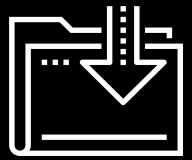
Let’s Begin Python!

Data Boot Camp

Lesson 3.1

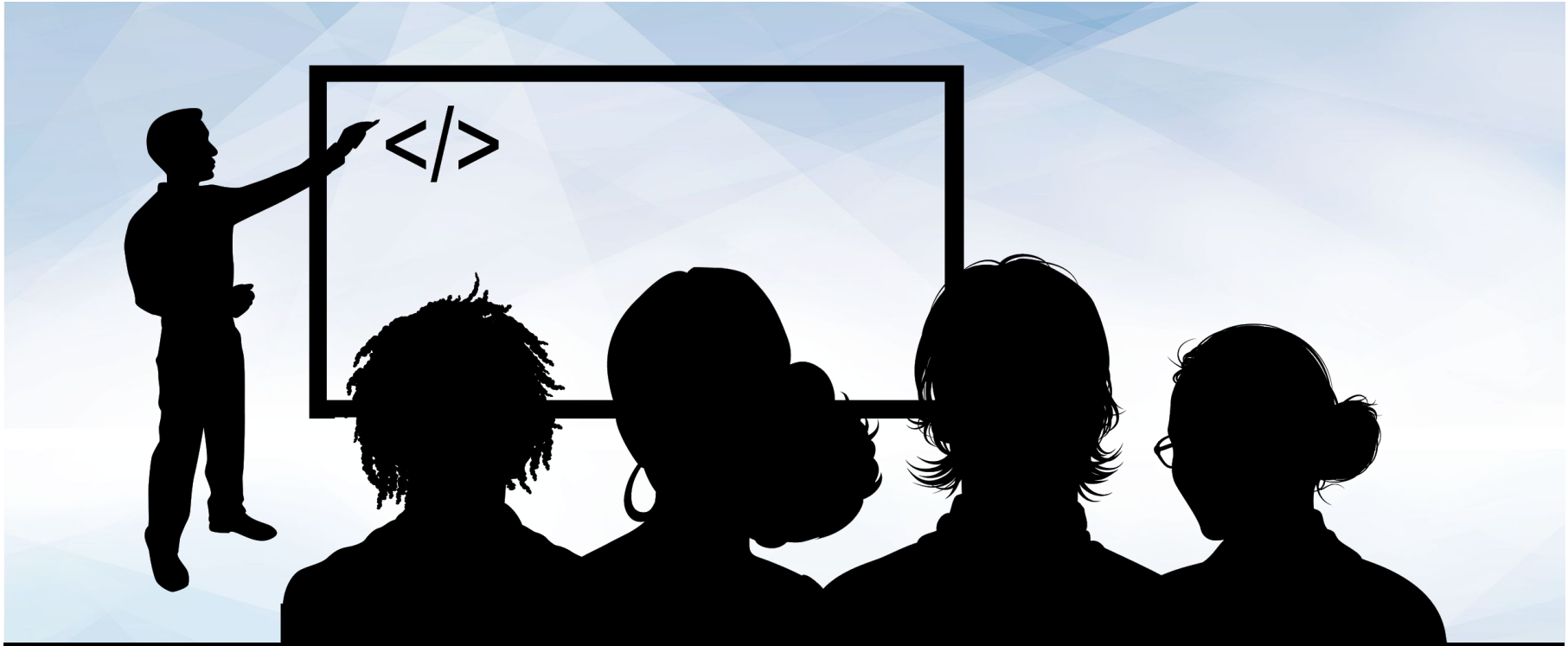


Class Objectives

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

- Successfully have Python 3 installed.
- Navigate your file system using the terminal/git-bash.
- Create and run Python script from terminal/git-bash.
- Differentiate between a sample and a population in regards to a dataset.

We Will Build on Concepts You Already Know



Instructor Demonstration

Introduction to Terminal/Git-Bash

Basic Terminal/Git-Bash Commands

Navigation

- `cd <path/to/directory>` (Changes directory to specified path).
- `cd ~` (Changes to the home directory).
- `cd ..` (Moves up one directory).
- `ls` (Lists files in and directories in the current directory).
- `pwd` (Shows the path of the current directory).

Creation

- `mkdir <FOLDERNAME>` (Creates a new directory with the FOLDERNAME).
- `touch <FILENAME>` (Creates a new file with the FILENAME).
- `rm <FILENAME>` (Deletes a file).
- `rm -r <FOLDERNAME>` (Deletes a folder, the -r).

Opening Files and Folders - Mac OS

- `open .` (Opens the current folder on Macs).
- `open <FILENAME>` (Opens a specific file on default program for that file type).

```
$ open .
$ open test.py
$
```

Opening Files and Folders - GitBash on Windows

- `explorer .` (Opens the current folder on GitBash).
- `explorer <FILENAME>` (Opens a specific file on GitBash).

Creating and Running our first Python Script

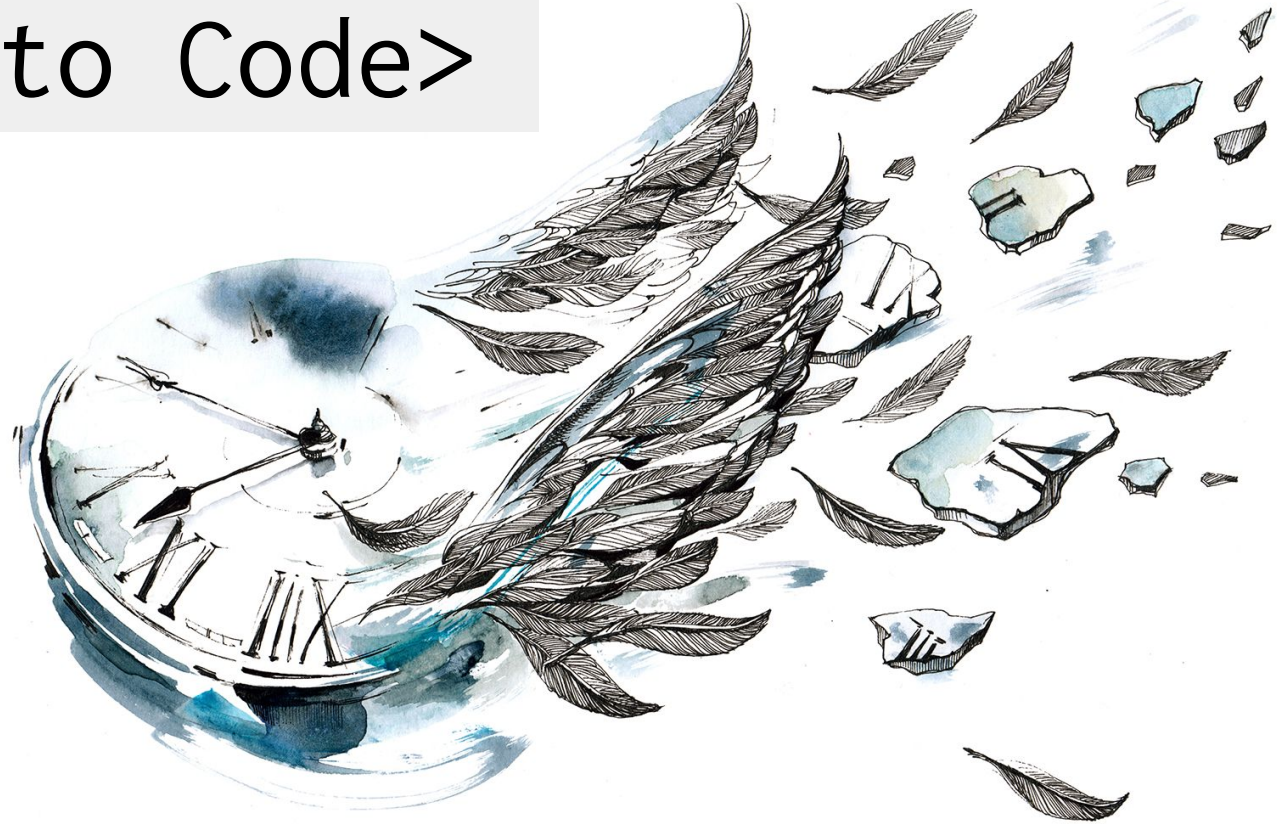
GitBash

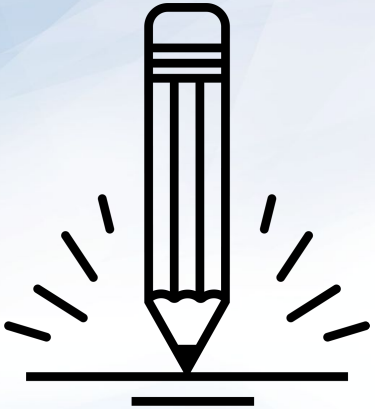
```
Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop
$ mkdir PythonStuff

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop
$ cd PythonStuff

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/PythonStuff
$ touch first_file.py

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/PythonStuff
$ explorer first_file.py

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/PythonStuff
$ python first_file.py
This is my first Python code
```

Mac Terminal

```
[$ mkdir PythonStuff
[$ cd PythonStuff/
[$ touch first_file.py
[$ open first_file.py
[$ python first_file.py
This is my first Python file
$
```

<Time to Code>





Activity: Terminal/Git-Bash

In this activity, you will now dive into the terminal, create three folders, and a pair of Python files which will print some strings of their own creation to the console.

