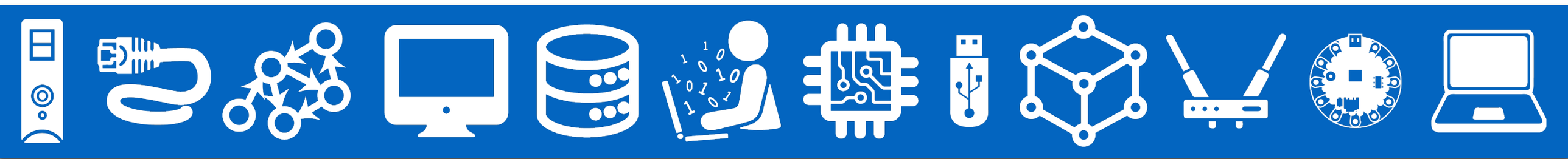
CS134:
Python Types and Expressions



Announcements & Logistics

- **HW 1** due today at 10 pm (Google form)
- **Lab 1** today/tomorrow, due Wed/Thur at 10pm
  - Goal: Setup computers, gain experience with the workflow and tools
  - Start with some short and sweet Python programs
  - *Important*: Login to Lab machines using **OIT credentials**
    - `clone/pull/push` to `evolene.cs.williams.edu` with **CS credentials**
      - You must have received an email about CS account info!
- **Student help hours and TA hours have started**
  - Check calendar on course webpage
- **Questions?**

Announcements & Logistics

- **POGILs:**
  - I don't collect them! They're practice for you!
    - Feel free to make mistakes, try things out!
    - Make your best guess! You'll remember guessing and being wrong better than me just telling you the content.
  - We often won't get through the entire activity: finish them outside of class
    - More practice = more learning
  - We don't have answer sheets for POGILs:
    - Encourages you to ask the computer
    - ...or to reach out and talk to peers, TAs, instructors, etc.
    - *Engage with the course content!*

Announcements & Logistics

- **Lecture Videos:**
  - We'll try to post lecture recordings or videos, as time allows
    - (although, it's been my experience that these are under-used)
  - They'll show up on the 'Lectures' page on the course website

CSCI 134

Introduction to Computer Science

Home | Lectures | Assignments | Resources | Williams CS

Course Schedule

The table below lists the topics we will discuss and any items associated with each class/date. Some of these resources will only be accessible from within the campus network. Information about the proxy server can be found here.

The schedule is subject to change: we may wish to explore new topics in response to current events or student interest. You're more than welcome to work ahead, but please check with us first!

Mon	Tue	Wed	Thu	Fri
9/2	9/3	9/4	9/5	9/6 **Welcome & overview** Course Syllabus Slides Activity Homework 1
9/9 **Expressions**	9/10	9/11 **Functions**	9/12	9/13 **Booleans**
9/16	9/17	9/18	9/19	9/20

Last Time

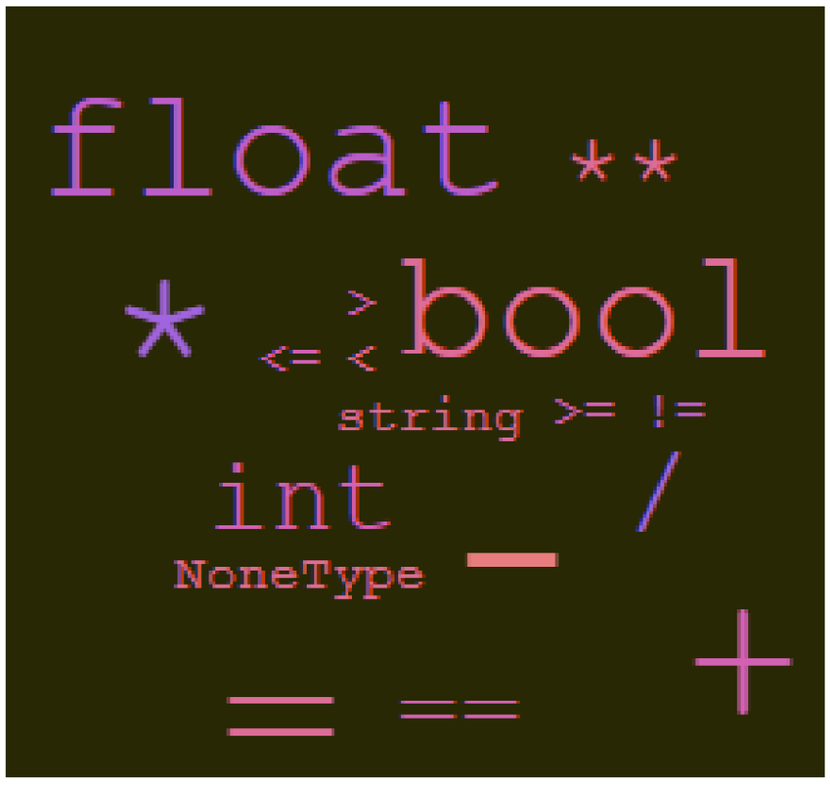
- Discussed course logistics
- **Reviewed** syllabus
- Important take-aways:
    - cs134 course website: place where everything is hosted
- Encouraged to use lab machines but resources to setup your personal machines are available on the website
    - Reach out to us or TAs if you get stuck

Today's Plan

- Learn lots of new vocabulary words!
- Discuss **data types** and **variables** in Python
  - `int, float, boolean, string`
- Learn about basic **operators**
  - arithmetic, assignment
- Experiment with built-in Python **functions** and expressions
  - `int(), input(), print()`
- Investigate different ways to run and interact with Python

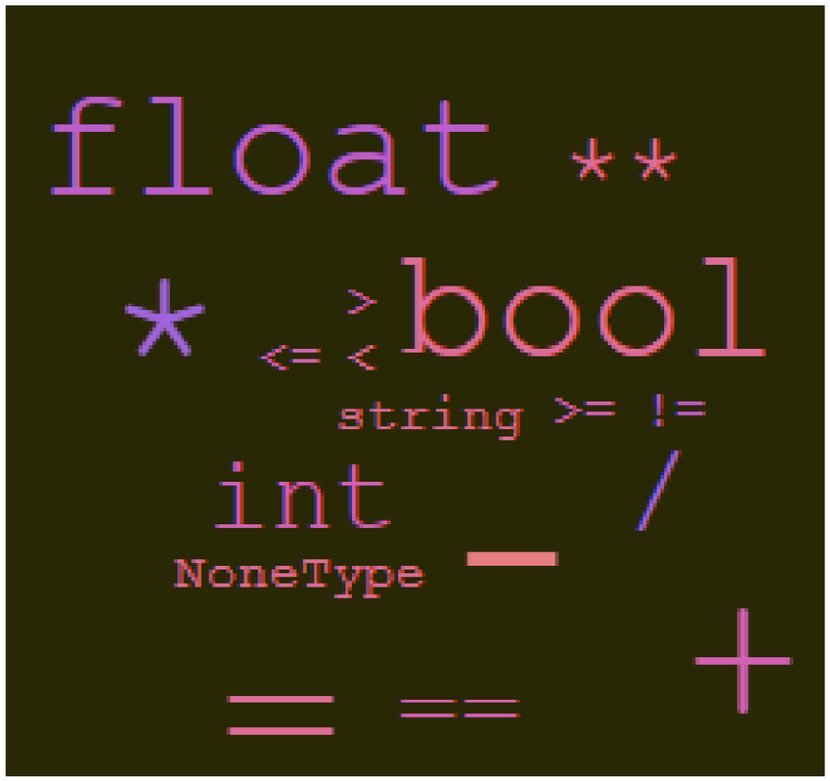
Aspects of Languages

- **Primitive constructs**
  - English:
    - words, punctuation
  - Programming languages:
    - numbers, strings, simple operators



Aspects of Languages

- **Syntax**
  - English:
    - "boy dog cat" (incorrect), "boy hugs cat" (correct)
    - "Let's eat grandma!" (probably incorrect), "Let's eat, grandma!" (correct)
  - Programming language:
    - `"hi"5` (incorrect), `4*5` (correct)



Aspects of Languages

- **Semantics** is the meaning associated with a syntactically correct string of symbols
  - **English:**
    - Can have many meanings (ambiguous), e.g.
    - "Flying planes can be dangerous"
    - Other examples?
  - **Programming languages:**
    - Must be *un*ambiguous
    - Can only have one meaning
    - Actual behavior is not always the intended behavior!

Python3

- Programming language used in this course
- Great introductory language
    - Better human readability and user friendly syntax than other PLs
- For this class, we need `Python 3.10`
- Checking version of Python on machine
    - Type `python3 --version` in Terminal
        - (VS Code Terminal for Windows)
- **Preinstalled on all lab machines**
- Installing Python3 on your machine: see setup guide on webpage

Interacting with Python

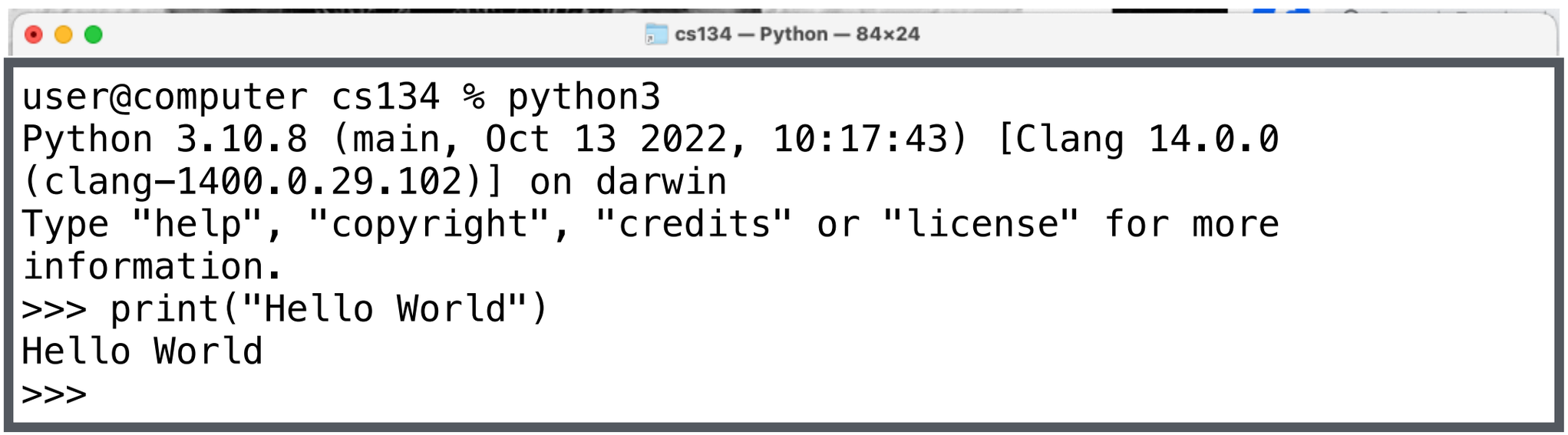
- You can run Python code in two ways:
  - As a **script**
    - Save code in a file, run from Terminal
  - **Interactively** (from Terminal)
    - Interactive session

