Intro to Python

Sunday, September 22, 2019 12:29 PM

Overview

- Today we are moving on to a more traditional programming language, Python. This will be the primary language for the next several weeks.
- The most significant change from Visual Basic will be syntax; the fundamental concepts are the same.
- I'll be slacking out a few files to you next. Which you can use as reference guides as we progress through Python.
    - Python_Reference_Guide.pdf
    - StudentGuide.md

Class Objectives

- Check Python 3 installation.
- Be able to navigate desktop via the terminal.
- Be able to create Python scripts and run them in the terminal.
- Be able to understand basic programming concepts in Python.

Terminal

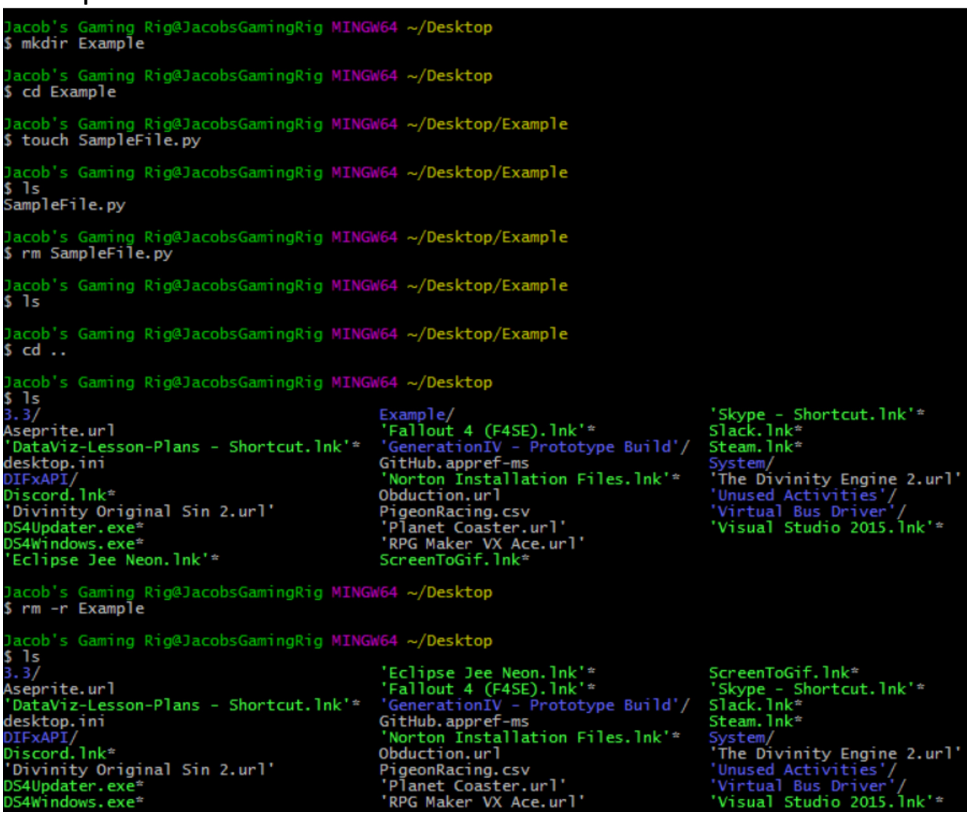
Sunday, September 22, 2019 12:56 PM

- Most of the Python code will be executed through either git-bash or the Mac terminal.
- Windows users should always use `git-bash` while Mac users should use the terminal.
- Commands (Slack these out)
  - `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 <FILENAME>` (Creates a new file with the FILENAME).
  - `rm <FILENAME>` (Deletes a file).
  - `rm -r <FOLDERNAME>` (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).

