Let’s Begin Python

Data Boot Camp

Lesson 3.1

Class Objectives

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

Perform Python 3 installation.

Navigate through folders and files via `terminal/git-bash`.

Create Python scripts and run them in `terminal/git-bash`.

Understand basic programming concepts in Python.

The Mighty Python

The Mighty Python

Few things to note before we move forward:

01

We are diving into a more traditional programming language, Python.

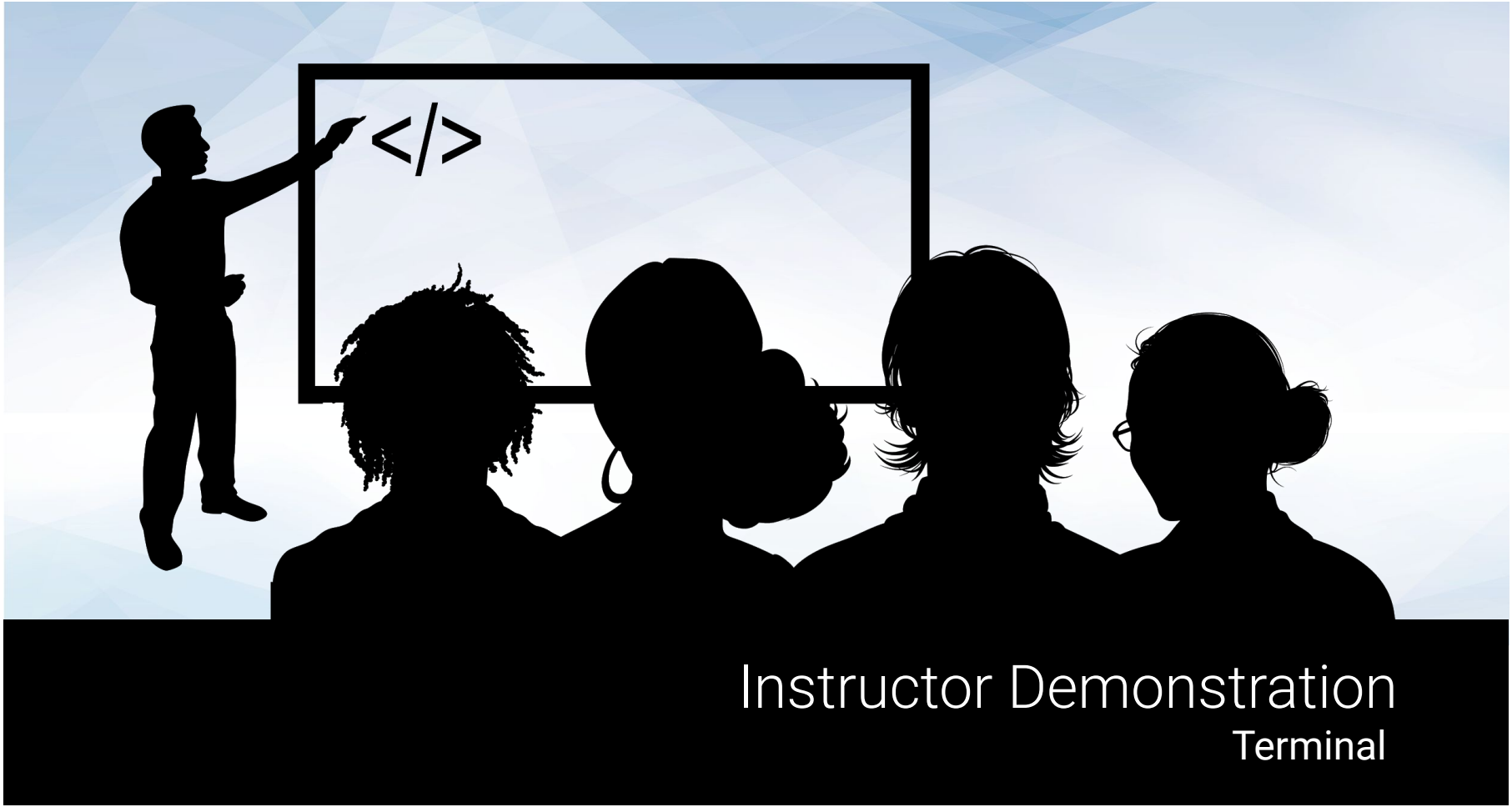
02

The fundamental concepts are still the same, and the most significant change in this transition period from VBA will be only the syntax.

03

Check your Slack.

slack



Instructor Demonstration

Terminal

Some Basic Commands

`cd`	Changes the directory
`cd ~`	Changes to the home directory
`cd ..`	Moves up one directory
`ls`	Lists files in the folder
`pwd`	Shows the current directory
`Mkdir <FOLDERNAME>`	Creates a new directory with the FOLDERNAME
`touch <FOLDERNAME>`	Creates a new file with the FILENAME
`rm <FOLDERNAME>`	Deletes a file
`rm -r`	Deletes a folder, make sure to note the -r
`open .`	Opens the current folder on Macs
`explorer .`	Opens the current folder on GitBash
`open <FILENAME>`	Opens a specific file on Macs
`explorer <FILENAME>`	Opens a specific file on GitBash

