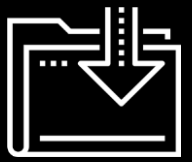
Reading and Writing Files

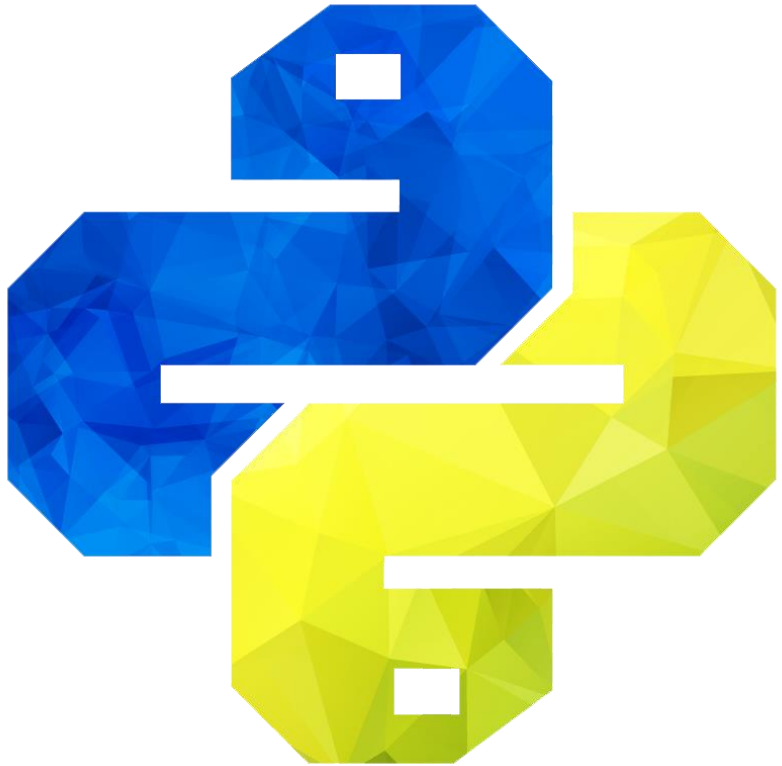
Some succeed because they are destined to;
most succeed because they are determined to.
-Unknown

Cybersecurity Boot Camp
Advanced Python: Day 1



Python: Reading and Writing Files

Today we begin using Python on a more advanced scale to write and read data to and from files.



Class Objectives

By the end of class today, you will be able to:

- ☐ Open and read text files using `open()` and `file.read()` method.
- ☐ Use the `string.split()` function to break a string into smaller strings
- ☐ Use `string.find()` function to search for specific text
- ☐ Create a command line application that searches for words within a text file
- ☐ Read and search through CSV files for specific information
- ☐ Write text to external files
- ☐ Append text to external files without overwriting existing text

Reading Text Files

Reading Text Files

So far, we've used the `input()` function to get external data into our Python application.

This function is useful for collecting small amounts of data, but it can get difficult and unwieldy using it for scripts and programs where large collections of data would be entered on the command line.

Reading Text Files

So far, we've used the `input()` function to get external data into our Python application.

This function is useful for collecting small amounts of data, but it can get difficult and unwieldy using it for scripts and programs where large collections of data would be entered on the command line.

To solve this problem, we can have scripts that were made to read through pre-existing external files in order to collect the desired information. This saves us lots of time and makes our applications more robust.

Reading Text Files

open ( ) function

```
# The open() function creates a connection to an external file
# The parameter passed into the function is the relative or
absolute path to file to open
diary_txt_file = open("Diary.txt","r")

# Using the .read() function then stringifies the file's
contents
diaryText = diary_txt_file.read()
print(diaryText)

# Closing the connection to the external file in order to save
memory
diary_txt_file.close()
```

In order to read Diary.txt file with our script file ReadingTextFiles.py, we will need to create a connection between the script file and the external file.

This connection is created by using the built-in open ( ) function in our script file.

The relative or absolute path to the file is provided as a parameter

Reading Text Files

open ( ) function

```
# The open() function creates a connection to an external file
# The parameter passed into the function is the relative or
absolute path to file to open
diary_txt_file = open("Diary.txt", "r")

# Using the .read() function then stringifies the file's
contents
diaryText = diary_txt_file.read()
print(diaryText)

# Closing the connection to the external file in order to save
memory
diary_txt_file.close()
```

The “r” indicates we will be reading the file.

