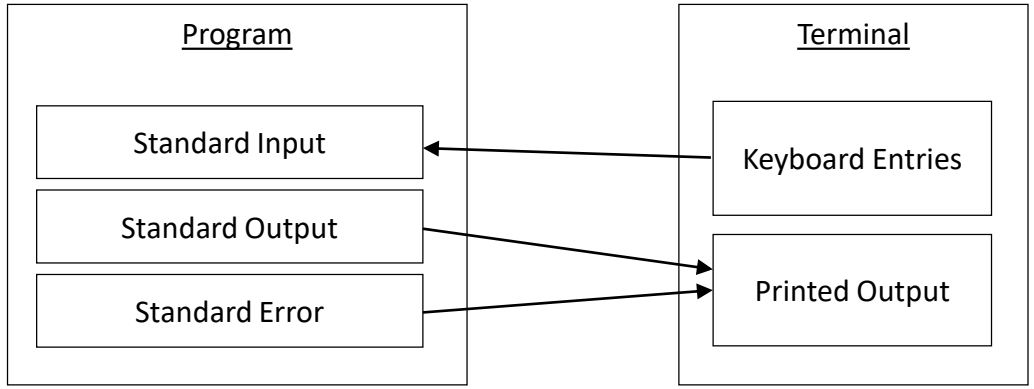
What are files?

- A ***file*** is stream of binary, digital information typically kept on a long-term storage device.
  - Word documents, Powerpoint presentations, and PDFs are all examples of different types of files.

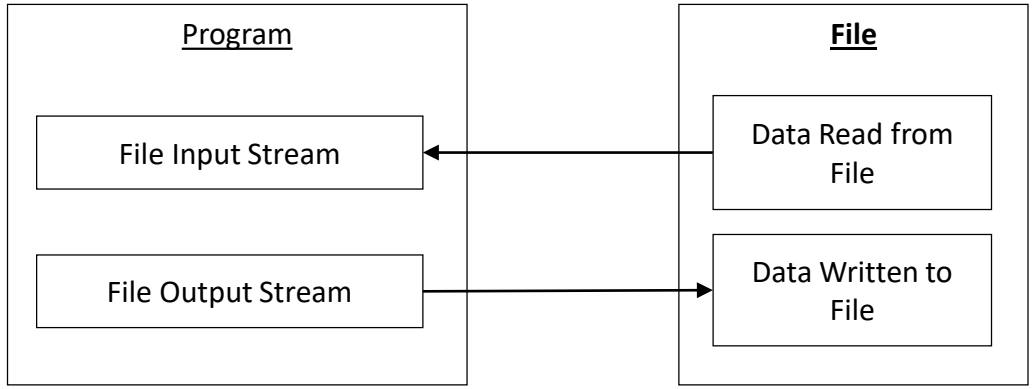
- Can be used as an input data stream. (“Reading a file”)

- Can be used as an output data stream. (“Writing to a file”)

Standard Data Streams (Shown Previously)



File Data Streams



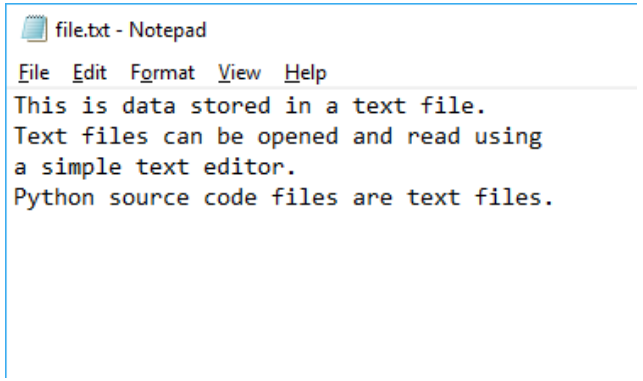
Extensions

- A file has a name which normally includes an extension.
  - Textfile**.txt**
  - WordDocument**.docx**