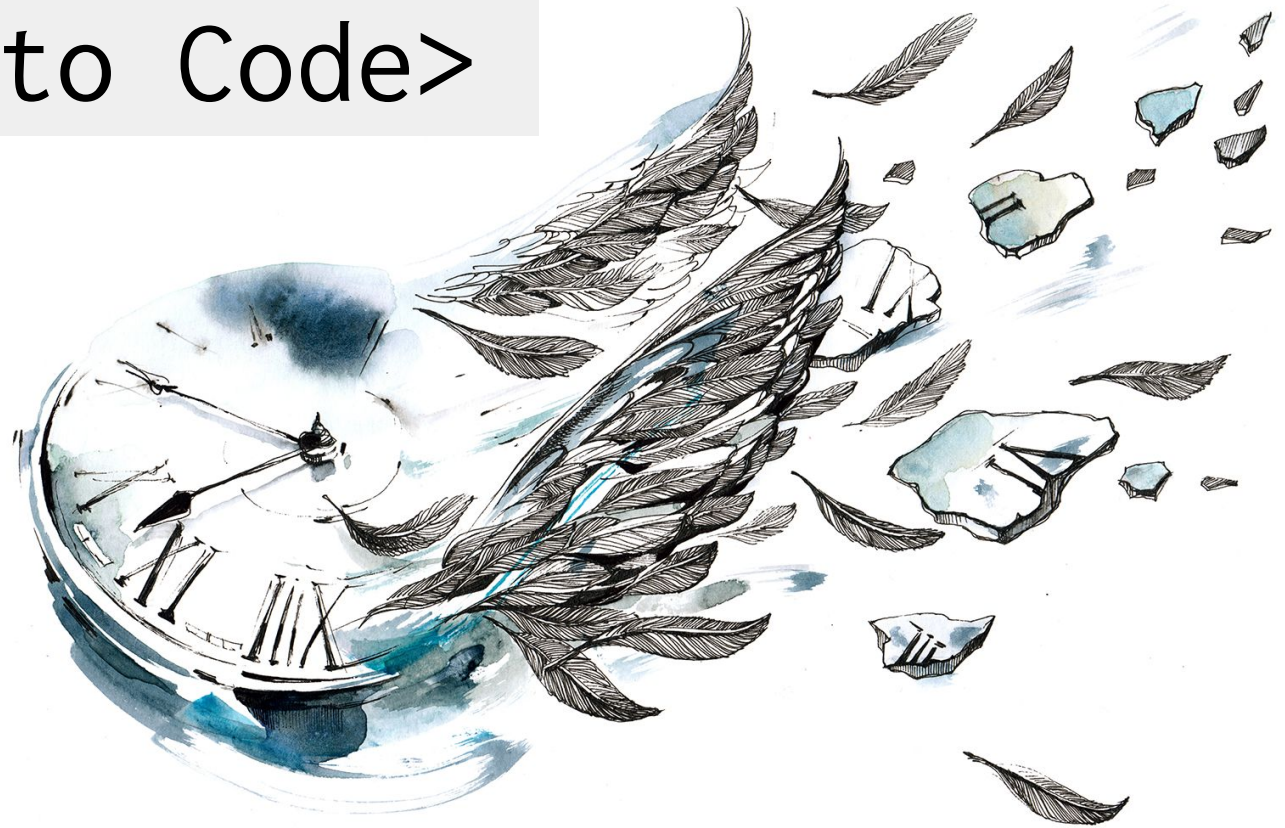
CommonCommands.txt

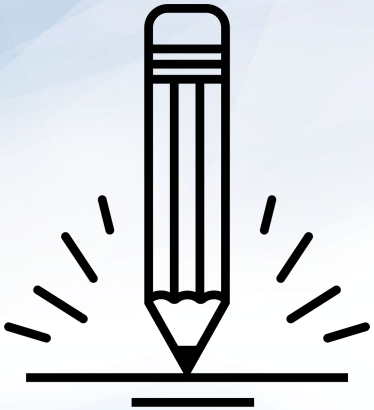
Common Commands

```
bash-3.2$ mkdir PythonStuff
bash-3.2$ cd PythonStuff
bash-3.2$ touch first_file.py
bash-3.2$ open first_file.py
```

```
bash-3.2$ python first_file.py
bash-3.2$ This is my first_file.py
```

<Time to Code>





Activity: Terminal

In this activity, you will 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.

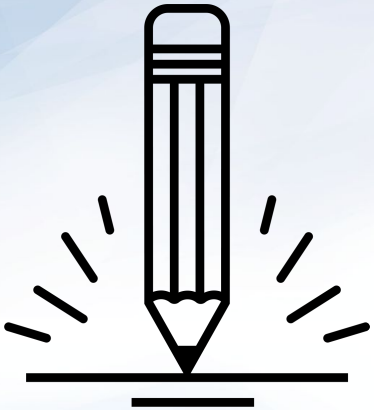
Suggested Time:
10 Minutes

Activity: Terminal

Write and execute the following commands:

- 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.



Everyone Do:

Check Anaconda Installation

In this activity, we will check if Anaconda is properly installed.

Suggested Time:
5 Minutes

Everyone Do: Check Anaconda Installation

Check if Anaconda is properly installed

01

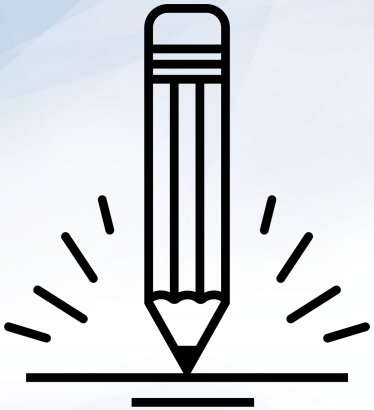
Open up terminal

02

Run `conda --version` and hit enter

03

Terminal output should return `conda 4.x.x`



Everyone Do:
Create a Virtual Environment

In this activity, we will create a virtual environment with all right dependencies for future class activities.

Suggested Time:
10 Minutes

Everyone Do: Create 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.

Everyone Do: Create a Virtual Environment

How to create a virtual environment and activate it?

01 Create environment

Run:

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

02 Activate new environment

Run:

```
source activate PythonData
```

Everyone Do: Create a Virtual Environment

Check python version in the new environment. Exit the environment.