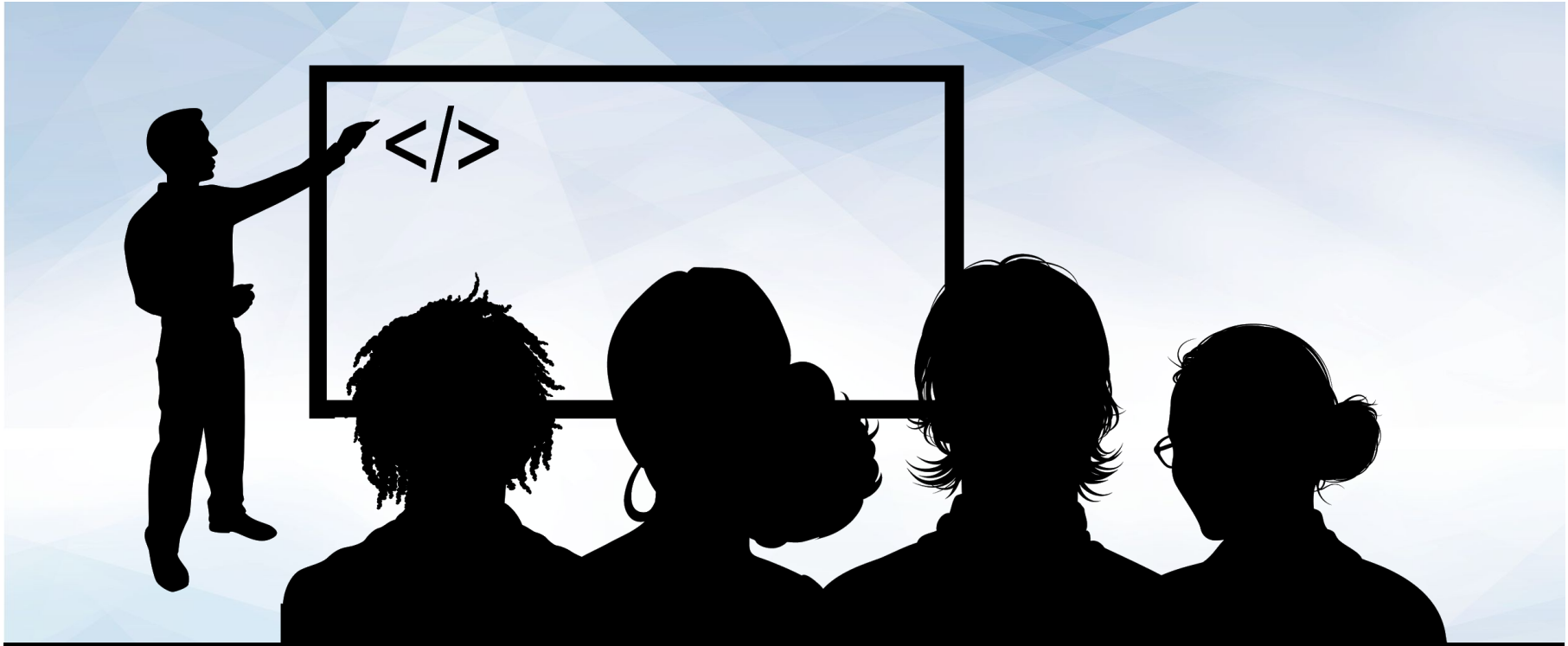
Suggested Time:
10 Minutes

Instructions: Terminal

Follow along with these instructions in your terminal and write the commands below:

- Create a folder called `LearnPython`.
- Navigate into the folder.
- Inside `LearnPython` create another folder called `Assignment1`.
- Inside `Assignment1` create a file called `quick_python.py`.
- Add a print statement to `quick_python.py`.
- Run `quick_python.py`.
- Return to the `LearnPython` folder.
- Inside `LearnPython` create another folder called `Assignment2`.
- Inside `Assignment2` create a file called `quick_python2.py`.
- Add a different print statement to `quick_python2.py`.
- Run `quick_python2.py`.

Time’s Up! Let’s Review.



Instructor Demonstration

Check Anaconda installation

Check Anaconda Installation

01

Open up your terminal

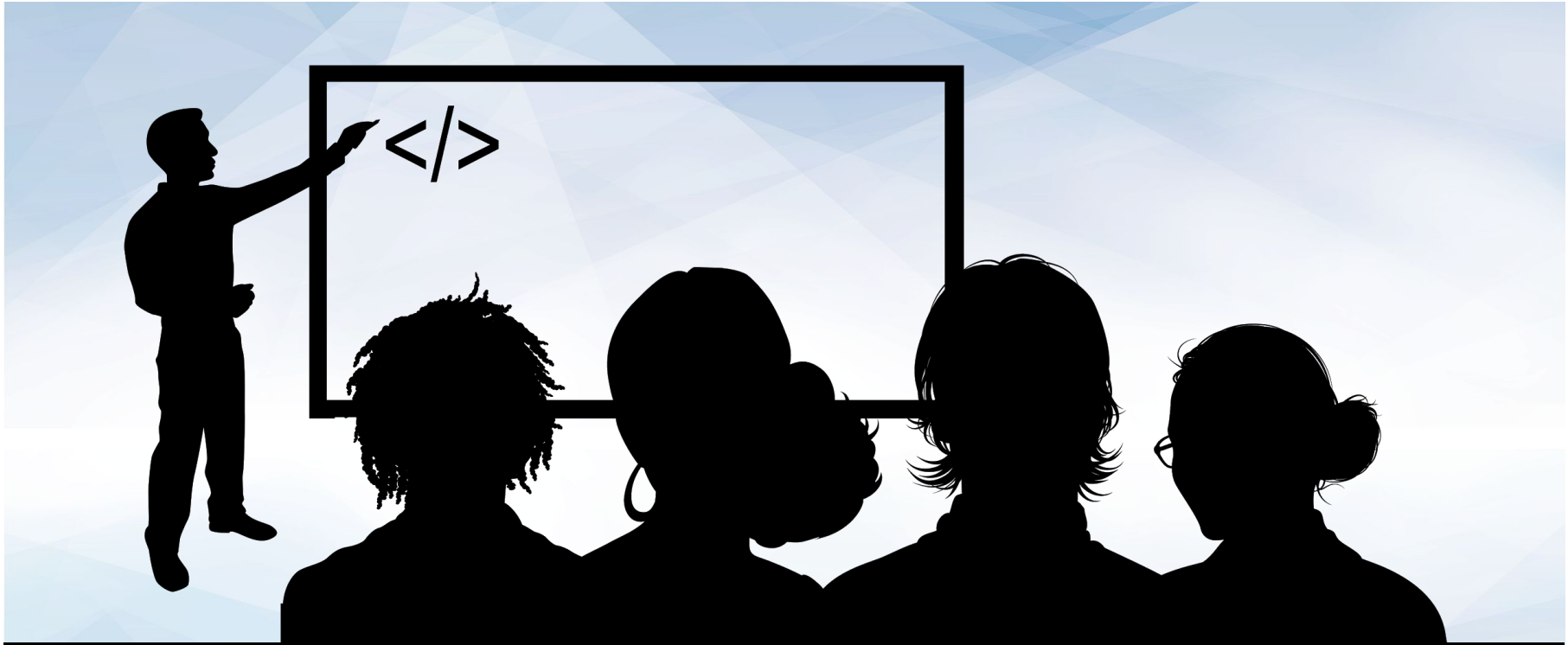
02

Type *conda --version* into your terminal then hit enter to run the command.

03

Check terminal output to make sure a version is displayed.

```
$ conda --version
conda 4.3.24
```



Instructor Demonstration

Creating a Virtual Environment

What is a virtual environment?

- Virtual environments create an isolated environment for Python projects.
- You may be working on different projects that have different dependencies.
- Different projects might also use different types and versions of libraries.
- This virtual environment will make sure the class has all the right dependencies for future class activities.
- Check the documentation for more information on managing environments at https://conda.io/docs/user-guide/tasks/manage-environments.html

Creating a virtual environment

01 Create environment

Run `conda create -n PythonData python=3.6 anaconda` in terminal/git-bash. Note that this will take a few minutes.

```
$ conda create -n PythonData python=3.6 anaconda
```

02 Enter the environment

Run `conda activate PythonData` or if that doesn't work try `source activate PythonData` in your terminal/git-bash.