Python: Program as a Script

- A **program** is a sequence of definitions and commands
  - Definitions are evaluated
  - Commands are executed and instruct the interpreter to do something
- Type instructions in a **file** that is read and evaluated sequentially
  - e.g., last lecture we wrote `helloworld.py` in a file and then executed it from the Terminal with
    `python3 helloworld.py`
  - **Standard method**: good for longer pieces of code or programs
  - We will use this method in our labs
  - Called "running the Python program as a *script*"

Python: Interactive

- Running Python **interactively** is great for introductory programming
- Launch the Python interpreter by typing `python3` in the Terminal
    - Opens up Interactive Python
    - Almost like a "calculator" for Python commands
    - Takes a Python expression as input and spits out the results of the expression as output
    - Great for trying out short pieces of code



Python Primitive Types

- Every data **value** has a data **type**. For example:
  - `10` is an integer (type: **int**)
  - `3.145` is a decimal number (type: **float**)
  - `'Williams'` or `"Williams"` is a sequence of characters (type: **string**)

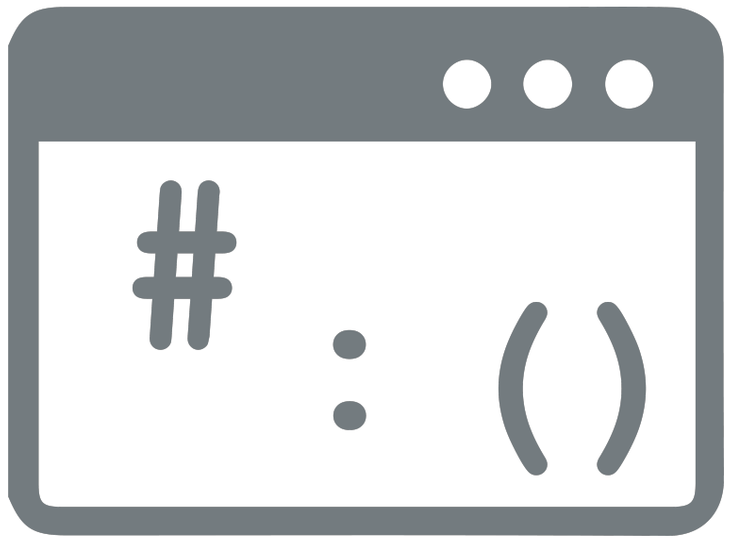
Knowing the **type** of a **value** allows us to choose the right ***operator*** for expressions.

Python Primitive Types

- Every data **value** has a data **type**. For example:
  - `10` is an integer (type: **int**)
  - `3.145` is a decimal number (type: **float**)
  - `'Williams'` or `"Williams"` is a sequence of characters (type: **string**)
  - `0` (`False`) and `1` (`True`) (type: **boolean** or **bool**)
    - Represent answers to decision questions (yes/no)
  - *Empty value* (type: **None**)
- We will revisit booleans and None types soon!

Knowing the **type** of a **value** allows us to choose the right ***operator*** for expressions.

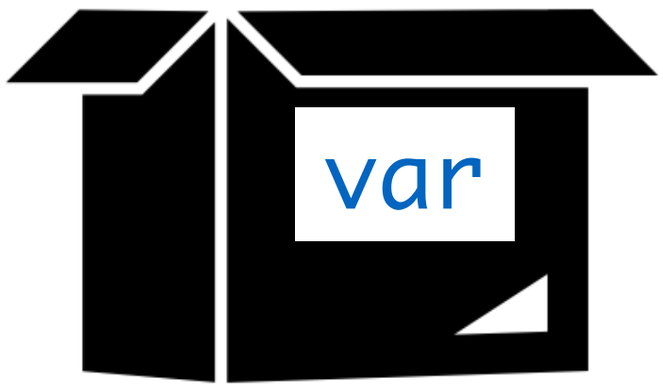
Examples

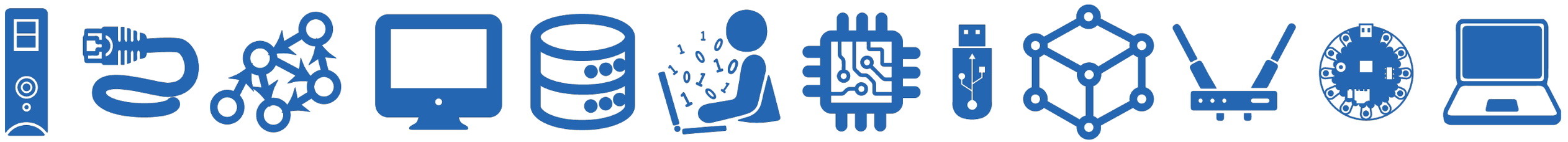


Python Operators

- **Arithmetic operators:**
  - **+** (addition), **-** (subtraction), ***** (multiplication)
  - **/** (floating point division, returns a value with a decimal point)
  - **//** (integer division, returns an integer)
  - **%** (modulo, or remainder)
  - ****** (power, or exponent)
- **Assignment operator:**
  - **=** ("is assigned or gets", not "equals")
  - Used to "assign" values to **variables**
  - ***Note.*** Not to be confused with mathematical equality, which is written as **==** in programming languages

Variables & Assignment



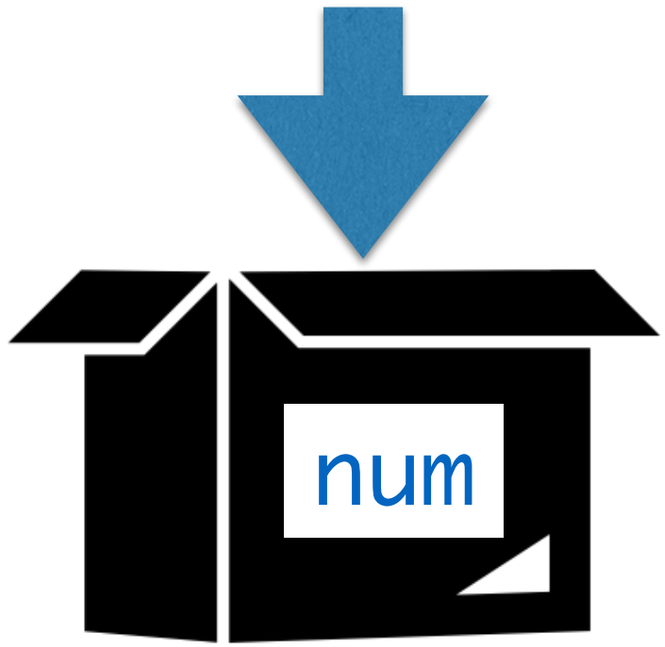


Variables and Assignments

- A **variable** names a value that we want to use later in a program
  - If we define `num = 17` then the value `17` essentially gets stored in a slot in memory with the label `num`
  - We are **assigning** `num` (a variable) the value `17`

```
>>> num = 17
```

17



Variables and Assignments

- A **variable** names a value that we want to use later in a program
  - If we define `num = 17` then the value `17` essentially gets stored in a slot in memory with the label `num`
  - We are **assigning** `num` (a variable) the value `17`

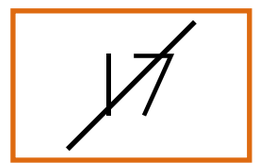
- Once defined, we can reuse variable names again, and later assignments can change the value in a variable box
  - `num = num - 5`
  - What is stored in `num` after this evaluates?

Math vs Programming. An assignment: expression on the right evaluated first and the value is stored in the variable name on the left

Variables and Assignments

- A **variable** names a value that we want to use later in a program
  - If we define `num = 17` then the value `17` essentially gets stored in a slot in memory with the label `num`
  - We are **assigning** `num` (a variable) the value `17`

- Once defined, we can reuse variable names again, and later assignments can change the value in a variable box
  - `num = num - 5`
  - What is stored in `num` after this evaluates?
  - var = <expression> (result of expression gets stored in the variable box var)



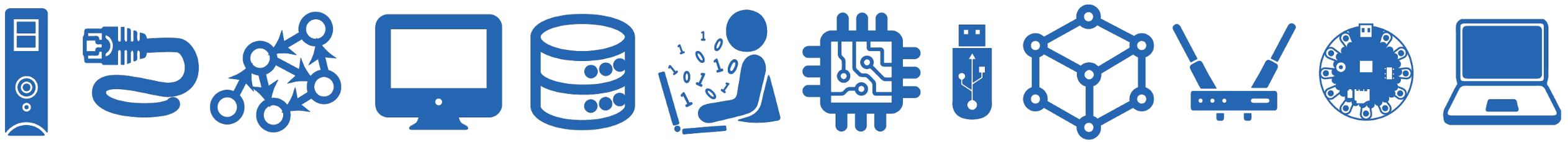
- **Question**. *Why would we want to name values or expressions?*

Abstracting Expressions

- Why give names to data values or the results of expressions?
  - To **reuse** names instead of values
  - Easier to change code later
- For example:

```
pi = 3.1415926  # useful to name
radius = 2.2
area = pi * (radius**2)
# suppose now we want to change radius
radius = 2.2 + 1
area = pi * (radius**2) # new area
```

Python Built-in Functions



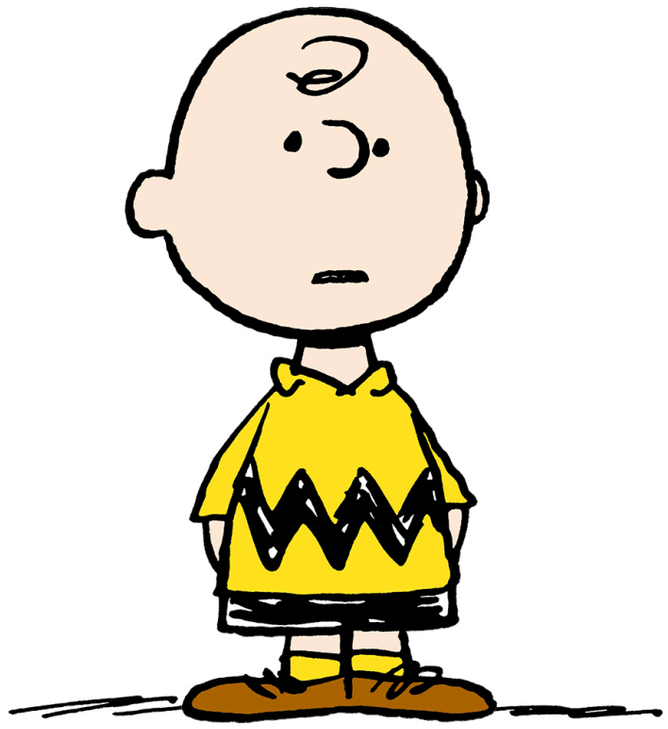
Built-In Functions

- Python comes with a ton of built-in capabilities in the form of **functions**
    - We will discuss the following built-in functions today
    - `input(), print()`
    - `int(), float(), str()`
- Will formally discuss functions on Friday