01 Check python version

Once in the new environment run the command:

```
python --version
```

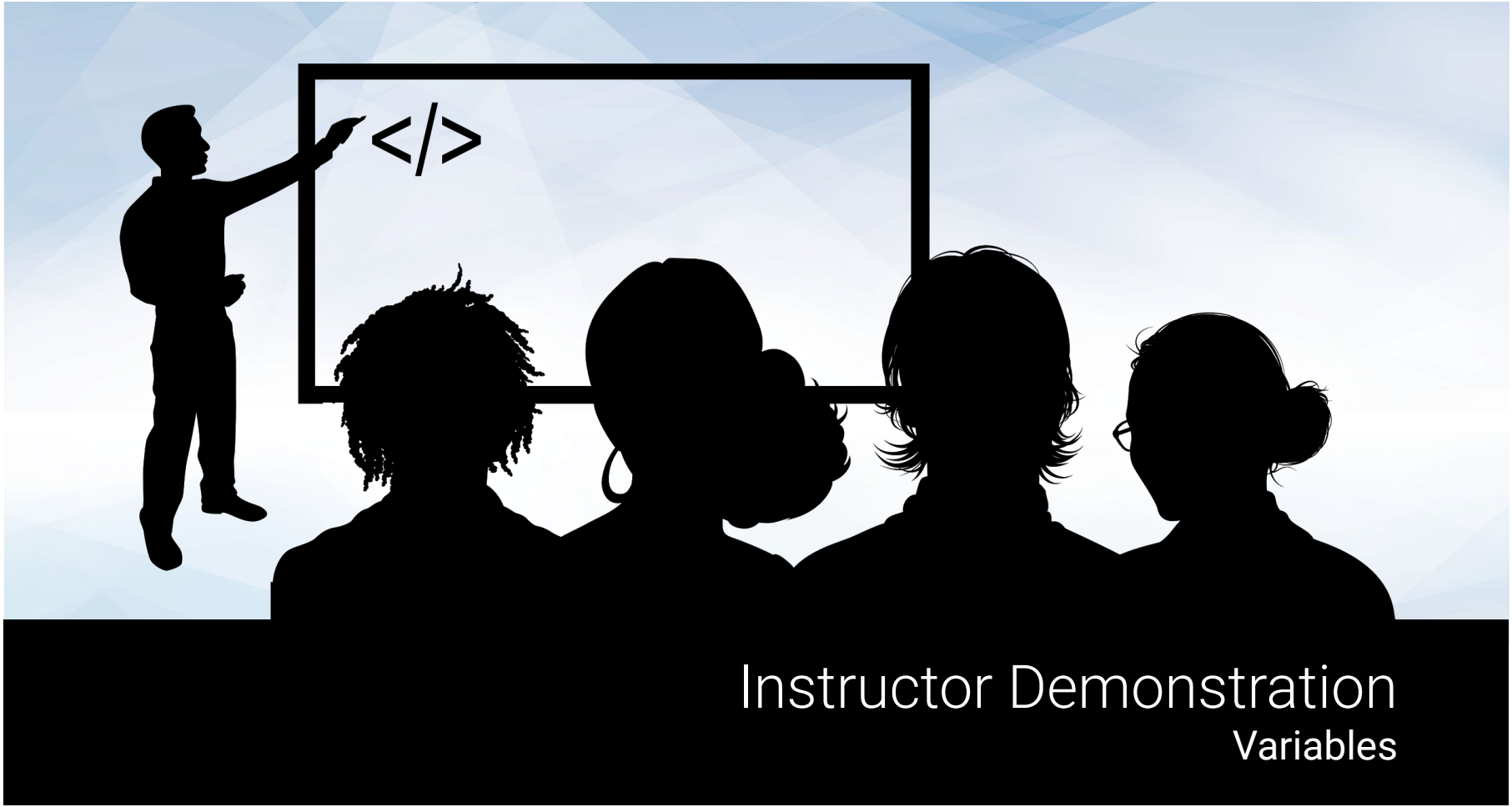
02 Deactivate new environment

Run:

```
source deactivate
```

Or:

```
conda deactivate
```



Instructor Demonstration

Variables

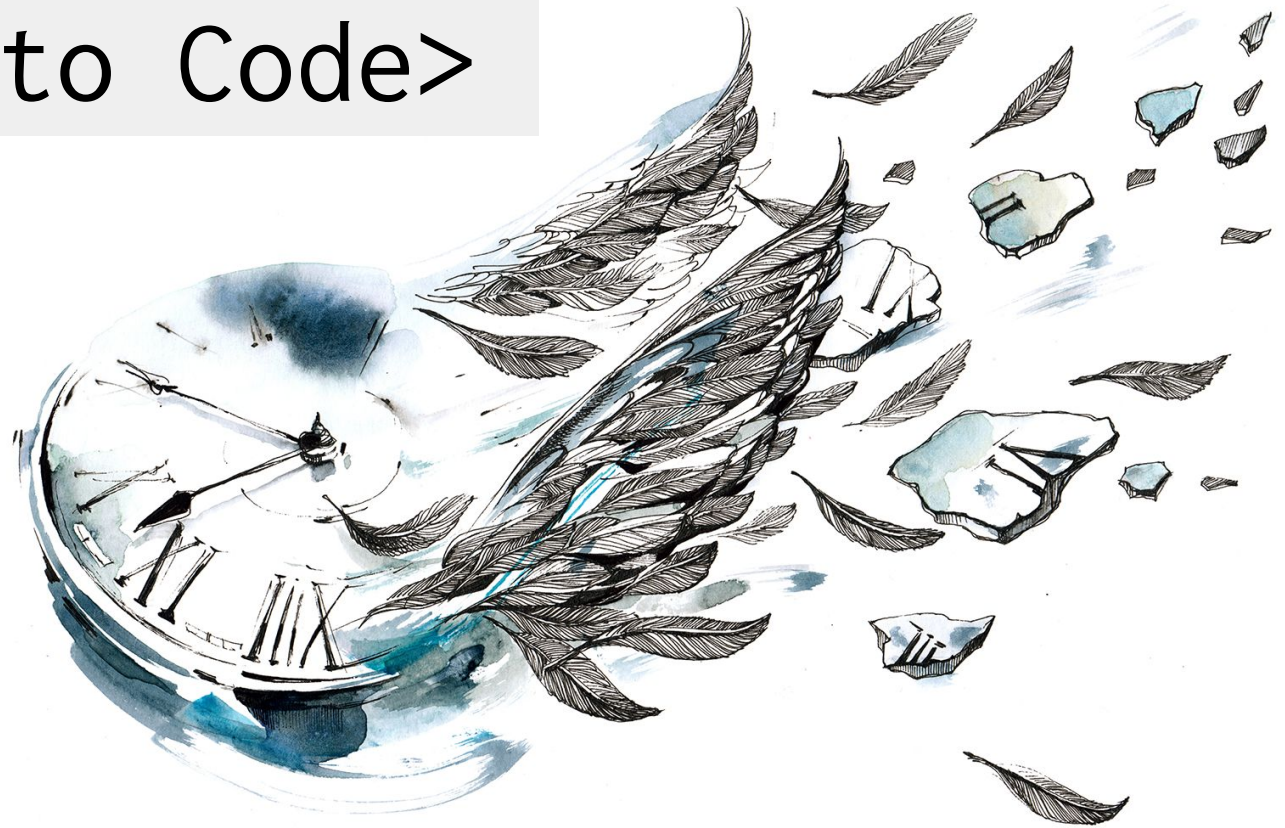
Variables

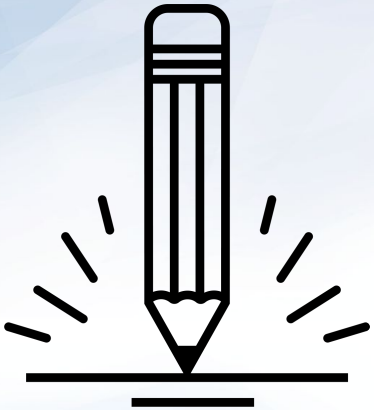
- Similar to values stored in VBA cells
- In Python, a value is being stored and given a name
- Variables can store different data types like strings, integers, and an entirely new data type called booleans which hold True or False values.

```
# 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
```

Python 3's f-Strings

<Time to Code>





Activity: Hello Variable World!

In this activity, you will create a simple Python application that uses variables.