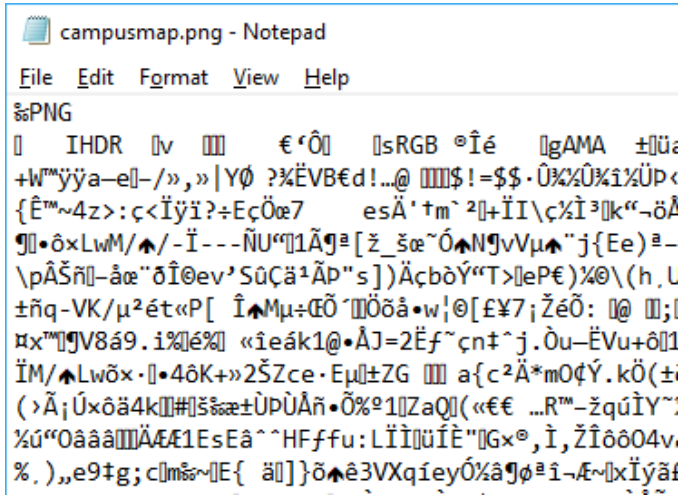
- You can have files without extensions.
  - Extensions are primarily used by the operating system, so it knows what program to use to open and read the file.
  - Some programs will only accept files with certain extensions.

Types of Files

- Text Files
  - The binary information contained in the file is encoded with ASCII plaintext.
  - Can be opened in any text editor (like Notepad.)
  - “Human readable”



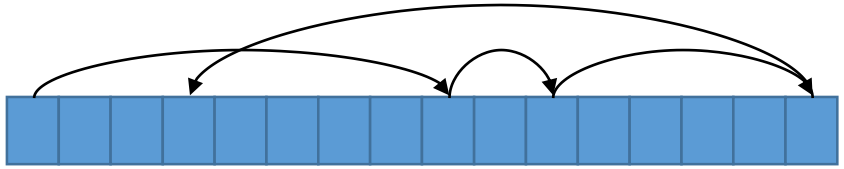
- Data Files
  - Files that are not stored in plaintext, like images and compiled programs.
  - Normally cannot be opened in any text editor.
  - Raw binary- “Computer readable”



File Access

- Using ***sequential access***, data is read/accessed from the beginning of the file through the end of the file.

- Using ***direct access***, data can be accessed from any location in the file.
  - A topic discussed in CSCI 112



Opening a File

- To open a (text file) data stream in Python, use its built-in **open** function.

- The open function accepts two arguments: The file's name and the mode in which the file is being used.
  - Both arguments are strings.

```
my_text_file = open(filename, mode)
```

- The object returned by the open function is a file object.

Specifying the File's name/path

- If the file you wish to access is in the same folder as the Python program opening the file, you only need to supply the file's name.

```
my_text_file = open("file.txt", mode)
```

- If the file is in a subfolder, you'll need to supply the path to the file beginning with the subfolder's name.

```
my_text_file = open("subfolder\\subfolder2\\file.txt", mode)
```

  - Remember, a backslash in a String literal indicates an escape sequence.

Specifying the File's name/path

- If the file is in an entirely different folder, you'll need to supply the full path to the file (beginning with the drive letter on Windows).

```
my_text_file = open("C:\\path\\to\\the\\file.txt", mode)
```

Writing Data to a Text File

- Specifying "w" as the mode will open the file in write mode.

```
my_output_file = open("output.txt", "w")
```

- In write mode, data can be written to the file.
  - If the specified file ***does not*** already exist (you want to make a new file) Python will create it.
  - If the specified file ***does*** exist, its contents will be **ERASED**.

Saving a File

- To when you are finished writing to the output stream, call the **close** function.

- This will save the file.
  - If you do not close the stream, the information you wrote will not be saved.