When you see the prompt start with `(PythonData)$` you will now be in the environment.

```
$ conda activate PythonData
```

```
$ source activate PythonData
(PythonData) $
```

Creating a virtual environment cont.

01 Double check Python version.

Inside your environment run the command `python --version`

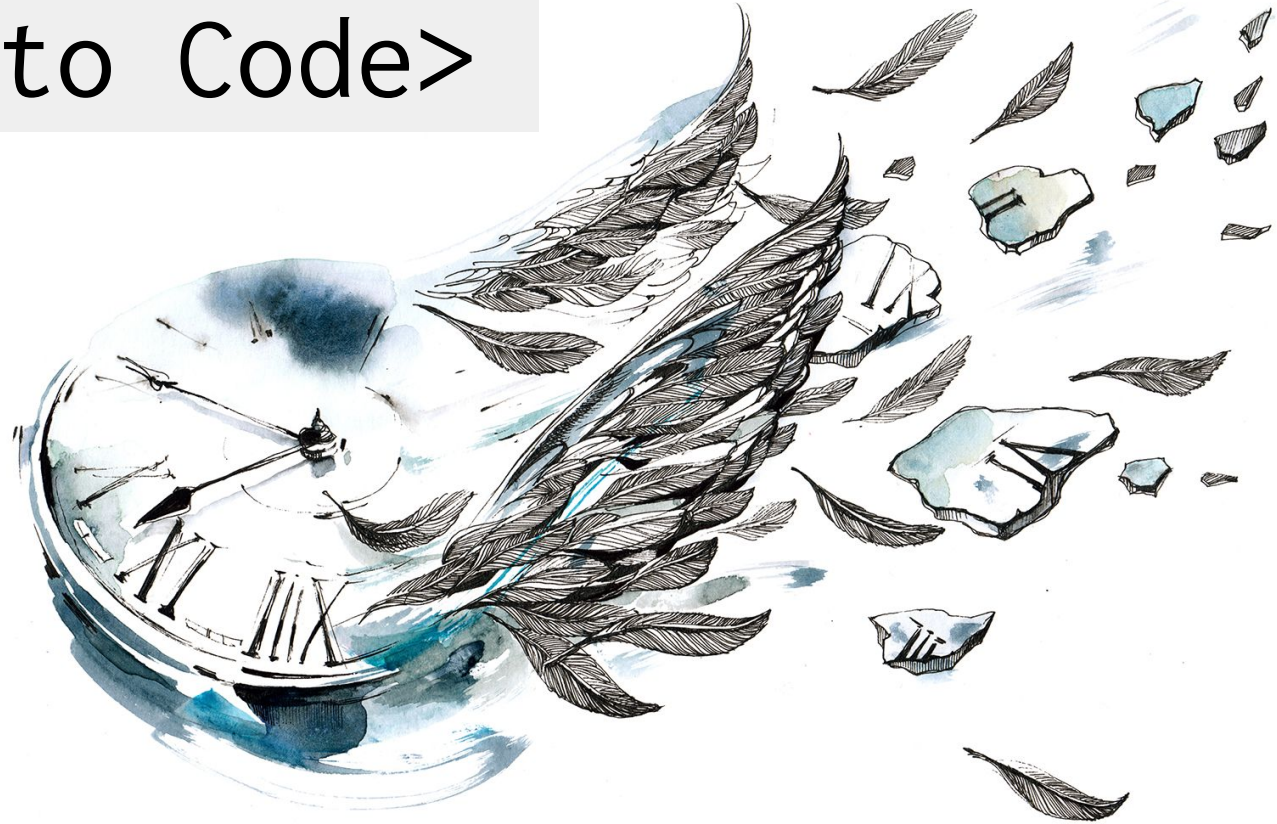
```
(PythonData) $ python --version
Python 3.6.8 :: Anaconda, Inc.
(PythonData) $
```

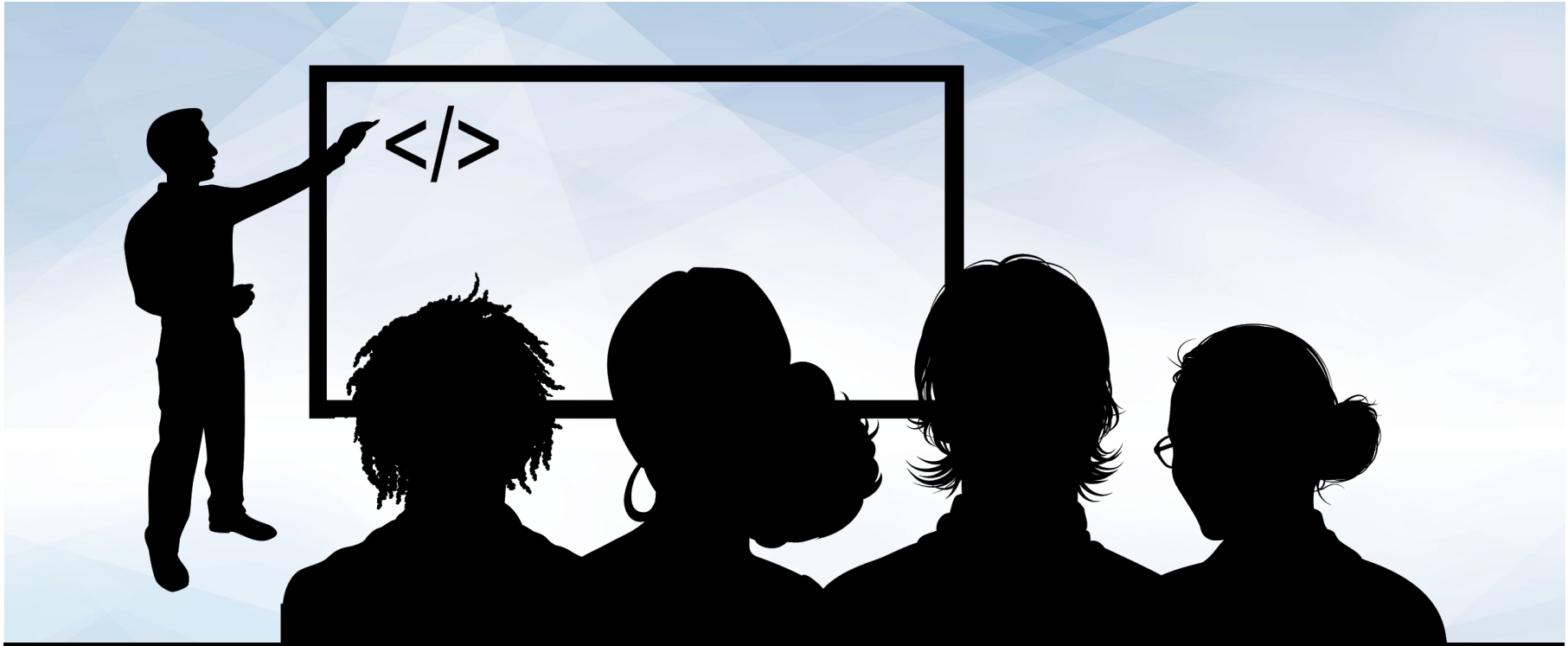
02 Exit the environment

You can exit the environment by entering `source deactivate`. If `source deactivate` does not work, try using `conda deactivate` instead. `(PythonData) $` Will no longer be displayed indicating you are no longer in the environment.

```
(PythonData) $ conda deactivate
$
```

<Time to Code>





Instructor Demonstration

Variables

Variables

- Remember in VBA, it accessed certain values when they referred to a specific cell. This is essentially what a variable is doing in Python, a value is being stored there.
- Variables can store different data types like strings, integers and an entirely new data type called booleans which hold `True` or `False` values.

