UC Berkeley
SCHOOL OF INFORMATION

Introduction

W200 Introduction to Data Science Programming (Python)

UC Berkeley
SCHOOL OF INFORMATION

Welcome! Today's Agenda

Each week we have an agenda

- Introduction
- Student survey
- Course Overview and Curriculum
- Grading
- Materials: Asynch, Zoom, Additional
- Breakout Rooms
- Github
- Recap

UC Berkeley
SCHOOL OF INFORMATION

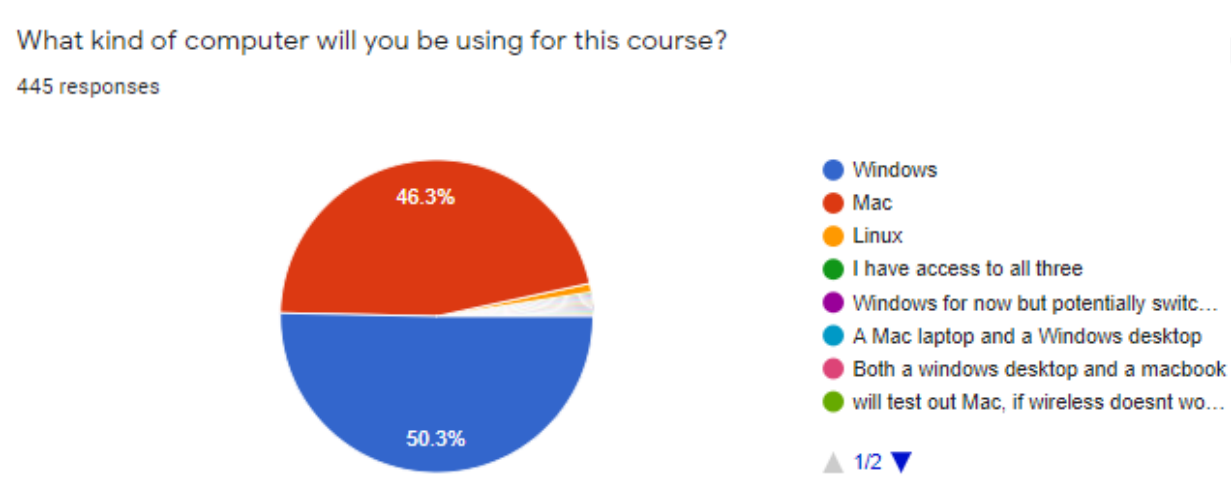
Welcome! Today's Agenda

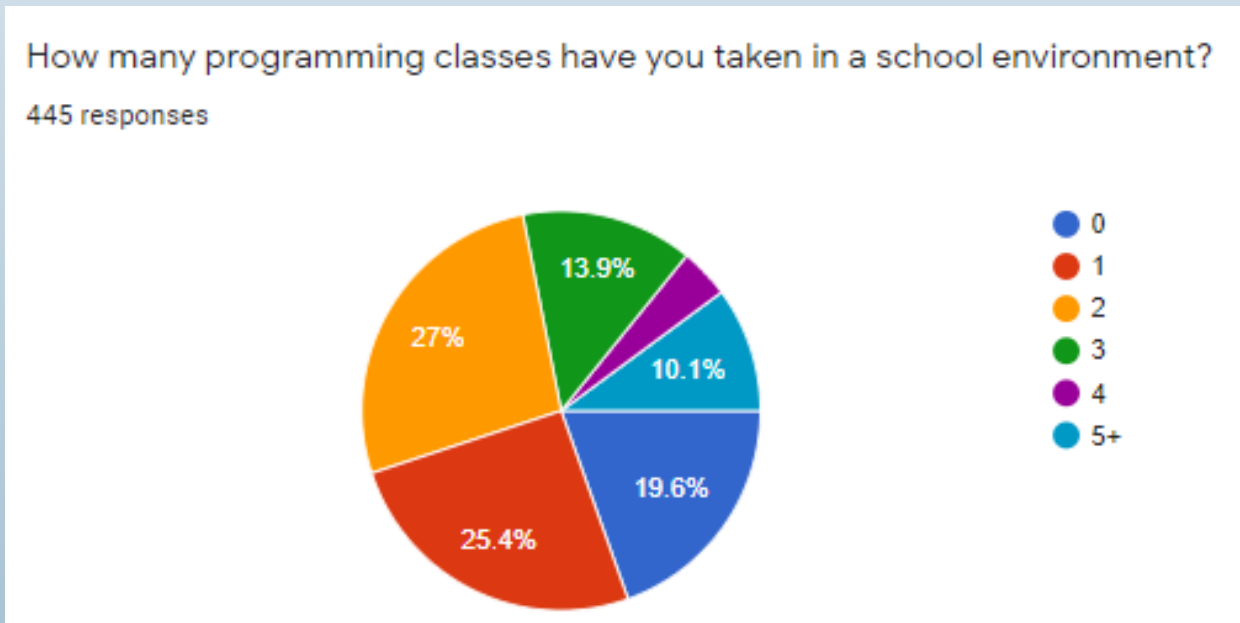
Weekly slides and/or Notebooks to guide discussion