Suggested Time:
10 Minutes

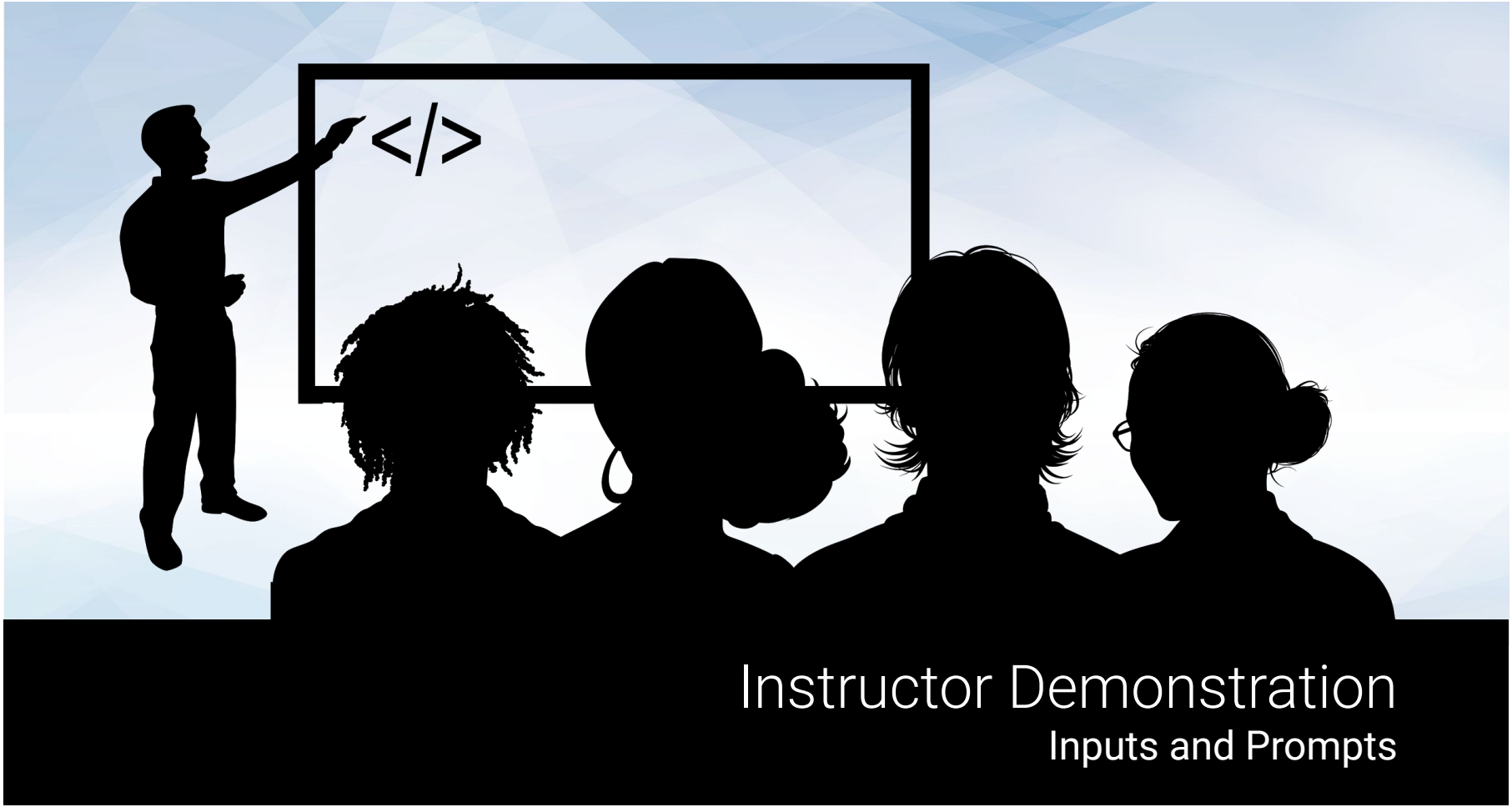
Activity: Hello Variable World!

Instructions:

- 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.

```
HelloVariableWorld.py
You live in Australia
You are 25 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

Print Statements

We can print statements which 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.

```
# 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))
```

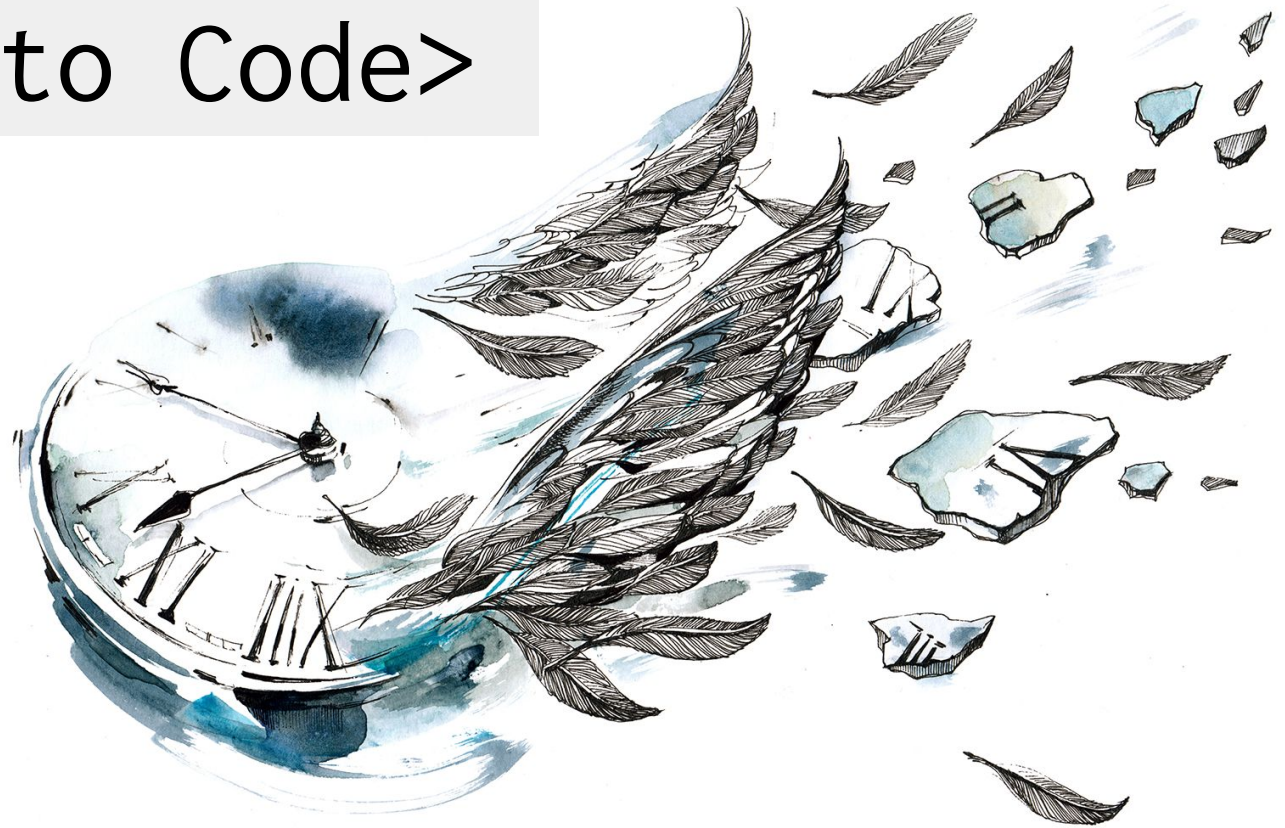
Alternatively, the '`f-string`' method of string interpolation allows strings to be formatted with different data types. Demonstrate the differences by refactoring the last print statement as an 'f-string':

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

Inputs and Prompts