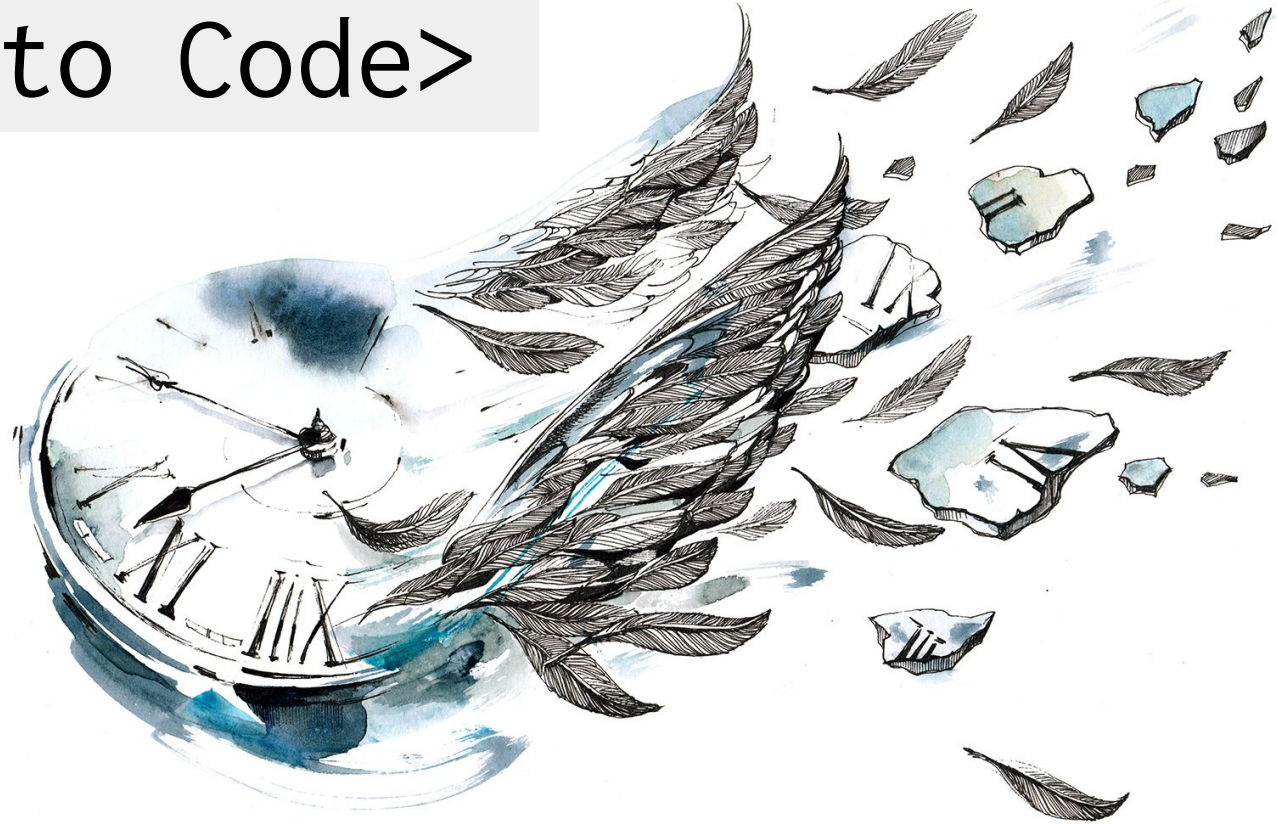
```
# Creates a variable with a string "Frankfurter"
title = "Frankfurter"

# Creates a variable with an integer 80
years = 80

# Creates a variable with the boolean value of True
expert_status = True
```

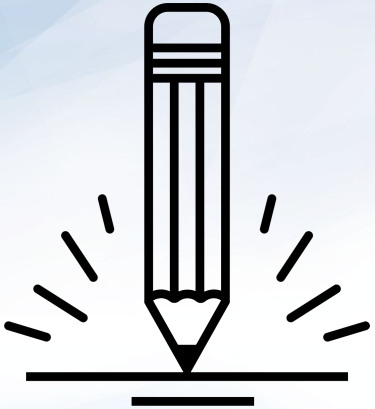
Printing Variables

- Print statements can include variables, but traditional Python formatting won't concatenate strings with other data types.
- This means integers and booleans must be cast as strings using the str() function.
- Alternatively, the "f-string" method of string interpolation allows strings to be formatted with different data types.

```
# Prints a statement adding the variable
print("Nick is a professional " + title)

# Convert the integer years into a string and prints
print("He has been coding for " + str(years) + " years")

# Converts a boolean into a string and prints
print("Expert status: " + str(expert_status))

# An f-string accepts all data types without conversion
print(f"Expert status: {expert_status}")
```

<Time to Code>





Activity: Hello Variable World!

In this activity, you will now create a simple Python application that uses variables. It will both run calculations on integers and print strings out to the console.

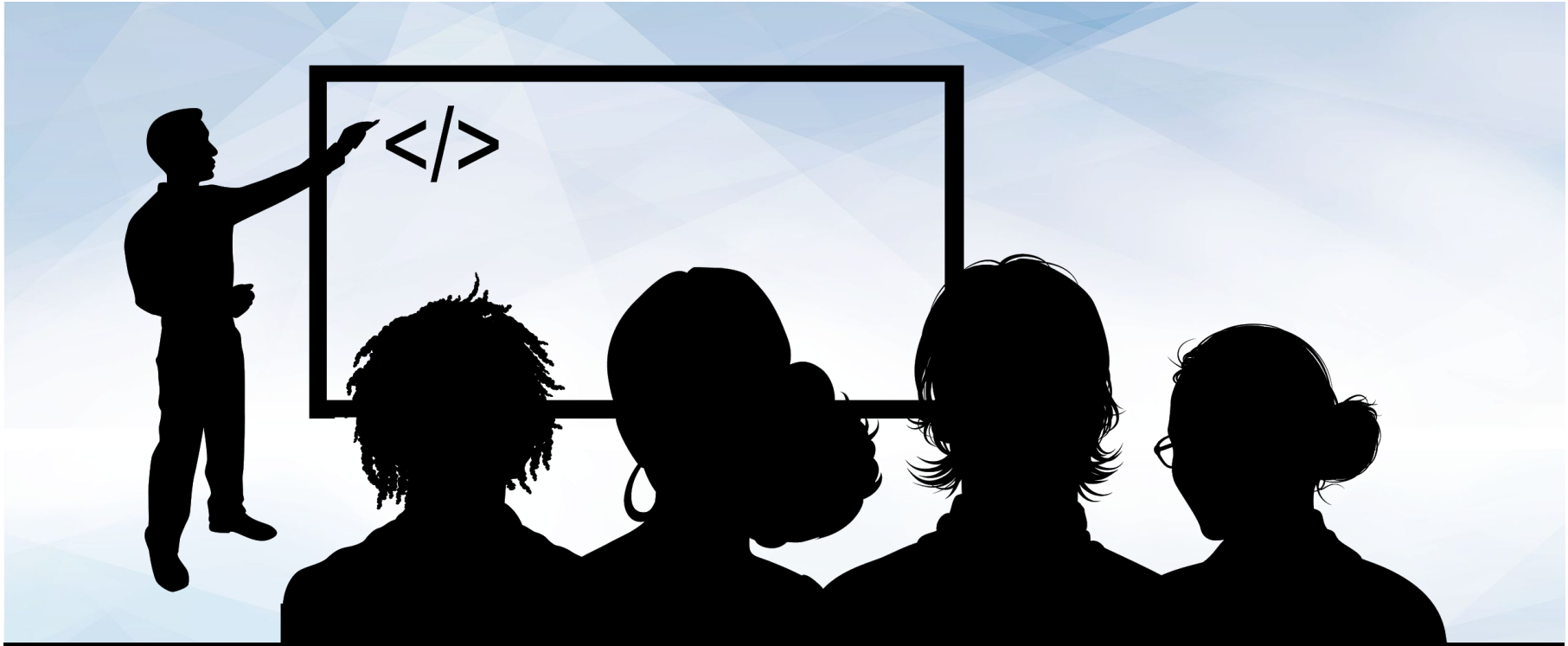
Suggested Time:
10 Minutes

Instructions: Hello Variable World!

- Use VS code to write your code and terminal/git-bash to run it.
- Create two variables called `name` and `country` that will hold strings.
- Create two variables called `age` and `hourly_wage` that will hold integers.
- Create a variable called `satisfied` which will hold a boolean.
- Create a variable called `daily_wage` that will hold the value of `hourly_wage` multiplied by 8.
- Print out statements using all of the above variables to the console.

```
$ python HelloVariableWorld.py
Hello George Washington!
You live in United States
You are 287 years old
You make 120 per day
Are you satisfied with your current wage? True
```

Time’s Up! Let’s Review.