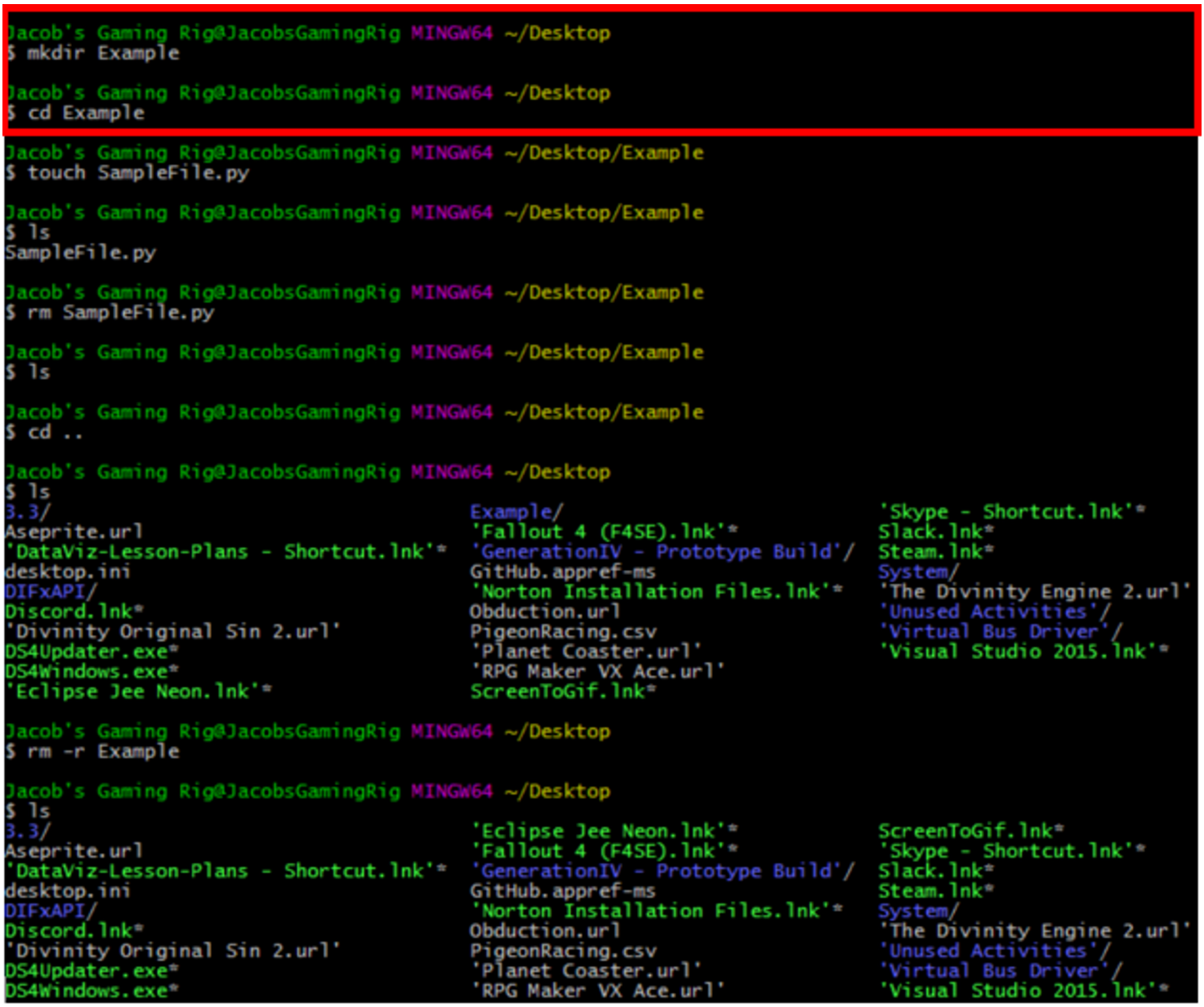
Terminal (continued)

