Reading, Writing, and Pyrithmetic

Data Boot Camp

Lesson 3.2

© 2020 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.

Class Objectives

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

- Read data into Python from CSV files.
- Write data from Python to CSV files.
- Zip two files together and know when this is helpful.
- Understand well how to create and use Python functions.

Activity: Python Check-Up

In this activity, you will start with a quick warm up activity to get the Python juices flowing!

Suggested Time:
10 Minutes

Activity: Python Check-Up

Create simple Python command line application. The application should:

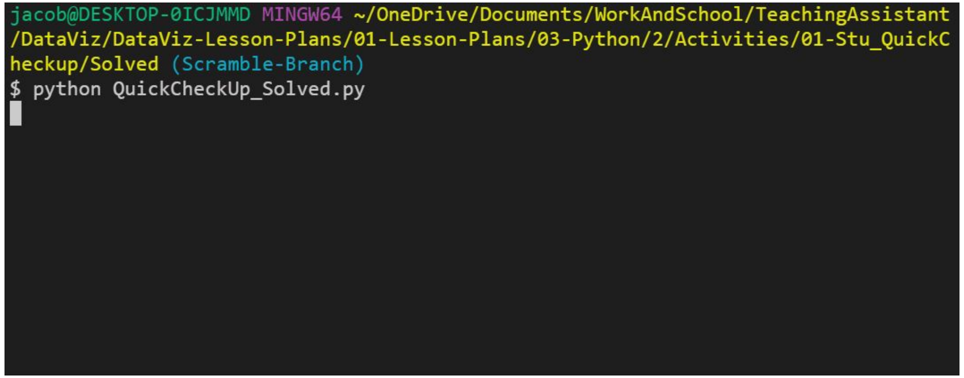
Print: “Hello User!”

Then ask: “What is your name?”

Then respond: “Hello <user’s name>”

Then ask: “What’s your age?”

Then respond: “Awwww… you’re just a baby!” or “Ah… A well traveled soul are ye”, depending on the user’s age.

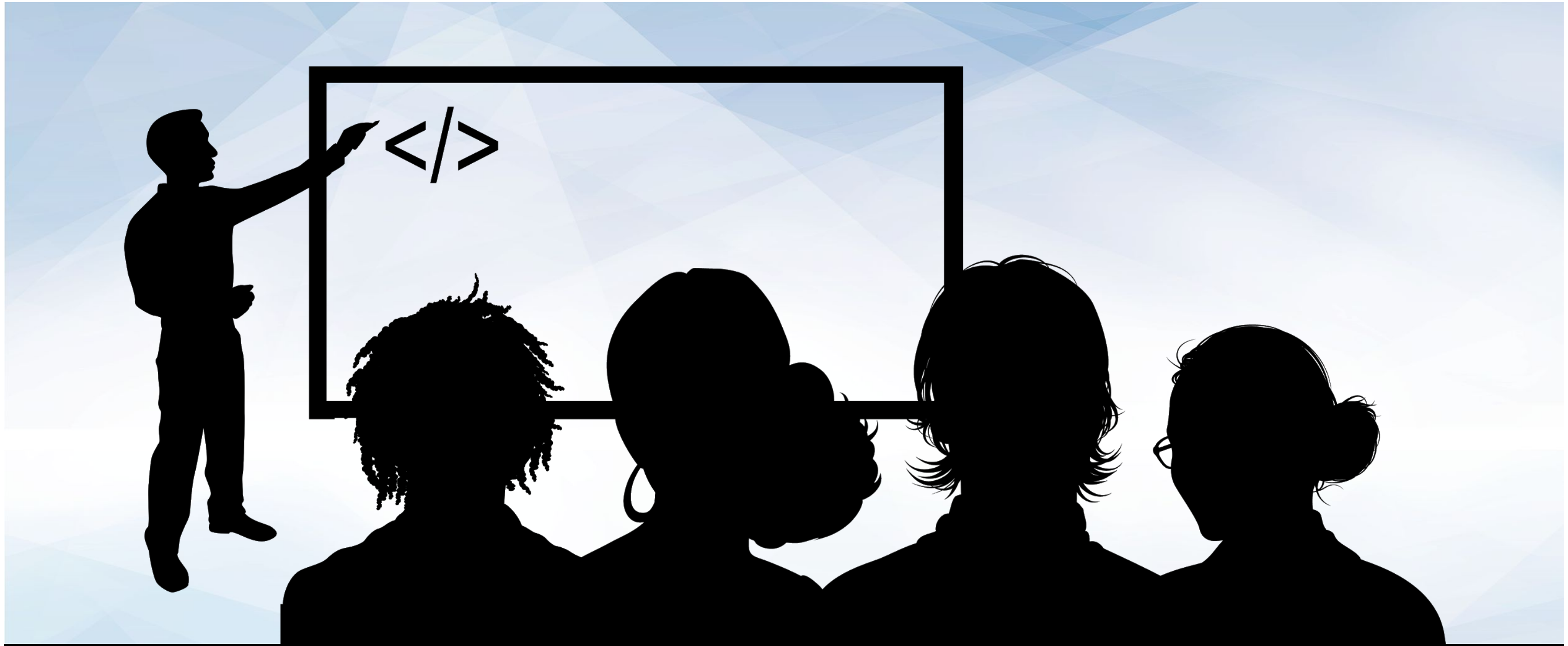


```
jacob@DESKTOP-0ICJMMD MINGW64 ~/OneDrive/Documents/WorkAndSchool/TeachingAssistant
/DataViz/DataViz-Lesson-Plans/01-Lesson-Plans/03-Python/2/Activities/01-Stu_QuickC
heckup/Solved (Scramble-Branch)
$ python QuickCheckUp_Solved.py
```

slack

01-Stu_QuickCheckup/quick_check_up.py

Time’s Up! Let’s Review.

