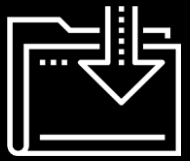
Modules in Python

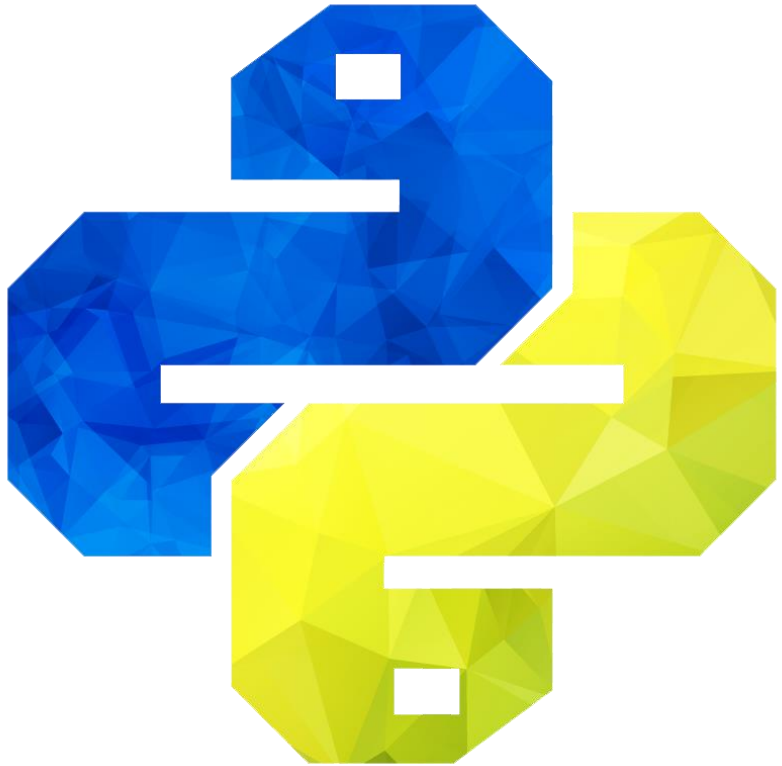
There is no such thing as work-life balance.
Everything worth fighting for unbalances your life.
-Alain de Botton

Cybersecurity Boot Camp
Advanced Python: Day 2



Python: Reading and Writing Files

Today we will cover Python's built-in modules.



Class Objectives

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

- ☐ Import the `string`, `random`, and `hashlib` modules to access the functions within them.
- ☐ Use the `csv` module to read and write data to external CSV files.
- ☐ Use the `secrets` module to generate random passwords.
- ☐ Use the `os` module to create file paths and check for specific files.
- ☐ Use the `os` module to navigate through a folder system and automate tasks.
- ☐ Use the `os` module to perform a simple attack on an operating system.

In previous classes, we covered:

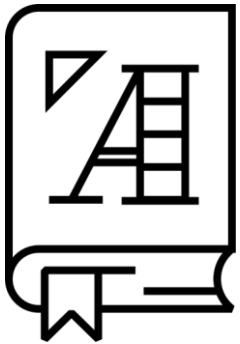
Data	Logic
1. Numbers	1. Operators
2. Strings	2. Conditionals
3. Booleans	3. Loops
4. Lists	4. Functions
5. Dictionaries	5. Modules

Today's Class:

Data	Logic
1. Numbers	1. Operators
2. Strings	2. Conditionals
3. Booleans	3. Loops
4. Lists	4. Functions
5. Dictionaries	5. Modules

Introduction to Built-In Modules

Built-In Modules



Modules are files containing a set of definitions for functions, classes, and / or variables which can be imported and used into another program and applications.

Built-In Modules

Python has many built-in modules for executing different tasks, like:

- `csv` modules allow users to easily work with external CSV files
- `os` modules allow users to work with the operating system
- `datetime` module contains functions for manipulating dates and times

In order for a program or file to use the code contained within a module, the module file has to be added into the script using a process called **importing**.