Sunday, September 22, 2019 5:46 PM

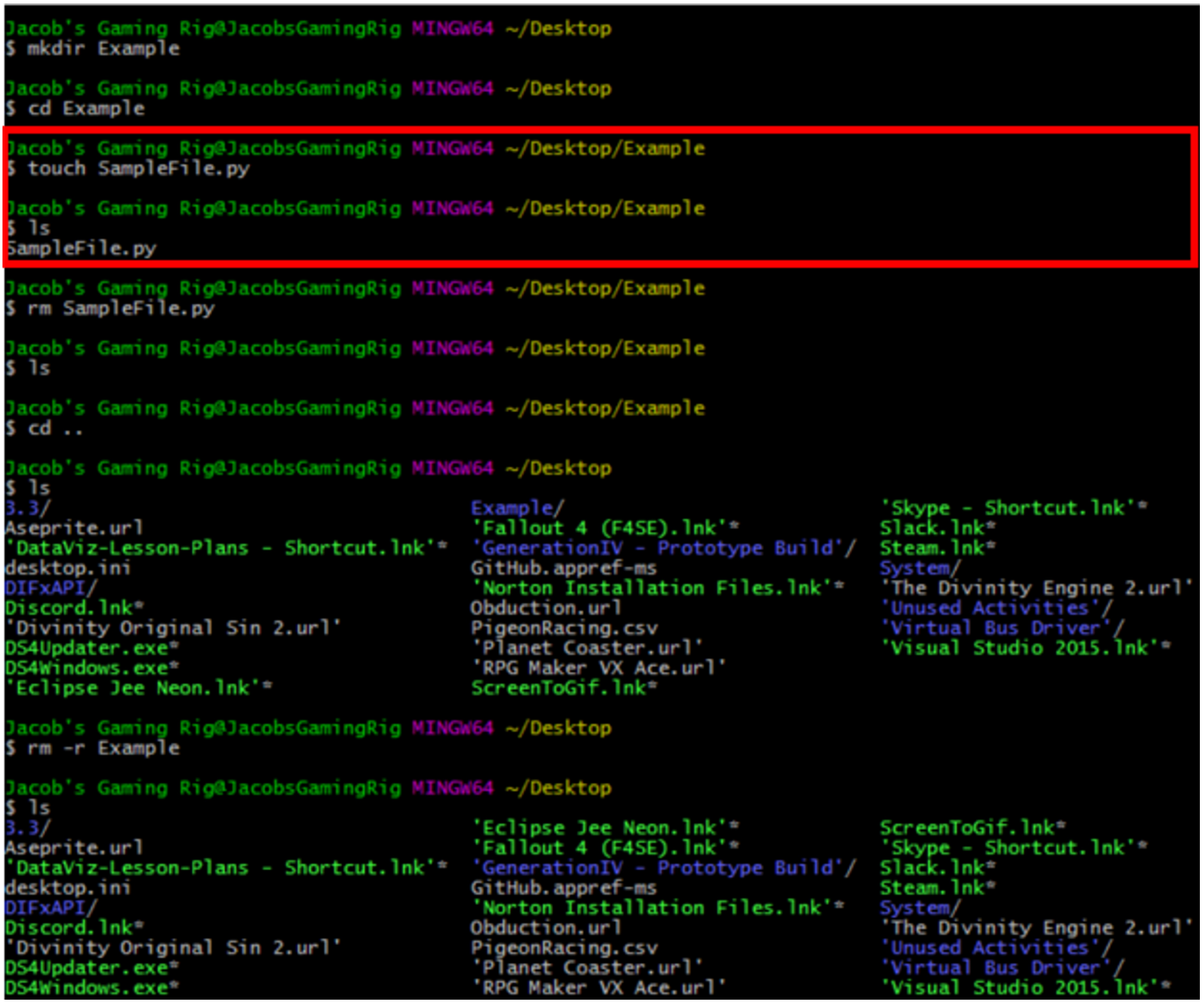
Example # 1



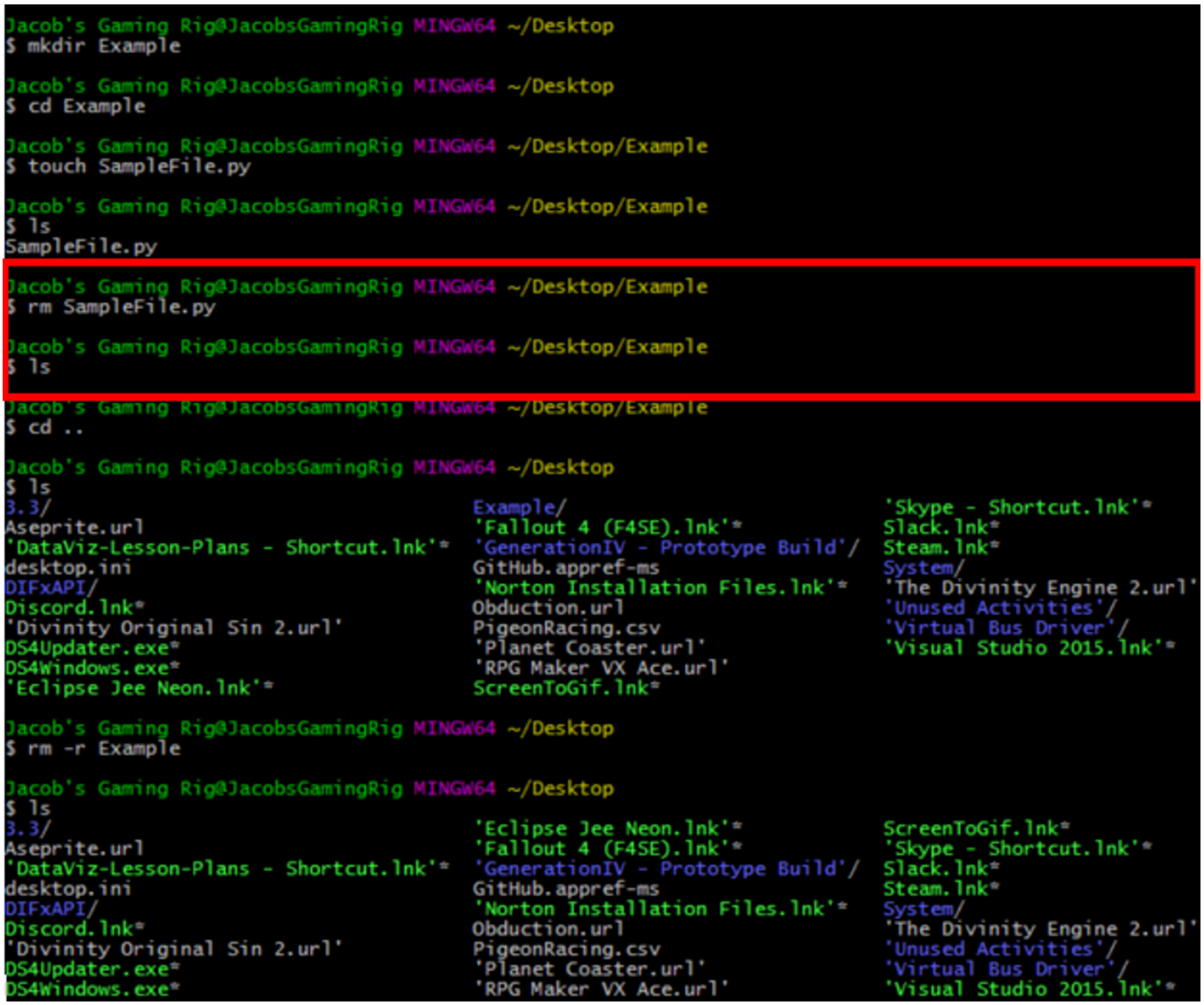
- Creates a folder (directory) called Example
- Changes directories to the Example folder that was just created