Instructor Demonstration

Inputs and Prompts

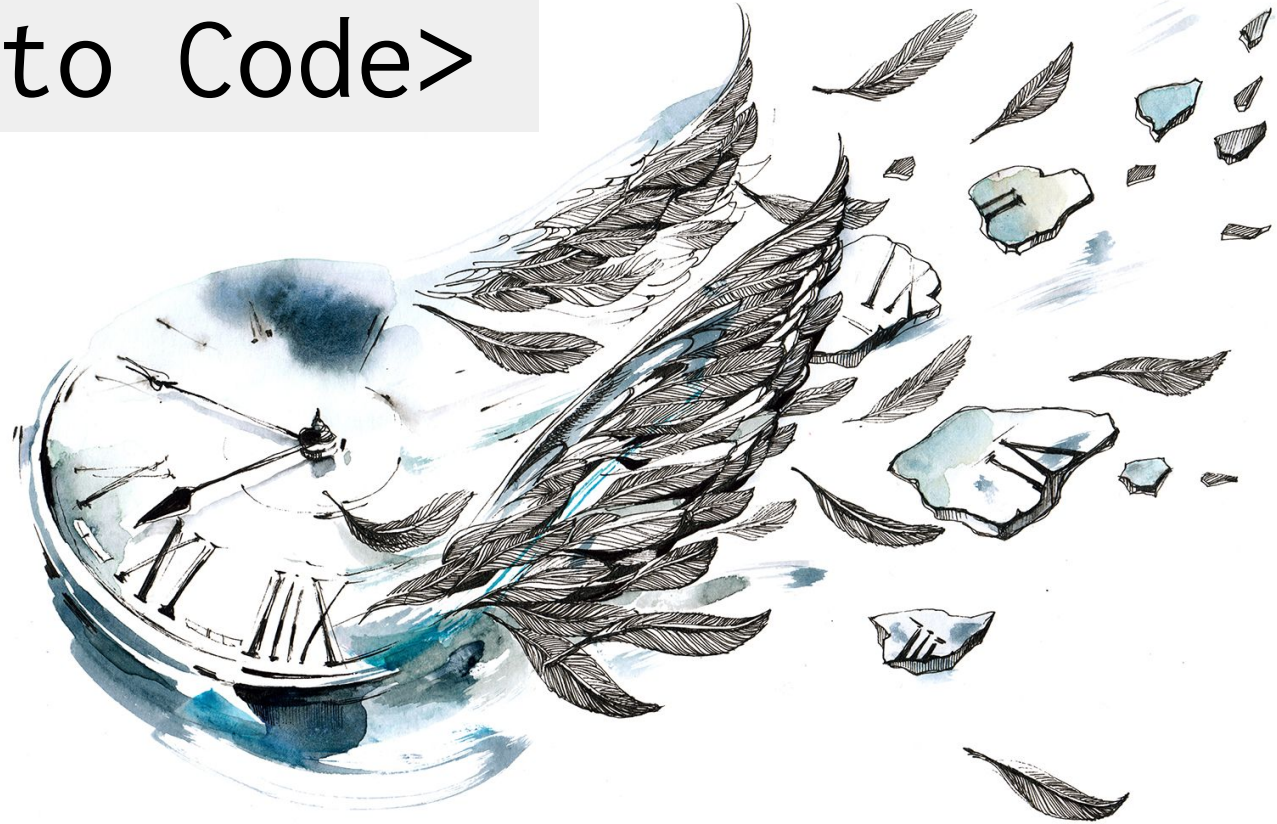
Inputs and Prompts

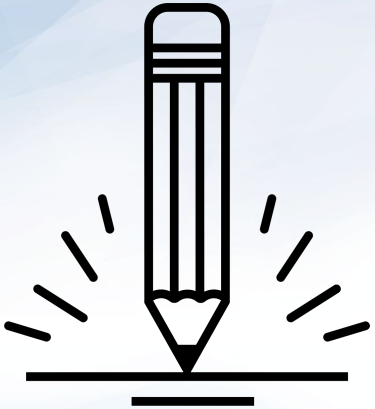
- Python allows you to take input from prompts in the terminal.
- Inputs can be stored in variables.
- Every input is considered a string.
- The bool() function will return True if any input is given

```
$ python inputs.py
What is your name? George
How old are you? 287
Is the input truthy? True
My name is George
I will be 288 next year.
The input was converted to True
```

```python
# Collects the user's input for the prompt "What is your name?"
name = input("What is your name? ")

# Collects the user's input for the prompt "How old are you?"
# and converts the string to an integer.
age = int(input("How old are you? "))

# Collects the user's input for the prompt "Is input truthy?"
# and converts it to a boolean. Note that non-zero,
#  non-empty objects are truth-y.
trueOrFalse = bool(input("Is the input truthy? "))

# Creates three print statements that to respond with the output.
print("My name is " + str(name))
print("I will be " + str(age + 1) + " next year.")
print("The input was converted to " + str(trueOrFalse))
```

<Time to Code>





Activity: Down to Input

In this activity, you will work on storing inputs from the command line and run some code based upon the values entered.

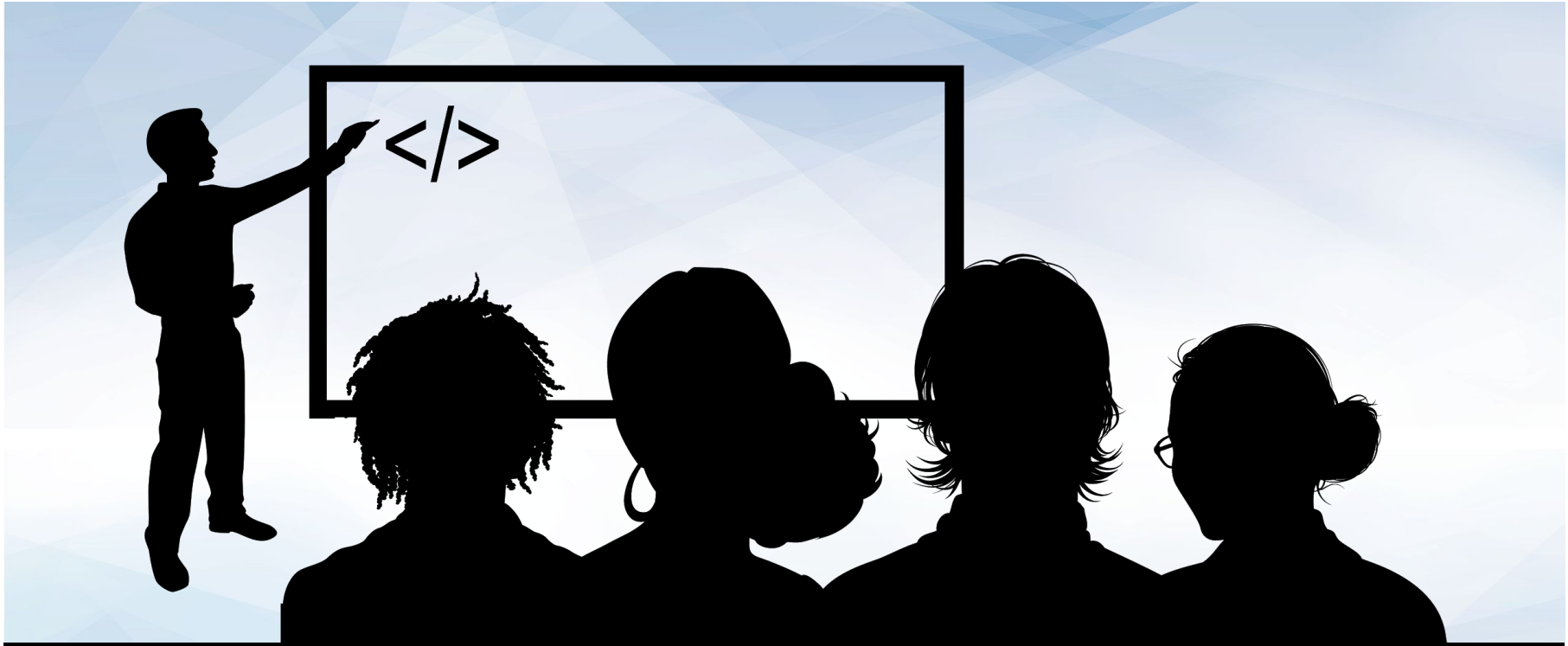
Suggested Time:
7 Minutes

Instructions: Down to Input

- Create two different variables that will take the input of your first name and your neighbor's first name.
- Create two more inputs that will ask how many months each of you has been coding.
- Finally, display a result with both your names and the total amount of months coding.

```
$ python DownToInput.py
What is your name? Jane Doe
What is your neighbor's name? Bob Smith
How many months have you been coding? 2
How many months has your neighbor been coding? 1
I am Jane Doe and my neighbor is Bob Smith
Together we have been coding for 3 months!
```

Time’s Up! Let’s Review.