```
[(PythonData) $ python inputs.py
What is your name? Gary
How old are you? 33
Is this statement true? Yes
My name is Gary
I am 33 years old.
The statement was true
```

<Time to Code>



Group 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:
10 Minutes

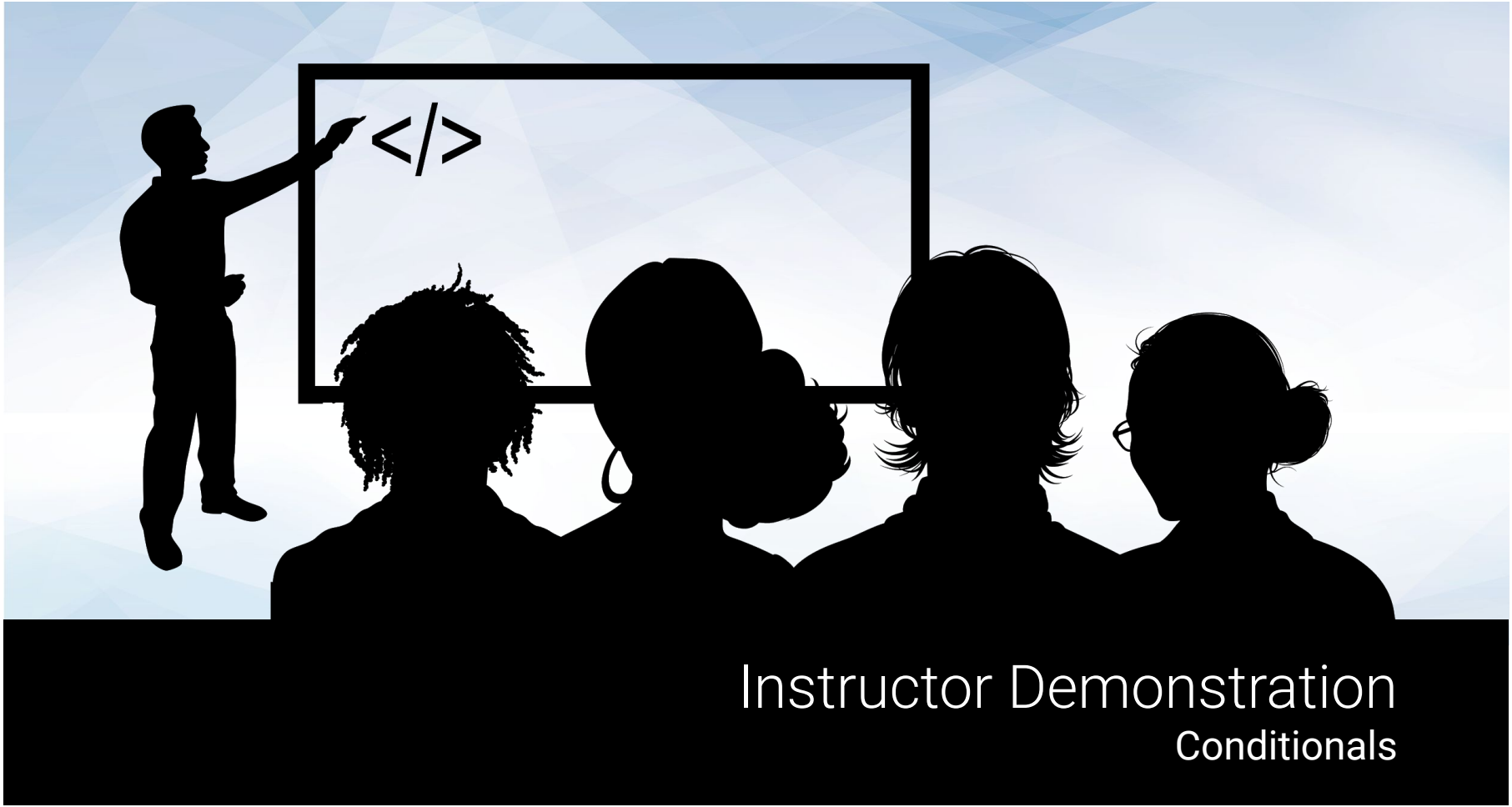
Group Activity: Down to Input

Instructions:

- Create two different variables that will take the input of your first name and your partner'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? Jacob Lee
What is your partner's name? Amelia Smith
How many months have you been coding? 24
How many months has your partner been coding? 12
I am Jacob Lee and my partner is Amelia Smith
Together we have been coding for 36 months!
```

Time’s Up! Let’s Review.



Instructor Demonstration

Conditionals

Conditionals: Few things to keep in mind

- Conditionals in Python carry nearly the same logic as in VBA. The primary difference is the syntax and indentation.
- Python uses `if`, `elif` and `else` for creating conditionals.
- Conditional statements are concluded with a colon but all lines after the colon must be indented to be considered a part of that code block. This is because Python reads blocks of code based on indentation.
- All sorts of operators like greater than, less than, equal to and much more can be used to create logic tests for conditionals.
- The condition is equal to uses == while variable assignment uses one equal sign.
- Multiple logic tests can be checked within a single conditional statement. Using the term and must mean both tests return True while or require that only one test return as true.
- Conditionals can even be nested, allowing programmers to run logic tests based upon whether or not the original logic test returned as True.

Conditionals

Indentation matters in Python!

```
>>> x = 1
>>> x = 10
>>>
>>> # Look what happens w/o indentation
... if x == 1:
... print('x is equal to 1')
  File "<stdin>", line 3
    print('x is equal to 1')
        ^
IndentationError: expected an indented block
```

```
>>> if x == 1:
```