“r” is always the default second parameter is none other is entered.

Reading Text Files

file.read ( ) function

```
# The open() function creates a connection to an external file
# The parameter passed into the function is the relative or
absolute path to file to open
diary_txt_file = open("Diary.txt","r")

# Using the .read() function then stringifies the file's
contents
diaryText = diary_txt_file.read()
print(diaryText)

# Closing the connection to the external file in order to save
memory
diary_txt_file.close()
```

Next, we use the file.read ( ) method to read through the text file and return a string version of its contents.

Reading Text Files

file.read ( ) function

```
# The open() function creates a connection to an external file
# The parameter passed into the function is the relative or
absolute path to file to open
diary_txt_file = open("Diary.txt","r")

# Using the .read() function then stringifies the file's
contents
diaryText = diary_txt_file.read()
print(diaryText)

# Closing the connection to the external file in order to save
memory
diary_txt_file.close()
```

The connection to a file can be closed at any time by using the file.close ( ) method.

While not always *necessary*, it is good practice to close a file and end a connection whenever possible (and your computer's memory will thank you).

Activity: Reading Rainbow

In this activity, you will create a command line application that asks the user for a color.

If this color has an associated file then the script will open up the file and print its content to the terminal.

Activities/ 02-Stu_ReadingRainbow

Suggested Time:
10 minutes

Sample: Reading Rainbow

Instructions:

1. Extract the contents of `Starter.zip` to some location on your computer and do your work within the `ReadingRainbow.py` file that was included.

2. Ask the user for a color and save their response to a variable.

3. Check to make sure that the color the user has entered has an associated file.

- If there is a matching file, then create a connection to this file, read through its contents, and print them out to the terminal before closing the connection.
- If there is no matching file, then simply print "No file associated with that color"

Hints:

- Look through the Colors subfolder before creating your conditional so as to figure out what values you should be checking for.
- Create one connection to one specific file/folder path before diving in and attempting to create a connection string that is different depending upon the value the user entered. i.e. your connection should be to the "Colors" subfolder and not multiple connections to the individual files.

Times Up! Let’s Review.

Reading Rainbow

String Functions

String Functions

`String.split( )` is used to work with large blocks of text.

```
# Since the file is now a string, it can be modified and worked
with using some string functions
# The split() function breaks a string apart into a list based
upon common words/characters that appear in the original string
diarySplit = diaryText.split(" ")


# Since the string was split on spaces, individual words will now
be printed when referenced
print(diarySplit[0])
print(diarySplit[1])
print(diarySplit[2])
print(diarySplit[3])
print("-----------")
```

The `string.split( )` function takes the original string provided and breaks it down into smaller chunks based upon whatever string value is passed into it as a parameter.

Then these chunks are placed into a list so they are more easily navigated through.

String Functions

find ( ) determines if the parameter occurs anywhere within the text

```
# The find() function will navigate through some text, determine whether or not the string passed into it is contained within, and return the index of that string
print(diaryText.find("malarkey"))

# This can be exceptionally useful when checking to see if a file contains some specific keywords
if diaryText.find("malarkey") > -1:
    print("Malarkey found!")

if diaryText.find("juice") > -1:
    print("Juice found")

print("-----------")
```

If the substring is found within the original text, then it will return the index of the first occurence of that substring.

The value of the index returned will be the character location where the substring began.

If the substring is not found within the original string, then a value of negative one (-1) is returned instead.

Activity: Word Search

In this activity, you will create a command line application that looks into a specific file and then asks the user to enter a word.

The application will then search through the file, find any instance of the word and then print out how many times the word was found.

Activities/04-Stu_WordSearch

Suggested Time:
10 minutes

Your Turn: Reading Rainbow

Instructions:

You are given a text file, and in your WordSearch.py file, create a function called `wordSearch()` which will take in a string as a parameter and will carry out the following tasks.

- Opens up and reads the text contained within "Monologue.txt"
- Searches through the text in order to uncover whether the string passed into `wordSearch()` can be found.
- Prints out how many times the string passed can be found within the original text.

