## CS 1110 Fall 2022

- **Outcomes:**
  - **Fluency** in (Python) procedural programming
    - Usage of assignments, conditionals, and loops
    - Ability read and test programs from specifications
  - **Competency** in object-oriented programming
    - Ability to recognize and use objects and classes
  - **Knowledge** of searching and sorting algorithms
    - Knowledge of basics of vector computation
- **Website:**
  - www.cs.cornell.edu/courses/cs1110/2022fa/

1

## Class Structure

- **Lectures.** Every Tuesday/Thursday
  - Not just slides; interactive demos almost every lecture
  - Because of enrollment, please stay with your section
  - **Semi-Mandatory**. 1% Participation grade from polling
- **Section/labs.** Phillips 318 or Hollister 401
  - Guided exercises with TAs and consultants helping out
    - Meets Tuesday/Thursday or Wednesday/Friday
    - **Only Phillips 318 has computers** (bring your laptop)
  - Contact Amy (ahf42@cornell.edu) for section conflicts
  - **Mandatory**. Missing more than 3 lowers your final grade

2

## What Do I Need for this Class?

- **Laptop Computer**
  - Capable of running Python (no ChromeBooks!)
  - Minimum of 8Gb of RAM
- **Python Installation**
  - Will be using the latest Anaconda version
  - See instructions on website for how to install
- **iClicker.** Acquire by **next Tuesday**
  - Credit for answering – even if wrong
  - iClicker App for smartphone **is not** acceptable

3

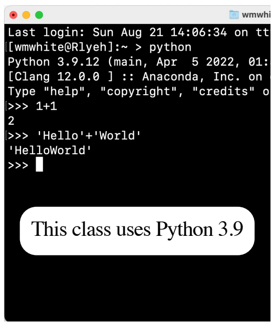
## Things to Do Before Next Class

- Visit the course website:
  - www.cs.cornell.edu/courses/cs1110/2021fa/
  - This IS the course syllabus, updated regularly
- Read **Get Started**
  - Enroll in **Ed Discussions**
  - Register your **iClicker** online
  - Sign into CMS and complete **Survey 0**
  - Install Python and complete **Lab 0**
  - Take the academic integrity quiz

4

## Getting Started with Python

- Will use the "command line"
  - OS X/Linux: **Terminal**
  - Windows: **PowerShell**
  - Purpose of the first lab
- Once installed type "python"
  - Starts an *interactive shell*
  - Type commands at >>>
  - Responds to commands
- Use it like a calculator
  - Use to evaluate *expressions*



```
Last login: Sun Aug 21 14:06:34 on tt
[wmwhite@Rlyeh]:~ > python
Python 3.9.12 (main, Apr  5 2022, 01:
[Clang 12.0.0 ] :: Anaconda, Inc. on
Type "help", "copyright", "credits" o
>>> 1+1
2
>>> 'Hello'+'World'
'HelloWorld'
>>> 
```

This class uses Python 3.9

5

## Expressions and Values

- An **expression** represents something
  - Python *evaluates it,* turning it into a **value**
  - Similar to what a calculator does
- Examples:

| | | |
|---|---|---|
| >>> 2.2 | Expression | (Literal) |
| 2.2 | Value | |
| >>> (3 * 7 + 1) * 0.1 | Expression | (Complex) |
| 2.2 | Value | |

6

## What Are Types?

- Think about + in Python:

```
>>> 1+2
3
>>> "Hello"+"World"
"HelloWorld"
```

adds numerically

glues together

- Why does + given different answers?
  - + is different on data of different *types*
  - This idea is fundamental to programming

7

## Example: int

- **Values:** integers
  - …, –1, 0, 1, …
  - Literals are just digits: 1, 45, 43028030
  - No commas or periods
- **Operations:** math!
  - +, – (add, subtract)
  - *, // (mult, divide)
  - ** (power-of)
- **Important Rule:**
  - **int** ops make **ints**
  - (if making numbers)
- What about division?
  - 1 // 2 rounds to 0
  - / is **not** an **int** op
- Companion op: %
  - Gives the remainder
  - 7 % 3 evaluates to 1

8

## Example: float

- **Values:** real numbers
  - 2.51, -0.56, 3.14159
  - Must have decimal
  - 2 is **int**, 2.0 is **float**
- **Operations:** math!
  - +, – (add, subtract)
  - *, / (mult, divide)
  - ** (power-of)
- Ops similar to **int**
- **Division** is different
  - Notice /, not //
  - 1.0/2.0 evals to 0.5
- But includes //, %
  - 5.4//2.2 evals to 2.0
  - 5.4 % 2.2 evals to 1.0
- Superset of **int**?

9

## Using Big float Numbers

- **Exponent notation** is useful for large (or small) values
  - –22.51e6 is –22.51 * $10^6$ or –22510000
  - 22.51e–6 is 22.51 * $10^{-6}$ or 0.00002251

A second kind of **float** literal

- Python *prefers* this in some cases

```
>>> 0.00000000001
1e-11
```

Remember: values look like **literals**

10

## Example: bool

- **Values:** True, False
  - That is it.
  - Must be capitalized!
- **Three Operations**
  - b and c
    (True if **both** True)
  - b or c
    (True if **at least one** is)
  - not b
    (True if b is **not**)
- Made by **comparisons**
  - **int**, **float** operations
  - But produce a **bool**
- Order comparisons:
  - i < j, i <= j
  - i >= j, i > j
- Equality, inequality:
  - i == j (**not** = )
  - i != j

11

## Example: str

- **Values:** text, or *sequence of characters*
  - String literals must be in quotes
  - Double quotes: "Hello World!", " abcex3$g<&"
  - Single quotes: 'Hello World!', ' abcex3$g<&'
- **Operation:** + (catenation, or concatenation)
  - 'ab' + 'cd' evaluates to 'abcd'
  - concatenation can only apply to strings
  - 'ab' + 2 produces an **error**

12