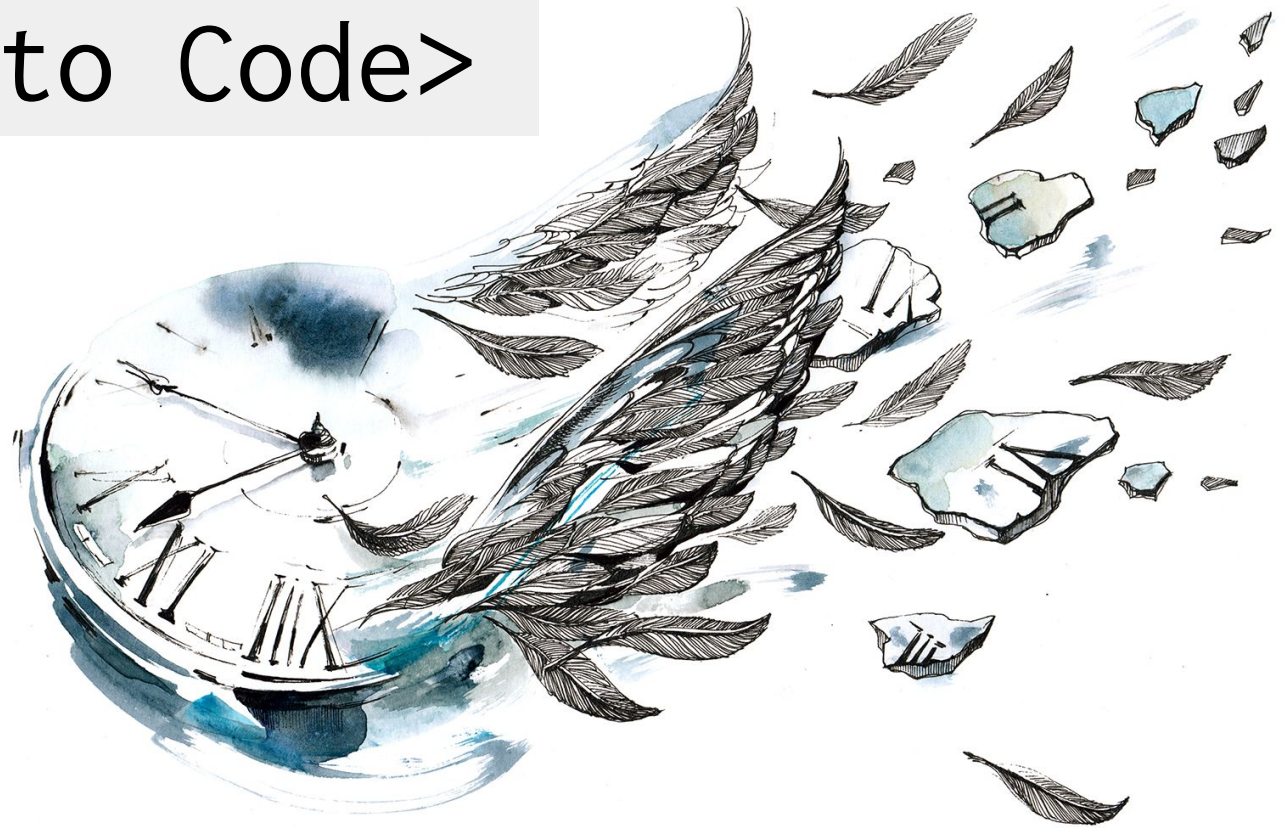
Hint: count four ‘space’ strokes on your keyboard or hit ‘tab’ once

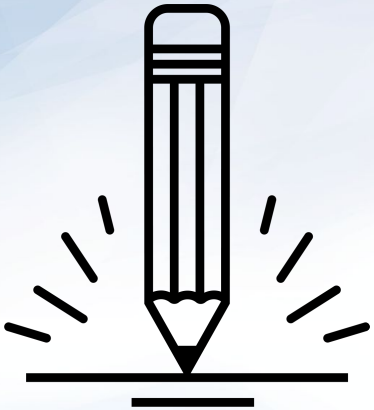
Conditionals

```
>>> # Checks if one value is equal to another
... if x == 1:
...     print("x is equal to 1")
...
x is equal to 1
>>> # Checks if one value is NOT equal to another
... if y != 1:
...     print("y is not equal to 1")
...
y is not equal to 1
>>> # Checks if one value is less than another
... if x < y:
...     print("x is less than y")
...
x is less than y
>>> # Checks if one value is greater than another
... if y > x:
...     print("y is greater than x")
...
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")
...
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")
...
Both values returned true
>>> # Checks if either of two conditions is met
... if x < 45 or y < 5:
...     print("One or more of the statements were true")
...
One or more of the 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'll look through some pre-written conditionals and attempting to figure out.

Suggested Time:
10 Minutes

Activity: Conditional Conundrum

Instructions:

01

Look through the conditionals within the provided code and figure out which lines will be printed to the console.

02

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.

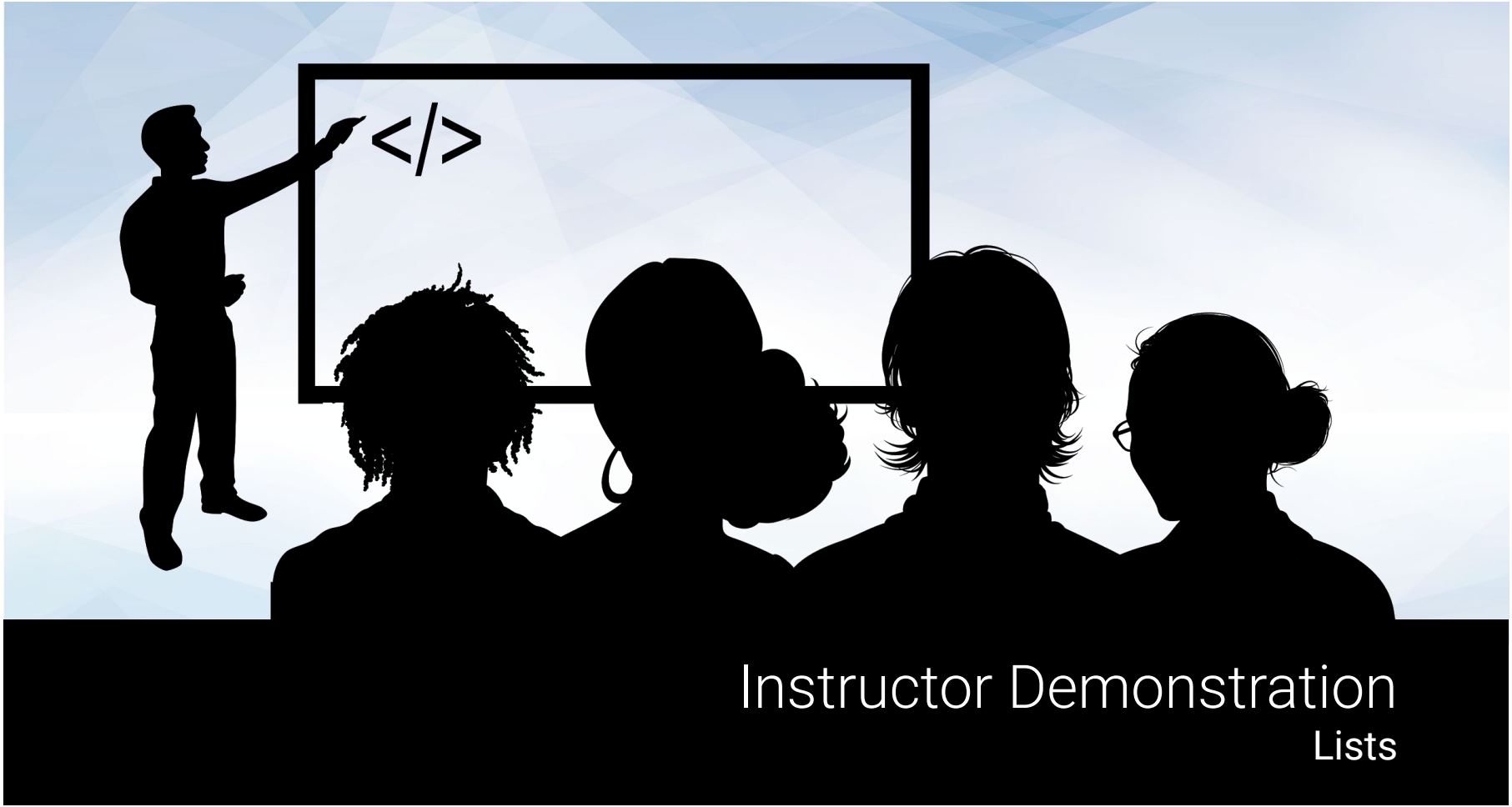
Time’s Up! Let’s Review.

Conditional Conundrum Solution

Countdown timer

15:00

(with alarm)



Instructor Demonstration

Lists

Lists

Couple of points to keep in mind before we move forward

01

Lists are the Python equivalent of arrays in VBA, functioning in much the same way by holding multiple pieces of data within one variable.

02

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.

Python has a set of built-in methods that you can use on lists

Lists Methods in Python

The append method can add elements on to the end of a list.

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

# Adds an element onto the end of the 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 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)
```

Lists Methods in Python

The `index` method returns the numeric location of a given value within a list.