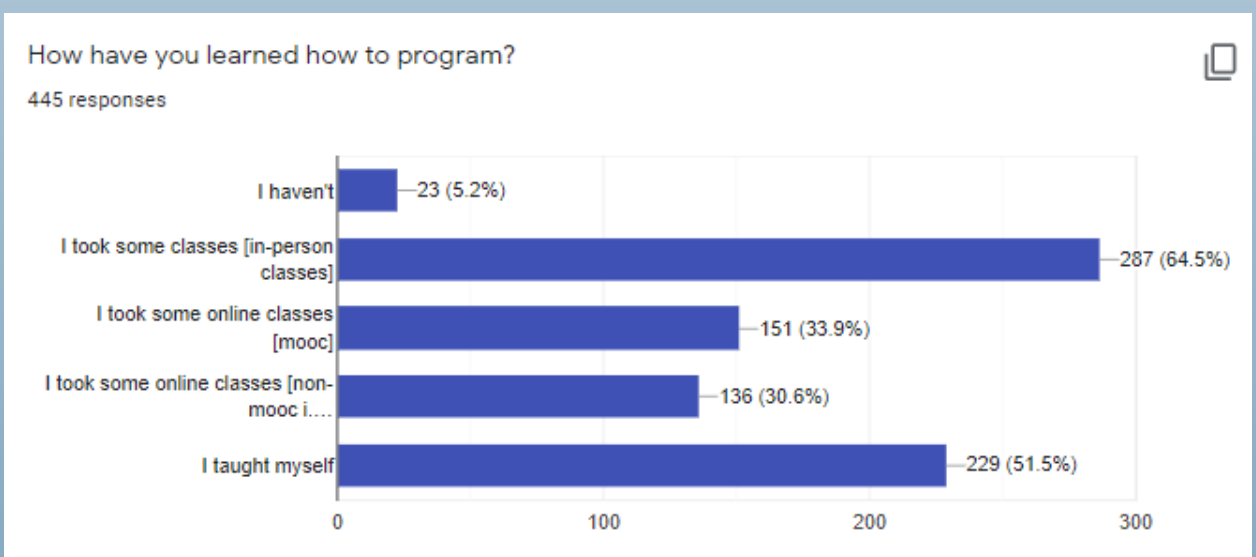
- Quick Intros: name, where do you live and work? Interests in Data Sci?
  - *Project 2 is a group project so good to know shared interests*
- What's your computer experience/ skills?
  - *Helps to know levels and background in computing for our teaching examples*

