Lecture 1

Course Overview, Types & Expressions

CS 1110 Fall 2012: Walker White

- **Outcomes:**
  - **Fluency** in (Python) procedural programming
    - Usage of assignments, conditionals, and loops
    - Ability to design Python modules and programs
  - **Competency** in object-oriented programming
    - Ability to write programs using objects and classes.
  - **Knowledge** of searching and sorting algorithms
    - Knowledge of basics of vector computation
- **Website:**
  - www.cs.cornell.edu/courses/cs1110/2012fa/

Why Did We Stop Teaching Java?

- Python is **easier for beginners**
  - A lot less to learn before you start “doing”
  - Designed with “rapid prototyping” in mind
- Python is **more relevant to non-CS majors**
  - NumPy and SciPy heavily used by scientists
- Python is a more **modern language**
  - Popular for web applications (e.g. Facebook apps)
  - Also applicable to mobile app development

Intro Programming Classes Compared

CS 1110: Python

- No prior programming experience necessary
- No calculus
- Non-numerical problems
- More about software design
- Focus is on training future **computer scientists**

CS 1112: Matlab

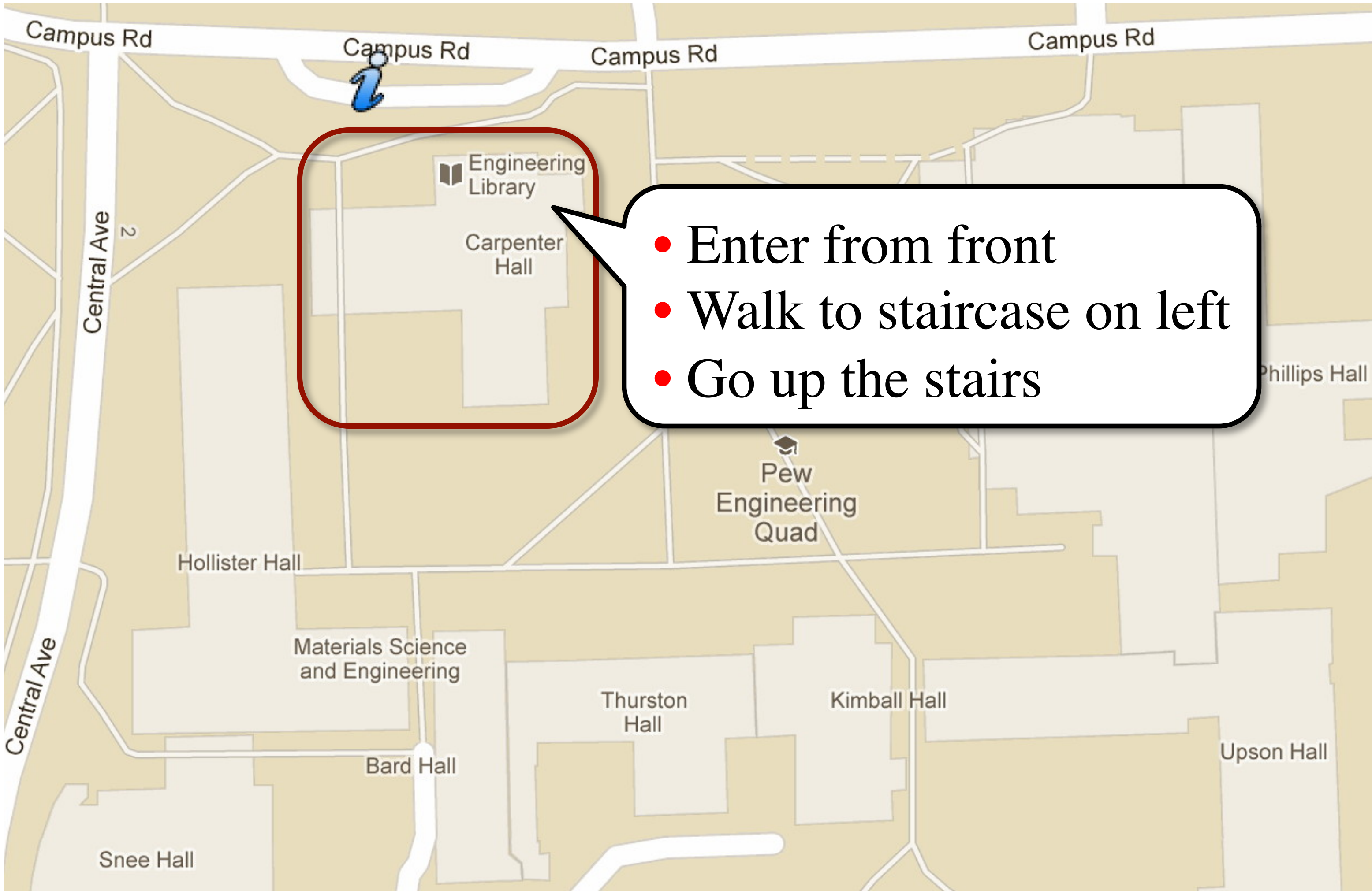
- No prior programming experience necessary
- One semester of calculus
- Engineering-type problems
- Less about software design
- Focus is on training future **engineers that compute**