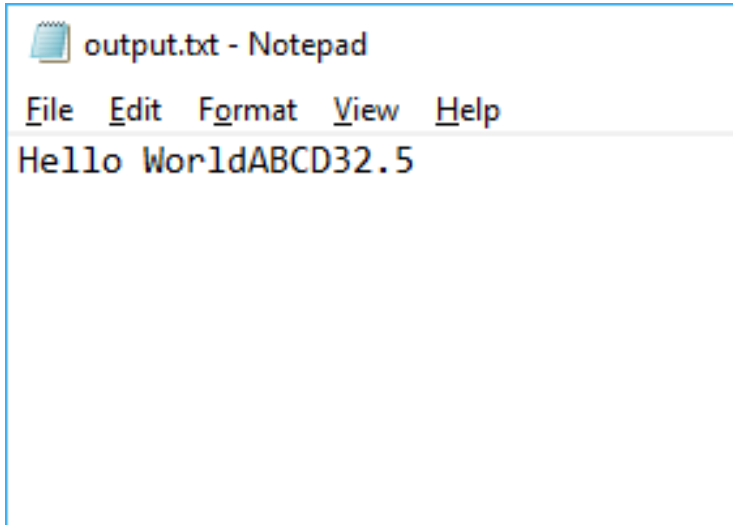
```
my_output_file.close()
```

Writing Data to a New Text File

- Once the stream is open, we can write data to the file.

- A file's **write** function will write <u>string</u> values to the file.
  - If the data is numeric (ints or floats) be sure to typecast the data to string form.

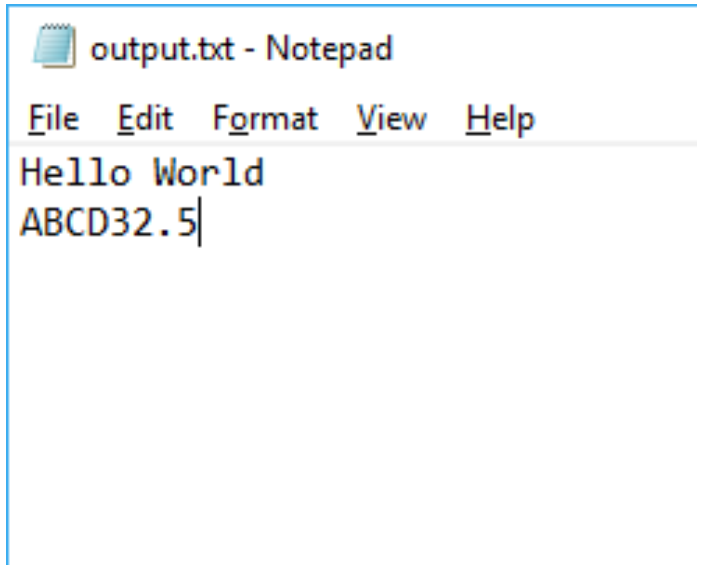
```
my_output_file = open("output.txt", "w")
my_output_file.write("Hello World")
my_output_file.write("ABCD")
my_output_file.write(str(32.5))
my_output_file.close()
```



Writing Data to a New Text File

- The write function does not add line feeds after each function call.
- To add line feeds, add (or concatenate) **\n** to the end of the line.

```
my_output_file = open("output.txt", "w")
my_output_file.write("Hello World\n")
my_output_file.write("ABCD")
my_output_file.write(str(32.5))
my_output_file.close()
```



Reading Data from a Text File

- Specifying "r" as the mode will open the stream in read-only mode.

```
my_text_file = open("file.txt", "r")
```

- No data can be written to a file opened in read-only mode.

Closing a File

- To when you are finished reading from the input stream, call the **close** function.

- Python can't have two instances of the same file open.
  - Always close your file when you are done reading from it.