Instructor Demonstration

Conditionals

Conditionals

- Python uses if, elif, and else for creating conditionals.
- Statement are concluded with a colon but all code contained in the conditional **must** be indented.

```
x = 1
y = 10

# Basic if statement that checks if one value is equal to another
# Very important to note that indentation matters in Python!
if(x == 1):
  print("x is equal to 1")
```

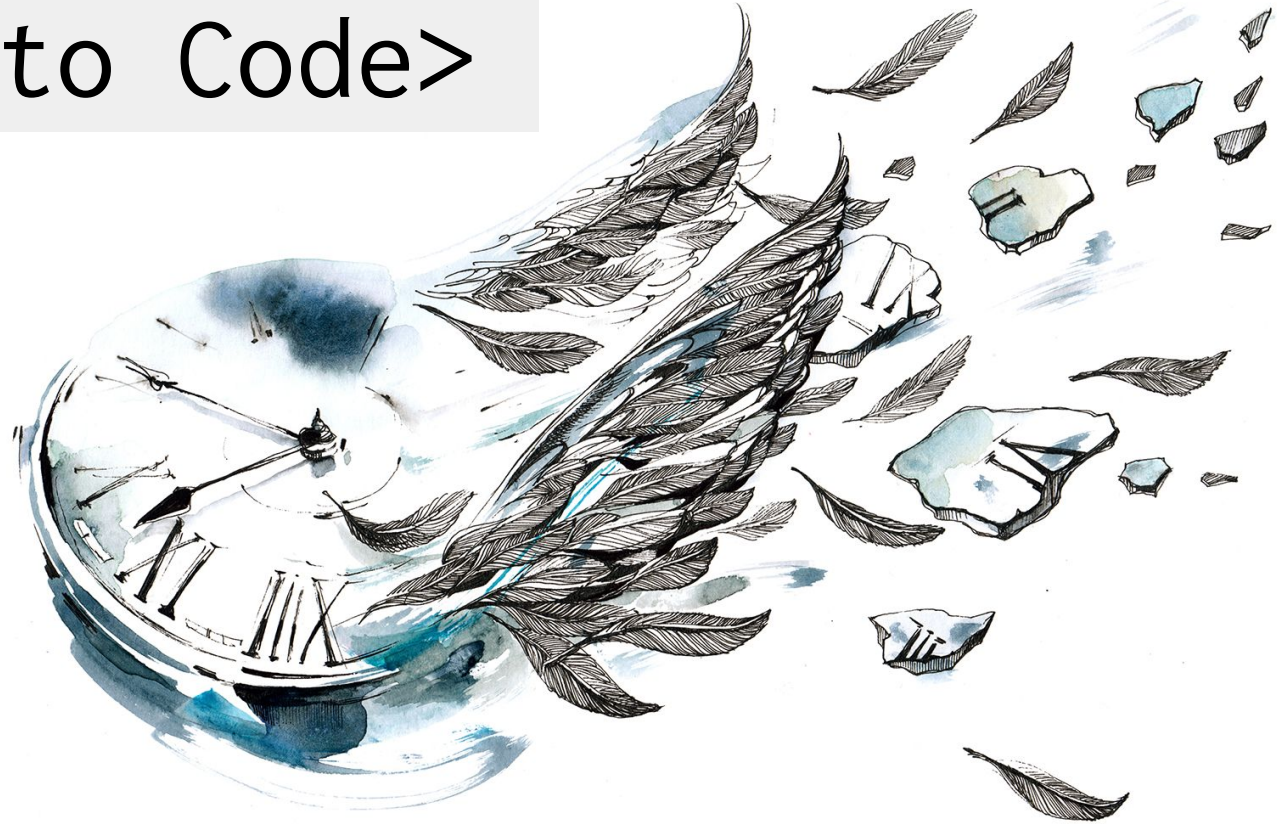
Indent

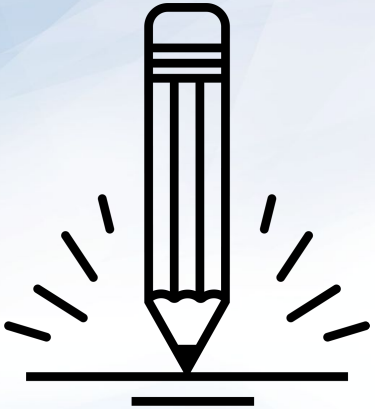
Conditionals cont.

- All sets of operators can be used like greater than and less than.
- Is equal to uses ==.
- The term and can be used to check if multiple conditions are True.
- The term or can used to check if at least condition is True.
- Conditionals can be nested.

```python
# Checks if one value is equal to another
if(x == 1):
  print("x is equal to 1")

# Checks if one value is NOT equal to another
if(y != 1):
  print("y is not equal to 1")

# Checks if one value is less than another
if(x < y):
  print("x is less than y")

# Checks if one value is greater than another
if(y > x):
  print("y is greater than x")

# Checks if a value is less than or equal to another
if(x >= 1):
  print("x is greater than or equal to 1")

# Checks for two conditions to be met using "and"
if(x == 1 and y == 10):
  print("Both values returned true")

# Checks if either of two conditions is met
if(x < 45 or y < 5):
  print("One or the other statements were true")

# Nested if statements
if(x < 10):
  if(y < 5):
    print("x is less than 10 and y is less than 5")
  elif(y == 5):
    print("x is less than 10 and y is equal to 5")
  else:
    print("x is less than 10 and y is greater than 5")
```

<Time to Code>





Activity: Conditional Conundrum

In this activity, you will be looking through some pre-written conditionals and attempting to figure out what lines will be printed to the console.

Suggested Time:
10 Minutes

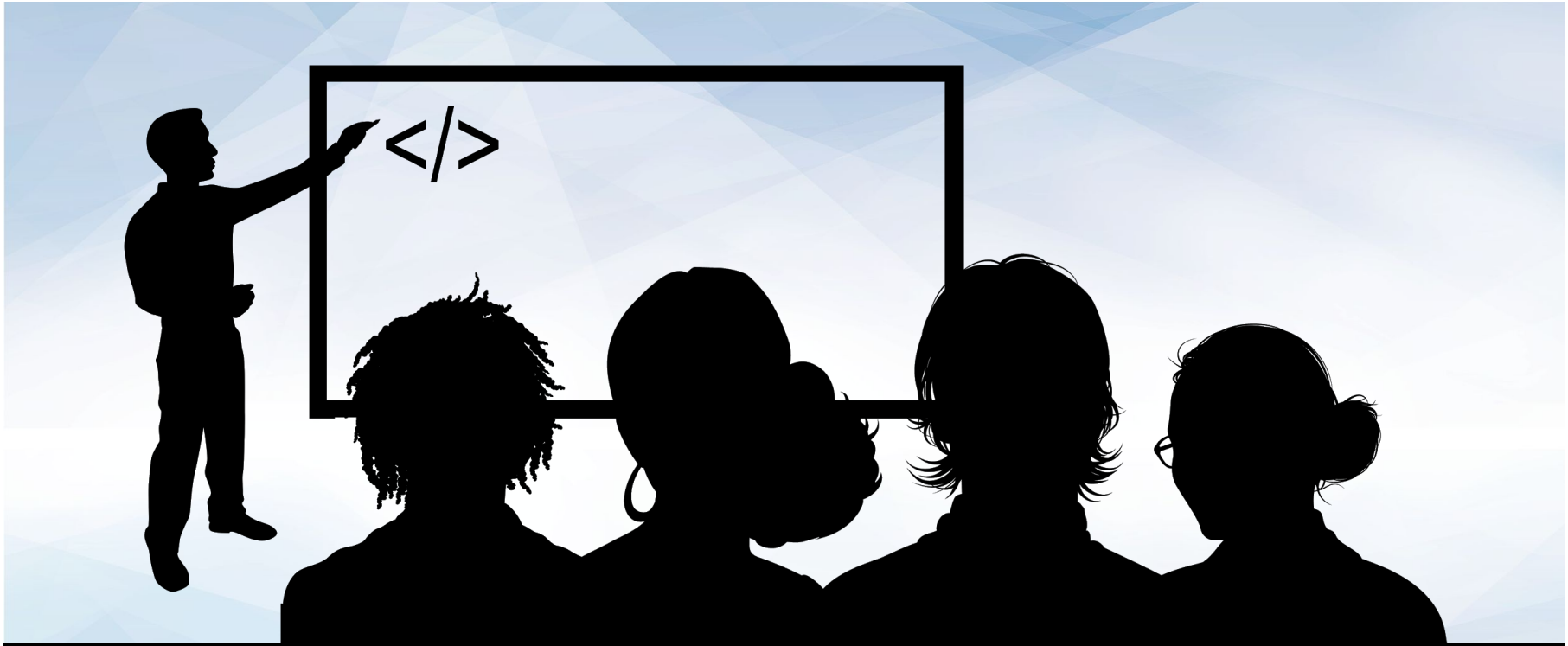
Instructions: Conditional Conundrum

- Look through the conditionals within the provided code and figure out which lines will be printed to the console.
- Do not run the application at first, see if you can follow the thought process for each chunk of code and then place a guess. Only after coming up with a guess for each section should you run the application.

Bonus

- After figuring out the output for all of the code chunks, create your own series of conditionals to test your fellow students. Once you have completed your puzzle, slack it out to everyone so they can test it.

Time’s Up! Let’s Review.



