



# A Deeper Dive Into Python

Data Boot Camp

Lesson 3.3



# Class Objectives

---

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



Create and use Python dictionaries.



Read in data from a dictionary.



Use list comprehensions.



Write and reuse Python functions.



Use coding logic and reasoning.



Add, commit, and push code to GitHub from the command line.



**WELCOME**



# Activity: Cereal Cleaner

In this activity, you will create an application that reads in cereal data from a CSV file and then prints only those cereals that contain 5 or more grams of fiber.

Suggested Time:

20 Minutes

# Activity: Cereal Cleaner

## Instructions

Open the file, `cereal.csv` and start by skipping the header row. See hints below for this.

Read through the remaining rows and find the cereals that contain five grams of fiber or more, printing the data from those rows to the terminal.

## Hint

- Every value within the CSV is stored as a string, and certain values have a decimal. This means that they will have to be cast to be used.
- `csv.reader` begins reading the CSV file from the first row. `next(csv_reader, None)` will skip the header row.
- Refer to this Stack Overflow post on [how to skip the header](#) for more information.
- Integers are whole numbers and, as such, cannot contain decimals. Decimal numbers will have to be cast as a `float` or `double`.

## Bonus

Try the activity again, but this time use `cereal_bonus`, which does not include a header.



Time's Up! Let's Review.

# Questions?



# Dictionaries





Another commonly used data type  
in Python is the dictionary, or **dict**.

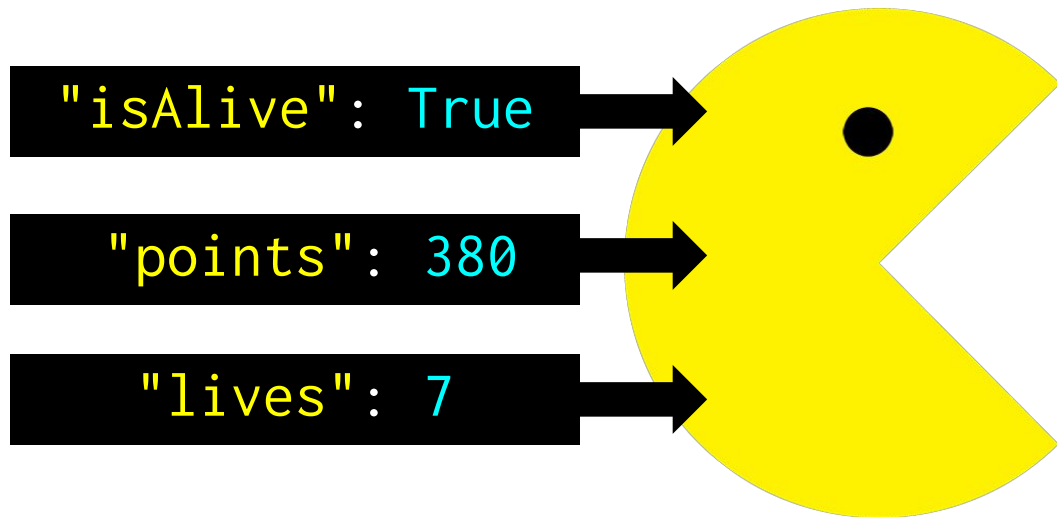


A **dictionary** is an object that stores a collection of data.

# Dictionaries

---

Like lists and tuples, dictionaries can contain multiple values and data types. The key in a dictionary is a string that can be referenced to collect an associated value. However, unlike lists and tuples, dictionaries store data in **key-value** pairs.



```
pacman = {  
    "isAlive": True,  
    "points": 380,  
    "lives": 7  
}
```

# Dictionaries


---

To use the example of a physical dictionary, the words in the dictionary would be considered the keys, and the definitions of those words would be the values.

word = key

definition = value

```
{"python" : "constricting_snake"}
```



**python** (**noun**): any of various large constricting **snakes** especially any of the large oviparous **snakes** (subfamily Pythoninae of the family Boidae) of Africa, Asia, Australia, and adjacent islands that include some of the largest existing **snakes**.

# Dictionaries

---

To initialize or create an empty dictionary, we use the syntax `actors = {}`.

```
# Create a dictionary to hold the actor's names.  
actors = {}
```

You can also create a dictionary with the built-in Python `dict()` function, or `actors = dict()`.

```
# Create a dictionary using the built-in function.  
actors = dict()
```

# Dictionaries

---

Values can be added to dictionaries at declaration by creating a key that is stored in a string, following it with a colon, and then placing the desired value after the colon.