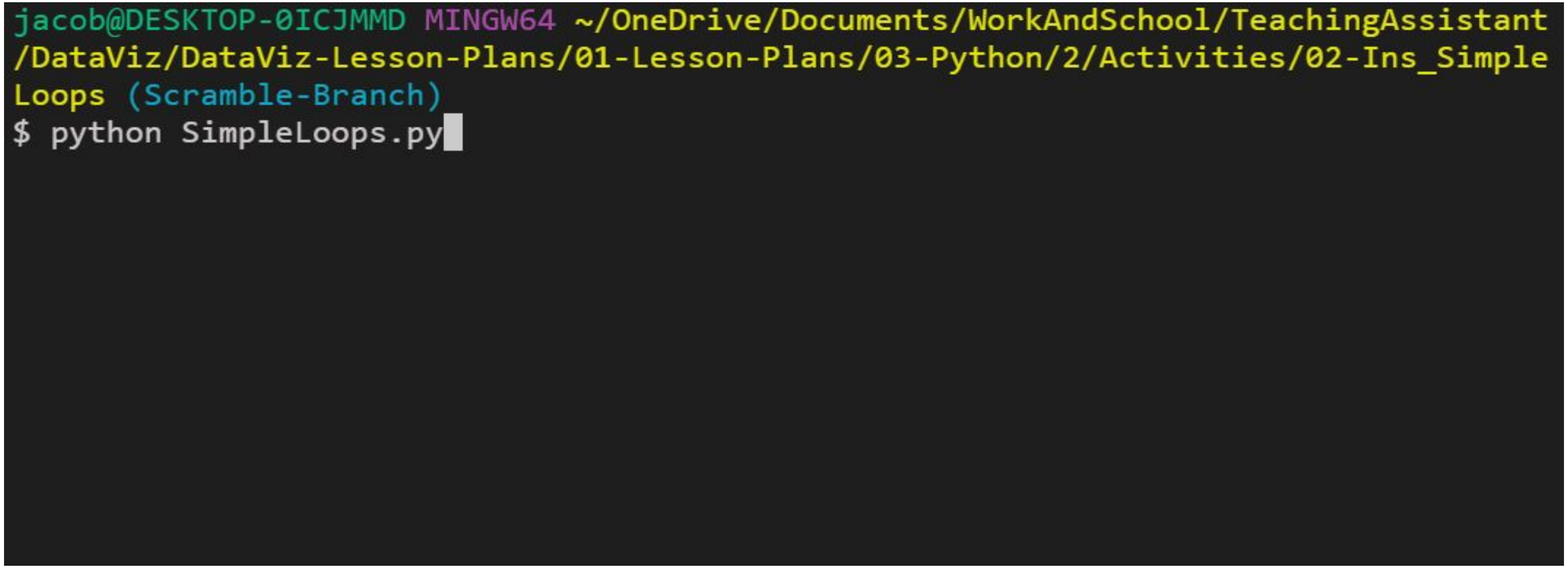


Instructor Demonstration

Loop Recap

Loop Recall: For loop

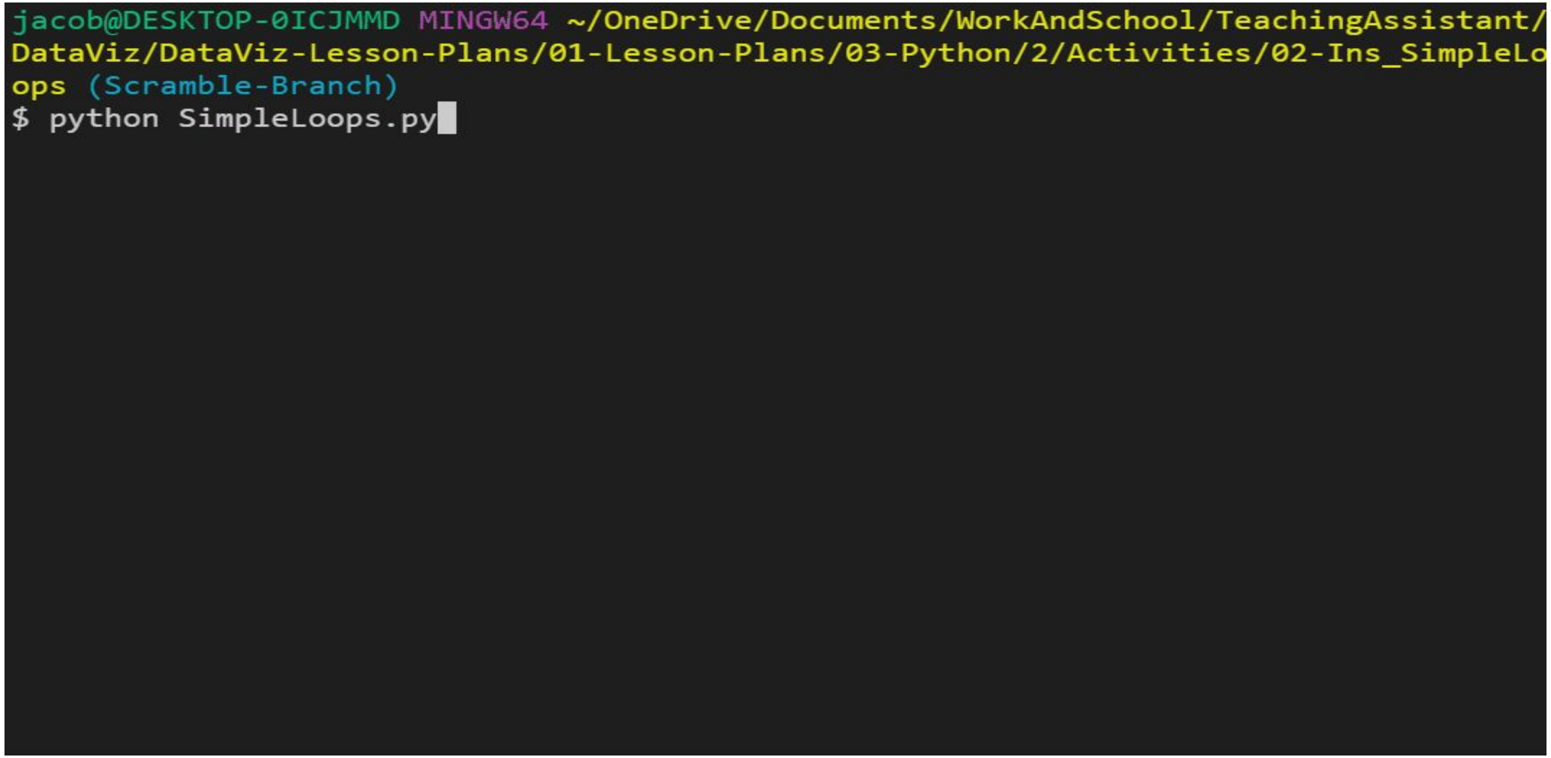
Loops through a range of numbers, the letters in a string, or the elements within a list one by one.



```
jacob@DESKTOP-0ICJMMD MINGW64 ~/OneDrive/Documents/WorkAndSchool/TeachingAssistant
/DataViz/DataViz-Lesson-Plans/01-Lesson-Plans/03-Python/2/Activities/02-Ins_Simple
Loops (Scramble-Branch)
$ python SimpleLoops.py
```

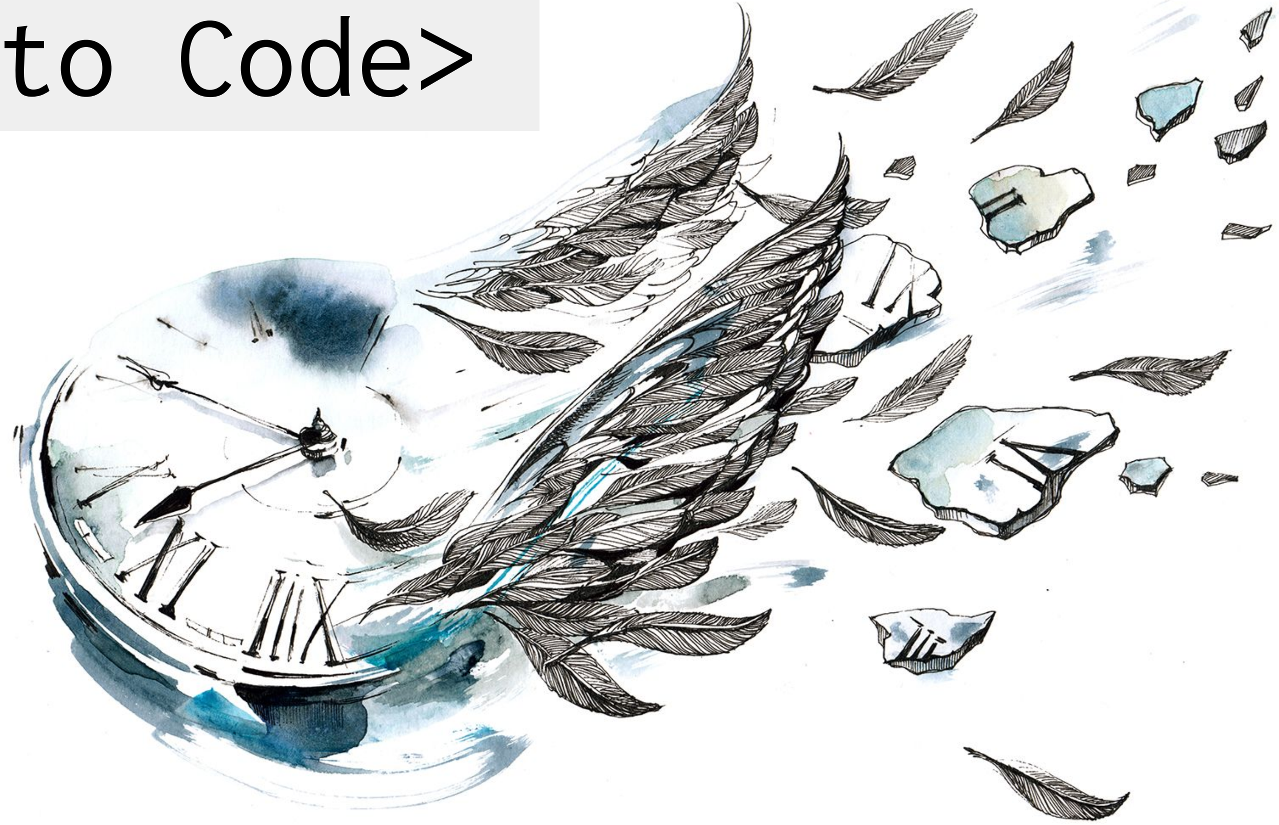
Loop Recall: While loop

Loops through the code contained inside of it until some condition is met.



```
jacob@DESKTOP-0ICJMMD MINGW64 ~/OneDrive/Documents/WorkAndSchool/TeachingAssistant/
DataViz/DataViz-Lesson-Plans/01-Lesson-Plans/03-Python/2/Activities/02-Ins_SimpleLo
ops (Scramble-Branch)
$ python SimpleLoops.py
```

<Time to Code>



Activity: Kid in a Candy Store

In this activity, you will pretend to be a kid going with their parents to the supermarket. After pestering your parents for a while, you're finally allowed to pick out some candy to take home.

Suggested Time:
15 Minutes

Activity: Kid in a Candy Store

Instructions:

- Create a loop that prints all of the candies in the store to the terminal with their index stored in brackets beside them.
  For example: `"[0] Snickers"`
- Create a loop that runs for a number of times as determined by the variable `allowance`.
  For example: If allowance is equal to five, the loop should run five times.
- Each time this second loop runs, take in a user's input - preferably a number - and then add the candy with a matching index to the variable `candy_cart`.
  For example: If the user enters ‘0’ as their input, ‘Snickers’ should be added into the `candy_cart` list.
- Use another loop to print all of the candies selected to the terminal.
- **Bonus:** Create a version of the same code which allows a user to select as much candy as they want up until they say they do not want any more.