Hints:

- Whenever `string.split()` is used to discover how many instances of a string appear within a block of text, the length of the list returned by the method is always one greater than it should be.
- Remember that you have to call a function in order to use it. Defining a function is never enough.
- The script file where you will write your solution has comments that guide you step by step through the code you will need to write.

Reading CSV Files

Reading CSV Files

CSV files offer a easier, more organized method for dealing with plain text files.

```
# Connect to the file and read in the text it contains
wrestling_csv = open("WWE-Data-2016.csv", "r")
wrestling_text = wrestling_csv.read()


# Since a CSV is broken into rows and columns, it will need to be split
twice
# The first split is to break the original text into rows by splitting on
each new line
wrestling_rows = wrestling_text.split("\n")
print(wrestling_rows[0])
print(wrestling_rows[1])
print(wrestling_rows[2])
print(wrestling_rows[3])
print("-----------")
```

A CSV is basically a text-formatted table, meaning that each piece of data is stored within a specific column and row.

CSV files can be opened in the exact same way as a text file.

Simply pass the path to the CSV into the open ( ) function and save the file object returned to a variable.

Reading CSV Files

Splitting CSV files:

```
# The first split is to break the original text into rows by splitting on
each new line
wrestling_rows = wrestling_text.split("\n")
print(wrestling_rows[0])
print(wrestling_rows[1])
print(wrestling_rows[2])
print(wrestling_rows[3])
print("-----------")

# The next split will then split the row into its respective columns on
commas
wrestling_cells = wrestling_rows[0].split(",")
print(wrestling_cells[0])
print(wrestling_cells[1])
print(wrestling_cells[2])
print(wrestling_cells[3])
```

Since a CSV is broken into rows and columns, it will need to be split twice

Use `string.split( )` to break the data down into more manageable chunks.

Activity: The User List

In this activity, you will be given a CSV file with usernames, passwords, and a list of IP addresses that users are regularly connected to.

You will read in the CSV, break into parts, and then create a Python dictionary version of the CSV.

Activities/06-Stu_TheUserList

Suggested Time:
10 minutes

Your Turn: The User List

Instructions:

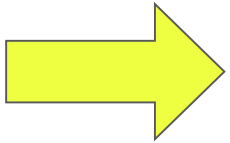
In this activity, you have been given a CSV file with usernames, passwords, and a list of IP addresses that these users are regularly connecting to.

Your job is to identify the users who are connecting to your company's private server and print out their information to the terminal.

You will code your solution in the ReadUserList.py file.

The company's private server is `229.62.232.190`

The user's information should be printed out in this format:



```
COMPANY PRIVATE SERVER FOUND
USER: Ryan
PASSWORD: QLeyzpdW
HOURS ONLINE: 292
COMMON IPs:
248.150.250.24
223.229.166.82
229.62.232.190
198.1.152.204
215.252.4.108
117.59.161.190
169.59.2.142
144.91.68.253
27.127.60.188
182.151.186.176
----------
COMPANY PRIVATE SERVER FOUND
USER: Clark
PASSWORD: uUd2p3VH
HOURS ONLINE: 282
COMMON IPs:
248.150.250.24
223.229.166.82
229.62.232.190
198.1.152.204
215.252.4.108
117.59.161.190
169.59.2.142
144.91.68.253
27.127.60.188
182.151.186.176
----------
```

Times Up! Let’s Review.

The User List

Take a Break!

Writing Files

Writing Files

Like reading files, open ( ) is used to write to files

```
# The open() function is also used for writing, though it
defaults to "r"ead
# so we have to use the "w" mode as the second argument
("w"rite, instead of "r"ead)
diary_file = open("MyPersonalDiary.txt", "w")

# The .write() function is then used to push the text into the
external file
diary_file.write("I don't write in diaries.")

# Since no spacing or newlines are added between .write()
functions, they have to be programmed into the application
manually
diary_file.write("\nPeriod.")
```

Writing files with Python is helpful in cybersecurity for when you have to write reports based on information uncovered during the investigation.

Note the inclusion of “w” parameter , standing for “write”.

“w” is write only, it can’t read files.

