Session 3

Git / Github

Basics

Flows

Practice Run

9.3.19

Link to Jupyter Notebooks:
https://mybinder.org/v2/gh/data-voyage-solutions/oag-session-mats/master

01

Wrap-up Prev. Session

2 hours

02

Git / GitHub

1 hour

03

Data Sets

5 min.

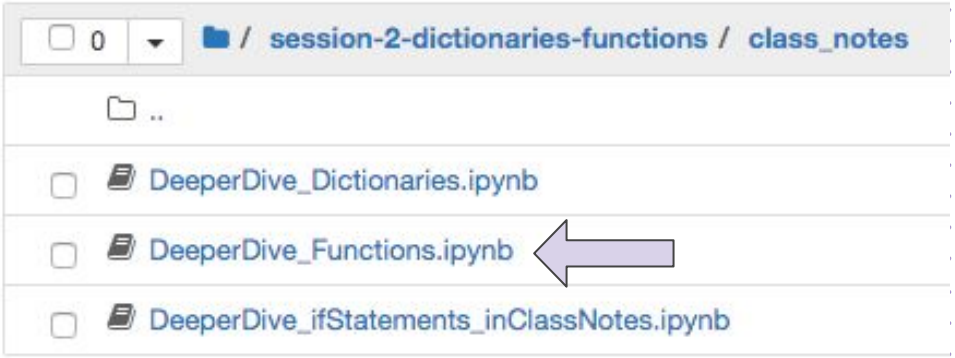
04

Session 2 Practice

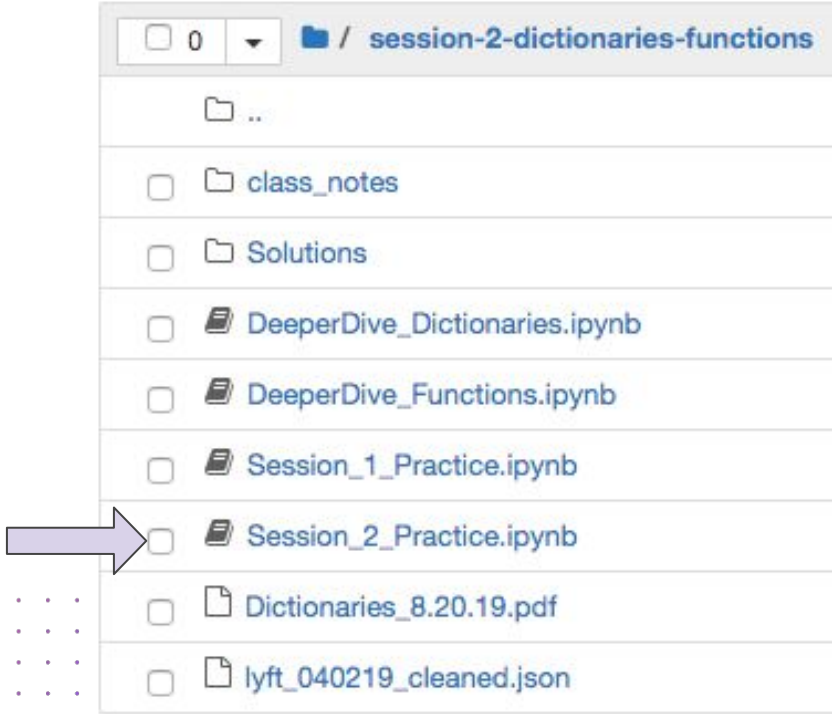
Remaining time

Pick up from last session

Functions



Practice Session 2 Material



Functions

Let's Consider a Repetitive Program…

Consider a program that prints a $5 shipping charge for products on a website:

```
print("You've purchased a Hanging Planter.")
print("Thank you for your order. There will be a $5.00 shipping charge for this order.")

# 10 minutes later...
print("You've purchased a Shell Mirror.")
print("Thank you for your order. There will be a $5.00 shipping charge for this order.")

# 5 minutes later...
print("You've purchased a Modern Shag Rug.")
print("Thank you for your order. There will be a $5.00 shipping charge for this order.")
```

What if there are 1,000 orders?

Seeing Functions in Action

So we *define* the function, then we can call the function by pairing its name with the parenthesis: `print_order()`.

```
def print_order():
 print("Thank you for your order. There will be a $5.00 shipping charge for this order.")

print("You've purchased a Hanging Planter.")
print_order()

print("You've purchased a Shell Mirror.")
print_order()

print("You've purchased a Modern Shag Rug.")
print_order()
```

Functions

We can write a **function** to print the order.

A function is simple — it's a reusable piece of code. We only define it once. Later, we can use its name as a shortcut to run that whole chunk of code.