Time’s Up! Let’s Review.

Activity: House of Pies

In this activity, you will construct an order form that will display a list of pies and then prompt users to make a selection. It will continue to prompt for selections until the user decides to terminate the process.

Suggested Time:

20 Minutes

Activity: House of Pies

Instructions: Part 1

- Create an order form that will display a list of pies to the user in the following way:

```
Welcome to the House of Pies! Here are our pies:

---------------------------------------------------------------------
(1) Pecan, (2) Apple Crisp, (3) Bean, (4) Banoffee, (5) Black Bun, (6) Blueberry, (7) Buko, (8) Burek
```

- Then prompt the user to select which pie they'd like to order via number.
- Immediately after, follow the order with `Great! We'll have that <PIE NAME> right out for you` and then ask if they would like to make another order. If so, repeat the process.
- Once the user is done purchasing pies, print the total number of pies ordered.

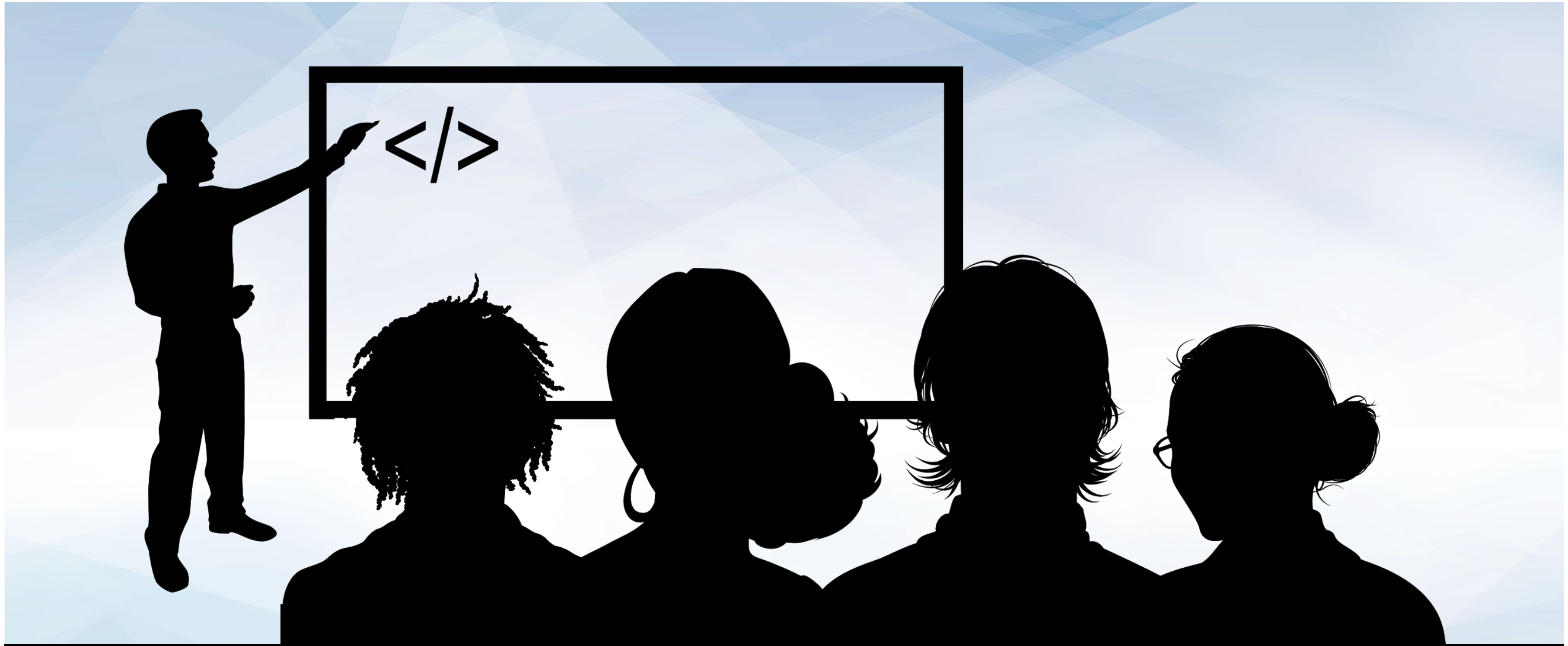
Activity: House of Pies

Instructions: Part 2

Modify the application once again, this time conclude the user's purchases by listing out the total pie count broken by ***each*** pie.

```
Welcome to the House of Pies! Here are our pies:
0 Pecan
0 Apple Crisp
0 Bean
2 Banoffee
0 Black Bun
0 Blueberry
0 Buko
0 Burek
0 Tamale
1 Steak
```

Time's Up! Let's Review.



Instructor Demonstration

Reading Text Files

Python can read data in from external text files to perform some tasks on it!

Reading Text Files

As we all need directions to move from point A to point B, Python is no different when dealing with external files. It requires very precise directions on what path to follow in order to reach the desired file.

In this case the desired file is located within a sub-folder called ‘Resources’, the path we need to tell Python would be `'Resources/FileName.txt'`.

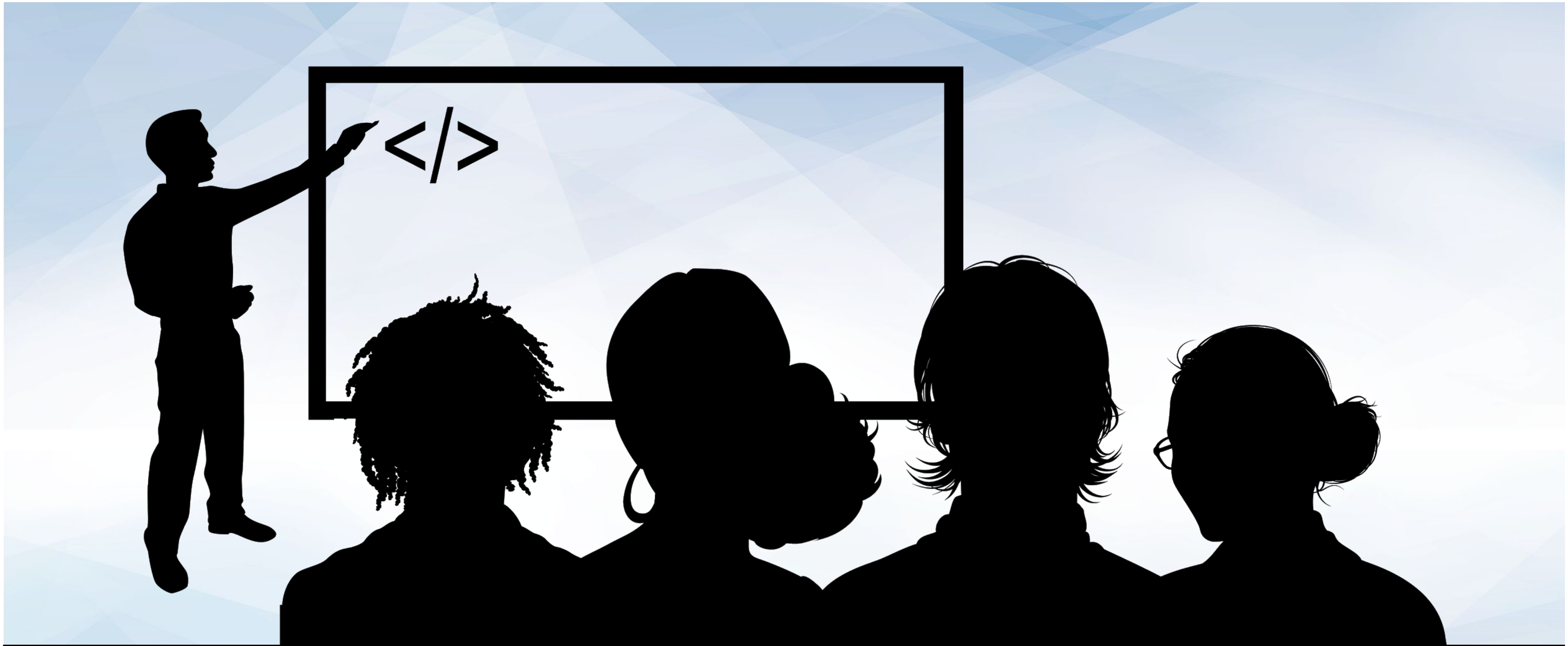
NOTE: Different operating systems use different ways to set their paths.

```
# Store the file path associated with the file
(note the backslash may be OS specific)
file = 'Resources/input.txt'
```

Reading Text Files

- `with` is a special syntactical block that allows to perform operations that require a safety ‘clean-up’ after the code block is completed.
- `open<File Path>, <Read/Write>` is the function Python uses to open a file. The function can be specified with `'r'`, `'w'`, or `'rw'`, in order to only read, only write, or perform both operations.
- `text.read()` reads the entire file converting to a string type.

```python
# Open the file in "read" mode ('r') and store
the contents in the variable "text"
with open(file, 'r') as text:
```