Built-in functions: input()

- `input()` displays its single argument as a prompt on the screen and waits for the user to input text, followed by **Enter/Return**
- ***Important***: interprets the entered value as a **string**

```
>>> input('Enter your name: ')
Enter your name: Charlie Brown
'Charlie Brown'
>>> age = input('Enter your age: ')
Enter your age: 8
>>> age
'8'
```



Prompts in Maroon. User input in blue.
Inputted values are by default a **string**

Built-in functions: print()

- `print()` displays a character-based representation of its argument(s) on the screen/Terminal.

```
>>> name = 'Peppermint Patty'
>>> print('Your name is ' + name)
Your name is Peppermint Patty
```

Need a space at end for string *concatenation*

```
>>> age = input('Enter your age : ')
Enter your age: 7
>>> print('The age of ' + name + ' is ' + age)
The age of Peppermint Patty is 7
```

Built-in functions: int()

- When given a string that's a sequence of digits, optionally preceded by `+/-`, `int()` returns the corresponding integer
- On any other string it raises a `ValueError`
- When given a float, `int()` returns the integer that results after truncating it towards zero
- When given an integer, `int()` returns that same integer

```
>>> int('42')
42
>>> int('-5')
-5
>>> int('3.141')
ValueError
```

Built-in functions: float()

- When given a string that's a sequence of digits, optionally preceded by `+/-`, and optionally including one decimal point, `float()` returns the corresponding floating point number.
- On any other string it raises a `ValueError`
- When given an integer, `float()` converts it to a floating point number.
- When given a floating point number, float returns that number

```
>>> float('3.141')
3.141
>>> float('-273.15')
-273.15
>>> float('3.1.4')
ValueError
```

Built-in functions: str()

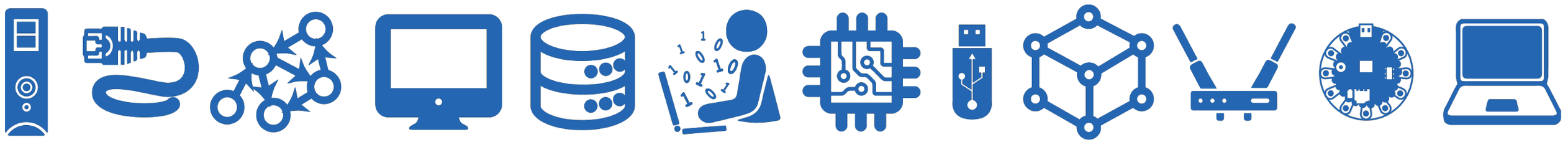
- Converts a given type to a **`string`** and returns it
- Returns a syntax error when given invalid input

```
>>> str(3.141)
'3.141'
>>> str(None)
'None'
>>> str(134)
'134'
>>> str($)
SyntaxError: invalid syntax
```

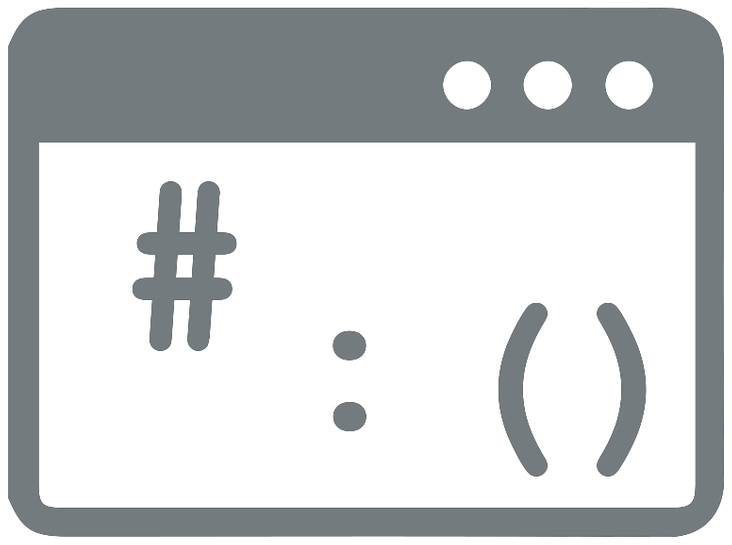
[Aside] Comments and Indenting

- Anything after **#** in Python is a comment
  - Ignored by the interpreter
  - Meant for humans reading the code
  - Useful for readability for large pieces of code
- Python is sensitive to **indentation**
  - Signify start of new "code block"
  - We will see how to use indents more in the coming lecture

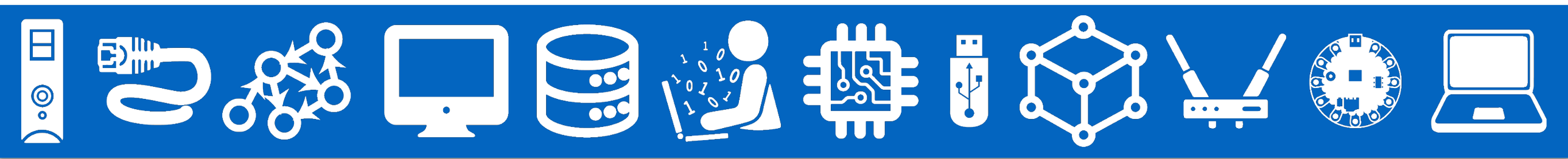
Practice in Interactive Python



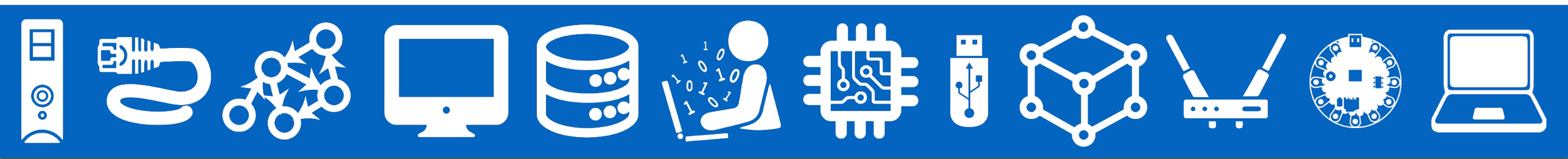
Examples: Interactive Python



The end!



CS134:
Lab 01



Today's Plan

Tools & resources for doing CS Labs!

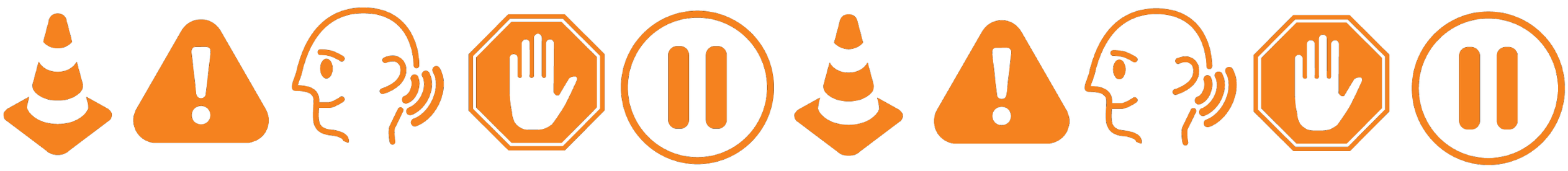
After this lesson, you should be able to:

1. Explain the difference between VS Code, Terminal, and python
2. Navigate your file structure with Terminal
3. Submit your lab assignments to be graded

Lab Deadlines

- Depend on your lab session day!

- Mondays —> Wednesday @10p
- Tuesdays —> Thursday @10p



Lab Instructions on Course Website!

- https://bit.ly/cs134f24

Labs

Lab assignments are due Wednesday 10 pm EST for students in Monday lab sections, and Thursday 10 pm EST for students in Tuesday lab sections. To request a 12-hour extension, please fill out this form *before* your lab's submission deadline.

Lab Date	Topic
Sept 09/10	Lab 1: Python/Git Workflow
Sept 16/17	Lab 2: Day of the Week
Sept 23/24	Lab 3: Madlibs
Sept 30/Oct 1	Lab 4: Voting
Oct 21/22	Lab 5: What's In A Name?
Oct 28/29	Lab 6: Thinking Recursively
Nov 4/5	Lab 7: AutoComplete
Nov 11/12	Lab 8: Boggle! [Part A] [Part B] [Graphics module]
Dec 2/3	Lab 9: Sorting

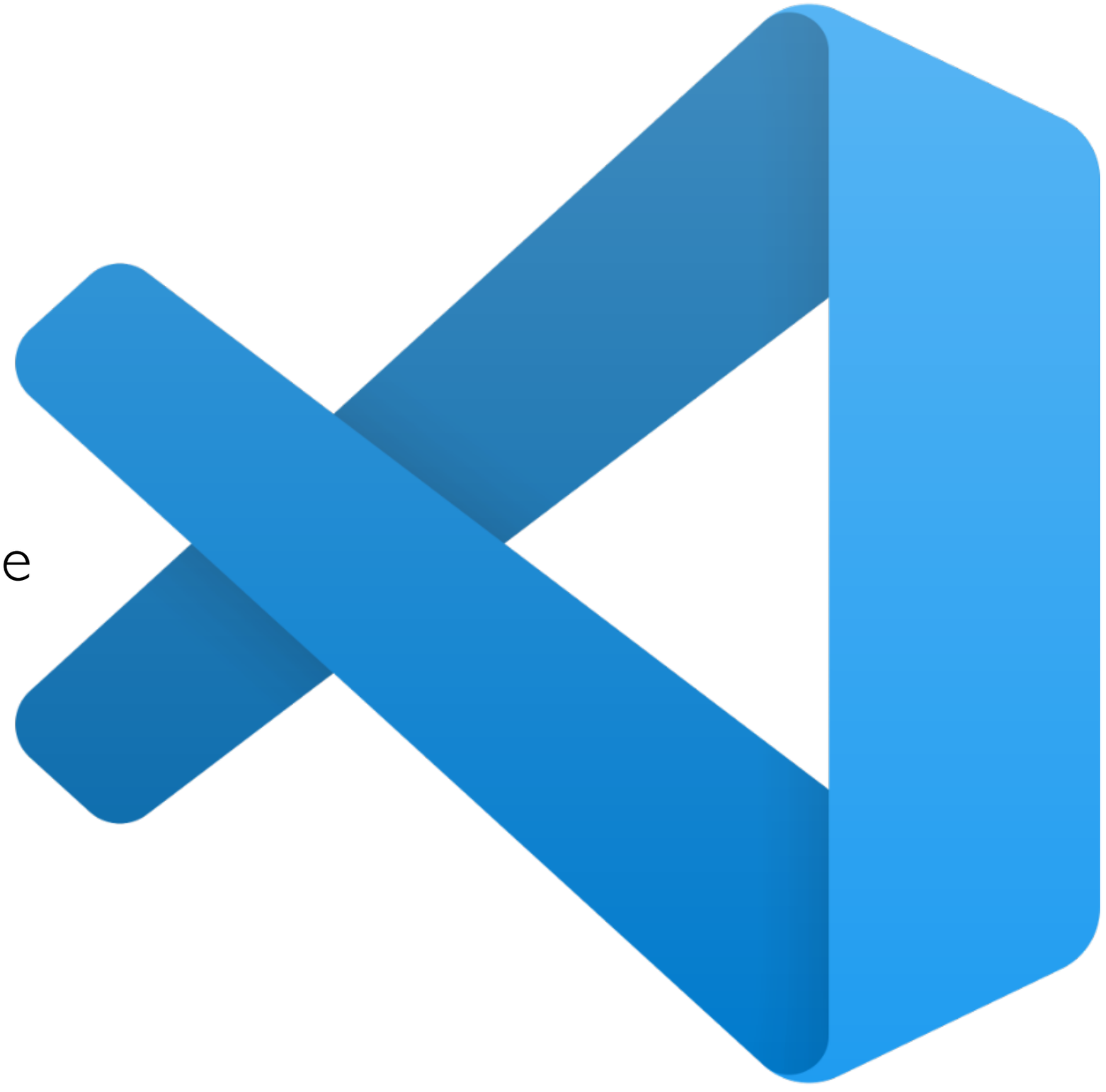
What do these 4 applications have in common?