Once the module has been imported, the functions and other attributes in the module can be referenced in the program to execute task

Built-In Modules

Getting started...

```
# In order to pull a module into script, it has to be imported
import math

# When working with the values/functions stored within a
module, generally you will have to name the module and then
use dot notation to access the function

# The `math.pow()` function takes in two values and returns
the value of the first raised to the power of the second
squaredNumber = math.pow(4,2)
print(squaredNumber)
```

First, use the `import` statement followed by the name of the module.

Once a module has been imported, the functions, methods, and values that it contains can be used anywhere in the program.

To access code stored within a module, you will have to name the modules and then use dot notation in the order to access the desired value or function.

Activity: Module Hunter

In this activity, you will be given a short worksheet that will ask them to import modules and then perform tasks using functions in that module.

Activities / 02-Stu_ModuleHunter

Suggested Time:
minutes

Your Turn: Module Hunter

Instructions:

- Using The Python Standard library as a reference and the provided file as your guide, import all of modules requested and then complete the tasks listed within the comments.

- To get you started, we've provided you with the first module that you will be importing.

`Hint`: If you get stuck on a problem, remember that there are other sources of information online that may help to clear up the confusion.

Module Hunter Review

Let’s Review:

The `string module` is used when the user wants to collect some common string constants or format a block of text.

The `random module` is used to add pseudo-random elements into a program. You will often want to use random numbers and values in your program.

the `random.shuffle()` function takes in a list, but instead of returning a new list that is shuffled, it actually modifies the original.

The `hashlib module` is an important module for cybersecurity and allows us to more easily work with specific hash algorithms.

The CSV Module

The CSV Modules *(Activities / 03-Ins_CSVModule/CSVModule.py)*

The CSV modules reads and writes tabular data

```
# Module for reading CSVs
import csv

csvFile = open("WWE-Data.csv")

# The `csv.reader()` method is used to read in the data within the file
contents = csv.reader(csvFile)
# Contents returned as a CSV object
print(contents)

# The CSV object can be looped through without any splitting
for row in contents:
    print(row)

    # rows are already created as lists of cells for us, so no need to
split
    print(row[0])
```

First, import the csv module.

Use csv.reader(file) to split contents of the external file into rows whenever there is a newline character and cells whenever there is a comma.

Therefore, there is no need to manually split the file.

Activity: User List Revisit

In this activity, you will be diving back into the code for the user list application from the last class and modify it to use the CSV modules instead of manually splitting up the data.

Activities / 04-Stu_UserListRevisit

Suggested Time:
10 minutes

Your Turn: User List Revist

Instructions

- The provided script contains the original solution from the last class. You must modify it so that it uses the CSV module to read through and parse the external data instead of requiring manual splitting.

- Modify the script so that, instead of printing out information to the screen, it instead pushes this information into a new text file called `PeopleToKeepEyesOn.txt`.

`Hint:` One manual split will be necessary in the completed application since the list of IP addresses is still contained within a single cell.

Secret Modules

Secret Modules

Secrets modules generates pseudo-random values

```
# Import the secrets module into our application
import secrets


# The `secrets.randbelow()` function takes in a single value and
will return a random integer between it and the number 0.
randInteger = secrets.randbelow(100)
print(randInteger)


# The `secrets.choice()` function takes in a list and selects a
random element from within it
myList = [1,2,3,4,5,6,7,8,9,10]
randElement = secrets.choice(myList)
print(randElement)
```

The secrets module generate cryptographically strong pseudo-random values that would be suitable for managing data, including passwords, account authentication, and security tokens.

Secret Modules

Secrets modules generates pseudo-random values

```
# Import the secrets module into our application
import secrets


# The `secrets.randbelow()` function takes in a single value and
will return a random integer between it and the number 0.
randInteger = secrets.randbelow(100)
print(randInteger)


# The `secrets.choice()` function takes in a list and selects a
random element from within it
myList = [1,2,3,4,5,6,7,8,9,10]
randElement = secrets.choice(myList)
print(randElement)
```

Use `secret.randbelow()` to collect a random integer between zero and a given number, with that number as the parameter.