- Creates a file called ‘SampeFile.py’
- Lists all the files within the Example folder (SampleFile.py file is shown)



- Deletes SampleFile.py
- Lists all the Files within the Example folder (SampeFile.py no longer shown)



- Changed directories to the previous folder (Desktop)
- Lists all the files (Example folder is shown)

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

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

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/Example
$ touch SampleFile.py

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/Example
$ ls
SampleFile.py

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/Example
$ rm SampleFile.py

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/Example
$ ls

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

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop
$ ls
3.3/                                Example/                        'Skype - Shortcut.lnk'*
Aseprite.url                        'Fallout 4 (F4SE).lnk'*         Slack.lnk*
'DataViz-Lesson-Plans - Shortcut.lnk'*  'GenerationIV - Prototype Build'/  Steam.lnk*
desktop.ini                         GitHub.appref-ms                System/
DIFxAPI/                            'Norton Installation Files.lnk'*  'The Divinity Engine 2.url'
Discord.lnk*                        Obduction.url                   'Unused Activities'/
'Divinity Original Sin 2.url'       PigeonRacing.csv                'Virtual Bus Driver'/
DS4Updater.exe*                     'Planet Coaster.url'            'Visual Studio 2015.lnk'*
DS4Windows.exe*                     'RPG Maker VX Ace.url'
'Eclipse Jee Neon.lnk'*             ScreenToGif.lnk*
Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop
$ rm -r Example

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop
$ ls
3.3/                                'Eclipse Jee Neon.lnk'*         ScreenToGif.lnk*
Aseprite.url                        'Fallout 4 (F4SE).lnk'*         'Skype - Shortcut.lnk'*
'DataViz-Lesson-Plans - Shortcut.lnk'*  'GenerationIV - Prototype Build'/  Slack.lnk*
desktop.ini                         GitHub.appref-ms                Steam.lnk*
DIFxAPI/                            'Norton Installation Files.lnk'*  System/
Discord.lnk*                        Obduction.url                   'The Divinity Engine 2.url'
'Divinity Original Sin 2.url'       PigeonRacing.csv                'Unused Activities'/
DS4Updater.exe*                     'Planet Coaster.url'            'Virtual Bus Driver'/
DS4Windows.exe*                     'RPG Maker VX Ace.url'          'Visual Studio 2015.lnk'*
```

- Removes the Example folder
- Lists all the files (Example folder is no longer shown)

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

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

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/Example
$ touch SampleFile.py

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/Example
$ ls
SampleFile.py

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/Example
$ rm SampleFile.py

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop/Example
$ ls

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

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop
$ ls
3.3/                                Example/                        'Skype - Shortcut.lnk'*
Aseprite.url                        'Fallout 4 (F4SE).lnk'*         Slack.lnk*
'DataViz-Lesson-Plans - Shortcut.lnk'*  'GenerationIV - Prototype Build'/  Steam.lnk*
desktop.ini                         GitHub.appref-ms                System/
DIFxAPI/                            'Norton Installation Files.lnk'*  'The Divinity Engine 2.url'
Discord.lnk*                        Obduction.url                   'Unused Activities'/
'Divinity Original Sin 2.url'       PigeonRacing.csv                'Virtual Bus Driver'/
DS4Updater.exe*                     'Planet Coaster.url'            'Visual Studio 2015.lnk'*
DS4Windows.exe*                     'RPG Maker VX Ace.url'
'Eclipse Jee Neon.lnk'*             ScreenToGif.lnk*

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop
$ rm -r Example

Jacob's Gaming Rig@JacobsGamingRig MINGW64 ~/Desktop
$ ls
3.3/                                'Eclipse Jee Neon.lnk'*         ScreenToGif.lnk*
Aseprite.url                        'Fallout 4 (F4SE).lnk'*         'Skype - Shortcut.lnk'*
'DataViz-Lesson-Plans - Shortcut.lnk'*  'GenerationIV - Prototype Build'/  Slack.lnk*
desktop.ini                         GitHub.appref-ms                Steam.lnk*
DIFxAPI/                            'Norton Installation Files.lnk'*  System/
Discord.lnk*                        Obduction.url                   'The Divinity Engine 2.url'
'Divinity Original Sin 2.url'       PigeonRacing.csv                'Unused Activities'/
DS4Updater.exe*                     'Planet Coaster.url'            'Virtual Bus Driver'/
DS4Windows.exe*                     'RPG Maker VX Ace.url'          'Visual Studio 2015.lnk'*
```