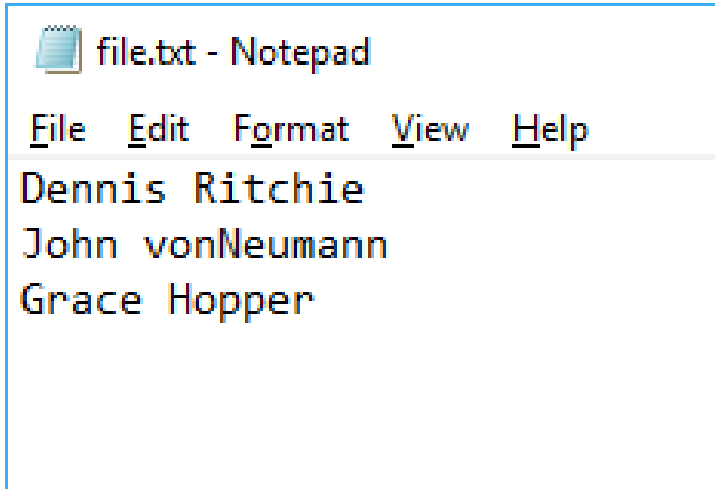
```
my_text_file.close()
```

Reading Data from a Text File

- Once the file is opened in read mode, we can read the contents of the file.
- To read a file, line-by-line, use the file's readline function.
  - The function will return a string containing the next line in the file.

```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline()
print(line1)
my_text_file.close()
```

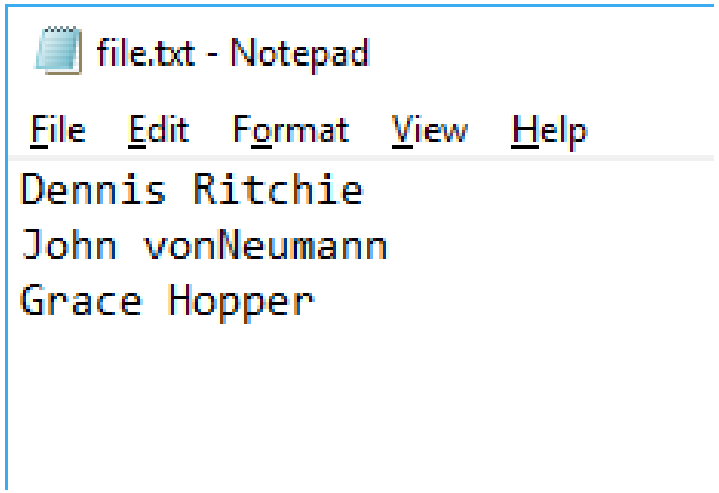
```
Dennis Ritchie
```



Reading Data from a Text File

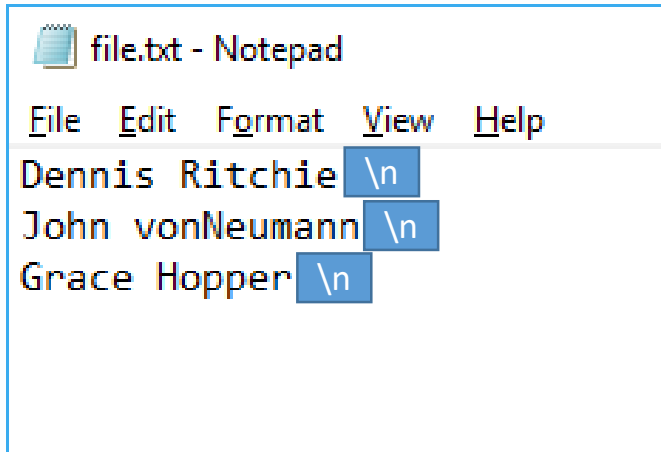
```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline()
line2 = my_text_file.readline()
line3 = my_text_file.readline()
print(line1)
print(line2)
print(line3)
my_text_file.close()
```

```
Dennis Ritchie

John vonNeumann

Grace Hopper
```



Reading Data from a Text File

- The extra lines are the result of the non-character line feed (**\n**) at the end of each line in the file.