UC Berkeley
SCHOOL OF INFORMATION

Student Survey: computer use



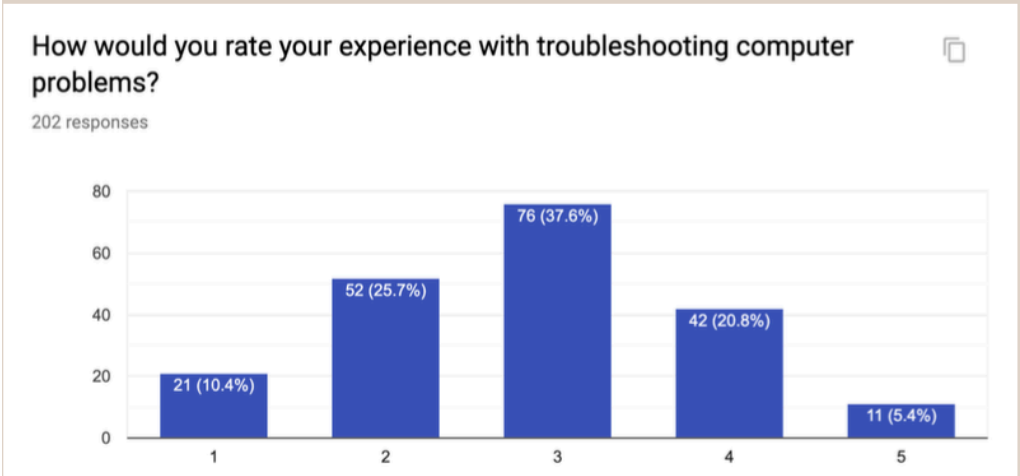




Figuring things out

a pov about learning computing in this environment

- Much of the time you'll have to **discover answers** on your own
- Learning to interpret & respond to **error messages** is vital
- If you're thinking it, someone else is, too!
- **Get to know each other** - for small group projects, networking, colleagues in the program



Curriculum

Fundamentals emphasize core python, problem-solving; then popular libraries

- Week 1: Computing Environment
  - *OS/Unix, Github, IDEs*
- 2-6: Fundamentals
  - *Data and sequence types; control of flow, algorithms functions, complexity; Big Theta*
- 7-8: Object-Oriented Design
  - *Classes, OOD, OOP*
  - *Project 1*
- 9-14: Data Analysis
  - *Text & Binary Data*
  - *Numpy*
  - *Pandas (3 lessons; exploratory data analysis)*
  - *Plotting*
  - *Files & Testing*
  - *Team Data Analysis & Presentation*
  - *Project 2*

Grading

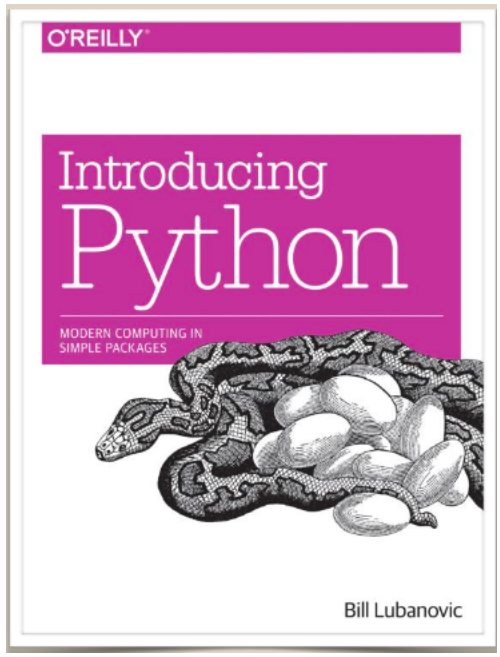
Distribution of points: projects tend to raise grades. Don't fall behind!

- Homeworks - 40%
  - Post on Gradebook; don't be late
- Projects - 30%
  - 15% for Project 1 (oop)
  - 15% for Project 2 (small teams)
- Online Exams - 20%
  - Midterm is 10%
  - Final is 10%, open for 2 weeks period in which to start; complete in a 48-hour window
- Participation - 10%