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

# Adds an element onto the end of the 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 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)
```

Lists Methods in Python

The `len` function returns the length of a list.

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

# Adds an element onto the end of the 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 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)
```

Lists Methods in Python

The remove method deletes a given value from a list.

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

# Adds an element onto the end of the 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 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)
```

Lists Methods in Python

The pop method can be used to remove a value by index.

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

# Adds an element onto the end of the 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 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

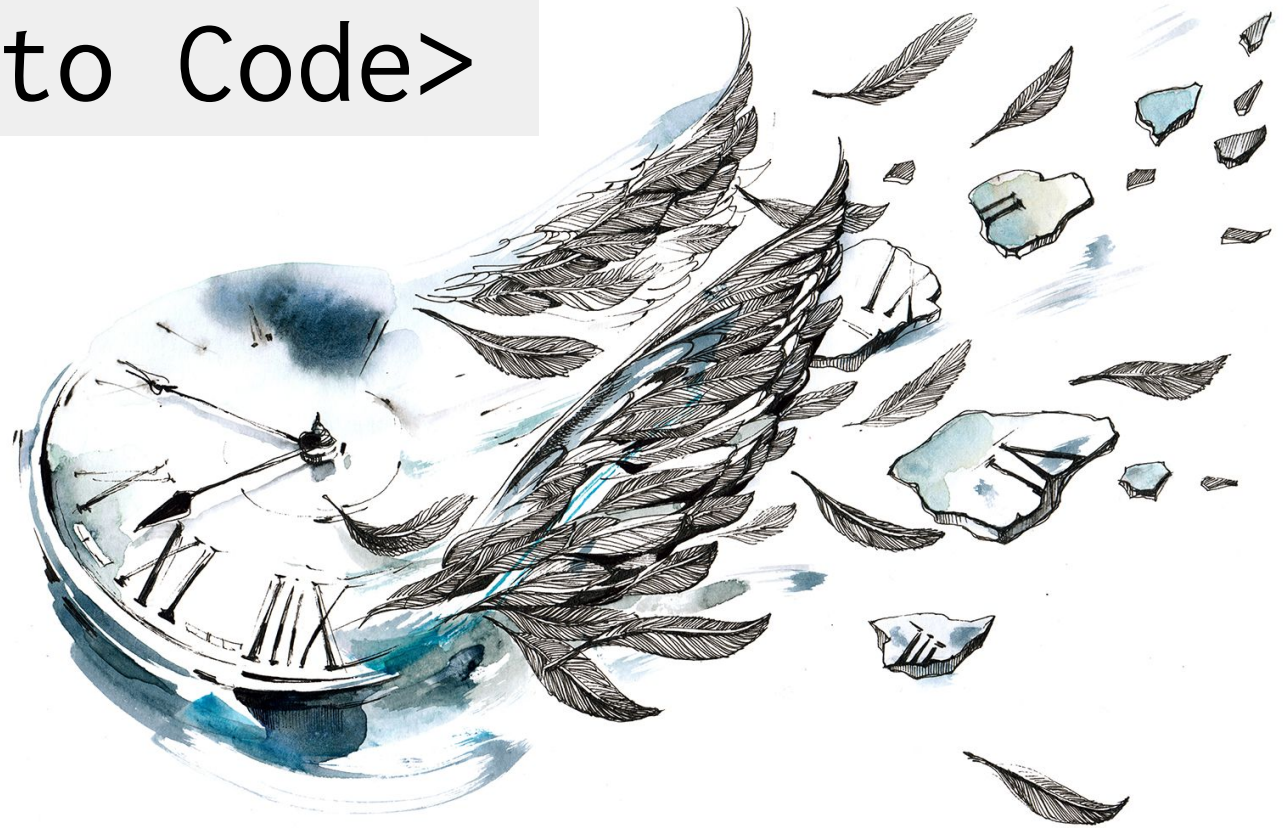
Tuples are functionally similar to lists in what they can store but are immutable

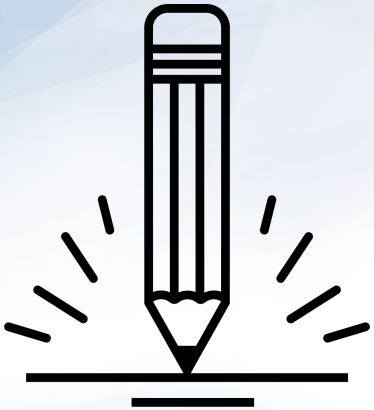
While lists in Python can be modified after their creation, tuples can never be modified after their declaration.

Tuples tend to be more efficient to navigate through than lists and also protect the data stored within from being changed.

```
# 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 create a simple game of Rock, Paper, Scissors that will run within the console.