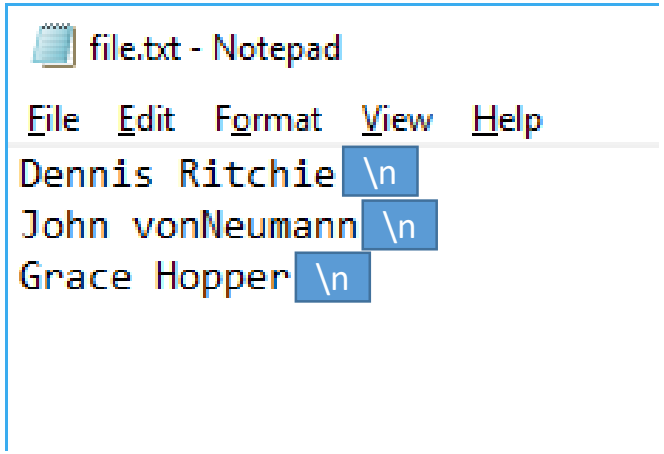



```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline()
line2 = my_text_file.readline()
line3 = my_text_file.readline()
print(line1)
print(line2)
print(line3)
my_text_file.close()
```

```
Dennis Ritchie

John vonNeumann

Grace Hopper
```

Reading Data from a Text File

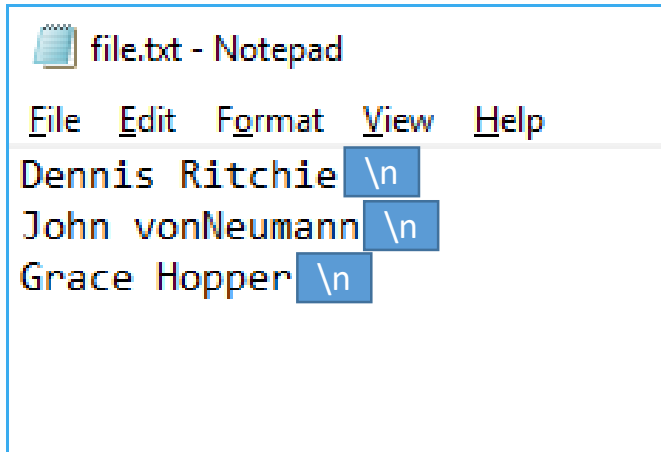
- To strip away the line feed, we can use the string's **rstrip** function.




```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline().rstrip("\n")
line2 = my_text_file.readline()
line3 = my_text_file.readline()
print(line1)
print(line2)
print(line3)
my_text_file.close()
```

```
Dennis Ritchie
John vonNeumann

Grace Hopper
```

Reading Data from a Text File




```
my_text_file = open("file.txt", "r")
line1 = my_text_file.readline().rstrip("\n")
line2 = my_text_file.readline().rstrip("\n")
line3 = my_text_file.readline().rstrip("\n")
print(line1)
print(line2)
print(line3)
my_text_file.close()
```

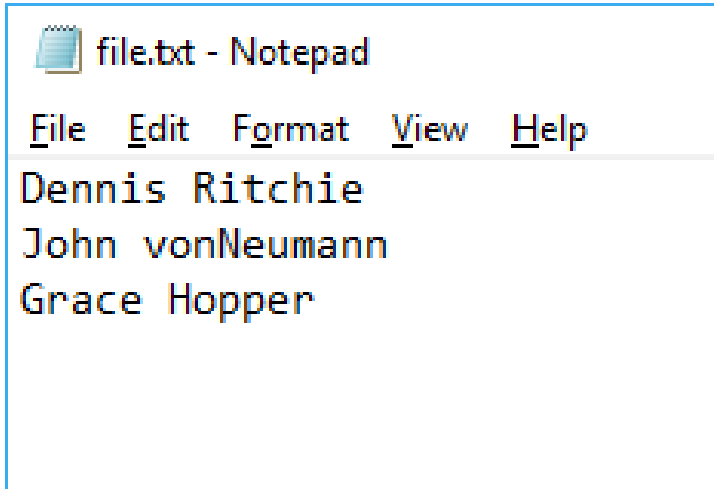
```
Dennis Ritchie
John vonNeumann
Grace Hopper
```