Instructor Demonstration

Introduction to Modules

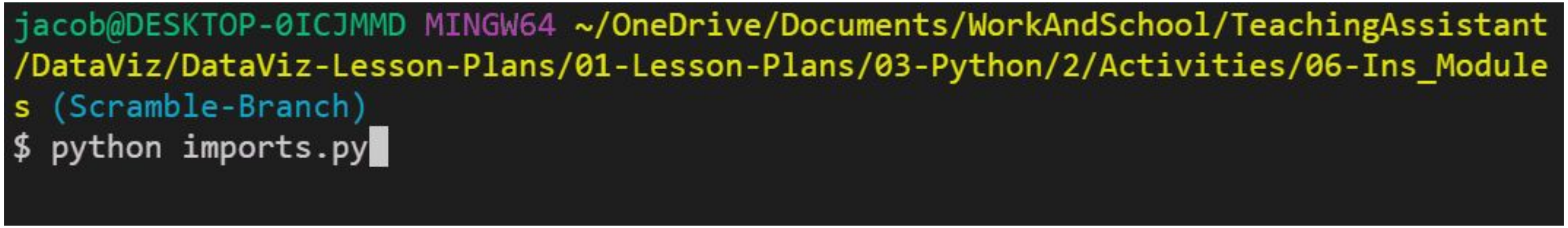
No built-in function for my specific task?

Relax! We can bring in external modules to perform the specific task.

Introduction to Modules

Import Modules

The `string` module contains many helpful constants and methods that pertain to strings. For example, users can use `string.ascii_letters` and Python will instantly grab a reference to every ascii character.



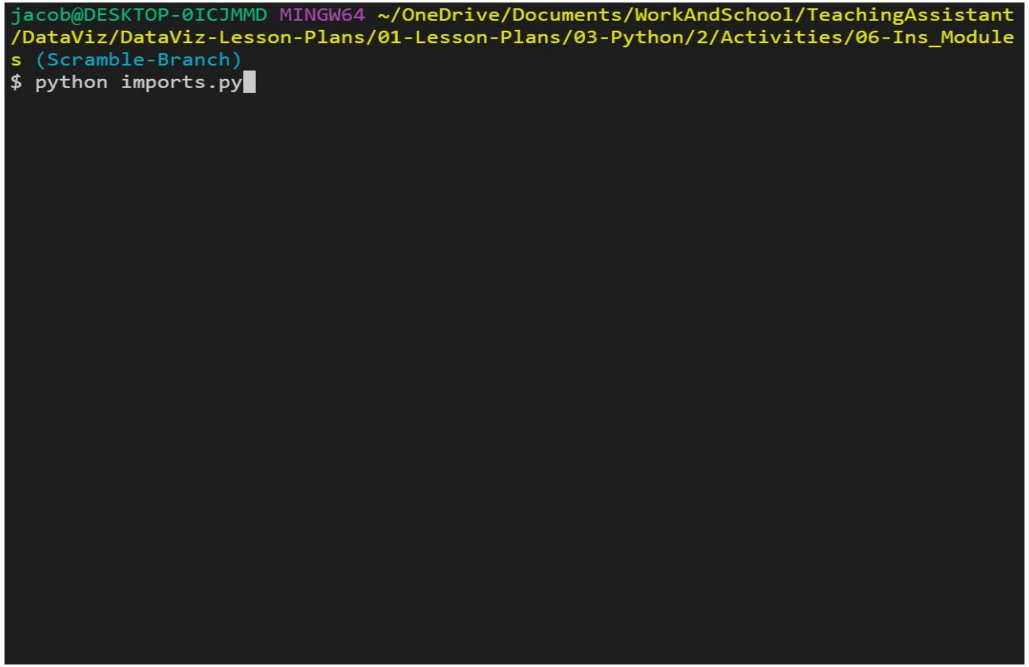
```
jacob@DESKTOP-0ICJMMD MINGW64 ~/OneDrive/Documents/WorkAndSchool/TeachingAssistant/DataViz/DataViz-Lesson-Plans/01-Lesson-Plans/03-Python/2/Activities/06-Ins_Modules (Scramble-Branch)
$ python imports.py
```

```
# Import the String Module
import string

# Utilize the string module’s custom method: ".ascii_letters"
import print(string.ascii_letters)
```

Import Modules

The `random` module does exactly what one might expect, it allows Python to randomly select values from set ranges, lists or even strings.



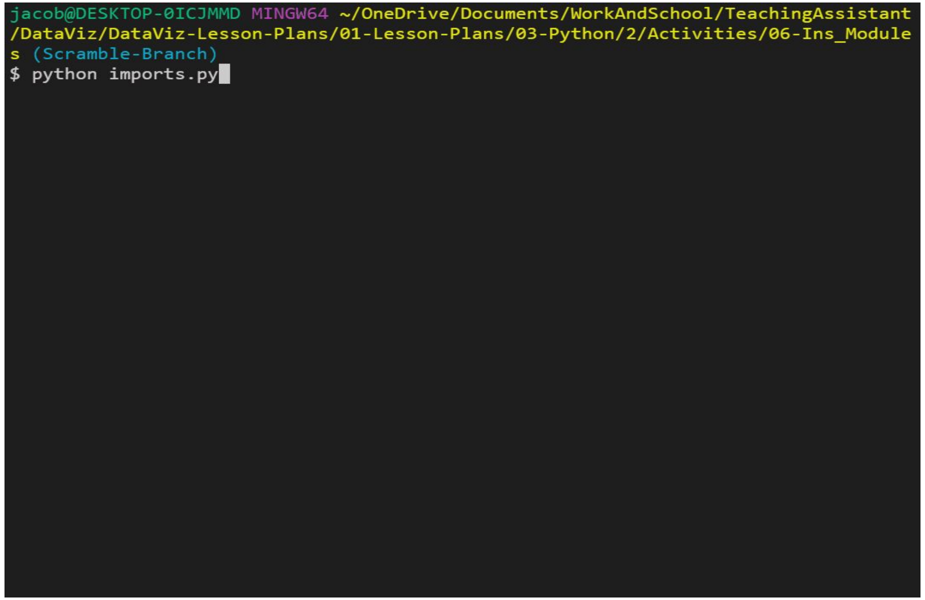
```
jacob@DESKTOP-0ICJMMD MINGW64 ~/OneDrive/Documents/WorkAndSchool/TeachingAssistant
/DataViz/DataViz-Lesson-Plans/01-Lesson-Plans/03-Python/2/Activities/06-Ins_Module
s (Scramble-Branch)
$ python imports.py
```

Import Modules

The `random` module does exactly what one might expect, it allows Python to randomly select values from set ranges, lists, or even strings.

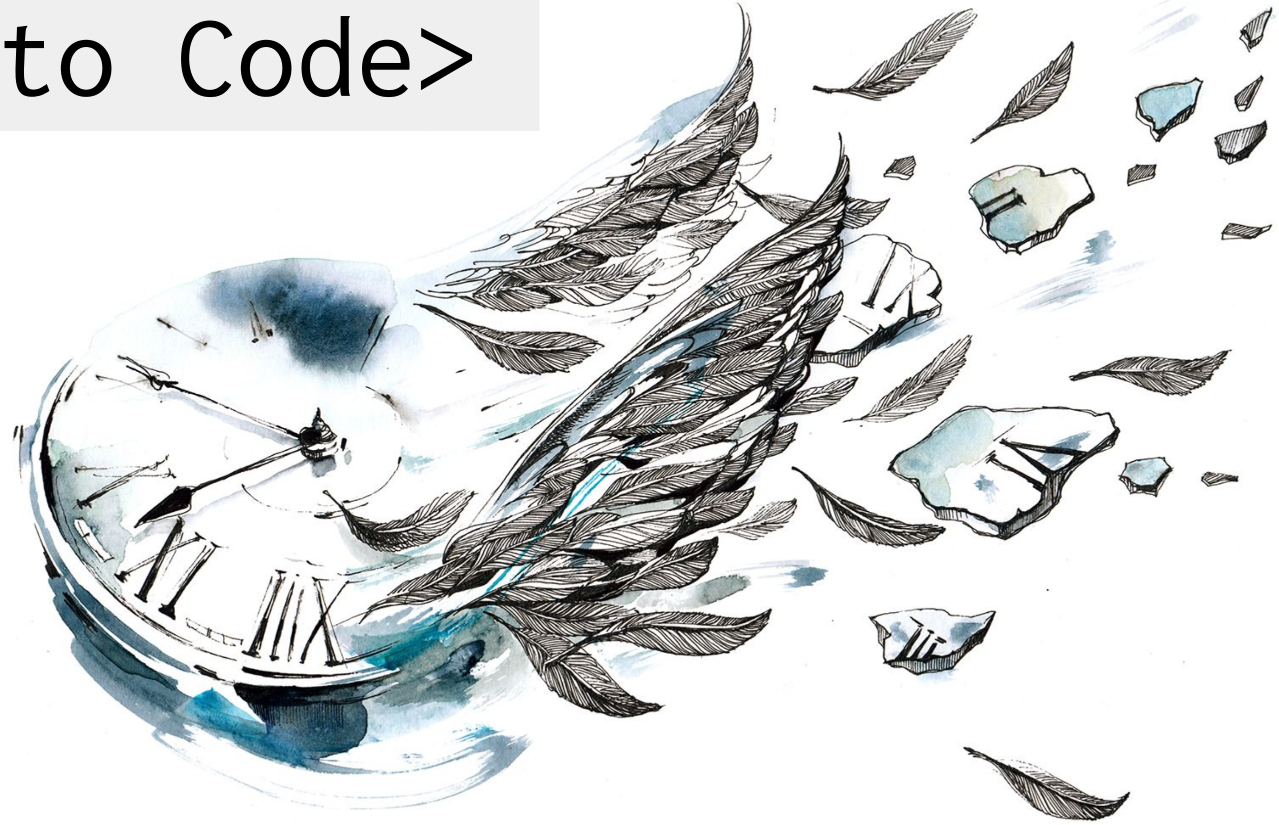
```python
# Import the Random Module
import random

# Utilize the random module's
custom method randint
print(random.randint(1,10))
```



```
jacob@DESKTOP-0ICJMMD MINGW64 ~/OneDrive/Documents/WorkAndSchool/TeachingAssistant
/DataViz/DataViz-Lesson-Plans/01-Lesson-Plans/03-Python/2/Activities/06-Ins_Module
s (Scramble-Branch)
$ python imports.py
```