- Functions are defined using the def syntax.
  - `def` stands for "define."
- In this case, we're *defining* a function named `function_name()`.

```
def function_name():
    # What you want the function to do.

# Call the function by name to run it:
function_name()
```

Pro tip: Don't forget the `()`, and be sure to indent!

Git
Version Control

Resources for *The Basics*

- https://try.github.io/
  - https://github.com/jlord/git-it-electron#what-to-install
  - https://learngitbranching.js.org/

What's the point?

Git is a program for keeping track of changes over time, known in programming as ***version control***.

If you've used a track changes feature in a text editing software then you're already familiar with the concept!

Lingo: Repository

- Collection of related files for a project.
- Think of it as a **project folder** that is tracked by Git.
- Called "repo" for short.

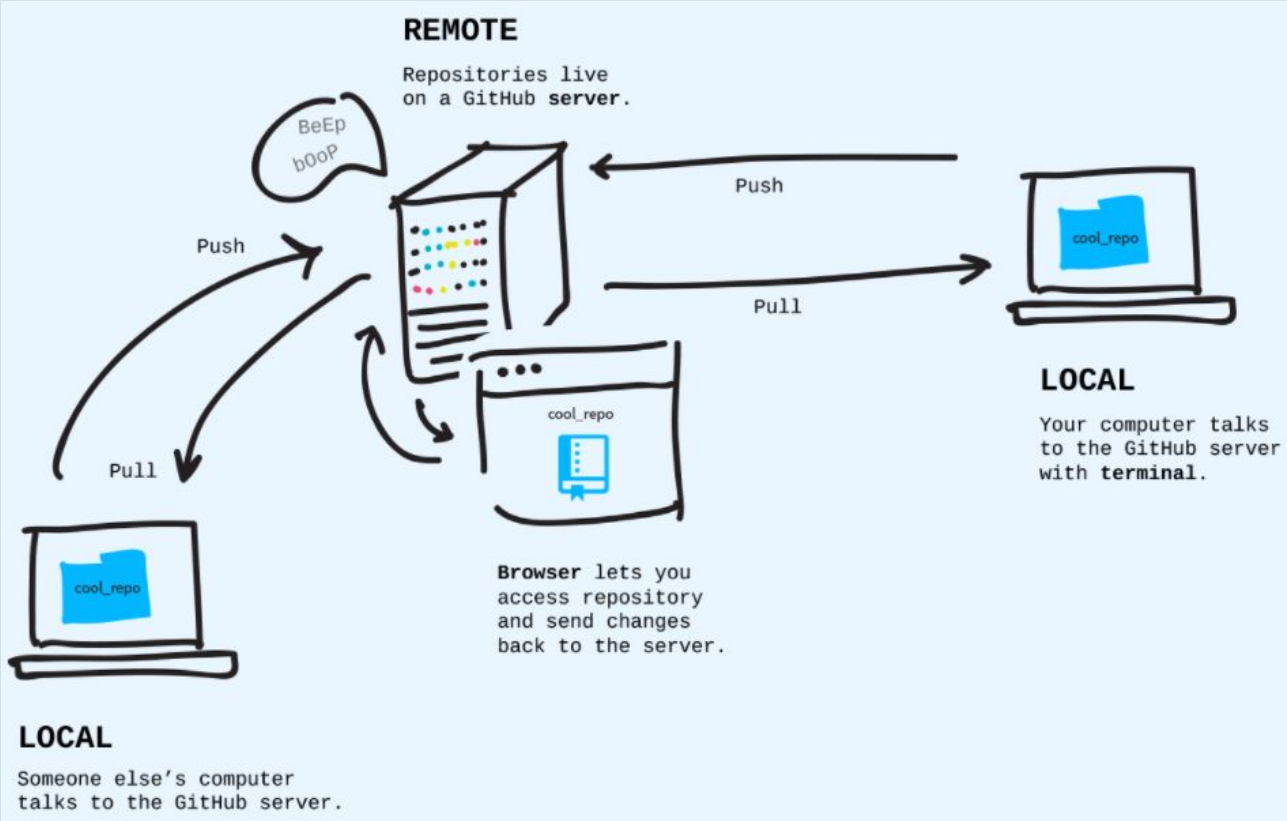
In order for you to be able to share and collaborate with others (without giving them access to your computer), you use GitHub.

- GitHub acts as a central repository for you and everyone else to share.
- Push changes to it and pull down changes from others.