Reading Data from a Text File

- A for loop can be used to read through a file sequentially.

```
my_text_file = open("file.txt", "r")

for line in my_text_file :
  print(line.rstrip("\n"))

my_text_file.close()
```

```
Dennis Ritchie
John vonNeumann
Grace Hopper
```



Appending Data to a Text File

- Specifying "a" as the mode will open an output stream in append mode.

```
my_existing_file = open("output.txt", "a")
```

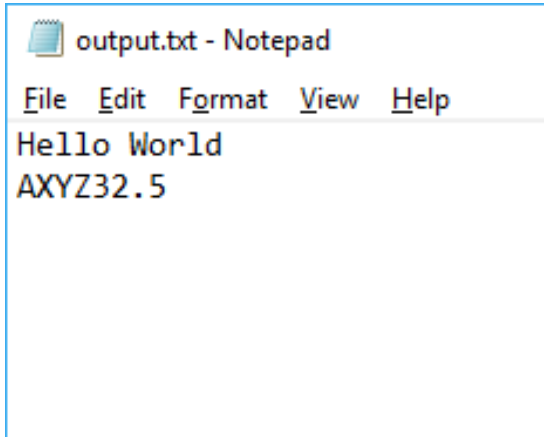
- In append mode, data can be written to a new or existing file.
  - If the file does not already exist, Python will create it.
  - If the file does exist, the file will be opened and wait for more data to be written to the end of the file.

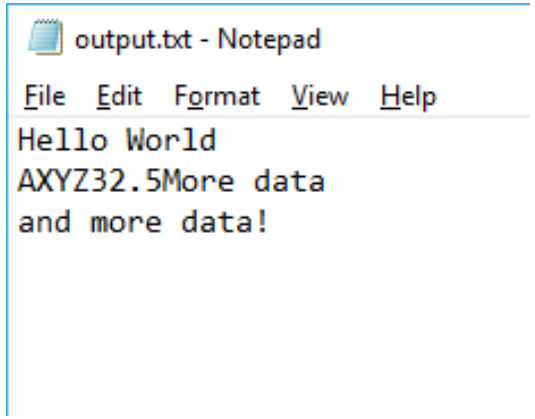
- Be sure to close the file when you are finished appending to it.

Appending Data to a Text File

- Once the file is open, we can continue writing data to the file.

```
my_output_file = open("output.txt", "a")
my_output_file.write("More data\n")
my_output_file.write("and more data!")
my_output_file.close()
```





Tokenizing Strings

- **Tokenization** is the process of splitting up a string into smaller units.

- Strings are tokenized using a ***delimiter*** (Usually spaces or commas but can be any number of characters.)
  - For example, the string “Community College of Philadelphia” could be tokenized using whitespace as the delimiter which would break it up into 4 separate strings or ***tokens***: “Community” “College” “of” and “Philadelphia”.

Tokenizing Strings

- Strings have a split function that can tokenize a string into a list of strings.

```
string_to_tokenize = "Alabama Alaska Arkansas Arizona"

tokens = string_to_tokenize.split()
```

- By default, the split function uses whitespace as the delimiter.