Use the `secrets.choice()` method to collect a random element from a list

There is no specific function that allows users to select a number in a given range, but we can use `secrets.choice()` and pass a `range()` into the method instead of a static list.

Random Password List (06-Stu_RandomPasswordList/RandomPasswordList.py)

Instructions

Import both the `secrets` and `string` libraries into the application.

Create a string that will hold all of the ASCII characters and digits inside of it.

Create a connection to an external text file called `PasswordList.txt`.

Create a loop that will run for 100 iterations and will push a new, 10 character password of random letters/numbers into the external file.

Take a Break!

Introduction to the OS Module

The OS module allows users to create scripts that are capable of looking into and working with the file or folder system of a computer regardless of its OS.

The OS Module

For example...

- Mac and Windows operating systems use different slashes (\ vs. /) to join paths.
- This is an issue if you are creating code that is supposed to work across multiple systems.
- If a mistake is made or something unexpected occurs, the code breaks without question.

The OS Module can provide a solution to this problem

The OS Module

An `os.path.join ()` function checks the computer's OS and inserts the appropriate slash.

```
# import the os library to use later
import os


# The os.path.join() function creates a file path which will
work for the current file system (so this code will work for
any filesystem)
real_path = os.path.join("Resources", "CoolText.txt")


# This path can then be used for open()
cool_text = open(real_path)
print(cool_text.read())
print("---------------")
```

As a solution, `os.path.join ()` function checks the computer's OS and inserts the for appropriate forward slash or backslash.

In the `os.path.join()` function, the parameters that you pass in to the function will be the folder and file names that the function will then modify to create the file path.

The OS Module

We can also use the OS modules to check whether a specified file exists or not.

```
# The os.path.isfile() function returns True if the file path provided points to an
actual file, false otherwise
fake_path = os.path.join("Resources", "NotAFile.txt")
print(os.path.isfile(real_path))
print(os.path.isfile(fake_path))
print("--------------")


# you can use isfile before you do a read to avoid errors if you're not sure the
file will be there:
fileName = input("Please enter the file you're looking for: ")


filePath = os.path.join("Resources", fileName)

if os.path.isfile(filePath):
    print("Found it!")
    # read the file here
else:
    print("The file doesn't exist")
```

If the file path exists, the value True will be returned.

And if the file does not exist, the value False will be returned.

Activity: Terminal Library

In this activity, you will be given a **Books** folder containing text files. You will need to create a program that prompts the user for a file name and then searches the **Books** folder for it.

Activities / 08-Stu_TerminalLibrary

Suggested Time:
15 minutes

Your Turn: Terminal Library

Instructions:

Unzip the Books folder, if necessary.

Create an application that will ask the user for the name of a text file and then checks to see if the file exists in the **Books** folder.

- If the text file exists, then print the text inside of the file to the terminal.
- If the text file does not exist, then print "Sorry! That book is not in our records! Please try again!"

Hint: When the user enters the file name, keep in mind that they will not use a file extension (.txt). Therefore, when you're creating your file path, remember that you'll need to have the file extension added to that user input.

Walking Through Folders and Files

OS Walk *(09-Ins_OSWalk/OSWalk.py)*

An expedited process for navigating through an entire folder to perform tasks

```
# import the os library to use later
import os

folder_path = os.path.join("Resources", "DiaryEntries")

# The os.walk() function is used to navigate through a collection of folders/files
# This function returns three values for each step it takes: root, dirs, and files
for root, dirs, files in os.walk(folder_path):

    # The root is the folder that is currently being searched through
    print("Currently inside of... " + root)

    # The dirs list stores all of the names of the folders inside the current root
    print("The folders in here are..." + str(dirs))

    # The files list stores all of the names of the files inside the current root
    print("The files in here are..." + str(files))
    print("~~~~~~~~~~")
```

`os.walk()` method takes a file path in as a parameter.