Instructor Demonstration

Lists

Lists

- Lists are the Python equivalent of arrays in VBA, functioning in much the same way by holding multiple pieces of data within one variable.
- Lists can hold multiple types of data inside of them as well. This means that strings, integers, and boolean values can be stored within a single list.

```
my_list = ['Data', 3000, True, 4.3]
```

Lists Methods

- Append
- Index
- Len
- Remove
- Pop
- **Remember** lists start at 0

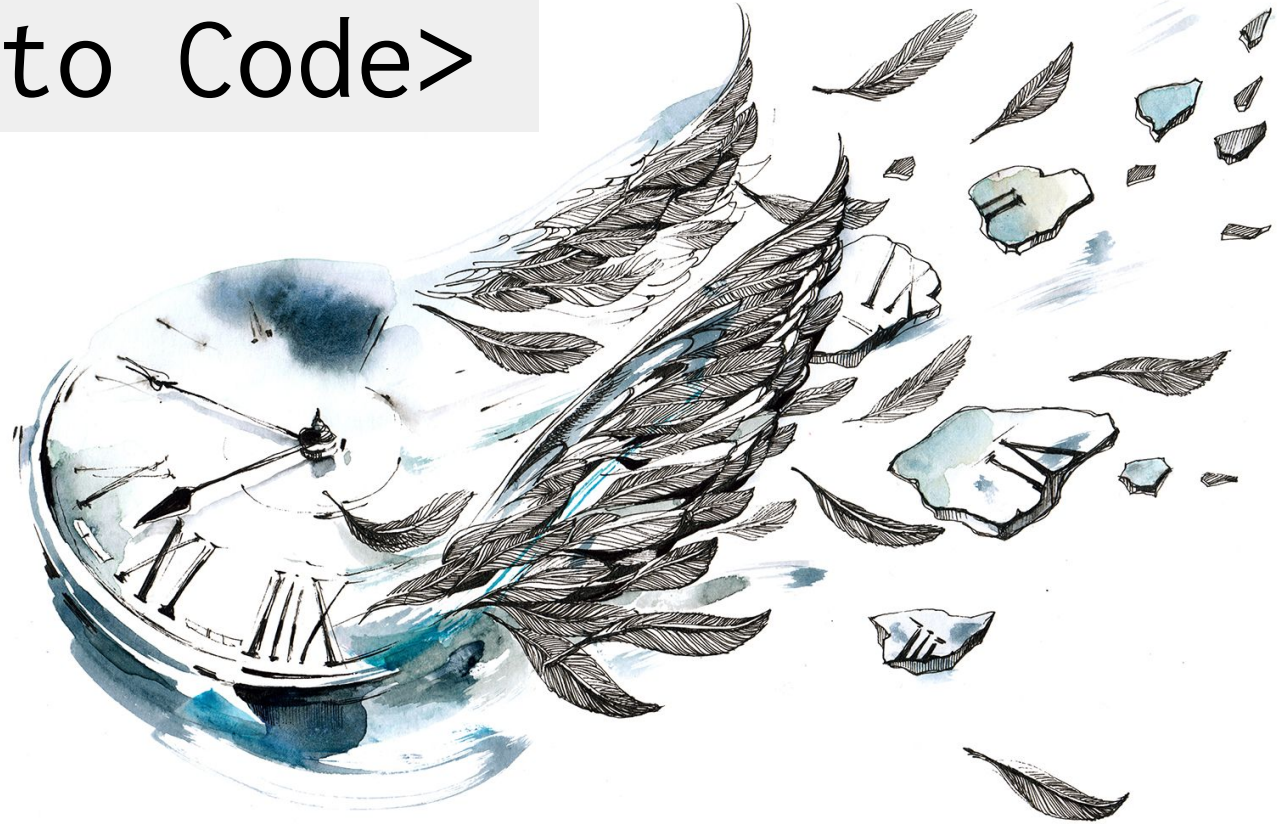
```
# Create a variable and set it as an List
myList = ["Jacob", 25, "Ahmed", 80]
print(myList)

# Adds an element onto the end of a List
myList.append("Matt")
print(myList)

# Changes a specified element within an List at the given index
myList[3] = 85
print(myList)

# Returns the index of the first object with a matching value
print(myList.index("Matt"))

# Returns the length of the List
print(len(myList))

# Removes a specified object from an List
myList.remove("Matt")
print(myList)

# Removes the object at the index specified
myList.pop(0)
myList.pop(0)
print(myList)
```

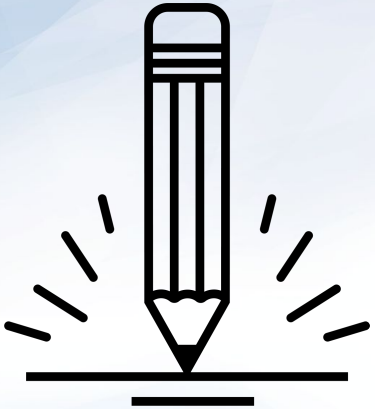
Tuples

- Lists that are immutable, or they can't be changed.
- Use parentheses instead of brackets.
- More efficient to navigate since they are read only after creation.
- Protect stored data from being changed.
- Visit https://www.quora.com/What-advantages-do-tuples-have-over-lists for more info.

```
# Creates a tuple, a sequence of immutable Python objects
# that cannot be changed
myTuple = ('Python', 100, 'VBA', False)
print(myTuple)
```

<Time to Code>





Activity: Rock, Paper, Scissors

In this activity, you will be creating a simple game of Rock, Paper, Scissors that will run within the console.

Suggested Time:
15 Minutes

Instructions: Rock, Paper, Scissors

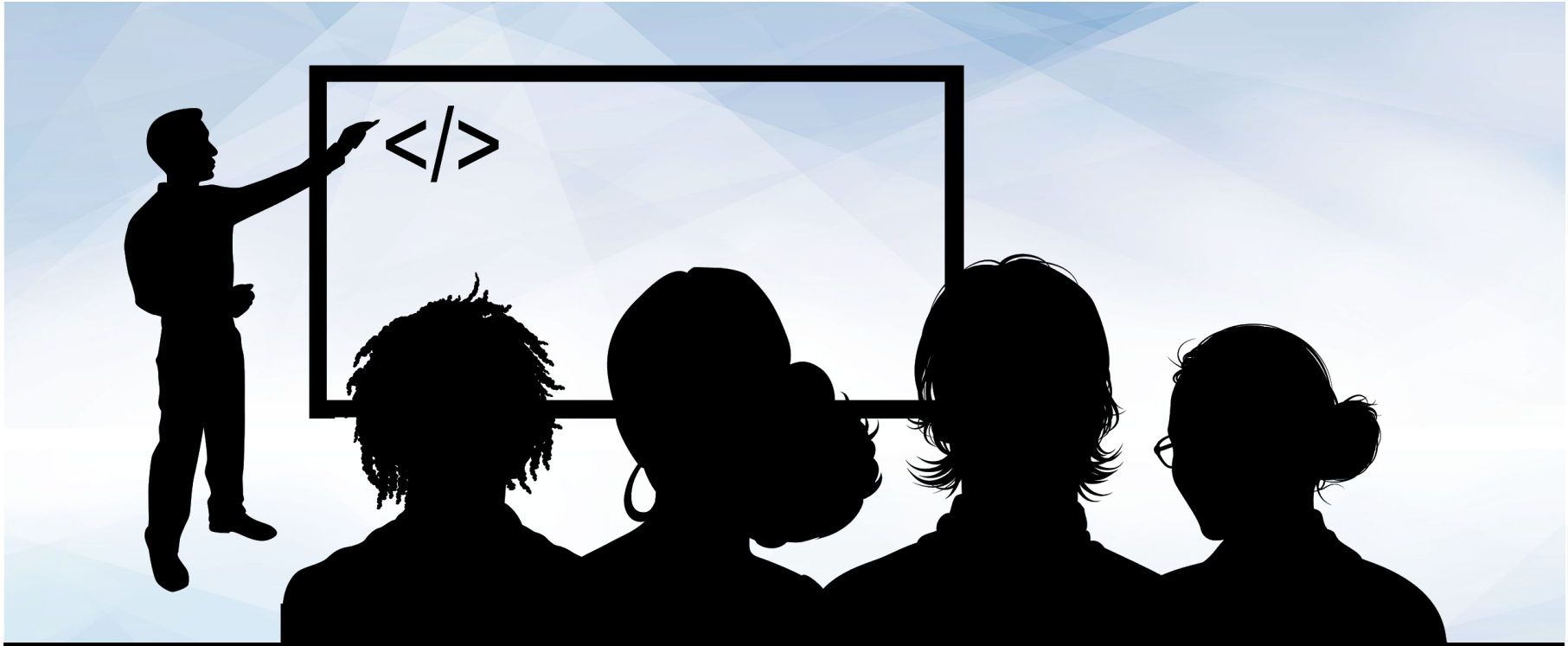
- Using the terminal, take an input of `r`, `p` or `s` which will stand for rock, paper, and scissors.
- Have the computer randomly pick one of these three choices.
- Compare the user's input to the computer's choice to determine if the user won, lost, or tied.

Hints

- Look into this https://stackoverflow.com/questions/306400/how-to-randomly-select-an-item-from-a-list question for help on using the `random` module to select a value from a list.

```
(PythonData) $ python RPS_Solved.py
Let's Play Rock Paper Scissors!
Make your Choice: (r)ock, (p)aper, (s)cissors? p
You chose paper. The computer chose rock.
Yay! You won.
```

Time’s Up! Let’s Review.