Google Docs

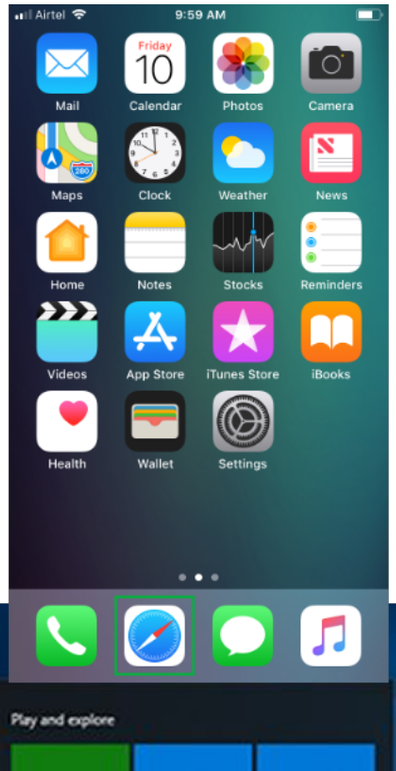
What do these 4 applications have in common?

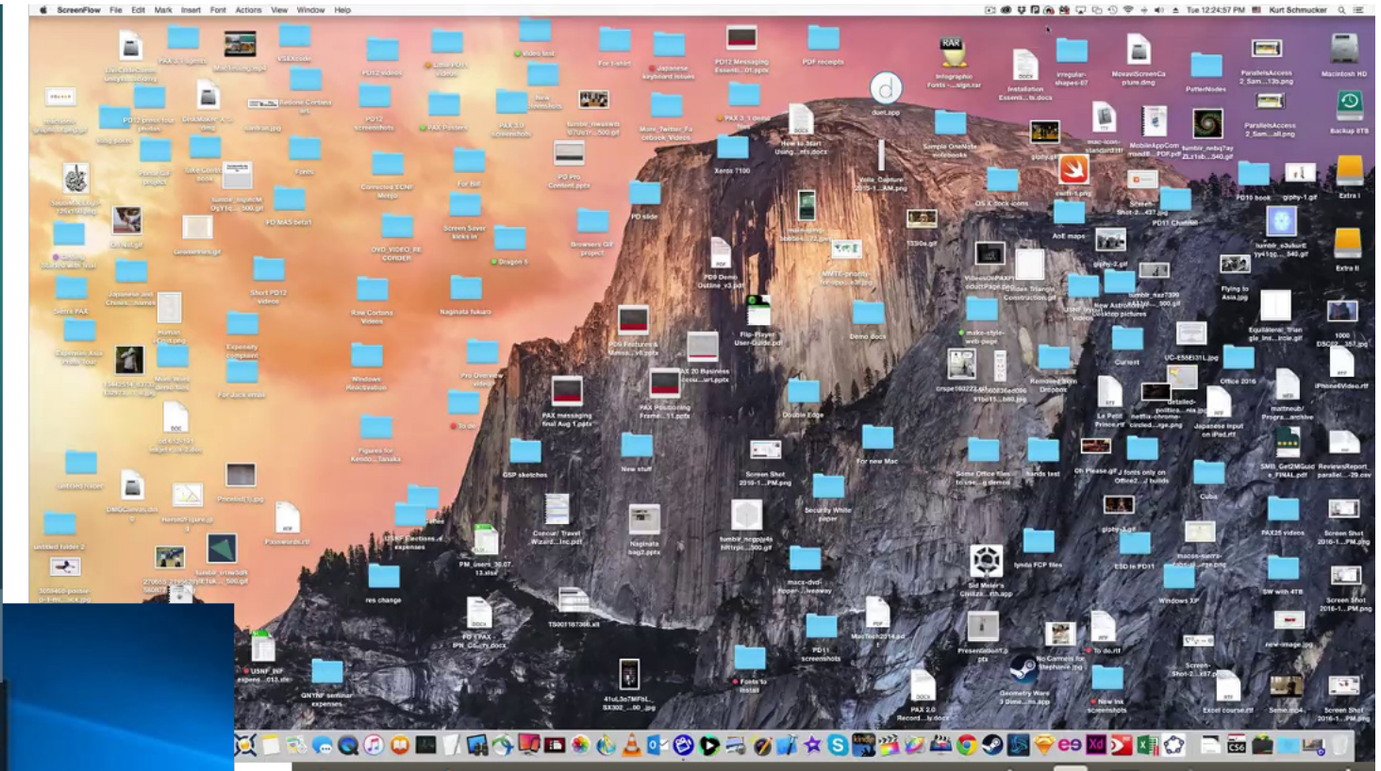
VS Code

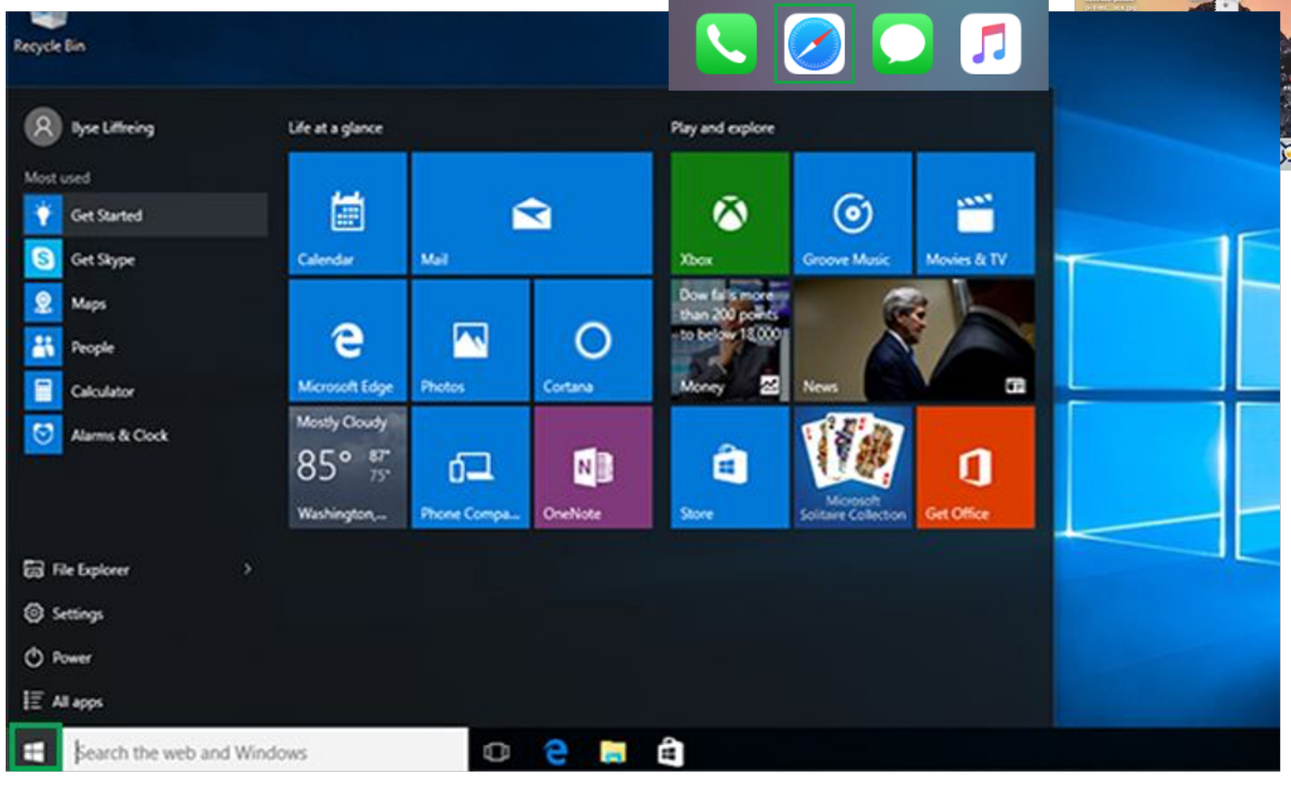
Visual Studio Code

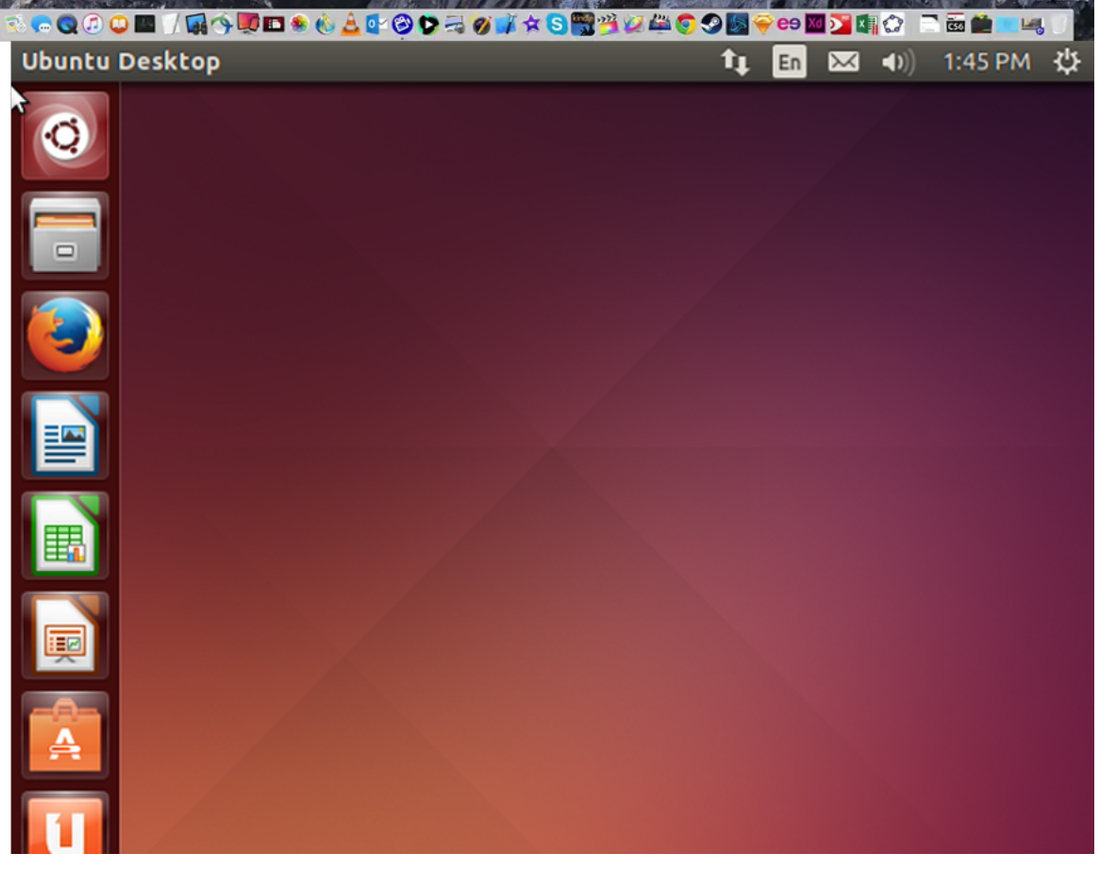


Similar…

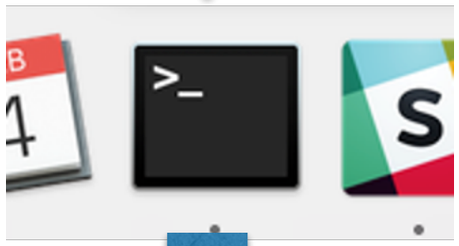

Airtel
9:59 AM
Mail
Calendar
Photos
Camera
Maps
Clock
Weather
News
Home
Notes
Stocks
Reminders
Videos
App Store
iTunes Store
iBooks
Health
Wallet
Settings


ScreenFlow File Edit Mark Insert Font Actions View Window Help
Tue 12:24:57 PM Kurt Schmucker


Recycle Bin
Ilyse Liffreing
Most used
Get Started
Get Skype
Maps
People
Calculator
Alarms & Clock
Life at a glance
Play and explore
Calendar
Mail
Xbox
Groove Music
Movies & TV
Microsoft Edge
Photos
Cortana
Money
News
Mostly Cloudy
85°
87°
75°
Washington,...
Phone Compa...
OneNote
Store
Microsoft Solitaire Collection
Get Office
File Explorer
Settings
Power
All apps
Search the web and Windows

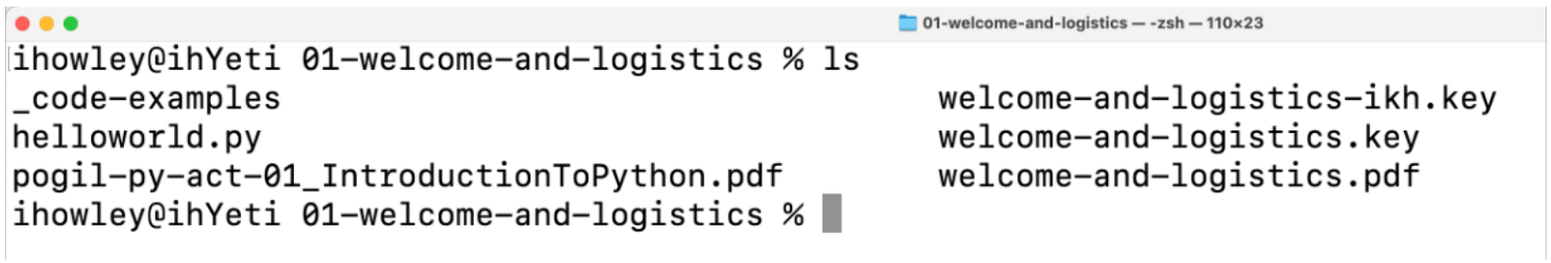