But either course serves as a pre-requisite to CS 2110

Class Structure

- **Lectures.** Every Tuesday/Thursday
  - Not just slides; interactive demos almost every lecture
  - You may attend *either* Lecture section (9 or 11)
  - **Semi-Mandatory**. 1% Participation grade from iClickers
- **Section/labs.** ACCEL Lab, Carpenter 2nd floor
  - Guided exercises with TAs and consultants helping out
  - Register for ANY section, but go to the one you want
    - Tuesday: 12:20, 1:25, 2:30, 3:35
    - Wednesday: 10:10, 11:15, 12:20, 1:25, 2:30, 3:35, 7:20
  - **Mandatory**. Missing more than 2 lowers your final grade

ACCEL Labs



- Enter from front
- Walk to staircase on left
- Go up the stairs

Class Materials

- **Textbook.** *Think Python* by Allen Downey
  - *Supplemental* text; does not replace lecture
  - Hardbound copies for sale in Campus Store
  - Book available for free as PDF or eBook
- **iClicker.** Acquire one by **next Tuesday**
  - Will periodically ask questions during lecture
  - Used to judge class understanding
  - Will get credit for answering – even if wrong
- **Python.** Necessary if you want to use own computer
  - See course website for how to install the software

Helping You Succeed: Other Resources

- **Consultants.** ACCEL Lab Green Room
  - Daily office hours (see website) with consultants
  - Very useful when working on assignments
- **AEW Workshops**. Additional discussion course
  - Runs parallel to this class – completely optional
  - See website; talk to advisors in Olin 167.
- **Piazza.** Online forum to ask and answer questions
  - Go here first **before** sending question in e-mail
- **Office Hours.** Talk to the professor!
  - Available in Hollister 202 between lectures

Assignments

- Major portion (40%) of your final grade
  - Larger projects due every two weeks
- First assignment requires **mastery**
  - Submit, get feedback, resubmit, … until correct
  - Everyone eventually scores 10/10
- Later assignments are designed to be fun
  - Examples: graphics, image manipulation
  - Final project is a Breakout game project
- Submitted via **Course Management System** (CMS)
  - Visit cms.csuglab.cornell.edu/ to check you are enrolled

Academic Integrity

- **Do not cheat**, in any way, shape, or form
  - Will be very explicit about this throughout course
  - Pay attention to all assignment instructions
- In return, we try to be fair about amount of work, grading the work, and giving you a course grade
- See website for more information
- Complete **Quiz: About the Course on CMS**

Participation: 2% of Final Grade

- **iClickers.** In lecture questions
  - Essentially a form of "stealth attendance"
  - Must answer 75% of questions for credit
  - But actually answers not graded
- **Surveys.** What do you think of the class?
  - This is the first semester teaching Python
  - Want data on who you are/why taking course?
  - What do you like/dislike about assignments?
  - Must answer 75% of surveys for full credit

Things to Do Before Next Class

1. Register your iClicker
   - Does not count for grade if not registered
2. Enroll in Piazza
3. Sign into CMS
   - Quiz: About the Course
   - Complete Survey 0
4. Read the textbook
   - Chapter 1 (browse)
   - Chapter 2 (in detail)

- Everything is on website!
  - Piazza instructions
  - Class announcements
  - Consultant calendar
  - Reading schedule
  - Lecture slides
  - Exam dates
- Check it regularly:
  - www.cs.cornell.edu/courses/cs1110/2012fa/