To reference a value within a dictionary, we simply call the dictionary and follow it up with a pair of brackets containing the key for the desired value.

```
# A dictionary of an actor.  
actors = {"name": "Tom Cruise"}  
print(f'{actors["name"]}')
```

# Dictionaries

---

Values can also be added to dictionaries by placing the key within single or double quotation marks inside brackets, and then assigning the key a value; then, values can be changed or overwritten by assigning the key a new value.

```
# Add an actor to the dictionary with the key "name"  
# and the value "Denzel Washington".  
actors["name"] = "Denzel Washington"
```

# Dictionaries

---

Dictionaries can hold multiple pieces of information by following up each key-value pairing with a comma and then another key-value pair.

```
# A list of actors
actors_list = [
    "Tom Cruise",
    "Angelina Jolie",
    "Kristen Stewart",
    "Denzel Washington"]

# Overwrite the value, "Tom Cruise", with the list of actors.
actors["name"] = actors_list
```



# Dictionaries

---

## Keys

Keys are immutable objects, like integers, floating-point decimals, or strings.

Keys cannot be lists or any other type of mutable object.

## Values

Values in a dictionary, as captured in the following image, can be objects of any type:

- integers
- floating-point decimals
- strings, Booleans
- `datetime` values
- lists

# Dictionaries

---

Items in a list in a dictionary can be accessed by calling the key and then using indexing to access the item, as in the following image.

```
# Print the first actor  
print(f'{actors["name"][0]}')
```



**You only need a basic understanding of this for now;  
when you get into APIs, you will get a lot more practice!**



**Dictionaries can also contain  
other dictionaries.**

# Dictionaries

---

To access the values inside nested dictionaries, simply add another key to the reference.

```
# A dictionary can contain multiple pairs of information
actress = {"name": "Angelina Jolie", "genre": "Action", "nationality": "United States"}

# -----

# A dictionary can contain multiple types of information
another_actor = {"name": "Sylvester Stallone", "age": 62, "married": True, "best movies": "Rocky"}
print(f'{another_actor["name"]} was in {another_actor["best movies"][0]}')

# -----

# A dictionary can even contain another dictionary
film = {"title": "Interstellar",
        "revenues": {"United States": 360, "China": 250, "United Kingdom": 73}}
print(f'{film["title"]} made {film["revenues"]["United States"]} in the US.')

# -----
```

# Questions?





# Activity: Hobby Book

In this activity, you will create and access dictionaries that are based on your own hobbies.

Suggested Time:

15 Minutes

# Activity: Hobby Book

---

## Instructions

Create a dictionary to store the following information:

- Your name
- Your age
- A list of a few of your hobbies
- A dictionary that includes a few days and the time you typically wake up on those days

Print out your name, how many hobbies you have, and a time you wake up during the week.



Time's Up! Let's Review.



# Questions?





# Instructor Demonstration

---

## List Comprehensions



# Activity: List Comprehensions

In this activity, you will use list comprehensions to compose a wedding invitation to send to every name on your mailing list.

Suggested Time:

10 Minutes

# Activity: List Comprehensions

---

## Instructions

Open the file called `comprehensions.py`.

Create a list that prompts the user for the names of five people that they know.

Run the provided program. Note that nothing forces you to write the name properly in title case—for example, as "Jane" instead of "jAnE". You will use list comprehensions to fix this.

- First, use list comprehensions to create a new list that contains the lowercase versions of the names your user provided.
- Then, use list comprehensions to create a new list that contains the title-case versions of each of the names in your lowercase list.

## Hint

Check out the documentation for the [`title`](#) method.



Time's Up! Let's Review.

# Questions?



# Functions

In software engineering,  
**Don't Repeat Yourself (DRY)**  
is a principle of software  
development that we can use  
functions and modules to  
avoid repeating code.

A person is wearing a red t-shirt. On the chest, there is a graphic that reads "< DRY >" in large, bold, white letters. Below this, in a white rectangular box, it says "DON'T REPEAT YOURSELF" in smaller, white, uppercase letters.

< DRY >  
DON'T REPEAT YOURSELF





**Are any disadvantages to writing code that does the same thing in three different places?**

# Functions

---

If we write the same code in different places and expect it to behave the same everywhere, we will also have to update it in several places whenever we make a change.