Live Demo: Terminal

Sunday, September 22, 2019 6:39 PM

LP

Exercise: Terminal (10 minutes)

Sunday, September 22, 2019 1:03 PM

Overview

Students will now dive into the terminal, create three folders, and 2 Python files which will print some strings of their own creation to the console

Instructions

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

Review: Terminal

Sunday, September 22, 2019 5:35 PM

```
# Create a folder called LearnPython
mkdir LearnPython

# Navigate into the folder
cd LearnPython

# Inside LearnPython create another folder called Assignment1
mkdir Assignment1

# Inside Assignment1 create a file called quick_python.py
touch quick_python.py

# Add a print statement to quick_python.py
# add print("This file works!") in a python file

# Run quick_python.py
python quick_python.py

# Return to the LearnPython folder
cd ..

# Inside LearnPython create another folder called Assignment2
mkdir Assignment2

# Inside Assignment2 create a file called quick_python2.py
touch quick_python2.py

# Add a different print statement to quick_python2.py
# add print("This file also works!") in a python file

# Run quick_python2.py
python quick_python2.py
```

Check Anaconda/VS Codecon Installation

Sunday, September 22, 2019 5:36 PM

- Let’s do a quick check to ensure everyone has have conda and added to their path.
- Open your console and type “conda - -version”
    - Displays the version of Anaconda install on your machine
- The most common problem with Windows is computers that don't have the Anaconda PATH variable set.
    - This can be fixed by manually adding the PATH to Windows' environment variables, but can be more easily solved by uninstalling/reinstalling Anaconda and making sure to check the "Install to Path" box that comes up in a menu.

Break (15 minutes)

Sunday, September 22, 2019 10:07 PM

Create a Virtual Environment

Sunday, September 22, 2019 5:36 PM

Virtual Environment

- Create an isolated environment for Python Projects
- Different projects can have different dependencies
  - Different projects might use different types and versions of libraries
- The virtual environment we create will make sure the we have all the right dependencies for future classes
- Link: https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

Creating a Virtual Environment

- We are going to create a virtual environment that will run Python 3.6. This will help solve issues where students have multiple versions of Python installed.
- Type “conda create -n PythonData python=3.6 anaconda” in the terminal
  - This might take a while
- Once complete, type “source activate PythonData” to activate the environment
- When “(PythonData)” appears, that means you are in the environment
- Type “python - - version” to make sure you are using the correct version of python
- **You will need to activate this environment each time you open a new console**
- You can exit the environment by entering “source deactivate” or “conda deactivate

Variables

Sunday, September 22, 2019 5:37 PM

- Variables let us store information that we can use later on
- In VBA, we accessed certain values when they referenced a specific cell. This is essentially what a variable is doing in Python, a value is being sored there
- **Open variables.py

- Variables can store different types like strings, integers, and 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
```

- We can print statements which include variables, but traditional Python formatting won't concatenate string with other data types. This means integers and booleans must be cast as string 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 string to be formatted with different data types.
    - Slack out: https://realpython.com/python-f-strings/

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

Exercise: Hello Variable World

Sunday, September 22, 2019 5:37 PM

Overview

Create a simple Python application that uses variables. This file will both run calculations on integers and print string out to the console

Instructor Demo[HelloVariableWorld.py]

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

Review: Hello Variable World

Sunday, September 22, 2019 5:37 PM

Key Points

- Each of the variables has to be declared individually but do not have to be cast at declaration. Python figures out data type on its own.
- Integer variables can easily be placed into calculations simply by using their name.
- Even though booleans look like strings, they do no use quotations in their declaration.

```
# Create a variable called 'name' that holds a string
name = "Jacob Deming"

# Create a variable called 'country' that holds a string
country = "United States"

# Create a variable called 'age' that holds an integer
age = 25

# Create a variable called 'hourly_wage' that holds an integer
hourly_wage = 15

# Calculate the daily wage for the user
daily_wage = hourly_wage * 8

# Create a variable called 'satisfied' that holds a boolean
satisfied = True
```

- When traditionally printing out integers and booleans with strings, the variables must be cast as string as well. Without casting them as strings, the console will return error

```
# Print out the user's age
print("You are " + str(age) + " years old")
```

- When using an f-string, integers and booleans do not need to be cast as strings. Also, the syntax is slightly different: variables are enclosed in curly braces, there are no plus signs, and a single set of quotation marks around the entire string.