We can see that we're passing the folder path we created using the `os.path.join` function.

The os.walk() method **always** needs to be used alongside a `for loop`.

OS Walk

Three main fields that os.walk () method can loop through:

```
# import the os library to use later
import os

folder_path = os.path.join("Resources", "DiaryEntries")

# The os.walk() function is used to navigate through a collection of folders/files
# This function returns three values for each step it takes: root, dirs, and files
for root, dirs, files in os.walk(folder_path):

    # The root is the folder that is currently being searched through
    print("Currently inside of... " + root)

    # The dirs list stores all of the names of the folders inside the current root
    print("The folders in here are..." + str(dirs))

    # The files list stores all of the names of the files inside the current root
    print("The files in here are..." + str(files))
    print("~~~~~~~~~~")
```

The first field returned is the root.

The root variable lets the application know the current directory that `os.walk()` is moving through and the value stored within this variable will change to reflect the current position of the application.

OS Walk

Three main fields that os.walk () method can loop through:

```
# import the os library to use later
import os

folder_path = os.path.join("Resources", "DiaryEntries")

# The os.walk() function is used to navigate through a collection of folders/files
# This function returns three values for each step it takes: root, dirs, and files
for root, dirs, files in os.walk(folder_path):

    # The root is the folder that is currently being searched through
    print("Currently inside of... " + root)

    # The dirs list stores all of the names of the folders inside the current root
    print("The folders in here are..." + str(dirs))

    # The files list stores all of the names of the files inside the current root
    print("The files in here are..." + str(files))
    print("~~~~~~~~~~")
```

The second field returned are the dirs.

The dirs variable is a list of all the folders that are located within the current root.

These are the folders that `os.walk()` will navigate into eventually.

OS Walk

Three main fields that os.walk () method can loop through:

```
# import the os library to use later
import os

folder_path = os.path.join("Resources", "DiaryEntries")

# The os.walk() function is used to navigate through a collection of folders/files
# This function returns three values for each step it takes: root, dirs, and files
for root, dirs, files in os.walk(folder_path):

    # The root is the folder that is currently being searched through
    print("Currently inside of... " + root)

    # The dirs list stores all of the names of the folders inside the current root
    print("The folders in here are..." + str(dirs))

    # The files list stores all of the names of the files inside the current root
    print("The files in here are..." + str(files))
    print("~~~~~~~~~~")
```

The third field returned are the files.

The files variable is a list of all the files that are located within the current root.

The user can create the path to these folders by looping through the list of files and then using `os.path.join()` to combine the value stored within root variable with the current file they want to view.

Activity: Get Wrekt!

In this activity, you will be taking on the role of a hacker tasked with overwriting the hard work of their victims with the phrase “GET WREKT!”

Activities / 10-Stu_GetWrekt

Suggested Time:
15 minutes

Get Wrekt! *(Activities / 10-Stu_GetWrekt)*

Instructions:

- Unzip the folder.
- Create an application that will check the Diaries folder that you just unzipped and automatically navigates through all of its subdirectories and files.
- Have the application print out the current root that the application is walking through.
- Have the application print out all the folders within the current root.
- Have the application print out all the files within the current root.
- Have your application automatically connect to each of the text files stored within this folder system and replace their text with the phrase "GET WREKT SKRUB!"

Bonus: Open your browser and locate and download an image. Then move the file into one of the folders within the Diaries folder. Run the application again and see what happens to the image file.

Class Objectives

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

- ✅ Import the `string`, `random`, and `hashlib` modules to access the functions within them.
- ✅ Use the `csv` module to read and write data to external CSV files.
- ✅ Use the `secrets` module to generate random passwords.
- ✅ Use the `os` module to create file paths and check for specific files.
- ✅ Use the `os` module to navigate through a folder system and automate tasks.
- ✅ Use the `os` module to perform a simple attack on an operating system.