This can quickly become unwieldy.



In large codebases, copying code in multiple places would often require us to waste time making the same change in several places.



It would also add the overhead of tracking duplicated code.



Efficiency is the motivation for the **d**on't **r**epeat **y**ourself mantra.



# Instructor Demonstration

---

## Functions



# Activity: Functions

In this activity, you will write a function that returns the arithmetic mean (average) for a list of numbers.

Suggested Time:

10 Minutes

# Activity: Functions

---

## Instructions

Write a function called `average` that accepts a list of numbers as a single argument.

- The function `average` should return the arithmetic mean (average) for a list of numbers

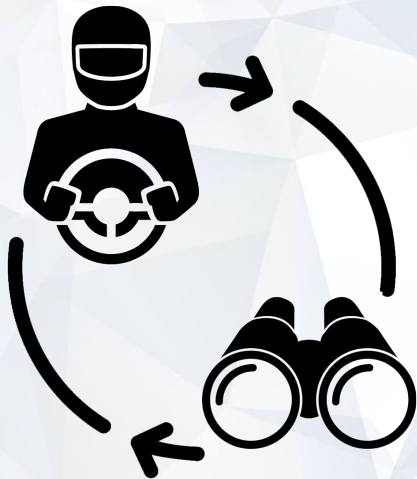
Test your function by calling it with different values and printing the results.



Time's Up! Let's Review.

# Questions?





## Pair Programming Activity:

---

# Graduating Functions

In this activity, you will create a function that searches a list of students and graduates by state to determine state graduation rates for public, private nonprofit, and private for-profit institutions.

Suggested Time:

15 Minutes



# Activity: Graduating Functions

## Instructions

Analyze the code and CSV file provided to determine what needs to be added to the application.

Using the starter code provided, create a function called `print_percentages` that takes in a parameter called `state_data` and does the following actions:

- The function uses the data stored within `state_data` to calculate the estimated graduation rates for each category of Title IV 4-year institutions (public, nonprofit private, and for-profit private).
- The function prints out the graduation rates for each school type for the state to the terminal.

## Note

Some states do not have nonprofit or for-profit private schools, so data must be checked for zeros.

## Bonus

Within the same `print_percentages()` function, calculate the overall graduation rate. Then, create a conditional that checks a state's overall graduation rate and prints "Graduation success" if the number was higher than 50 or "State needs improvement" if the number was lower than 50.



Time's Up! Let's Review.



Countdown timer

**40:00**

(with alarm)



# Intro to Git

# Intro to Git

---

So far, we have used GitHub as a sort of “drop box” to store our files.

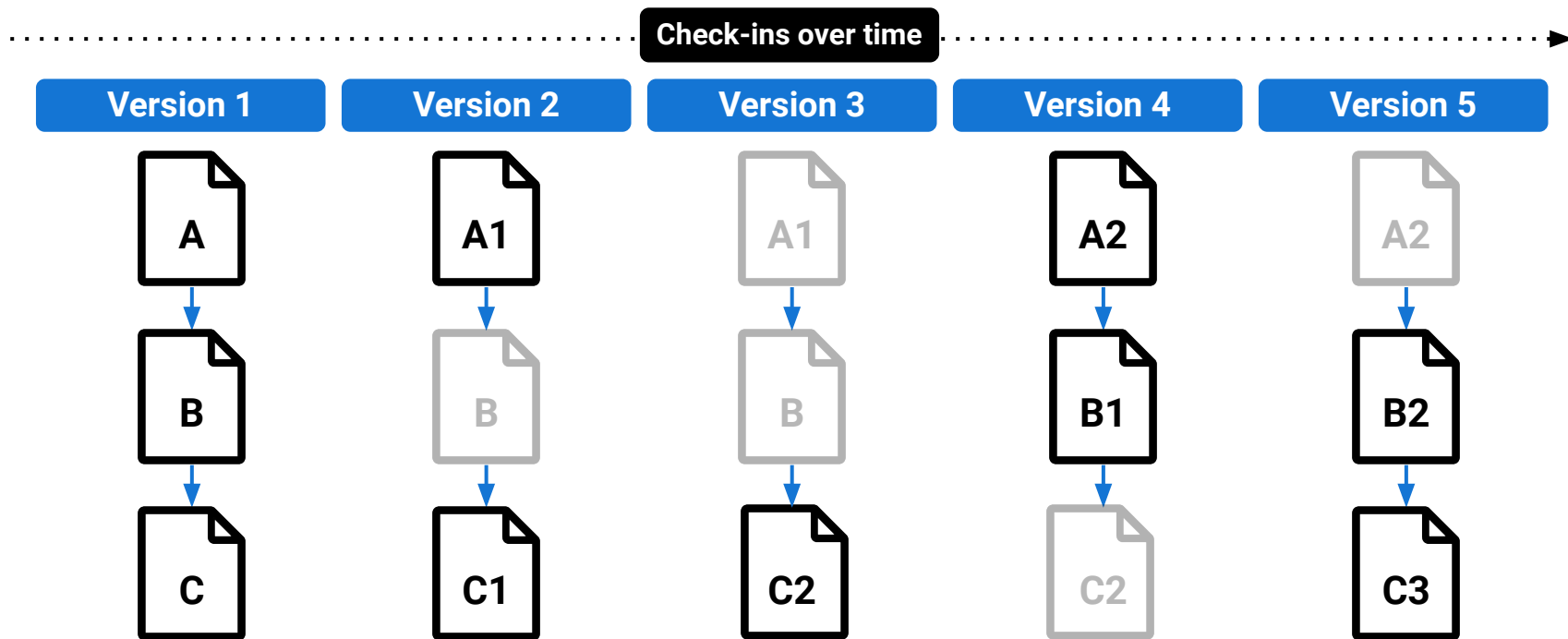
Although this is one way to use GitHub, it has much deeper capabilities.