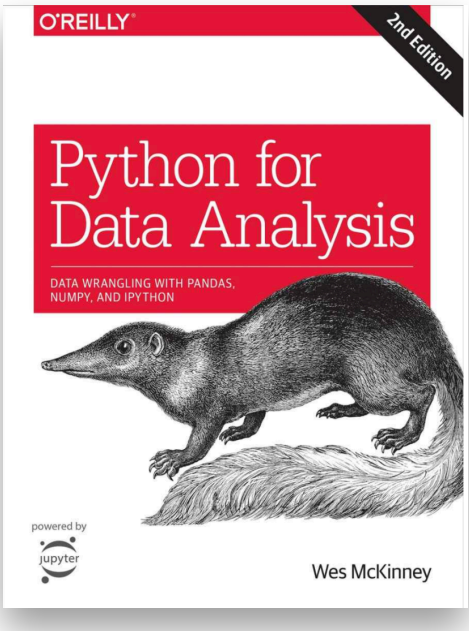
Materials for the course

Required, optional, and just fyi resources

- Complete asynch before live session.
  - Wide-range of student experiences so many optional guides are available.



Optional materials



McKinney, W (2017). *Python for Data Analysis*. Sebastopol, CA: O'Reilly Media. [can be viewed using the UC Berkeley library VPN]

Breakout Rooms

Most weeks we break into small group video chat rooms

- Reading and coding is great; problem-solving with a small team is great, too!
  - Most weeks we have time for a breakout room where you'll work on a problem with colleagues.
  - At times you'll gossip code and share techniques.
  - Usually 1 student shares the screen and the group discusses the code.

Resources

Online links to helpful sites/tools/emails

- Syllabus & Calendar: all are available only on bCourses site
- Homeworks:
  - HW are released on bCourses; submit your homework via Gradescope [post final version on your GitHub]
- Instructor & Student Announcements: all on bCourses now
- Slack Channel (students): is datasci-200-python

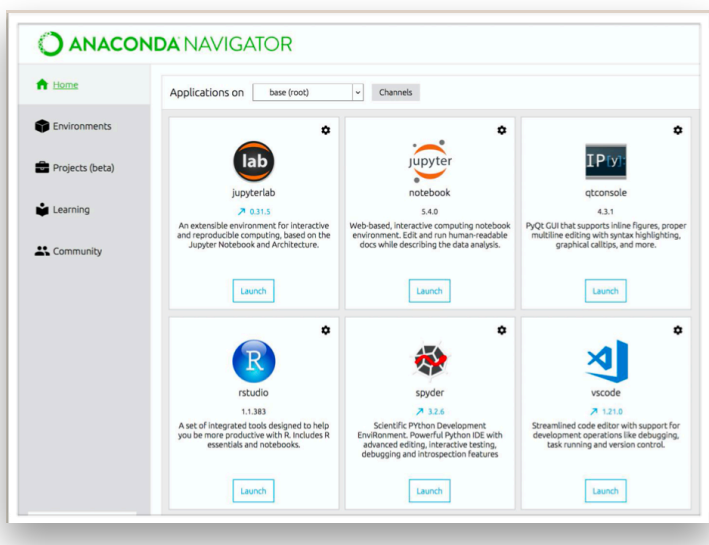
UC Berkeley
SCHOOL OF INFORMATION

Programs & Tools for the course

Programs & Tools for the course

Tools you'll use.

- Unix shell
- Python 3.x
- Anaconda / Jupyter Notebook
- GitHub
- Text editor or an IDE