Getting Started with Python

- Designed to be used from the "command line"
  - OS X/Linux: **Terminal**
  - Windows: **Command Prompt**
  - Purpose of the first lab
- Once installed type "python"
  - Starts an *interactive shell*
  - Type commands at >>>
  - Shell responds to commands
- Can use it like a calculator
  - Use to evaluate *expressions*

This class uses Python 2.7.x
- Python 3 is too cutting edge
- Minimal software support

Expressions vs Statements

Expression

- **Represents** something
  - Python *evaluates it*
  - End result is a value
- Examples:
  - `2.3` Value
  - `(3+5)/4` Complex Expression

Statement

- **Does** something
  - Python *executes it*
  - Need not result in a value
- Examples:
  - `print "Hello"`
  - `import sys`

Will see later this is not a clear cut separation

Representing Values

- **Everything** on a computer reduces to numbers
  - Letters represented by numbers (ASCII codes)
  - Pixel colors are three numbers (red, blue, green)
  - So how can Python tell all these numbers apart?

Memorize this definition!

Write it down several times.

- **Type:**
  A set of values and the operations on them.
  - Examples of operations: +, -, /, *
  - The meaning of these depends on the type

Type: Set of values and the operations on them

- Type **int**:
  - values: …, –3, –2, –1, 0, 1, 2, 3, 4, 5, …

    “Whole” numbers w/o decimals
  - operations: +, –, *, /, **, unary –

    multiply

    to power of
- **Principal**: operations on int values must yield an int
  - **Example:** 1 / 2 rounds result down to 0
    - Companion operation: % (remainder)
    - 7 % 3 evaluates to 1, remainder when dividing 7 by 3
  - Operator / is not an int operation in Python 3 (use // instead)

Type: Set of values and the operations on them

- Type **floating point** (or **float**):
  - values: fractions and/or real numbers
    - If you add a decimal, Python assumes it is a **float** (e.g. **2.0**)
    - Without a decimal, Python assumes it is an **int** (e.g. **2**)
  - operations: +, –, *, /, **, unary –
    - But meaning is different for floats
    - **Example**: 1.0/2.0 evaluates to 0.5
- **Exponent notation** is useful for large (or small) floats
  - 22.51e6 is $-22.51 * 10^{6}$ or –22510000
  - 22.51e–6 is $22.51 * 10^{-6}$ or 0.00002251
  - Must start with an integer: 1e5 is ok, but e5 is not

Representation Error

- Python stores floats as **binary fractions**
  - Integer mantissa times a power of 2
  - Example: 12.5 is $\underbrace{10}_{\text{mantissa}} * 2^{\overbrace{-3}^{\text{exponent}}}$
- Impossible to write every number this way exactly
  - Similar to problem of writing 1/3 with decimals
  - Python chooses the closest binary fraction it can
- This approximation results in **representation error**
  - When combined in expressions, the error can get worse
  - **Example**: type 0.1 + 0.2 at the prompt >>>

Type: Set of values and the operations on them

- Type boolean or **bool**:
  - values: **True**, **False**
  - operations: not, and, or
    - not b: **True** if b is false and **False** if b is true
    - b and c: **True** if both b and c are true; **False** otherwise
    - b || c: **True** if b is true or c is true; **False** otherwise
- Often come from comparing **int** or **float** values
  - Order comparision: i < j i <= j i >= j i > j
  - Equality, inequality: i == j i != j

==, not =

Casting: Converting Value Types

- Basic form: *type*(*value*)
  - float(2) casts value 2 to type **float** (value now 2.0)
  - int(2.56) casts value 2.56 to type **int** (value is now 2)
- Narrow to wide: **bool ⇒ int ⇒ float**
  - *Widening* Cast. Python automatically if needed
    - **Example**: 1/2.0 evaluates to 0.5 (casts 1 to **float**)
  - *Narrowing* Cast. Python *never* does automatically
    - Narrowing casts cause information to be lost
    - **Example**: float(int(2.56)) evaluates to 2.0

Type: Set of values and the operations on them

- Type String or **str**:
  - values: any sequence of characters
  - operation(s): + (catenation, or concatenation)
- **String literal**: sequence of chars in quotes
  - Double quotes: `" abcex3$g<&"` or `"Hello World!"`
  - Single quotes: `'Hello World!'`
- Concatenation can only apply to Strings.
  - `"ab" + "cd"` evaluates to `"abcd"`
  - `"ab" + 2` produces an **error**

Cannot cast to or from the other types