<Time to Code>



Activity: Module Playground

In this activity, you will have the opportunity to explore some of Python's modules and play around with them.

Suggested Time:
5 Minutes

Activity: Module Playground

Instructions:

There are tons of built-in modules for Python and there is no possible way that a single class could cover all of them. Source yourself Built-In Python modules and share with the class.

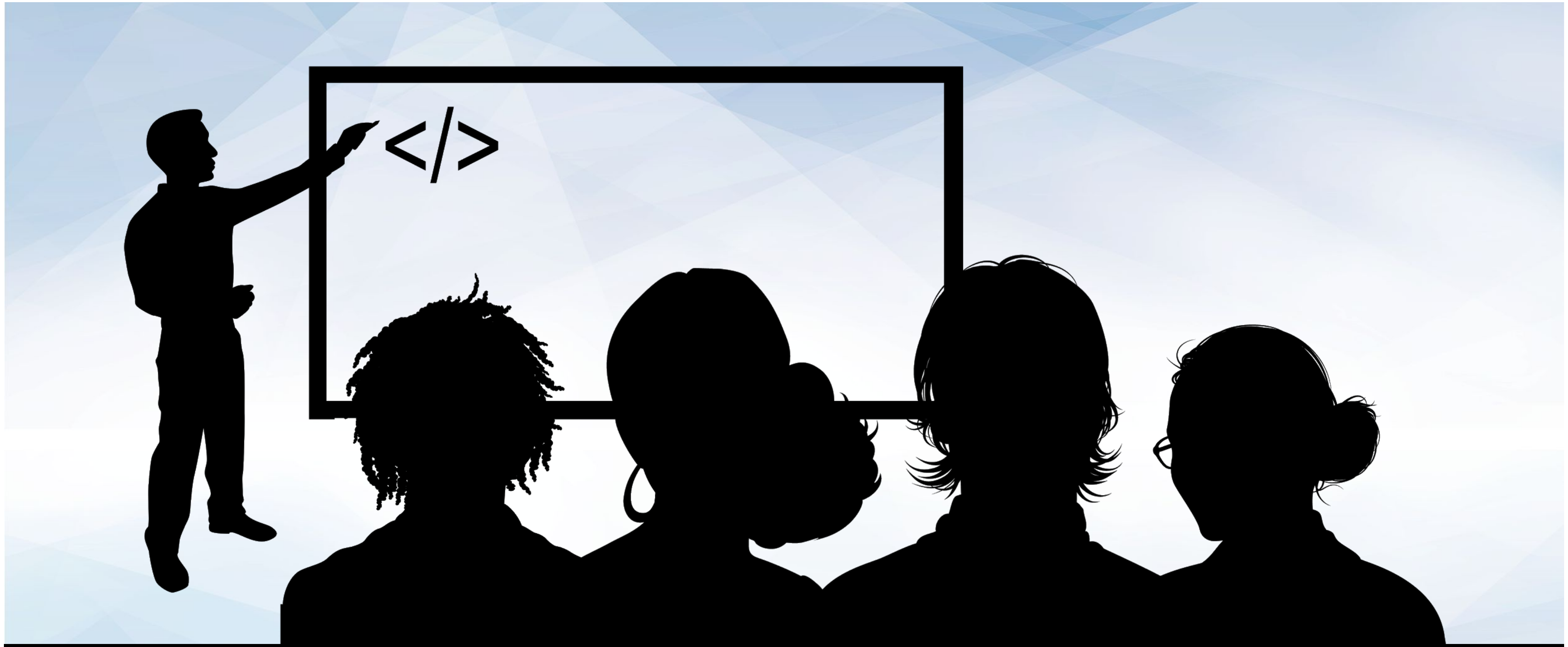
Hint: Use your Google-fu skills.

Time's Up! Let's Review.

Countdown timer

15:00

(with alarm)



Instructor Demonstration

Reading in CSV Files

It is more likely within the data industry to run across files known as **CSV files**.

Reading in CSV Files

Comma Separated Values

- **CSV** stands for **Comma Separated Values** and is essentially a table that has been converted into text format with each row and column being separated by specified symbols.
- More often than not each row is located on a new line and each column is separated by a comma. Seems simple enough since this is why the file type is called Comma Separated Values.

```
First Name,Last Name,TFN
Charlotte,Smith,865414088
Henry,Jones,656077298
Grace,Williams,125486619
Isla,Brown,551953801
Olivia,Wilson,615315318
Ethan,Taylor,907974668
James,Johnson,459599230
Lucas,Walker,996506823
Amelia,Harris,243494429
Eva,Lee,796801997
Noah,Robinson,754601340
```

Python has a module called `csv` which pulls in data from external **CSV** files and perform some operations upon them.

Reading in CSV Files: os module

The first major piece of code to point out is the importing and usages of the os module. This module allows Python programmers to very easily create dynamic paths to external files that function across different operating systems.

```
# First we’ll import the os module
# This will allow us to create file paths across operating systems
import os

csvpath = os.path.join('Resources', 'accounting.csv')
```

Reading in CSV Files: csv reader

Instead of `text.read()`, this new code instead utilises `csv.reader()` to translate the object being opened by Python. It is critical to note the `delimiter=','` parameter being used as this tells Python that each comma within the CSV should be seen as moving into a new column for a row.

```
import csv
with open(csvpath, newline='') as csvfile:

    # CSV reader specifies delimiter and variable that holds contents
    csvreader = csv.reader(csvfile, delimiter=',')

    print(csvreader)

    # Each row is read as a row
    for row in csvreader:
        print(row)
```

Reading in CSV Files: csv reader

The code then loops through each row of the CSV and prints out the contents. Each value is being shown as a string and all of the rows are lists.

```
$python ReadCSV.py
['First Name', 'Last Name', 'TFN']
['Charlotte', 'Smith','865414088']
['Henry', 'Jones', '656077298']
['Grace', 'Williams', '125486619']
['Isla', 'Brown', '551953801']
['Olivia', Wilson', '615315318']
['Ethan', 'Taylor', '907974668']
['James', 'Johnson', '459599230']
['Lucas', 'Walker', '996506823']
['Amelia', 'Harris', '243494429']
['Eva', 'Lee', '796801997']
['Noah', 'Robinson', '754601340']
```

<Time to Code>

Activity: Reading Netflix

In this activity, you will use a CSV file containing data taken from Netflix to create an application which searches through the data for a specific movie/show and returns the name, rating, and review score for it.