Today, we will delve into Git and how to use it through the terminal to interact with GitHub.



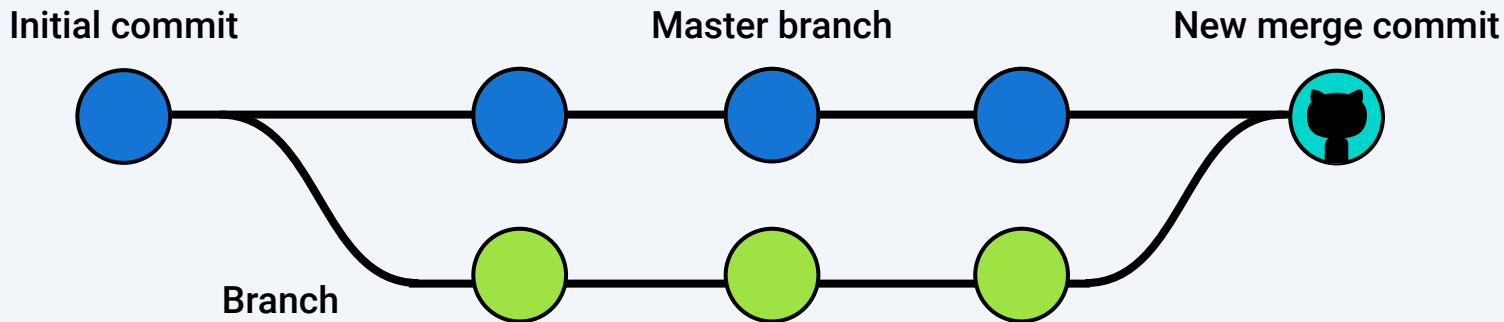
# Intro to Git

Git is essentially a way for us to keep track of our work over time. Whenever we get another piece of a project working, we can save the change with Git.



# Git Commit

A Git “save” is called a **commit**. It represents a checkpoint for our project where we save and describe our work.