Instructor Demonstration

Loops

Loops

- x variable could theoretically be anything.
- The range method tell the loop how many times to go through.
- Stops one number before the final number.
- Given two numbers, it starts at the first and stops at the number before the second.

```
# Loop through a range of numbers (0 through 4)
for x in range(5):
  print(x)

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

# Loop through a range of numbers (2 through 6)
for x in range(2, 7):
  print(x)

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

Looping through strings

- Python can loop through the contents of a string
- letters can be any variable name but word is the variable to loop through.
- Format is for <variable> in <string or list>:
- Similar to strings and numbers, Python can loop through lists.

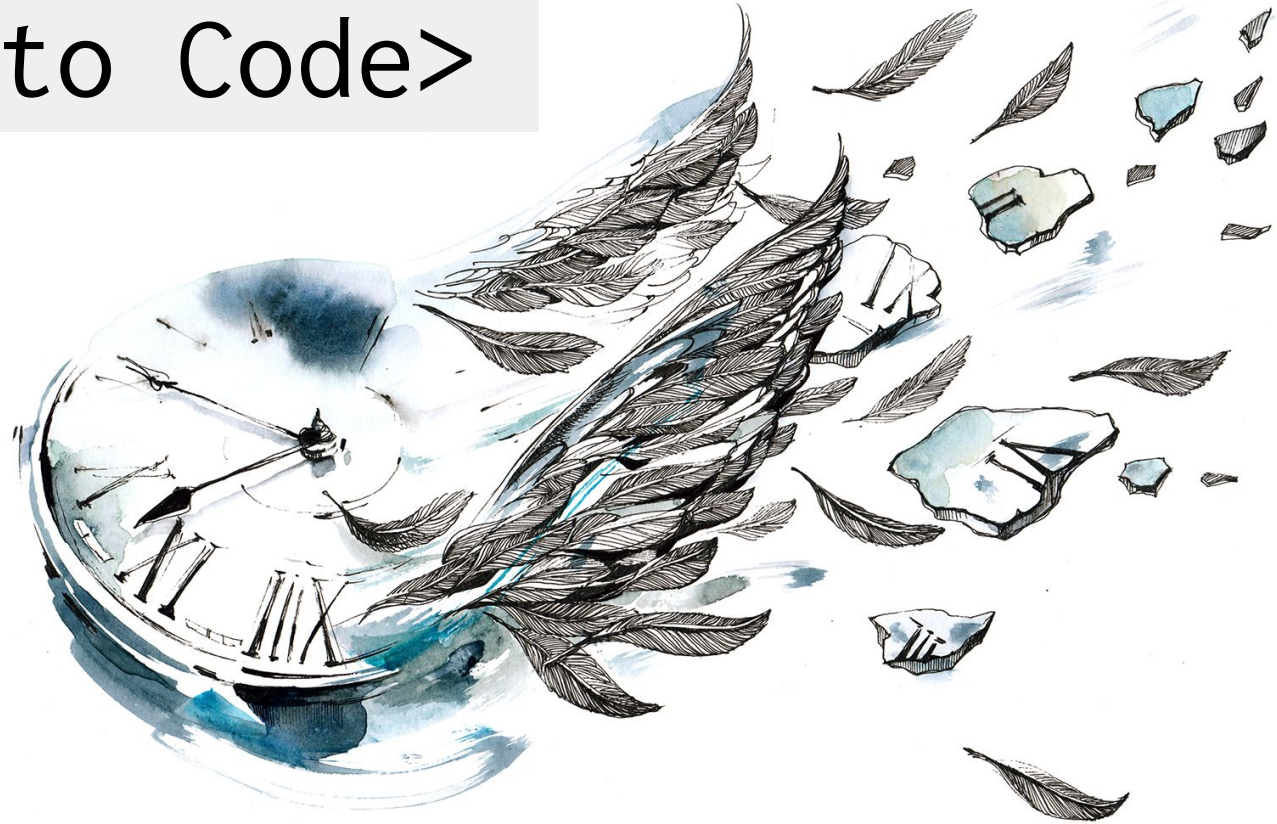
```
# Iterate through letters in a string
word = "Peace"
for letters in word:
  print(letters)

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

# Iterate through a list
zoo = ["cow", "dog", "bee", "zebra"]
for animal in zoo:
  print(animal)

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

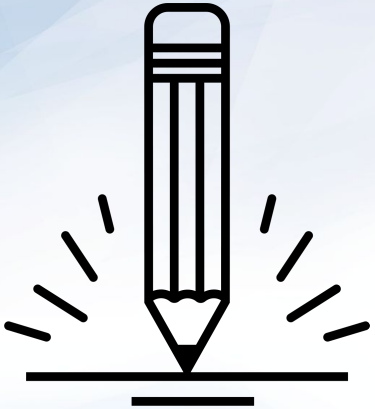
While loops

- Runs blocks of code just like a for loop.
- This time a condition must be met for the loop to stop.

```
# Loop while a condition is being met
run = "y"

while run == "y":
    print("Hi!")
    run = input("To run again. Enter 'y'" )
```

<Time to Code>





Activity: Number Chain

In this activity, you will take user input and print out a string of numbers.

Suggested Time:
14 Minutes

Instructions: Number Chain

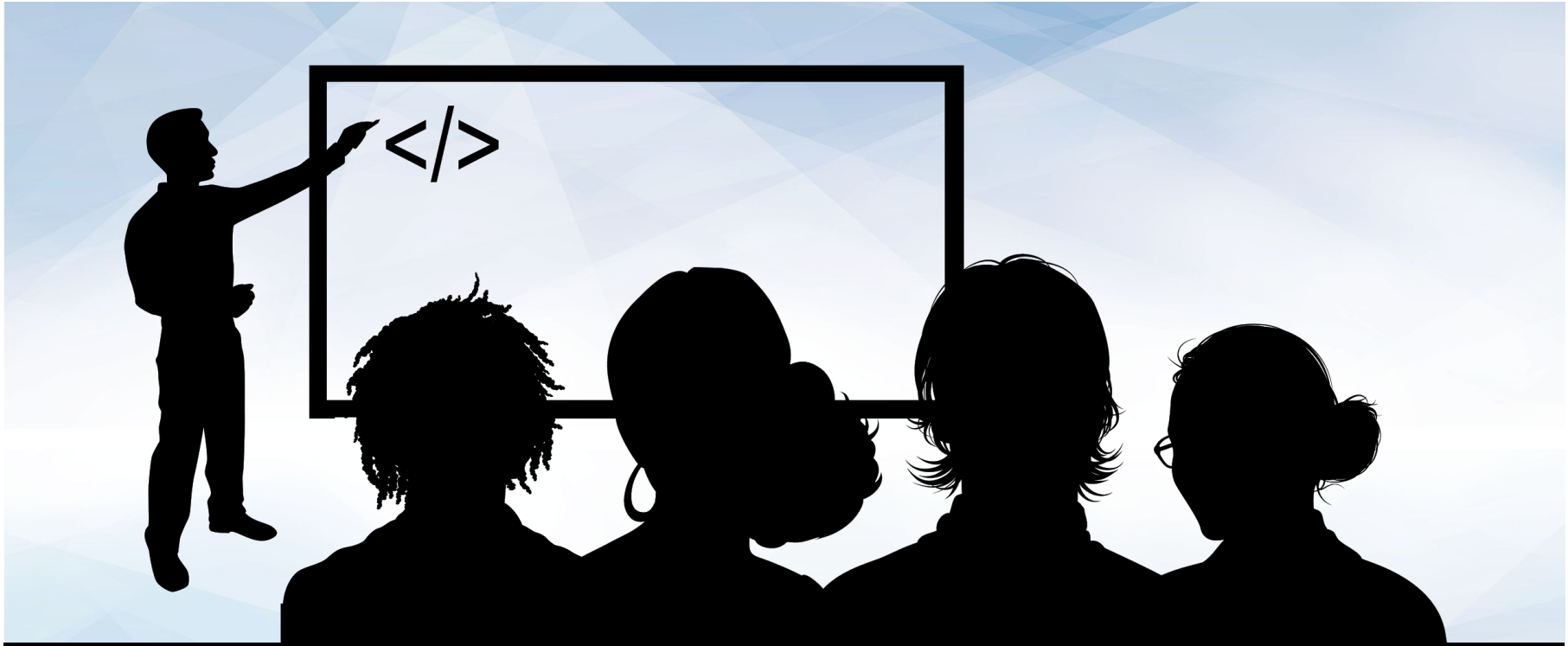
- Using a `while` loop, ask the user "How many numbers?", and then print out a chain of ascending numbers from 0 to the number input.
- After the results have printed, ask the user if they would like to continue. If "y" is entered, keep the chain running by inputting a new number and starting a new count from 0 to the number input. If "n" is entered, exit the application.

Bonus

- Rather than just displaying numbers starting at 0, have the numbers begin at the end of the previous chain.

```
$ python NumberChainBonus_Solved.py
How many numbers?
```

Time’s Up! Let’s Review.



Instructor Demonstration

Preview Homework