Ubuntu Desktop
En
1:45 PM

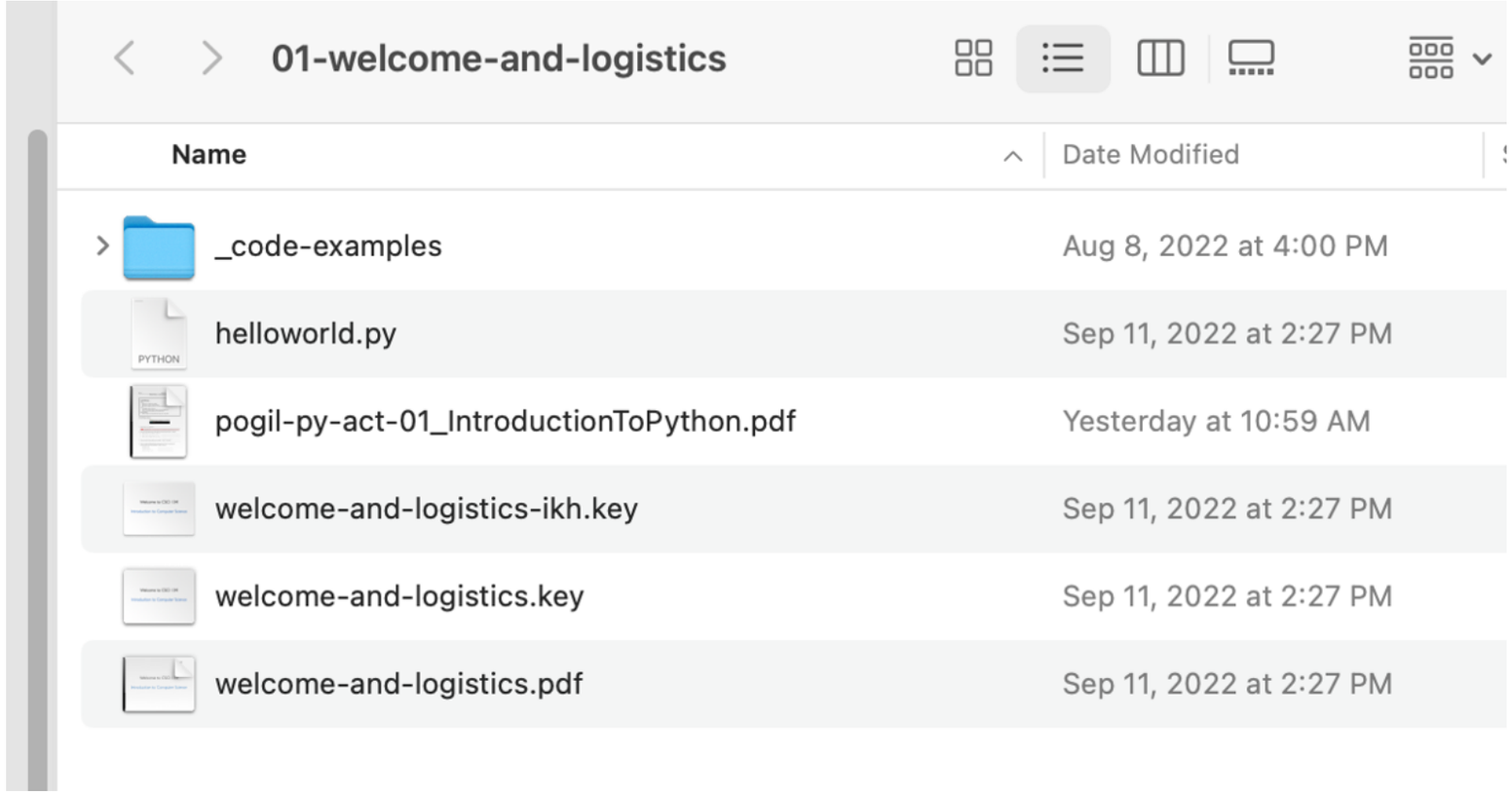
Terminal



Terminal vs. File Explorer

- Terminal is a text-based view of your **directory** structure
  - Like File or Explorer / Windows Explorer, but in text



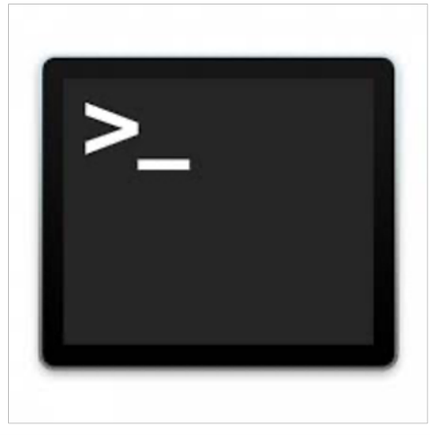


Python

- Written into a **script** and then interpreted by Python
  - `python3 hello.py`

- **Interactive** mode
  - `python3`
  - `>>> print(“hello!”)`

- Terminal & Python are two separate apps!



Terminal Keyboard Shortcuts

- Type a letter or two, then hit 

–> Tab Complete!

-  —> Cycles through all previous Terminal commands!

Useful Keyboard Shortcuts

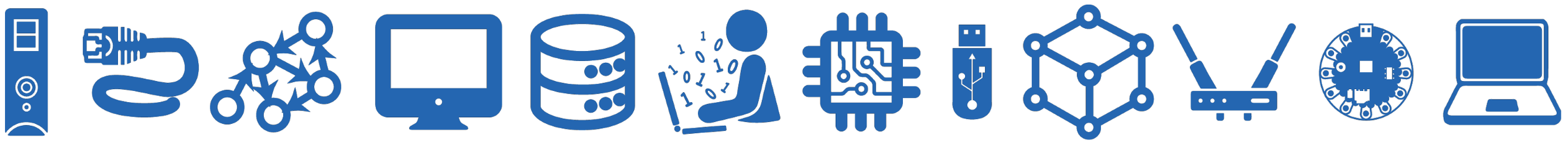
- ⌘ (command) + C → Copy
- ⌘ (command) + X → Cut
- ⌘ (command) + V → Paste
- ⌘ (command) + A → Select All
- ⌘ (command) + S → Save
- ⌘ (command) + R → Refresh
- ⌘ (command) + Z → Undo

- Ctrl+ A → First character of line
- Ctrl + E→ End of line
- Option +  → Right one word
- Option +  → Left one word