UC Berkeley
SCHOOL OF INFORMATION

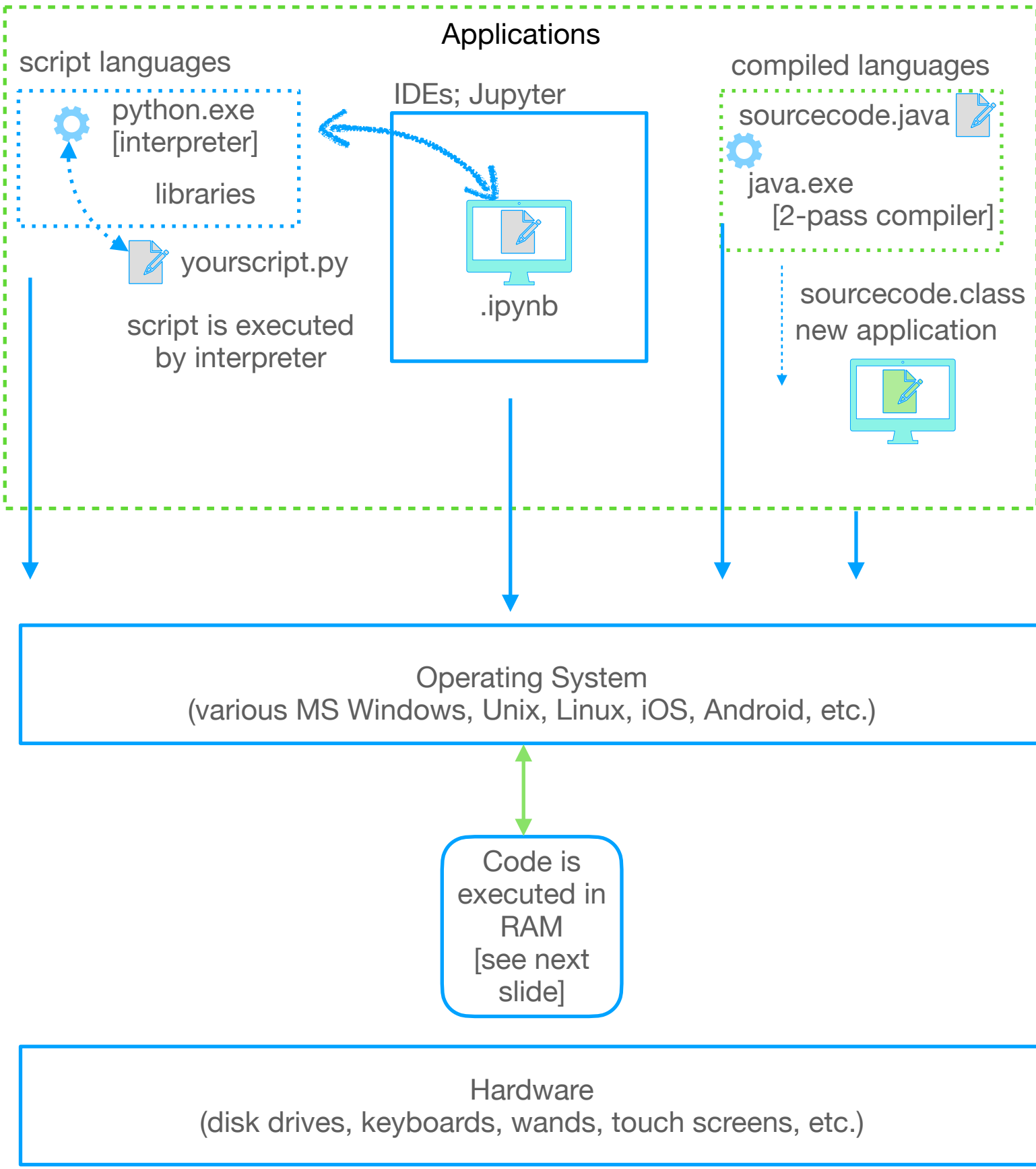
Relationships of code + hardware

Key to the next slide ...

- **Scripting** languages ("JIT") - source script is converted by executable interpreter (e.g., python.exe) and generates object code code;
- **Compiled** languages - two passes to review syntax and then create object code; may be run w/ executable or as a new stand-alone application
- Jupyter Notebook is a (kind of) integrated development environment (**IDE**) that is an application for programmers; when executing scripts in Jupyter, it's Jupyter that communicate with python ... **Operating Systems** can be used instead of an IDE to pass the name of the script to the executable, in the terminal, e.g., `%> python3 myscript.py`
- **RAM**: how we code determines partly where the names and declarations of variables are stored and where the values will be stored in RAM (heap and stack).

Optional

Relationships of code, os, + hardware



Optional: for details see the week_01_notes.txt

How does python get its speed?