Suggested Time:
15 Minutes

Activity: Reading Netflix

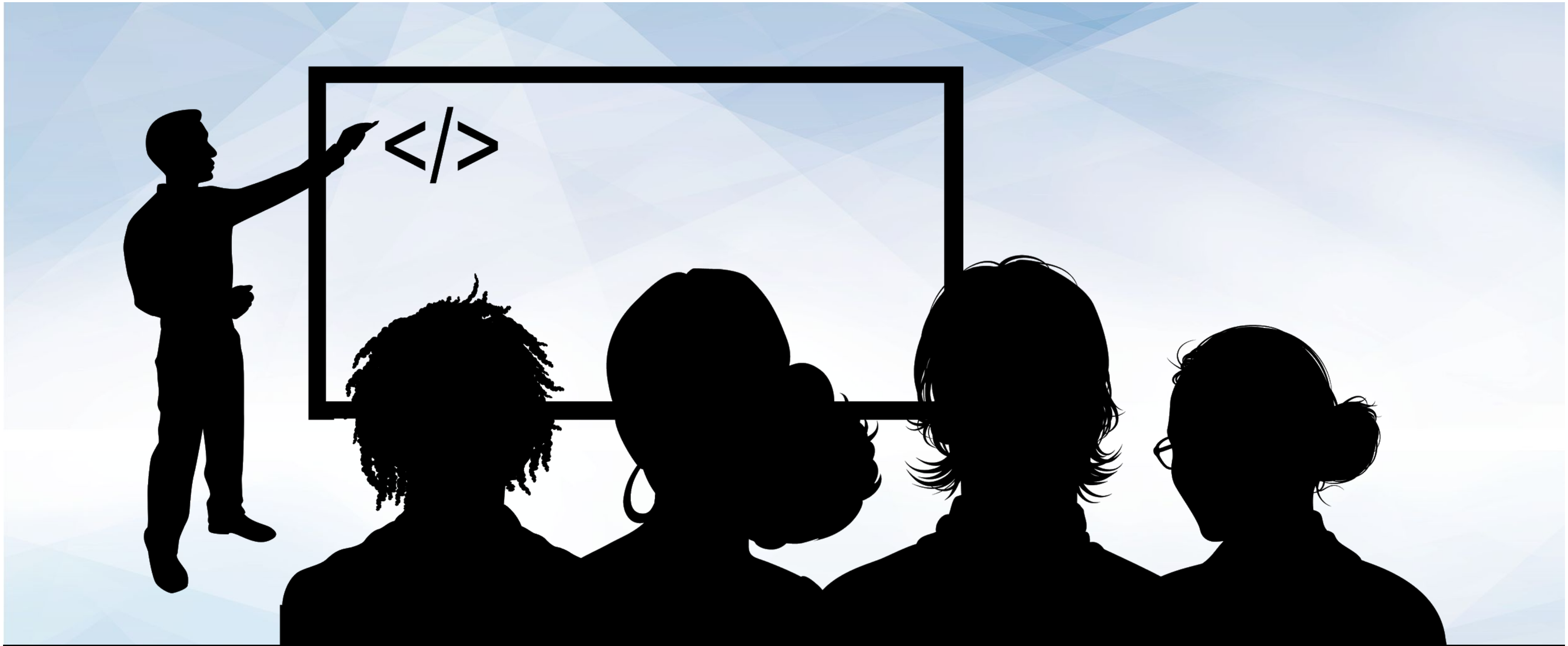
Instructions:

- Prompt the user for what video they are looking for.
- Search through the `netflix_ratings.csv` to find the user's video.
- If the CSV contains the user's video then print out the title, what it is rated, and the current user ratings. For example: `'Grease is rated PG with a rating of 86'`

Bonus

- You may have noticed that there is more than one listing for some videos.
- Edit your code to have the title, the rating, and user rating printed out only once.
- Set a variable to `False` to check if we found the video.
- In the `for loop` change the variable to confirm that the video is found.
- Insert a `break` statement into the `for loop` to stop the loop when the first movie is found. Check your Slack for documentation.
- If the CSV does not contain the user's video then print out a message telling them that their video could not be found.

Time's Up! Let's Review.



Instructor Demonstration

Writing CSV Files

We all know Python can read data in from CSVs.

Now, can we also write data to a CSV? Let’s find out!

Writing CSV Files

`os.path.join("..", "output", "new.csv")` commands Python the file to write to while assign it to the variable `output_path`.

```python
# Dependencies
import os
import csv

# Specify the file to write to
output_path = os.path.join("..", "output", "new.csv")

# Open the file using "write" mode. Specify the variable
to hold the contents
with open(output_path, 'w') as csvfile:

    # Initialize csv.writer
    csvwriter = csv.writer(csvfile, delimiter=',')

    # Write the first row (column headers)
    csvwriter.writerow(['First Name', 'Last Name', 'TFN'])

    # Write the second row
    csvwriter.writerow(['Charlotte', 'Smith', '865414088'])
```

Writing CSV Files

`with open(output_path, 'w') as csvfile:` is telling Python to open the file using ‘write’ mode while holding the contents in `output_path`.

```python
# Dependencies
import os
import csv

# Specify the file to write to
output_path = os.path.join("..", "output", "new.csv")

# Open the file using "write" mode. Specify the variable
to hold the contents
with open(output_path, 'w') as csvfile:

    # Initialize csv.writer
    csvwriter = csv.writer(csvfile, delimiter=',')

    # Write the first row (column headers)
    csvwriter.writerow(['First Name', 'Last Name', 'TFN'])

    # Write the second row
    csvwriter.writerow(['Charlotte', 'Smith', '865414088'])
```

Writing CSV Files

`csv.writer()` is telling Python that this application will be writing code into an external CSV file.

```python
# Dependencies
import os
import csv

# Specify the file to write to
output_path = os.path.join("..", "output", "new.csv")

# Open the file using "write" mode. Specify the variable
to hold the contents
with open(output_path, 'w') as csvfile:

    # Initialize csv.writer
    csvwriter = csv.writer(csvfile, delimiter=',')

    # Write the first row (column headers)
    csvwriter.writerow(['First Name', 'Last Name', 'TFN'])

    # Write the second row
    csvwriter.writerow(['Charlotte', 'Smith', '865414088'])
```

Writing CSV Files

`csv.writerow()` is the code to write a new row into a CSV file.

```python
# Dependencies
import os
import csv

# Specify the file to write to
output_path = os.path.join("..", "output", "new.csv")

# Open the file using "write" mode. Specify the variable
to hold the contents
with open(output_path, 'w') as csvfile:

    # Initialize csv.writer
    csvwriter = csv.writer(csvfile, delimiter=',')

    # Write the first row (column headers)
    csvwriter.writerow(['First Name', 'Last Name', 'TFN'])

    # Write the second row
    csvwriter.writerow(['Charlotte', 'Smith', '865414088'])
```

The syntax for writing into a CSV file is thankfully very similar to that used to read data in from an external file.

Writing CSV Files

First, the code references the path that will point into the CSV file the user would like to write to.

```python
# Specify the file to write to
output_path = os.path.join('output', 'new.csv')

# Open the file using "write" mode. Specify the variable
to hold the contents
with open(output_path, 'w', newline='') as csvfile:

    # Initialize csv.writer
    csvwriter = csv.writer(csvfile, delimiter=',')

    # Write the first row (column headers)
    csvwriter.writerow(['First Name', 'Last Name', 'TFN'])

    # Write the second row
    csvwriter.writerow(['Charlotte', 'Smith', '865414088'])
```

Writing CSV Files

Next, the `with open()` statement is used once more but with one significant difference. Instead of the parameter `'r'` being passed and directing Python to read a file, the parameter `'w'` is passed instead to inform Python to write to the file.

```python
# Specify the file to write to
output_path = os.path.join('output', 'new.csv')

# Open the file using "write" mode. Specify the variable
to hold the contents
with open(output_path, 'w', newline='') as csvfile:

    # Initialize csv.writer
    csvwriter = csv.writer(csvfile, delimiter=',')

    # Write the first row (column headers)
    csvwriter.writerow(['First Name', 'Last Name', 'TFN'])

    # Write the second row
    csvwriter.writerow(['Charlotte', 'Smith', '865414088'])
```

Writing CSV Files

Instead of `read.csv()`, `csv.write()` is used to once again inform Python that this application will be writing code into an external CSV file.

```
# Specify the file to write to
output_path = os.path.join('output', 'new.csv')

# Open the file using "write" mode. Specify the variable
to hold the contents
with open(output_path, 'w', newline='') as csvfile:

    # Initialize csv.writer
    csvwriter = csv.writer(csvfile, delimiter=',')

    # Write the first row (column headers)
    csvwriter.writerow(['First Name', 'Last Name', 'TFN'])

    # Write the second row
    csvwriter.writerow(['Charlotte', 'Smith', '865414088'])
```

Writing CSV Files

To write a new row into a CSV file, simply use the `csv.writerow(<DATA FILE>)` function and pass in an array of data as the parameter.