```
# With an f-string, print out the daily wage that was calculated
print(f"You make {daily_wage} per day")

# With an f-string, print out whether the users were satisfied
print(f"Are you satisfied with your current wage? {satisfied}")
```

Inputs and Prompts

Sunday, September 22, 2019 5:38 PM

**Open inputs.py file

Take a closer look at the code

```
# 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))
```

- The variable “name” will store the user's response to the prompt.

```
# 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))
```

- Every response to an input is stored as a string regardless of the characters entered.
- As such, variables that are intended to be integers must converted to be used in calculations. Uses Int() function

```
# 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))
```

- The `bool()` function always returns `True` if any text is inside of it (technically, any non-empty string will evaluate to True).

```
# 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))
```

- Print statements can be concatenated with variables so long as they are also strings or are cast as strings.

```
# 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))
```

Exercise: Down to Input

Sunday, September 22, 2019 5:38 PM

Overview

Students will get a chance to work on storing inputs form the command line and run some code based upon events
Entered

Instructor Demo [DownToInput.py]

Instructions

- 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.

Review: Down to Input

Sunday, September 22, 2019 5:39 PM

Key Points

```
# Take input of you and your neighbor
your_first_name = input("What is your name? ")
neighbor_first_name = input("What is your neighbors name? ")

# Take how long each of you have been coding
months_you_coded = input("How many months have you been coding? ")
months_neighbor_coded = input("How many months has your neighbor been coding? ")

# Add total month
total_months_coded = int(months_you_coded) + int(months_neighbor_coded)

# Print results
print("I am " + your_first_name + " and my neighbor is " + neighbor_first_name)
print("Together we have been coding for " + str(total_months_coded) + " months!")
```

- The variables "your_first_name" and "neighbor_first_name" are set using tow inputs and since they are strings, we won't have to cast them later on

```
# Take input of you and your neighbor
your_first_name = input("What is your name? ")
neighbor_first_name = input("What is your neighbors name? ")

# Take how long each of you have been coding
months_you_coded = input("How many months have you been coding? ")
months_neighbor_coded = input("How many months has your neighbor been coding? ")

# Add total month
total_months_coded = int(months_you_coded) + int(months_neighbor_coded)

# Print results
print("I am " + your_first_name + " and my neighbor is " + neighbor_first_name)
print("Together we have been coding for " + str(total_months_coded) + " months!")
```

- The "months_you_coded" and "months_neighbor_coded" are set using two inputs, but need to be cast to be acknolwded as integers

```
# Take input of you and your neighbor
your_first_name = input("What is your name? ")
neighbor_first_name = input("What is your neighbors name? ")

# Take how long each of you have been coding
months_you_coded = input("How many months have you been coding? ")
months_neighbor_coded = input("How many months has your neighbor been coding? ")

# Add total month
total_months_coded = int(months_you_coded) + int(months_neighbor_coded)

# Print results
print("I am " + your_first_name + " and my neighbor is " + neighbor_first_name)
print("Together we have been coding for " + str(total_months_coded) + " months!")
```

- After calculating the total number of months, this new integer variable will have to be cast as a string to be printed

```
# Take input of you and your neighbor
your_first_name = input("What is your name? ")
neighbor_first_name = input("What is your neighbors name? ")

# Take how long each of you have been coding
months_you_coded = input("How many months have you been coding? ")
months_neighbor_coded = input("How many months has your neighbor been coding? ")

# Add total month
total_months_coded = int(months_you_coded) + int(months_neighbor_coded)

# Print results
print("I am " + your_first_name + " and my neighbor is " + neighbor_first_name)
print("Together we have been coding for " + str(total_months_coded) + " months!")
```

Conditionals

Sunday, September 22, 2019 5:39 PM

**Open conditionals.py

- Python uses **if**, **elif**, and **else** for creating conditionals (pay attention to the letter case and spelling!).
- 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.