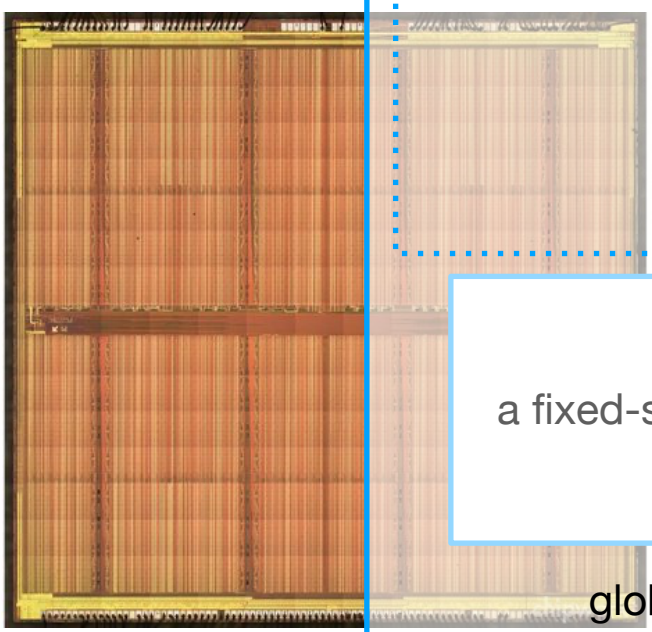
We have a limited amount of RAM available when we code.

Python lets us create global variables that stay in one place in RAM, but it's not a best practice.

We can declare variables that won't change and access them quickly in the "stack." Variables that will change are stored in the "heap".

Some libraries of code, like Pandas and Numpy, use the interaction between heap and stack very efficiently.

As you'll learn, python's syntax and commands allow us to use programming techniques that use RAM more efficiently, too.



RAM looks like a screen door of x,y coordinates (called addresses).

high memory address

heap
(most of our variables go here)

stack
a fixed-sized block of ram - holds our functions, their (local) variables)

global variables & static variables

machine instructions (OS and python and the compiled code) and other text (our scripts)

low memory

UC Berkeley
SCHOOL OF INFORMATION

Unix shells

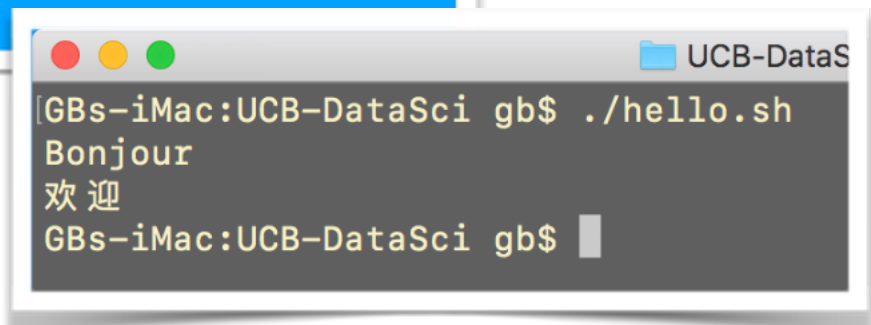
many versions: zsh, csh, tsch, bash. same functionality, different commands.

- Interacting with unix (or the OS) is important for writing/running scripts and for checking access permissions.
- Reading terminal commands …
  - What do you know about the permissions (`rwx, r, xr`; octal `0755`)
  - stdout, stdin, stderr

```
$  chmod 755
hello.sh
$  ./hello.sh
```

Read each line and character. What does it/ they does/do?

```
#!/bin/sh
# a comment
echo Bonjour  # a comment in French
echo "欢迎"
```



Virtual Environments

Common in education and on-the-job before releasing live

- Programming projects often require a different set of Python packages (sets of programs) and these sets can be incompatible.
- An environment lets you run compatible sets of packages together and keep them away from other environments.
- The steps for using environments are:
  - Create the environment adding dependencies

```
conda create --name myenv
```

  - Step into (activate the environment)

```
conda activate myenv
```

  - Add additional dependencies

```
conda install pandas
```

- Then code!
- You will practice in HW1.

Wrap up

UC Berkeley
SCHOOL OF INFORMATION

- Enjoy the course - it's can be a challenge but keep at it!
- If you're working along for > 1 hour, take a break and have a "sanity check." [*Really.*]
- Keep in touch. The TAs and tutors are a good starting point for questions outside of class - as are your classmates & the Net.