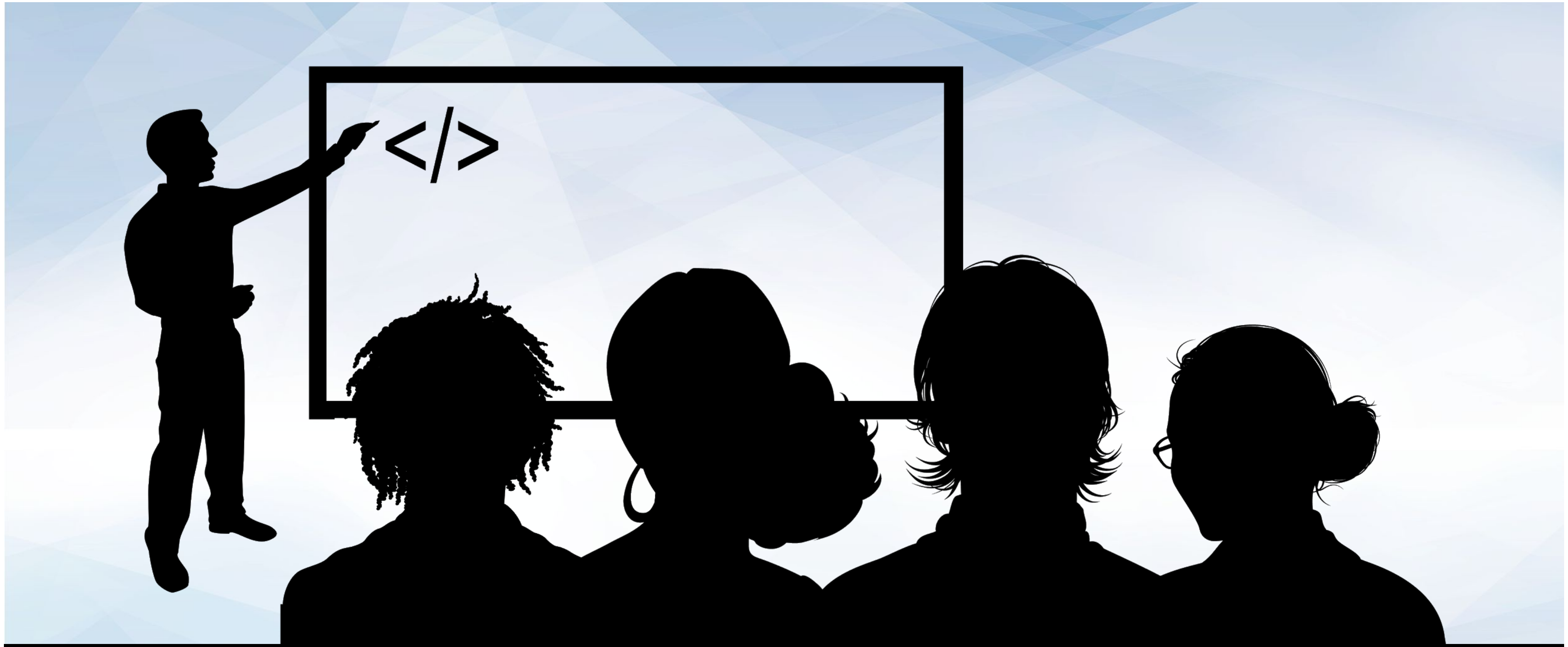
```
# Specify the file to write to
output_path = os.path.join('output', 'new.csv')

# Open the file using "write" mode. Specify the variable
to hold the contents
with open(output_path, 'w', newline='') as csvfile:

     # Initialize csv.writer
     csvwriter = csv.writer(csvfile, delimiter=',')

     # Write the first row (column headers)
     csvwriter.writerow(['First Name', 'Last Name', 'TFN'])

     # Write the second row
     csvwriter.writerow(['Charlotte', 'Smith', '865414088'])
```



Instructor Demonstration

Zipping Lists

Python users can far more efficiently write data into a new CSV file by using the `zip()` function.

Zipping Lists

`zip()` takes in a series of lists as its parameters and joins them together into a stack.

```
$ python zipper.py
(1, 'Charlotte', 'Manager')
(2, 'Henry', 'Sales')
(3, 'Grace', 'Sales')
(4, 'Isla', 'HR')
```

```python
# Three Lists
indexes = [1, 2, 3, 4]
employees = ["Charlotte", "Henry", "Grace", "Isla"]
department = ["Manager", "Sales", "Sales", "HR"]

# Zip all three lists together into tuples
roster = zip(indexes, employees, department)

# Print the contents of each row
for employee in roster:
    print(employee)
```

Zipping Lists

By zipping these lists together, there is now a single joined list whose indexes reference all three of the lists inside.

```
$ python zipper.py
(1, 'Charlotte', 'Manager')
(2, 'Henry', 'Sales')
(3, 'Grace', 'Sales')
(4, 'Isla', 'HR')
```

```python
# Three Lists
indexes = [1, 2, 3, 4]
employees = ["Charlotte", "Henry", "Grace", "Isla"]
department = ["Manager", "Sales", "Sales", "HR"]

# Zip all three lists together into tuples
roster = zip(indexes, employees, department)

# Print the contents of each row
for employee in roster:
    print(employee)
```

Zipping Lists

Each zipped object can only be used once. For example, you can write the zipped object to a CSV or print to the terminal, but not both.

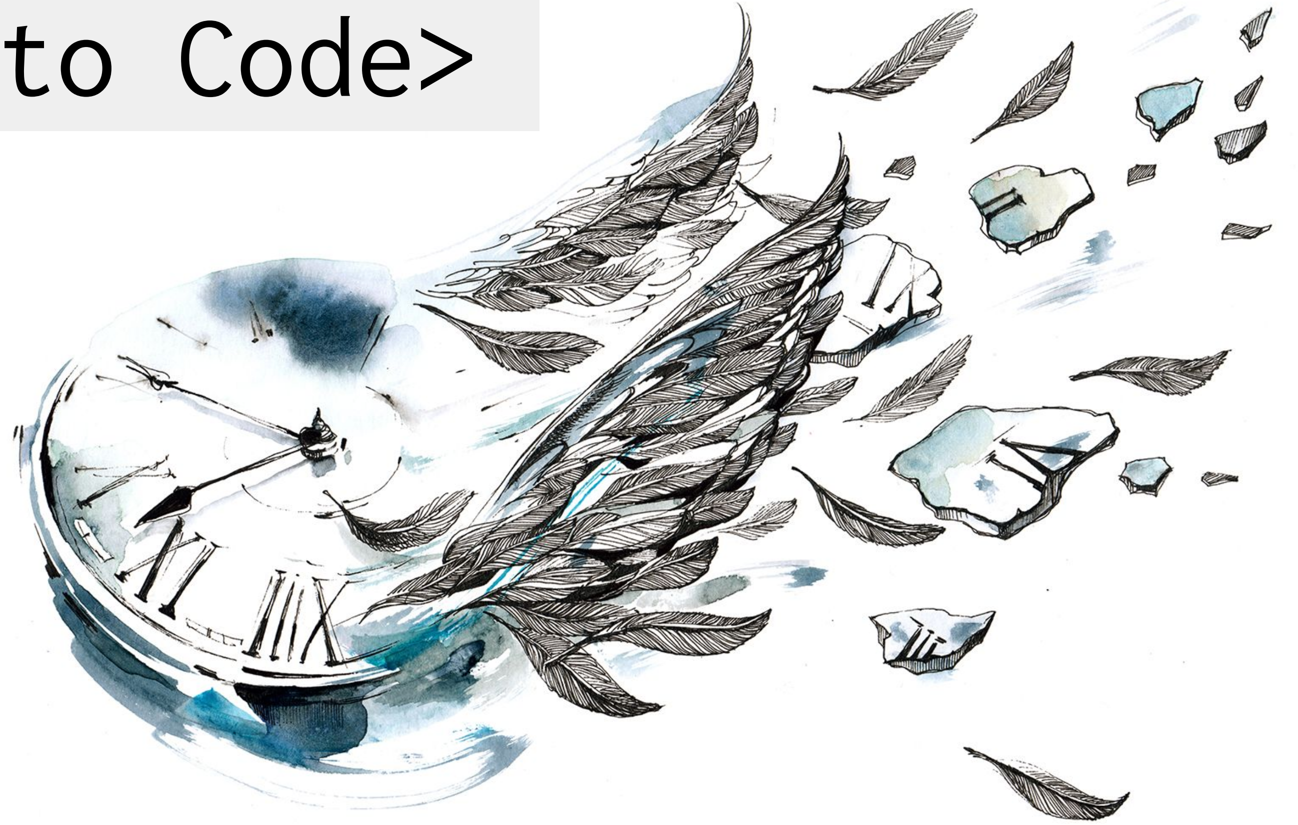
```
$ python zipper.py
(1, 'Charlotte', 'Manager')
(2, 'Henry', 'Sales')
(3, 'Grace', 'Sales')
(4, 'Isla', 'HR')
```

```python
# Three Lists
indexes = [1, 2, 3, 4]
employees = ["Charlotte", "Henry", "Grace", "Isla"]
department = ["Manager", "Sales", "Sales", "HR"]

# Zip all three lists together into tuples
roster = zip(indexes, employees, department)

# Print the contents of each row
for employee in roster:
    print(employee)
```