Writing Files

Like reading files, open ( ) is used to write to files

```
# The open() function is also used for writing, though it
defaults to "r"ead
# so we have to use the "w" mode as the second argument
("w"rite, instead of "r"ead)
diary_file = open("MyPersonalDiary.txt", "w")

# The .write() function is then used to push the text into the
external file
diary_file.write("I don't write in diaries.")

# Since no spacing or newlines are added between .write()
functions, they have to be programmed into the application
manually
diary_file.write("\nPeriod.")
```

Important: This method of writing files is somewhat dangerous: the “w” mode for open( ) will cause any new lines to overwrite the original text inside the file

Writing Files

Use `file.write ( )` to write text into a file.

```
# The open() function is also used for writing, though it
defaults to "r"ead
# so we have to use the "w" mode as the second argument
("w"rite, instead of "r"ead)
diary_file = open("MyPersonalDiary.txt", "w")

# The .write() function is then used to push the text into the
external file
diary_file.write("I don't write in diaries.")

# Since no spacing or newlines are added between .write()
functions, they have to be programmed into the application
manually
diary_file.write("\nPeriod.")
```

Whatever string is written in the parameter will be written into the external file.

It is important to be very specific with syntax.

If a newline is added, the `\n` character must be used.

There is no spacing added by default between multiple `file.write` commands.

Activity: Terrible Word Application

In this activity, you will create a command line application that allows you to write lines of text into an external file.

Activities/08-Stu_TerribleWordApp

Suggested Time:
10 minutes

Your Turn: Terrible Word Application

Instructions:

You will need to create a text file called "Notes.txt." Then use the .py file provided to code your solution.

1. Establish a connection to the external text file called "Notes.txt" and ensure that the mode of the connection is set to write.
2. Check if the user would like to add a new line of text into the external file that they connected to. You will need to use a `while` loop to accomplish this.
3. Allow the user to write some text to the terminal using the `input()` function before writing this text into the external file the application is connected to.

Hints:

- Use the script file comments to guide you step by step through the code you will need to write.

Times Up! Let’s Review.

Terrible Word Application

The Append Mode

The Append Mode

Used to add new information onto the end of the file.

```
# The "a" mode stands for append and allows the application to
add new text onto the end of an existing file
notesFile = open("Notes.txt", "a")


# The .write() method in conjunction with the append mode will
write to the end of a file
notesFile.write("\nThis is a completely new line of text
created by the APPEND mode.")


# Closing the file
notesFile.close()
```

The append mode is activated whenever the parameter “a” is passed into the open ( ) function.

The append mode is basically a less destructive version of the write mode.

Like the write function, spacing is not added between the preexisting content and the new content that is being added.

Through a combination of reading, writing, and appending, Python programmers can create complex and efficient applications that automatically take in data from one source and create reports in another.

Activity: The Watchlist

In this activity, students will be creating a command line application that allows them to read through and update a company's watchlist of "dangerous" individuals from the terminal.

Activities/10-Stu_TheWatchlist

Suggested Time:
20 minutes

Your Turn: The Watchlist

Instructions:

You will create a command line application that allows its users to read through and add to a private company's watchlist of "dangerous" individuals.

The CSV file you have been given has four column categories: Name, Birth Year, Reason to Watch, Threat Level. You can see that there are two entries in there so far.

Your application should accomplish all of the following.

- Have a function that asks the user whether they would like to read the watchlist, write to the watchlist, or quit out of the application altogether.
- Have another function that allows users to write new individuals into the watchlist. Each entry should have a name, a birth year, a reason for being on the list, and a threat level.
- Have a third function that reads in each row from the watchlist and prints the data out one at a time to the terminal.
- Loops until the user decides to quit out of the application.

Hints

- This is not an easy activity, so feel free to work with some of the people around you to find a solution.

Times Up! Let’s Review.

The
Watchlist

Class Objectives

By the end of class today, you will be able to:

- ✅ Open and read text files using `open()` and `file.read()` method.
- ✅ Use the `string.split()` function to break a string into smaller strings
- ✅ Use `string.find()` function to search for specific text
- ✅ Create a command line application that searches for words within a text file
- ✅ Read and search through CSV files for specific information
- ✅ Write text to external files
- ✅ Append text to external files without overwriting existing text