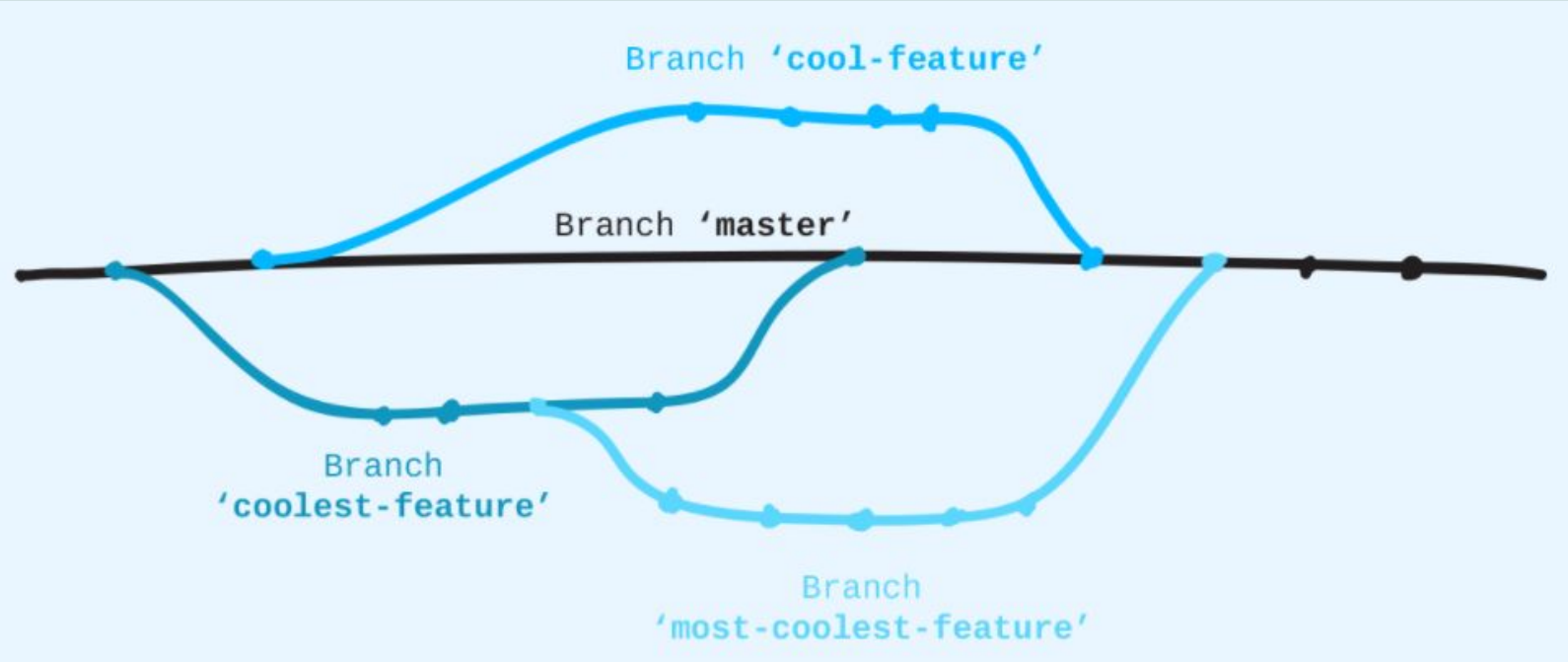
Lingo: Remote Repository

- A repo that lives on one of GitHub's servers.
- By **pushing your local changes** to a remote, you are updating the remote repo.
- By **pulling** your updated changes **down from the remote** repo, collaborators can get the latest from your work.

Diagram about Repos



Feature Branch Workflow



Our Git Workflows

- Functions folder/Project Templates
- Team Member A's projects
- Team Member B's projects

- Master and "DA Stage" Branch
  - Project A
    - Branch: EDA
    - Branch: Cleaning/Preprocessing
    - Branch: Analysis 1
  - Project B
  - Project C

NOTE:

We will use **`git rebase`** for our merges.

Let's have a practice run!

Assuming Git is installed and already configured on your local computer:

- ❏ Open terminal or shell
- ❏ Navigate to a desired ***parent*** directory
- ❏ One way to set-up: Clone a remote repo on GitHub
- ❏ Navigate to cloned repo on your local computer
- ❏ Make some changes to the local repo
- ❏ Push changes to the remote repo:
  - ❏ git status
  - ❏ git diff
  - ❏ git add <filename> or .
  - ❏ git commit -m "ur commit msg" (aka save history...with a short message)

“Round Robin” Game

1. Starting spot: https://github.com/orgs/data-voyage-solutions/dashboard
2. Create a new remote repo
3. Add collaborators
4. Kelly starts the round:
    a. **git clone a remote repo to local**
    b. **make some changes and save**
    c. **save history of changes**
    d. **push changes to remote repo from local (update remote repo)**
5. Next person up! Complete #4 steps, one person at a time.

“Round Robin” Game -- Round 2

Once Round 1 has been completed:

1. Kelly starts the round:
    a. **Check status of local repo**
    b. **Do a `git pull`! It’s like an update...**
    c. **Check the logs....vs a diff**
2. Everyone else, at the same time (except the last person that pushed changes)! Complete #1 steps.