```
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

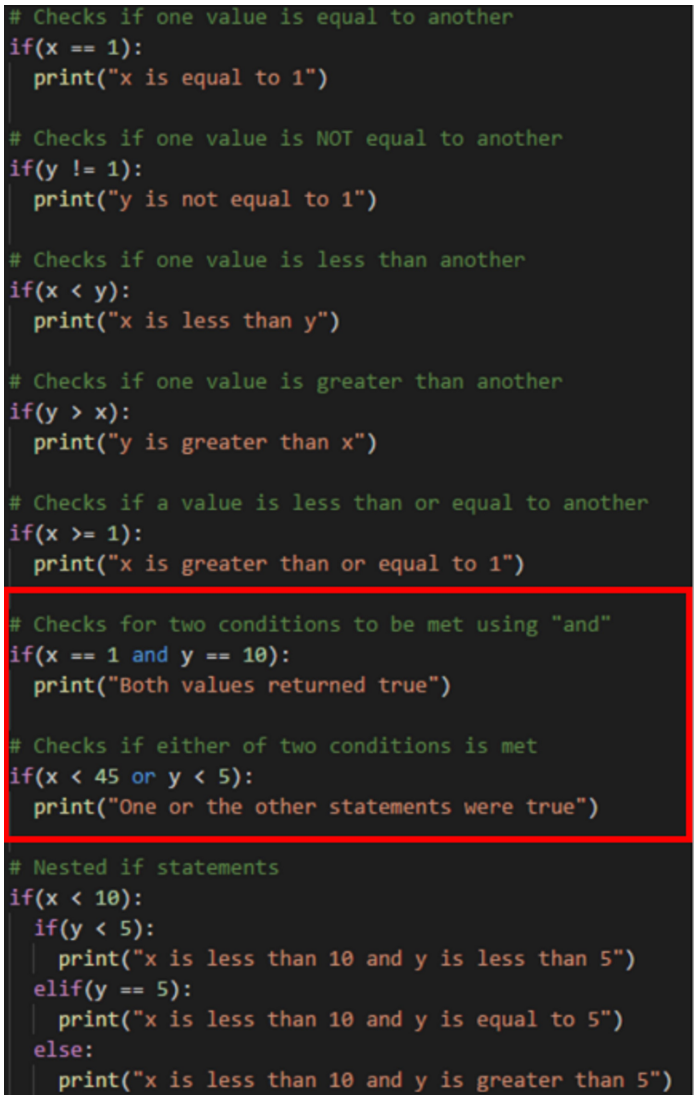
- All sorts of operators like greater than, less than, equal to, and much more can be used to create logic tests for conditionals.

```
# 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")
```

- The condition **is equal to** uses **==** while variable assignment uses one equal sign.

```
# 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")
```

- Multiple logic tests can be checked within a single conditional statement.
- Using the term **and** must mean both tests return **True**
- **or** requires that only one test return as **True**.



```
# 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")
```

- Conditionals can even be nested, allowing programmers to run logic tests based upon whether or not the original logic test returned as **True**.

```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")
```

Exercise: Conditional Conundrum

Sunday, September 22, 2019 5:39 PM

Overview

Student will be looking through some pre-written conditionals and attempting to figure out what lines will be printed to the console

No Instructor Demo

File Needed: conditionals_unsolved.py

Instructions

- 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

Review: Conditionals Conundrum

Sunday, September 22, 2019 5:39 PM

Key Points

- The **if** statement for the first code chunk checks whether **10 > 10**, which is false. As such the code will return *"ooo needs some work".*

```
# 1. oooo needs some work
x = 5
if (2 * x > 10):
    print("Question 1 works!")
else:
    print("oooo needs some work")
```

- The length of *"Dog"* is 3 and x is 5, thus making the statement for the second chunk of code true and returning *"Question 2 works!"*

```
# 2. Question 2 works!
x = 5
if (len("Dog") < x):
    print("Question 2 works!")
else:
    print("Still missing out")
```

- The addition of the **and** statement to the third chunk of code means both logic tests need return **True** to run. Thankfully both do and thus *"GOT QUESTION 3!"* is printed.

```
# 3. GOT QUESTION 3!
x = 2
y = 5
if ((x**3 >=y) and (y**2 < 26)):
    print("GOT QUESTION 3!")
else:
    print("Oh good you can count")
```

- Conditionals work by going form the top down. The logic tests in

chunk four do not return as **True** until the third conditional and, as such, *"Dan is in group three"* is printed.

```
# 4. Dan is in group three
name = "Dan"
group_one = ["Greg", "Tony", "Susan"]
group_two = ["Gerald", "Paul", "Ryder"]
group_three = ["Carla", "Dan", "Jefferson"]

if (name in group_one):
    print(name + " is in the first group")
elif (name in group_two):
    print(name + " is in group two")
elif (name in group_three):
    print(name + " is in group three")
else:
    print(name + " does not have a group")
```

- Within chunk five, one of the conditions are met in the third conditional, but it's not until getting into the **or** statement of the fourth conditional that the logic test finally returns as **True**

```
# 5. Can ride bumper cars
height = 66
age = 16
adult_permission = True

if ((height > 70) and (age >=18)):
    print("Can ride all the roller coasters")
elif ((height > 65) and (age >=18)):
    print("Can ride moderate roller coasters")
elif ((height > 60) and (age >= 18)):
    print("Can ride light roller coasters")
elif (((height > 50) and (age >= 18)) or ((adult_permission) and (height > 50))):
    print("Can ride bumper cars")
else:
    print("Sitck to lazy river")
```

Lists

Sunday, September 22, 2019 5:40 PM

- 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.
- Open lists.py

```
# 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)
```

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

```
# 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"))
```

```
# 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)
```

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

```
# 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)
```

- The **len** function returns the length of a list.

```
# 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)
```

- The **remove** method deletes a given value from a list.

```
# 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)
```

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

```python
# 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)
```

- Python also has a data type called "tuples" that 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.