Useful Terminal/Unix Commands

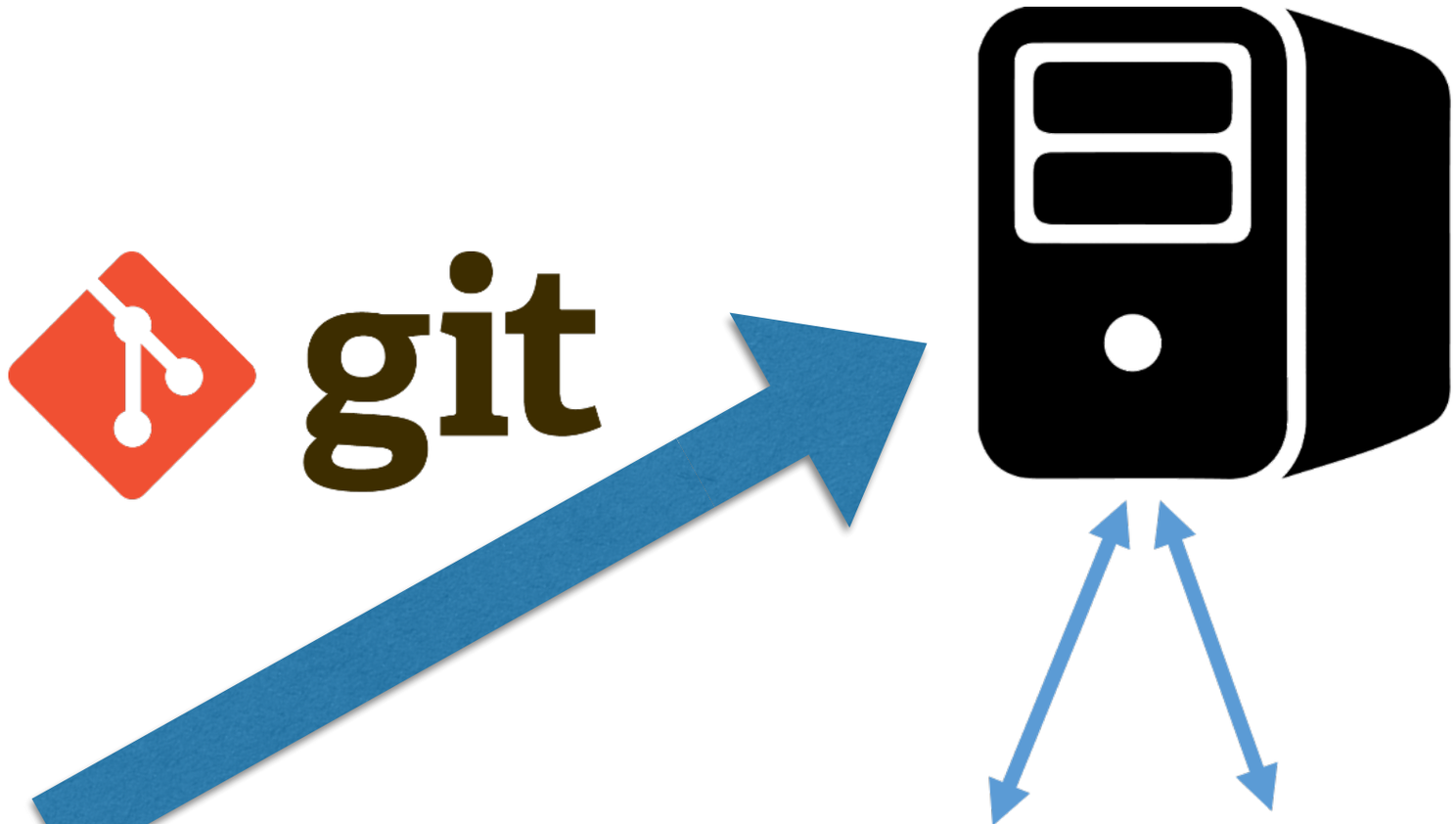
- `pwd` - print working directory
- `mkdir <dir name>` - make new directory (or folder)
- `cd <dir name>` - change directory (like moving into a folder)
- Special directory names in Unix
  - single dot, current directory
  - `..` two dots, parent directory
  - `~` tilde, home directory
- `cd ..` - takes you to the parent directory
- `cd` - takes you "home"
- `ls` - shows contents of current directory

git

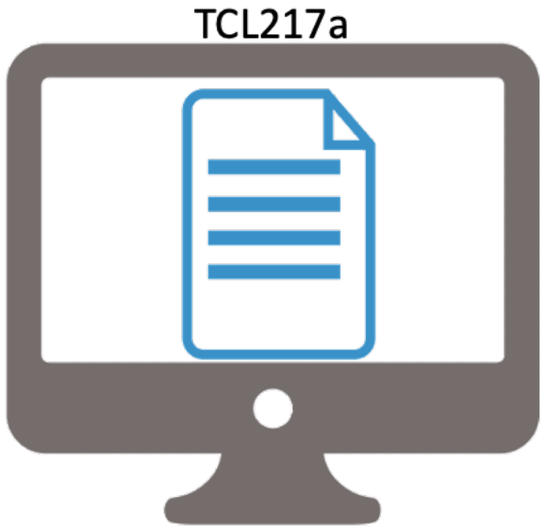


Working on Labs

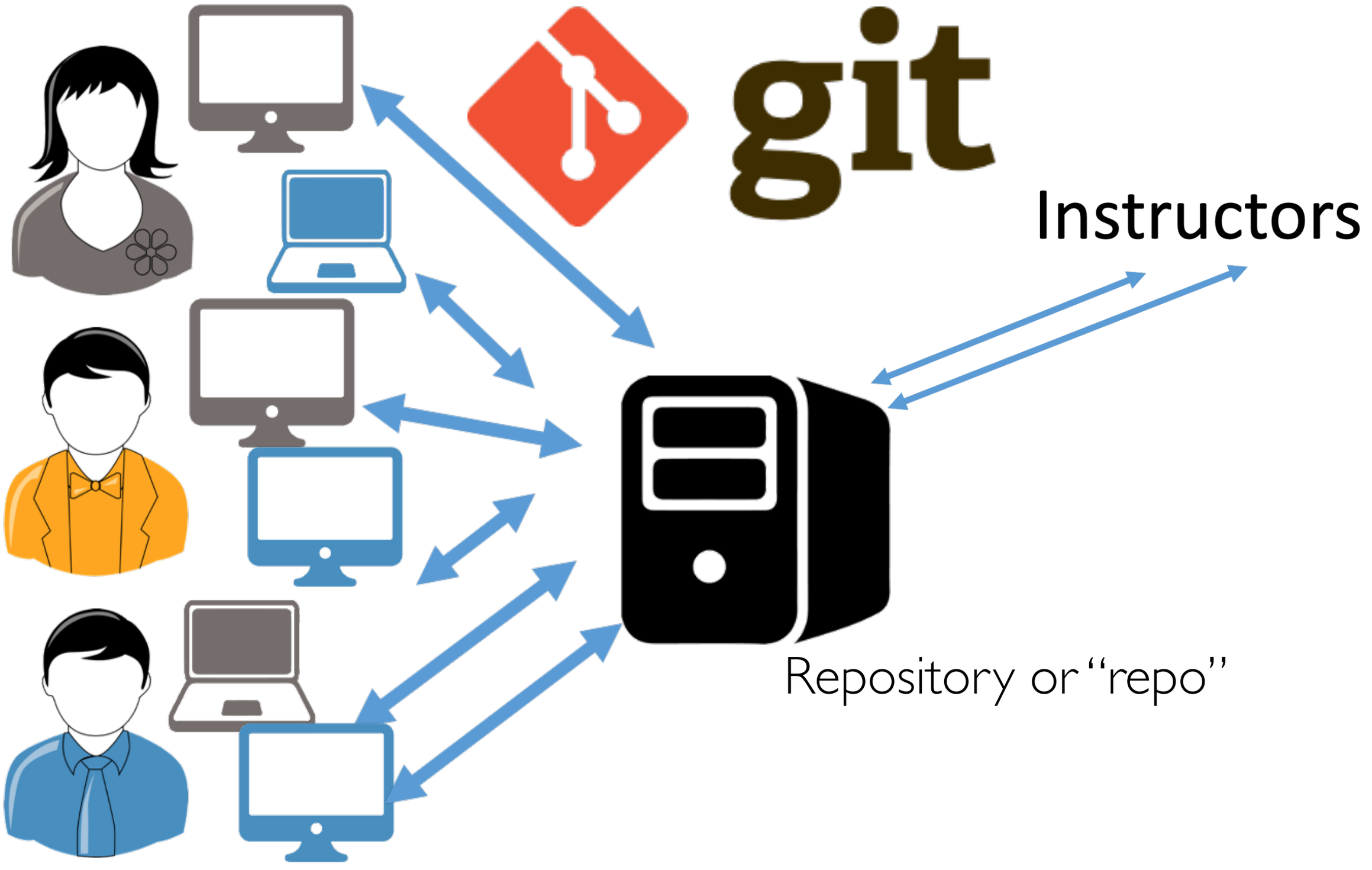
Repository or "repo"



Instructors

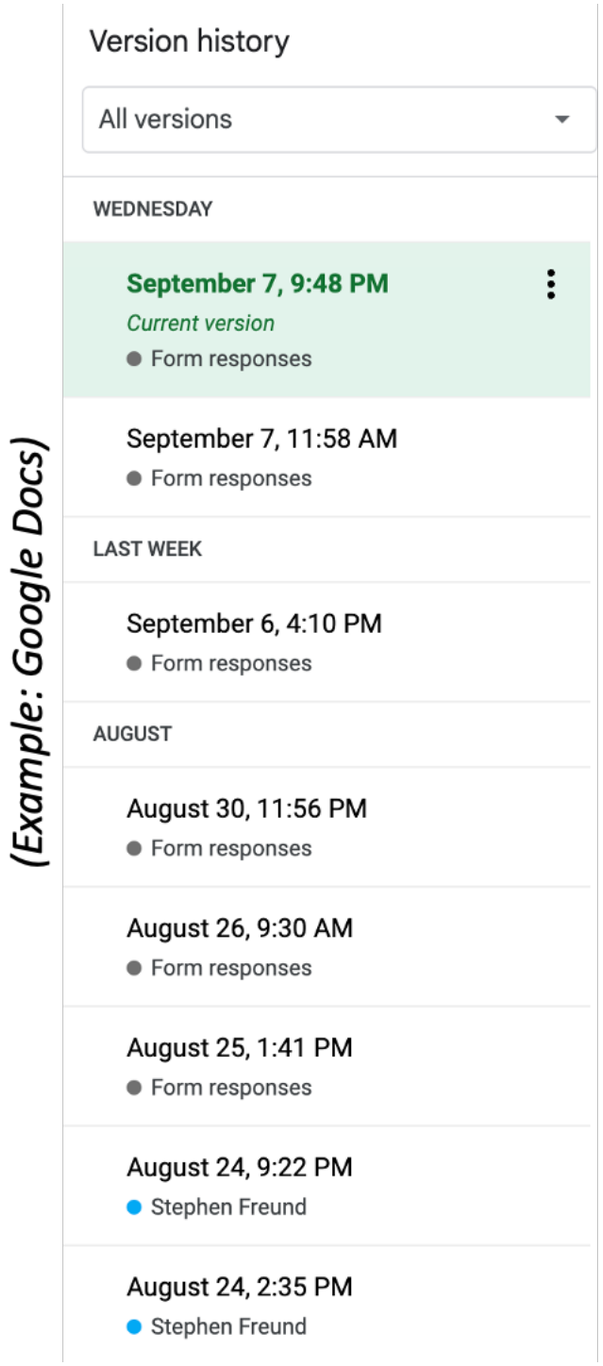


Working on Labs



Why git?!