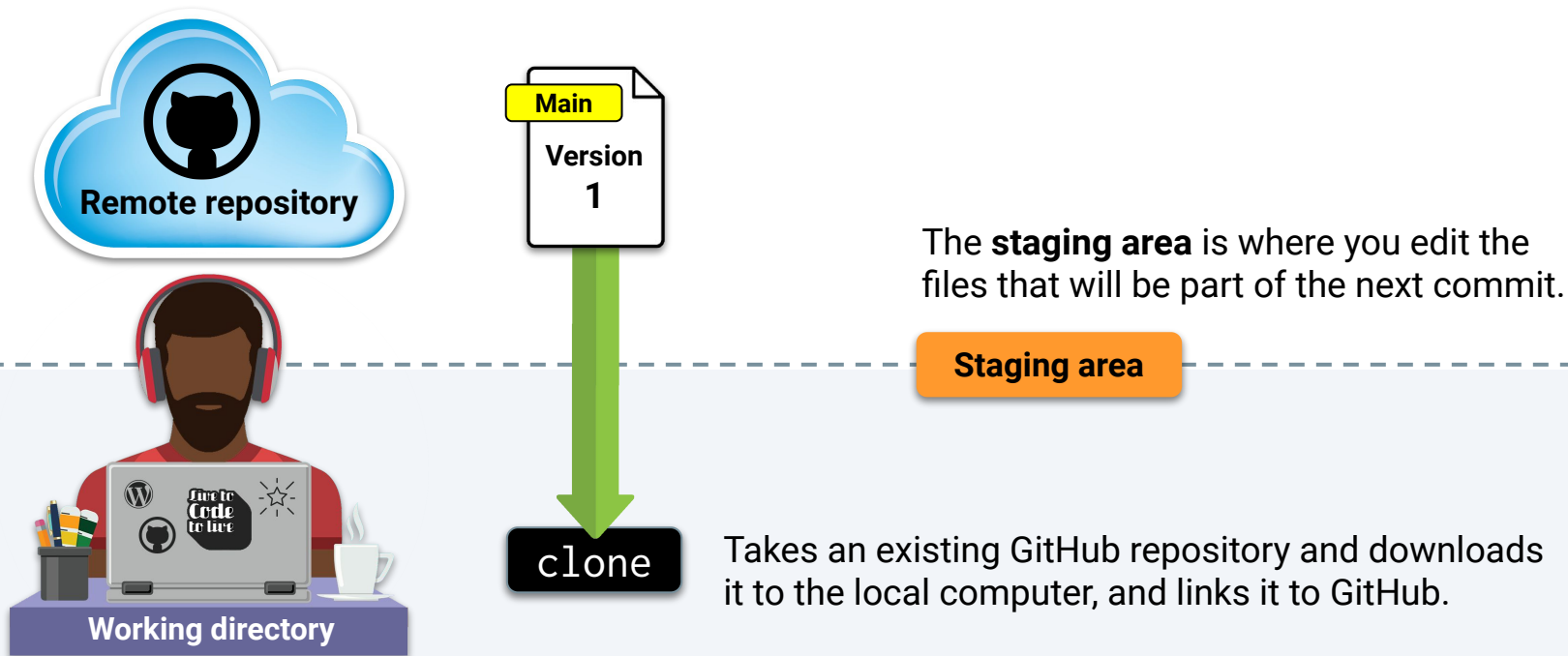


If we break something while working on our code, this system allows us to restore working code from earlier. Since Git remembers these checkpoints, we can work on several different concerns all at once.

# Git Version Control

**Scenario:** Your group has been working with Uber's rider data, and you've decided to analyze the average age of the riders:

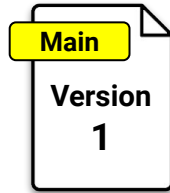
The root code for the project is called **main**.





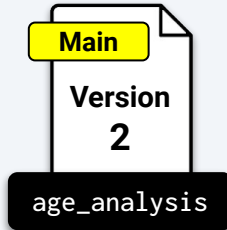
# Git Version Control

Git essentially allows us to write this code and save it with the name `age_analysis`.



The **staging area** is the where you edit the files that will be part of the next commit.

**Staging area**



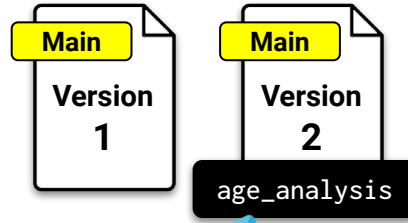
`git commit`



Your staged changes are saved once you commit.

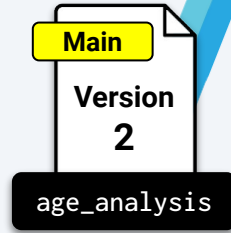
# Git Version Control

**age\_analysis** is a branch that originates from the main branch. It contains updates that will be added to the main branch when it's ready to **merge**.



The **staging area** is the where you edit the files that will be part of the next commit.

**Staging area**



# Popular Git CLI Commands

---

<code>git clone</code>	Clones a git repository onto the local file system.
<code>git add</code>	Adds changed files to the queue of tracked files ready to be committed.
<code>git commit</code>	Adds tracked files as a bulk checkpoint ready to be pushed to the remote git repository.
<code>git push</code>	Uploads changed files from the local git repository to the remote git repository and updates the remote files.
<code>git pull</code>	Downloads changed files from the remote git repository to the local git repository and updates the local files.