Tokenizing Strings

```
string_to_tokenize = "Alabama Alaska Arkansas Arizona"
tokens = string_to_tokenize.split()

print("Total tokens=", len(tokens))

for token in tokens :
  #Print Each Token
  print("Token:", token)

print("Done")
```

```
Total tokens= 4
Token: Alabama
Token: Alaska
Token: Arkansas
Token: Arizona
Done
```

Tokenizing Strings

```
string_to_tokenize = "Philadelphia, PA"
tokens = string_to_tokenize.split()

print("Total tokens=", len(tokens))

for token in tokens :
  #Print Each Token
  print("Token:", token)

print("Done")
```

```
Total tokens= 2
Token: Philadelphia,
Token: PA
Done
```

Tokenizing Strings

- To specify a custom delimiter, pass it as a string argument to the split function.

```
string_to_tokenize = "Alabama,Alaska,Arkansas,Arizona"

tokens = string_to_tokenize.split(",")
```

- The custom delimiter can be any number of characters long.

Tokenizing Strings

```
string_to_tokenize = "Alabama,Alaska,Arkansas,Arizona"
tokens = string_to_tokenize.split(",")

print("Total tokens=", len(tokens))

for token in tokens :
  #Print Each Token
  print("Token:", token)

print("Done")
```

```
Total tokens= 4
Token: Alabama
Token: Alaska
Token: Arkansas
Token: Arizona
Done
```

Tokenizing Strings

```
string_to_tokenize = "Philadelphia, PA"
tokens = string_to_tokenize.split(",")

print("Total tokens=", len(tokens))

for token in tokens :
  #Print Each Token
  print("Token:", token)

print("Done")
```

```
Total tokens= 2
Token: Philadelphia
Token:  PA
Done
```

Trimming Strings

- Occasionally, strings may have extra whitespace at the start or end of its character sequence.

```
string_to_tokenize = "   Alabama,Alaska,Arkansas,Arizona   "
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token: ", token, ".", sep="")

print("Done")
```

```
Token:    Alabama.
Token: Alaska.
Token: Arkansas.
Token: Arizona   .
Done
```

Trimming Strings

- The string's lstrip (left strip) function removes leading whitespace.

```
string_to_tokenize = "   Alabama,Alaska,Arkansas,Arizona   "
string_to_tokenize = string_to_tokenize.lstrip()
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token: ", token, ".", sep="")

print("Done")
```

```
Token: Alabama.
Token: Alaska.
Token: Arkansas.
Token: Arizona   .
Done
```

Trimming Strings

- The string's rstrip (right strip) function removes trailing whitespace.

```
string_to_tokenize = "   Alabama,Alaska,Arkansas,Arizona   "
string_to_tokenize = string_to_tokenize.lstrip()
string_to_tokenize = string_to_tokenize.rstrip()
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token: ", token, ".", sep="")

print("Done")
```

```
Token: Alabama.
Token: Alaska.
Token: Arkansas.
Token: Arizona.
Done
```

Trimming Strings

```
string_to_tokenize = "   Alabama,Alaska,Arkansas,Arizona   "
string_to_tokenize = string_to_tokenize.lstrip().rstrip()
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token: ", token, ".", sep="")

print("Done")
```

```
Token:  Alabama.
Token:  Alaska.
Token:  Arkansas.
Token:  Arizona.
Done
```

Trimming Strings

- Sometimes, extra whitespace may be captured as part of a token.
- Trimming the token removes that extra leading/trailing whitespace.

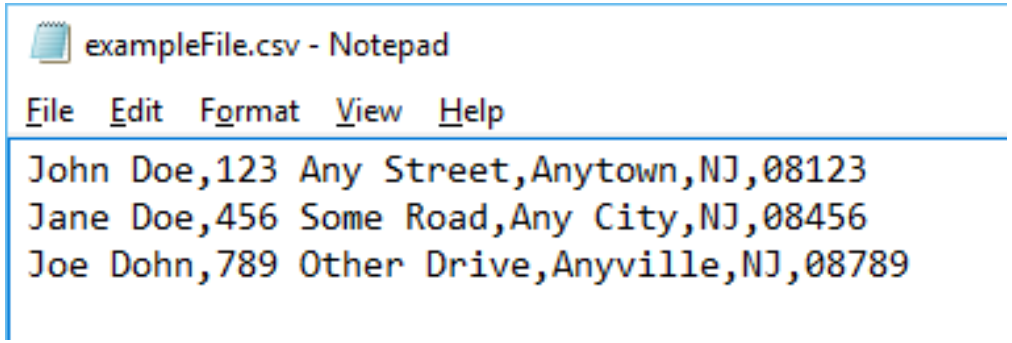
```
string_to_tokenize = "Philadelphia, PA"
tokens = string_to_tokenize.split(",")

for token in tokens :
  #Print Each Token
  print("Token:", token.lstrip().rstrip(), ".", sep="")

print("Done")
```

```
Total tokens= 2
Token: Philadelphia
Token: PA
Done
```