- Version history!
- Access files from anywhere!
- Great for collaboration!
- Great for maintaining large code bases!



(Example: Google Docs)

`git`:: Get the starter code first

- Copying files —> "I need to get a copy of the lab files my instructor made."

```
git clone https://URL-here.git
```

`git`:: 3 Steps: `git add`

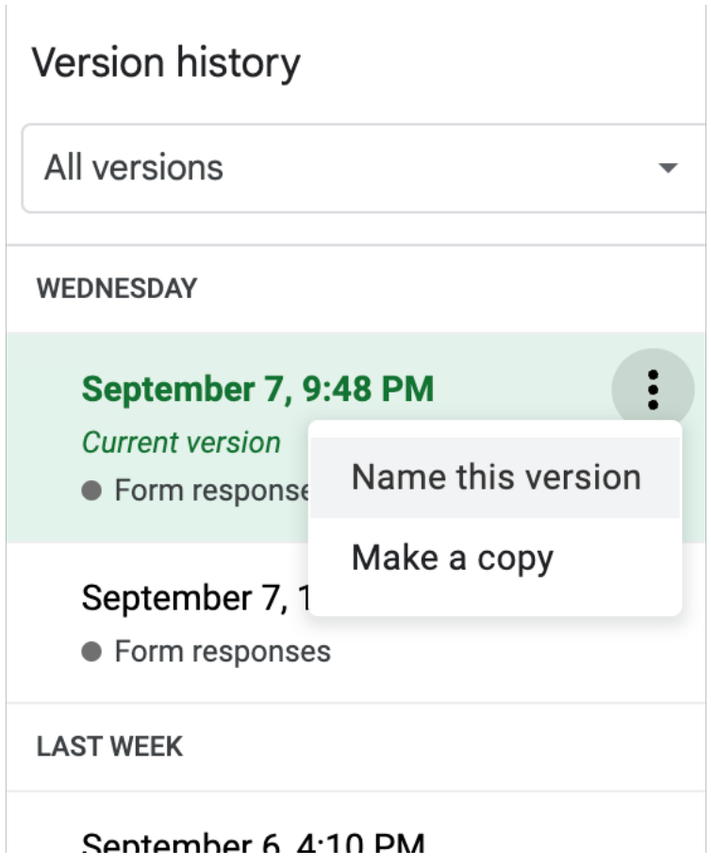
- Staging files —> “I edited this file, I want to include it in the next snapshot of my code .”

```
git add myfilename.py
```

`git`:: 3 Steps: `git commit -m`

- Committing files —> “Take a snapshot of my code that I’ve added so far and assign it a version number.”

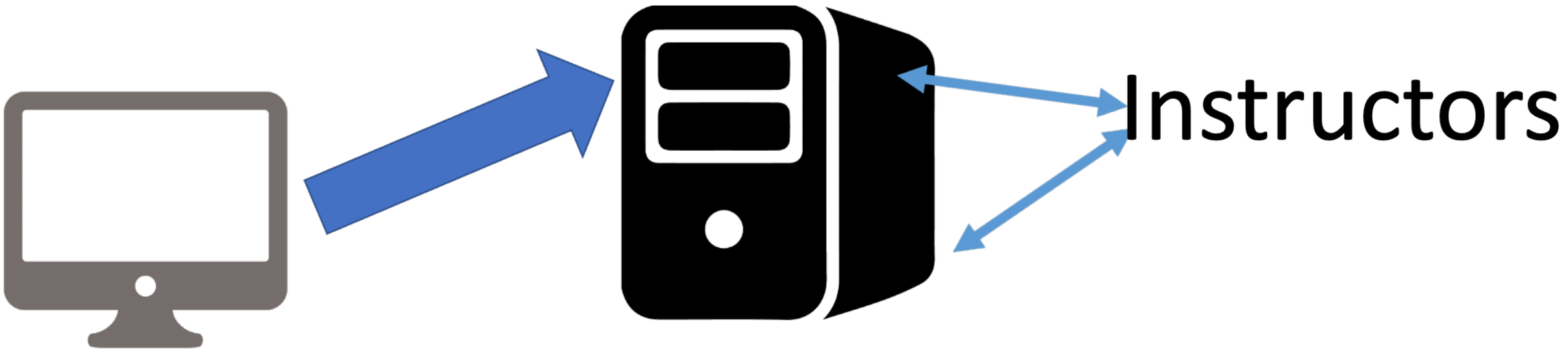
```
git commit -m “Message here”
```



`git`:: 3 Steps: `git push`

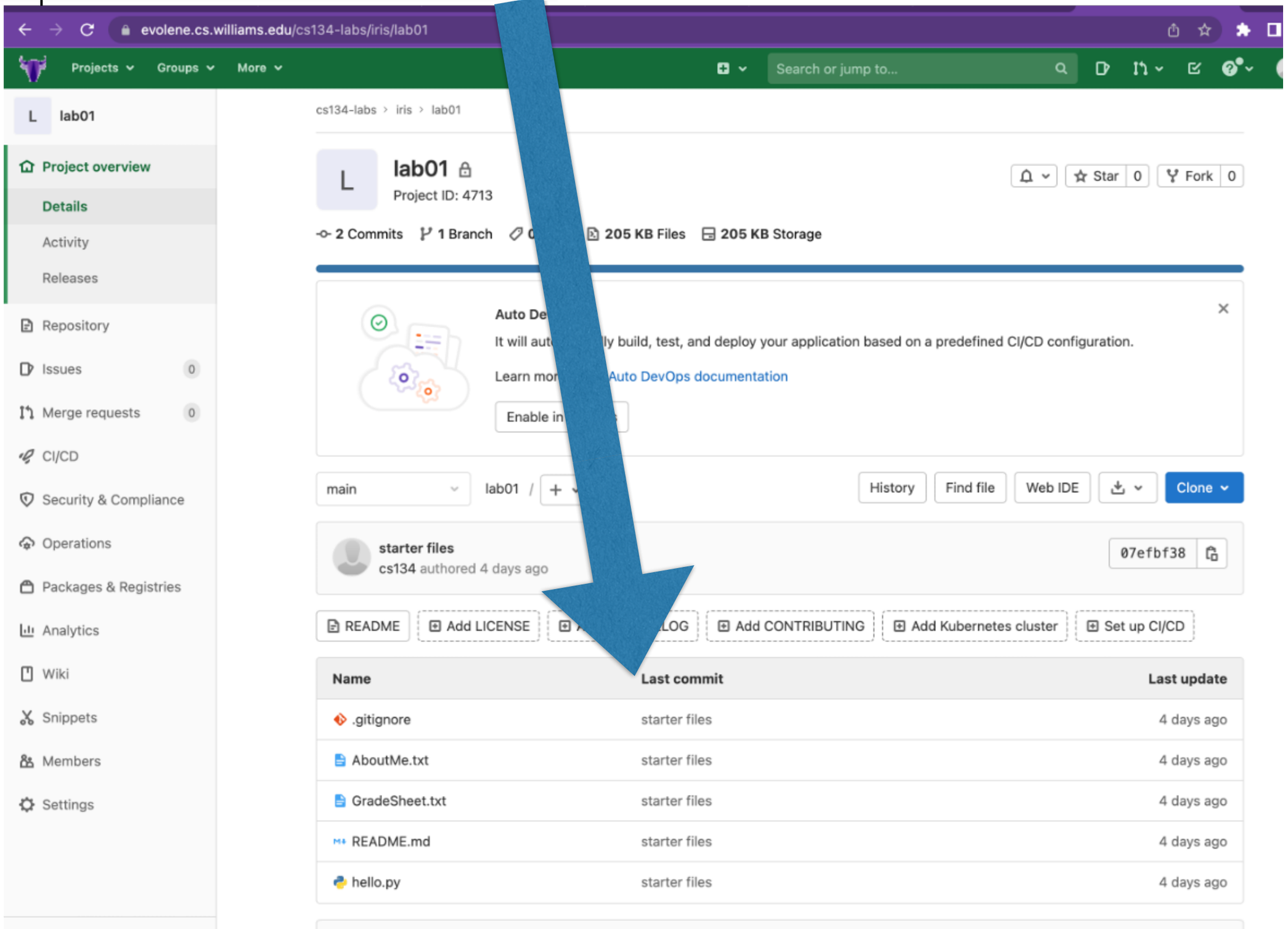
- Pushing files —> “Send all my snapshots up to the server!”

`git push`



`git`:: Check the Website to See Your Edits

- https://evolene.cs.williams.edu



`git`:: 3 Steps

- `git clone`: copy code from server to a **new** machine for the first time. Only run this once for each assignment on each machine!

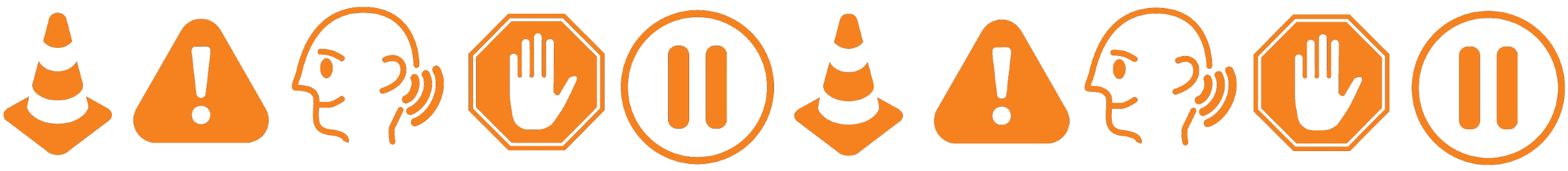
- `git add <files>`: add new or modified files to the next commit (this basically allows you to choose which files you plan to commit)
- `git commit -m "<message>"`: create a local snapshot of the added files (this does **not** copy anything back to the server!)
- `git push`: copy changes from your machine back to our server

- `git pull`: copy latest version of code from our server to your local machine (this can only be done **after** you have run **`git clone`** on this machine)
- `git commit -am "<message>"`: commits an already added file (a shortcut)

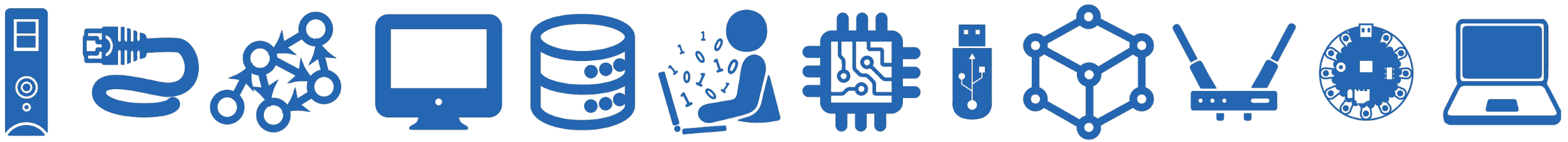
Notes

- We use `git` commands in Terminal
- You need your CS account to log-in to evolene (the gitlab server with the lab starter files)
- Always `git add/commit/push` before you leave lab!