A commit in GitHub is like a snapshot of what your project or file looks like at a particular moment in time. If a file doesn't contain any changes, the file is not stored again; instead, Git provides a link to the identical file that it previously stored.





# Time to Code

## Adding Files from the Command Line

Suggested Time:

---

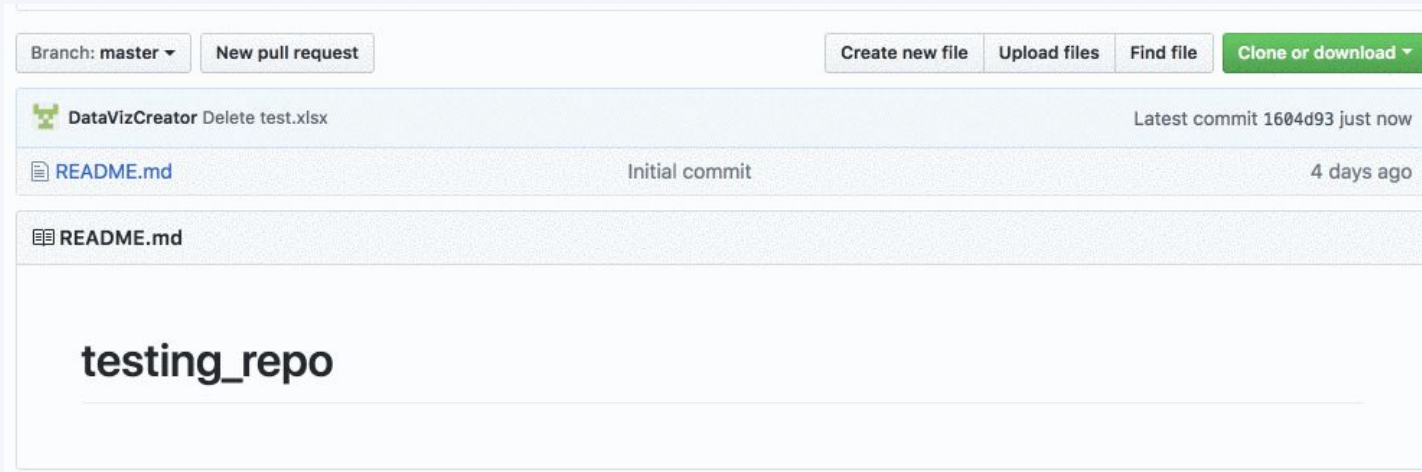
10 Minutes

# Activity: Adding Files from the Command Line

## Instructions

Create a new repo.

From the repo page, click the green box in the top-right labeled "Clone or download", select "Use SSH", and copy the link to the clipboard, as captured in the following GIF:



# Activity: Adding Files from the Command Line

---

## Instructions

Open Terminal (or Git Bash for Windows users), and navigate to the home folder using `cd ~`.

Type `git clone <repository link>` in the terminal to clone the repo to the current directory. Once this code has run, everyone should find a folder with the same name as the repo:

```
$ git clone git@github.com:DataVizCreator/testing_repo.git
```

Open the folder in VS Code, and create two python script files, named `script01.py` and `script02.py`.

# Activity: Adding Files from the Command Line

---

## Instructions

Then, open Terminal/Git Bash, and navigate to the repo folder. Run the following lines:

```
# Displays that status of files in the folder  
git status
```

```
# Adds all the files into a staging area  
git add .
```

```
# Check that the files were added correctly  
git status
```

```
# Commits all the files to your repo and adds a message  
git commit -m <add commit message here>
```

```
# Pushes the changes up to GitHub  
git push origin main
```

Navigate to the repo on [Github.com](https://github.com) to confirm that the changes have been pushed up.





# Activity: Adding More to the Repo

In this activity, you will make or add changes to the repo that we just created.

Suggested Time:

15 Minutes

# Activity: Adding More to the Repo

---

## Instructions

Using the repo that we just created, make or add the following changes:

- Add new lines of code to one of the Python files
- Create a new folder.
- Add a file to the newly created folder.
- Add, commit, and push the changes.
- Delete the new folder.
- Add, commit, and push the changes again.



Time's Up! Let's Review.

# Questions?



*The  
End*