CSV files

- ***Comma separated values*** (or CSV) is a widely recognized text file format where each line of the file contains values that are separated by commas.

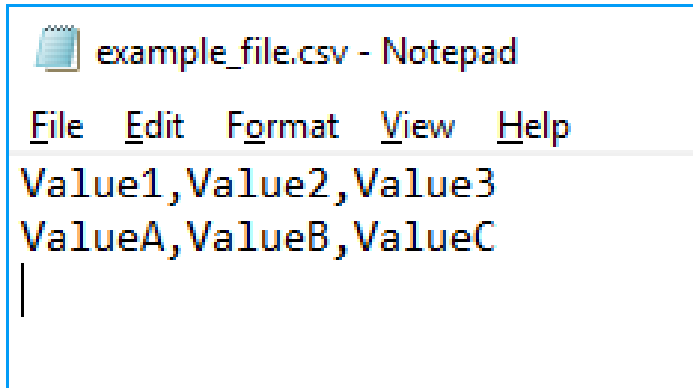
- Many database and spreadsheet programs use CSV format to export and import data.
  - Filename ends with .csv



Writing CSV files

- There is no special object for writing a CSV file.
  - Write the comma separated values as you would normally write to a file.

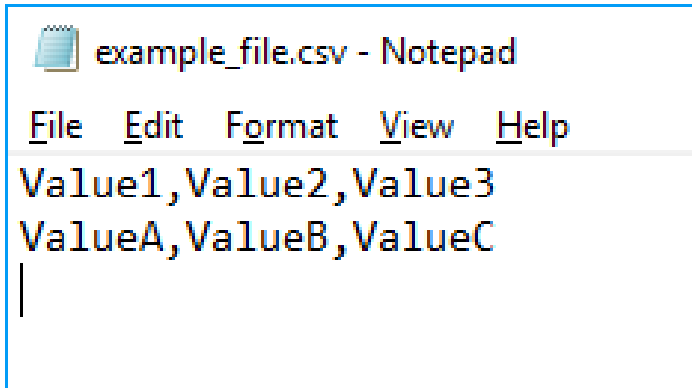
```
my_csv_file = open("example_file.csv", "w")
v1 = "ValueA"
v2 = "ValueB"
v3 = "ValueC"

my_csv_file.write("Value1,Value2,Value3\n")
my_csv_file.write(v1 + "," + v2 + "," + v1 + "\n")

my_csv_file.close()
```



Reading CSV files

- There is no special object for reading a CSV file.
  - Read the file as you would read any text file.
  - For each line in the file, split the line using a comma as the delimiter.

```
my_csv_file = open("example_file.csv", "r")

for line in my_csv_file :
  tokens = line.rstrip("\n").split(",")
  #Use the tokens list to access
  #the individual values of that line

my_csv_file.close()
```



Reading CSV files

```
my_csv_file = open("example_file.csv", "r")

for line in my_csv_file :
  tokens = line.rstrip("\n").split(",")
  print(tokens[0])
  print(tokens[1])
  print(tokens[2])

my_csv_file.close()
```

example_file.csv - Notepad

File Edit Format View Help

```
Value1,Value2,Value3
ValueA,ValueB,ValueC
```

```
Value1
Value2
Value3
ValueA
ValueB
Valuec
```