Suggested Time:
15 Minutes

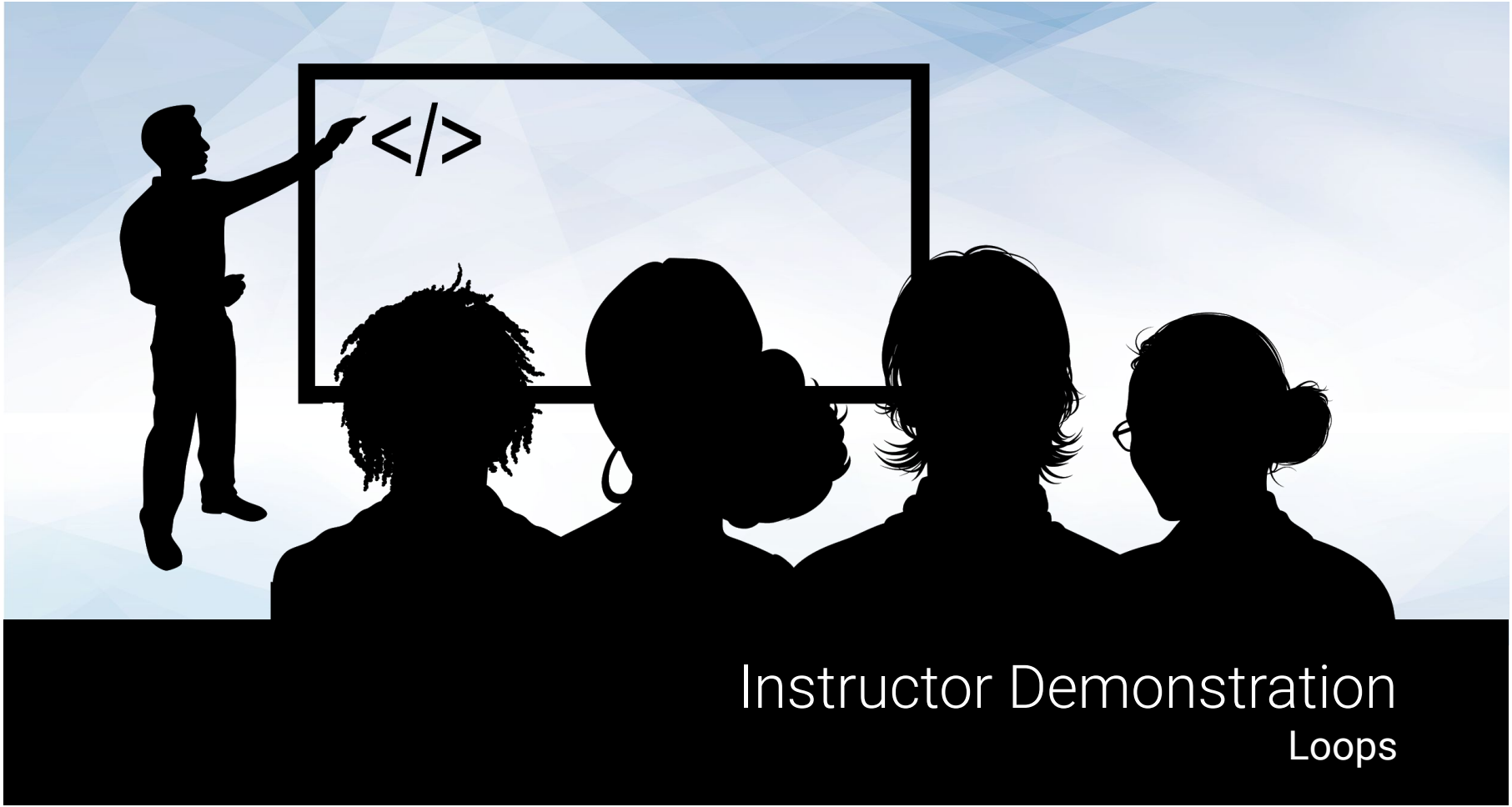
Activity: Rock, Paper, Scissors

Instructions:

- 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.

```
(PythonData) $ python RPS_Solved.py
Let’s play Rock Paper Scissors!
Make your choice: (r)ock, (p)aper, (s)cissors? p
You choose paper. The computer choose rock.
Congratulations! You won.
```

Time’s Up! Let’s Review.



Instructor Demonstration

Loops

Loops is also a concept we covered during VBA!

Loops

The variable `x` is created within the loop statement and could theoretically take on any name so long as it is unique.

```
# 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("--------------------------------------------")
```

Loops

When looping through a range of numbers, Python will halt the loop one number before the final number. For example, when looping from 0 to 5, the code will run five times, but x will only ever be printed as 0 through 4.

```
# 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("-----------------------------------------")
```

Loops

When provided with a single number, `range()` will always start the loop at 0. When provided with two numbers, however, the code will loop from the first number until it reaches one less than the second number.

```
# 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("-----------------------------------------")
```

Python can also loop through all of the letters within a string

Looping Through Strings

The syntax is for `<variable>` in `<string>`:

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

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

Python can also loop through all of the values within a list

Looping Through Lists

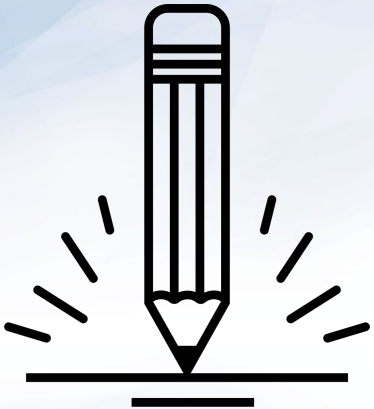
The syntax is for `<variable>` in `<list>`:

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

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

While Loops

Just like a for loop but will continue looping for as long as a condition is met

```
# Loop while a condition is being met
run = 'y'
while run == 'y':
    print('Hi!')
    run = input("To run again. Enter 'y'")
```



Activity: Number Chain

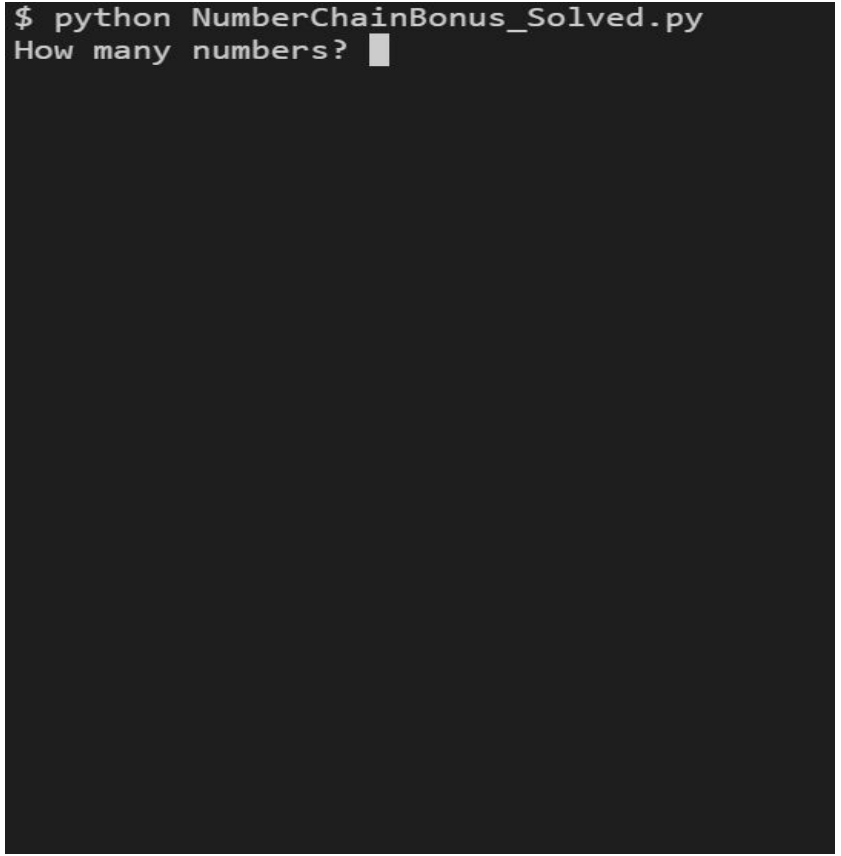
In this activity, you will take user input and print out a string of numbers.

Suggested Time:
15 Minutes

Activity: Number Chain

Instructions:

- 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.



Time’s Up! Let’s Review.


The End