<Time to Code>



Activity: Udemy Zip

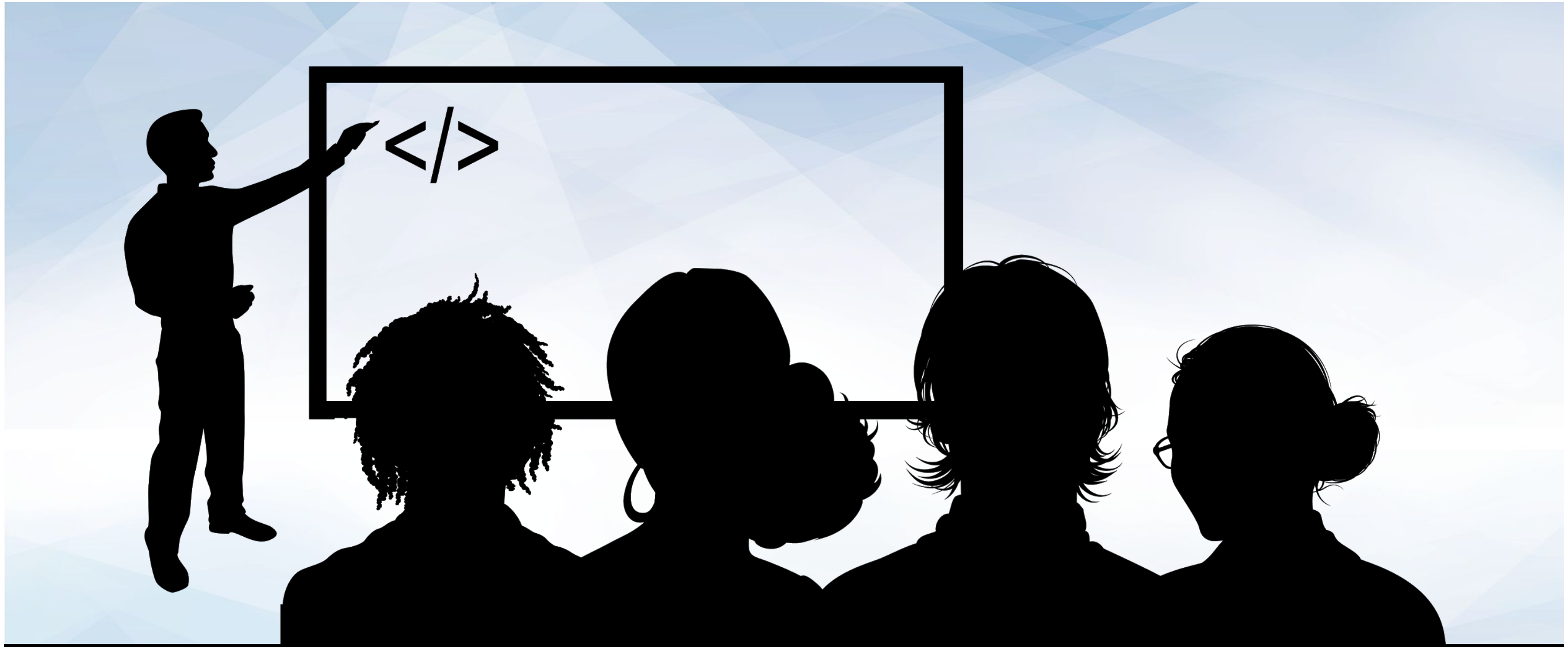
In this activity, you will receive a large dataset from Udemy. Your goal is to clean the file up, make it easier to comprehend, and finally to create a CSV file.

Suggested Time:
20 Minutes

Activity: Udemy Zip

Instructions	**Notes**	**Bonus**
Create a Python application that reads the data on Udemy Web Development offerings.	As, with many datasets, the file does not include the header line. Use the below as a guide on the columns: `‘id, title, url, isPaid, price, numSubscribers, numReviews, numPublishedLectures, instructionalLevel, contentInfo, publishedTime’`	Find the percent of subscribers that have also left a review on the course. Include this in your final output.
Then store the contents of the Title, Price, Subscriber Count, Number of Reviews, and Course Length into Python Lists.		
Then zip these lists together into a single tuple.		Parse the string associated with course length, such that we store it as an integer instead of a string. (i.e., ‘4 hours’ should be converted to 4).
Finally, write the contents of your extracted data into a CSV. Make sure to include the titles of these columns in your CSV.		

Time’s Up! Let’s Review.



Instructor Demonstration

Introduction to Functions

DRY—It stands for **D**on’t **R**epeat **Y**ourself.

It is a very popular acronym among the tech community, and many coders live by it.

Introduction to Functions

Preventing repetition by liberal usage of Python functions.

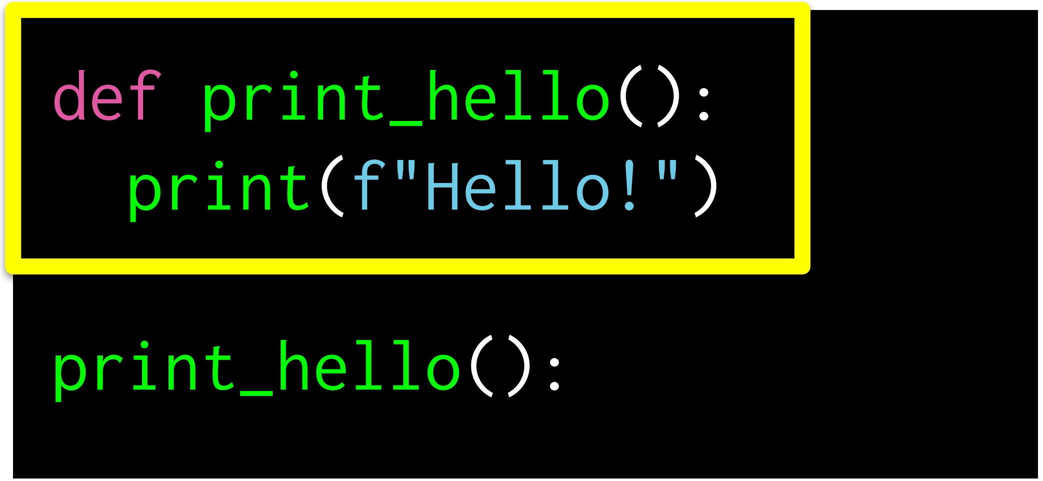
A function is a block of organised, reusable code that is used to perform a single, related action. In other words, functions are placeable blocks of code that perform a specific action.

```
def print_hello():
  print(f"Hello!")

print_hello():
```

Preventing repetition by liberal usage of Python functions.

To create a new function, simply use def `<FUNCTION NAME>():` and then place the code that you would like to run within the block underneath it.

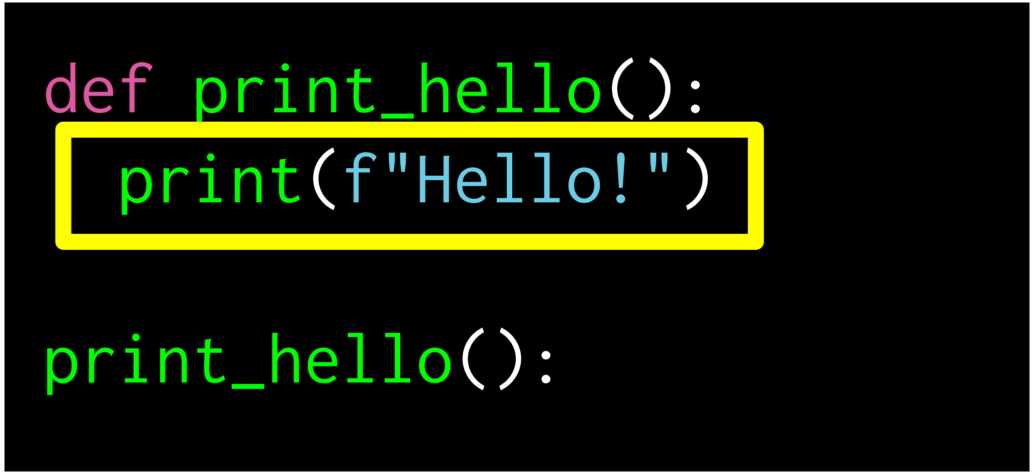


```
def print_hello():
  print(f"Hello!")

print_hello():
```

Preventing repetition by liberal usage of Python functions.

In order to run the code stored within a function, the function itself must be called within the program. Functions will never run unless called upon.



```
def print_hello():
  print(f"Hello!")

print_hello():
```

Preventing repetition by liberal usage of Python functions.

Functions that take in parameters can also be created by simply adding a variable into the parentheses of the function's definition. This allows specific data to be passed into the function for usage.

```python
def print_name(name):
    print("Hello" + name + "!")

print_name("Noah Robinson")
```

The End