- Lab instructions are on the course Website
- You'll *submit* your assignments through GradeScope.



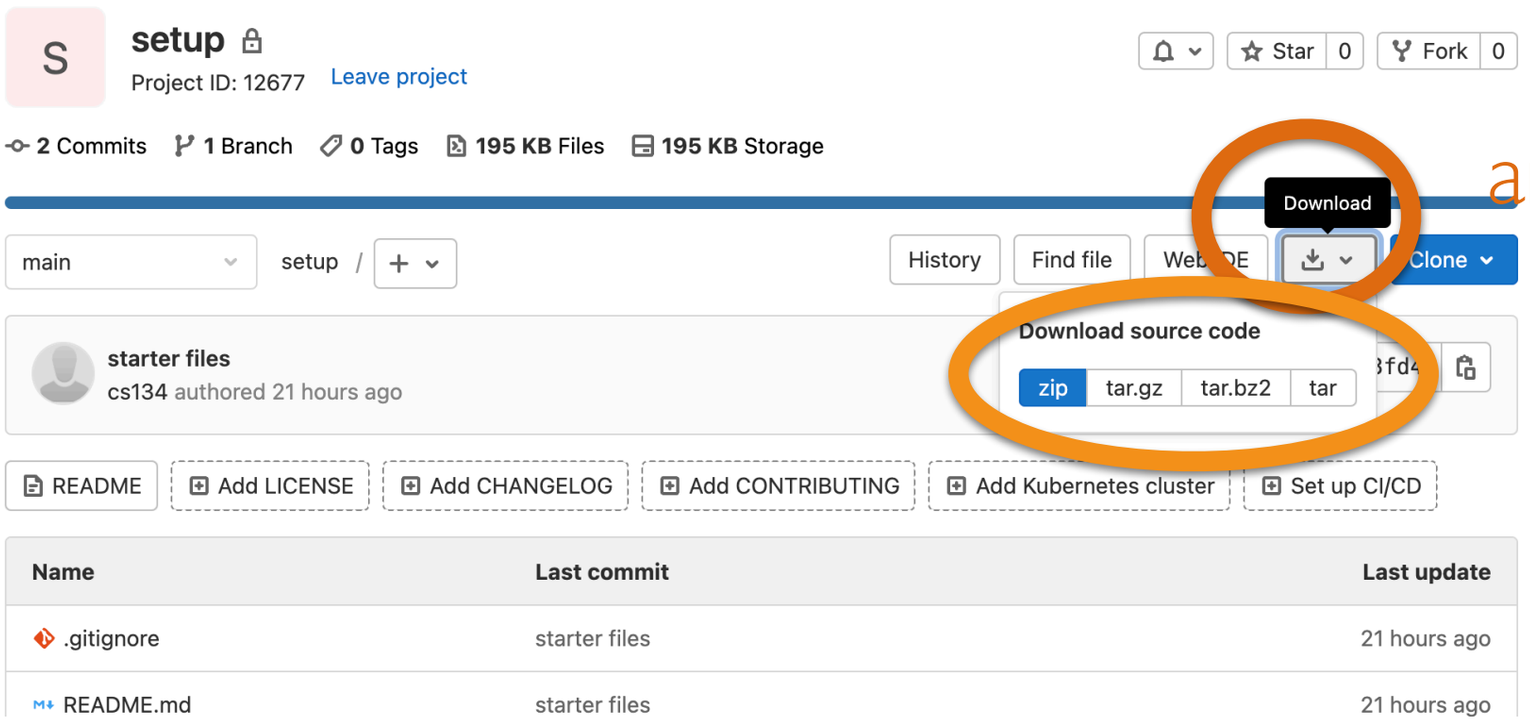
Submitting Your Lab Assignment Through Gradescope



Gradescope

- First, you'll need to go to evolene/gitlab and download your lab assignment as a .zip file:

  (This screenshot is for a different lab, but the buttons should be the same!)



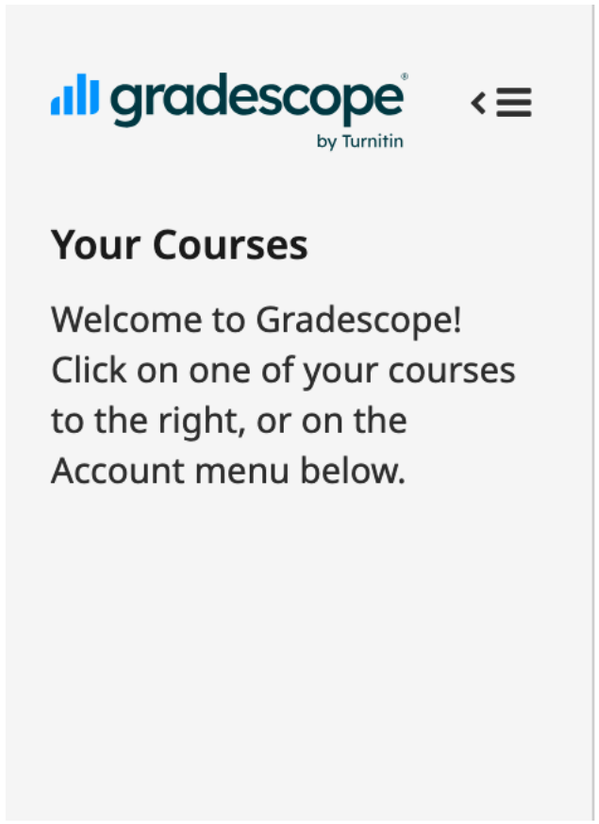
Select the downward arrow Download button

Then select 'zip' as the Download source code option

Remember where you've downloaded the **`lab01-main.zip`** file!

Gradescope

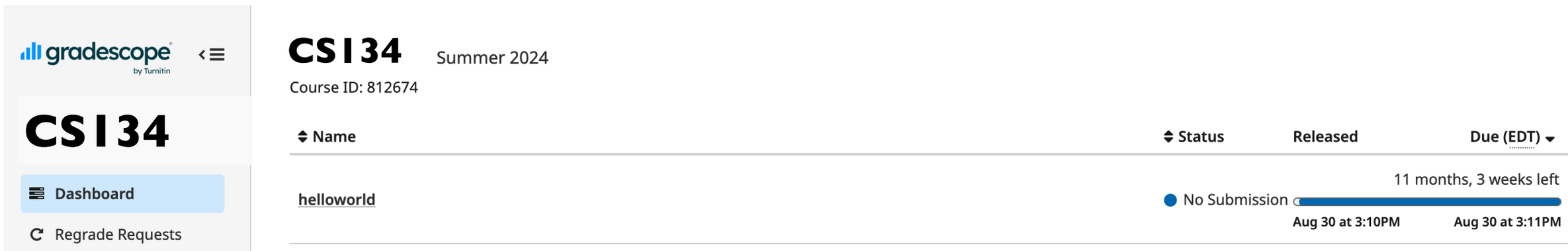
- Use the Gradescope invitation email you receive to create an account associated with CS134
- From the Gradescope Dashboard, select CS134



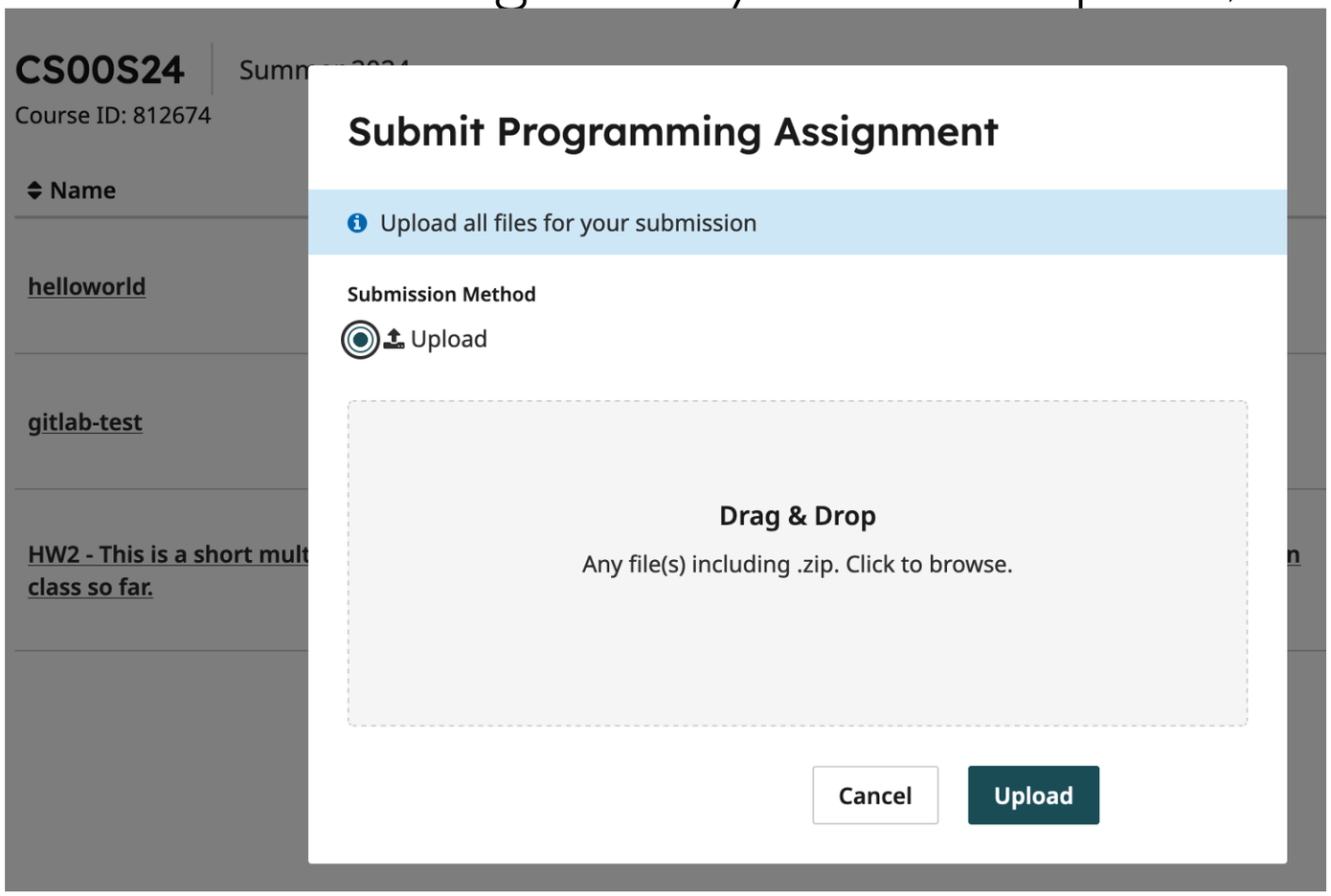
Your Dashboard will look *something* like this!

Gradescope

- Select the appropriate assignment (for this week, lab 1)



- You should navigate to your Lab .zip file, and upload it



Remember to press 'Upload!

The end!