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

Exercise: Rock, Paper, Scissors

Sunday, September 22, 2019 5:40 PM

Overview

Students will be creating a simple Rock, Papers, Scissors that will run within the console

Instructor Demo [RPS_Solved.py]

File Needed: RPS_Unsolved.py

This starter code imports a module called **random** that will allow the computer to make a choice randomly from a list of actions. We'll be diving deeper into modules next class

Random documentation:
https://docs.python.org/3.6/library/random.html

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.

Hints

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

Review: Rock, Paper, Scissors

Sunday, September 22, 2019 5:40 PM

Starter Code

- In the starter code, **random.choice** will pick a random choice within the **options** list for the computer and store its pick within a variable called **computer_choice**.
- The application prompts the user for their option and stores it within a variable called **user_choice**

```
# Incorporate the random library
import random

# Print Title
print("Let's Play Rock Paper Scissors!")

# Specify the three options
options = ["r", "p", "s"]

# Computer Selection
computer_choice = random.choice(options)

# User Selection
user_choice = input("Make your Choice: (r)ock, (p)aper, (s)cissors? ")
```

- Knowing that rock beats scissors, scissors beats paper, and paper beats rock the code can be organized into a series of conditional statements to compare the user's choice to the computer's choice

```
if (user_choice == "r" and computer_choice == "p"):
  print("You chose rock. The computer chose paper.")
  print("Sorry. You lose.")

elif (user_choice == "r" and computer_choice == "s"):
  print("You chose rock. The computer chose scissors.")
  print("Yay! You won.")

elif (user_choice == "r" and computer_choice == "r"):
  print("You chose rock. The computer chose rock.")
  print("A smashing tie!")

elif (user_choice == "p" and computer_choice == "p"):
  print("You chose paper. The computer chose paper.")
  print("A smashing tie!")
```

Loops

Sunday, September 22, 2019 5:41 PM

**Open LoopDeeLoop.py

The syntax for loops in Python is different than VBA

- The variable x is created within the loop statement and could theoretically take on any name so long as it is unique.
- 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.
- 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 or all of the values stored within a list by using the syntax **`for <variable> in <string or list>:`**

```
# 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** will run blocks of code just like a **for loop** does 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'" )
```

Exercise: Number Chain

Sunday, September 22, 2019 5:41 PM

Overview

This exercise will take user input and print out a string of numbers.

Instructor Demo [NumberChain_solved.py]

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

Review: Number Chain

Sunday, September 22, 2019 5:41 PM

Key Points

- The initial value for `user_play` is set to "y" so that the `while` loop will run initially. This loop will continue to run so long as the value of `user_play` is "y" at the end of the code block.

```
# Initial variable to track game play
user_play = "y"

# While we are still playing...
while user_play == "y":

    # Ask the user how many numbers to loop through
    user_number = input("How many numbers? ")

    # Loop through the numbers. (Be sure to cast the string into an integer.)
    for x in range(int(user_number)):

        # Print each number in the range
        print(x)

    # Once complete...
    user_play = input("Continue: (y)es or (n)o? ")
```

- An input number is asked for and then a `for` loop will then run to count from 0 to that number.

```
# Initial variable to track game play
user_play = "y"

# While we are still playing...
while user_play == "y":

    # Ask the user how many numbers to loop through
    user_number = input("How many numbers? ")

    # Loop through the numbers. (Be sure to cast the string into an integer.)
    for x in range(int(user_number)):

        # Print each number in the range
        print(x)

    # Once complete...
    user_play = input("Continue: (y)es or (n)o? ")
```

- The user is then prompted to either enter "y" if they would like to create a new number chain or "n" if they would like to terminate the application.

```
# Initial variable to track game play
user_play = "y"

# While we are still playing...
while user_play == "y":

    # Ask the user how many numbers to loop through
    user_number = input("How many numbers? ")

    # Loop through the numbers. (Be sure to cast the string into an integer.)
    for x in range(int(user_number)):

        # Print each number in the range
        print(x)

    # Once complete...
    user_play = input("Continue: (y)es or (n)o? ")
```

Homework (Due 10/5)

Sunday, September 22, 2019 5:42 PM

- You will be reading from a csv and formulating the results using Python. You'll be given two data sets where their scripts will need to work for each.