CS 1110:
Introduction to Computing Using Python

Lecture 1

Course Overview, Python Basics

[Andersen, Gries, Lee, Marschner, Van Loan, White]

Interlude: Why learn to program?

(which is subtly distinct from, although a core part of, computer science itself)

From the Economist: "Teach computing, not Word"
http://www.economist.com/blogs/babbage/2010/08/computing_schools

Like philosophy, computing qua computing is worth teaching less for the subject matter itself and more for the habits of mind that studying it encourages.

The best way to encourage interest in computing in school is to ditch the vocational stuff that strangles the subject currently, give the kids a simple programming language, and then get out of the way and let them experiment. For some, at least, it could be the start of a life-long love affair.

Interlude, continued

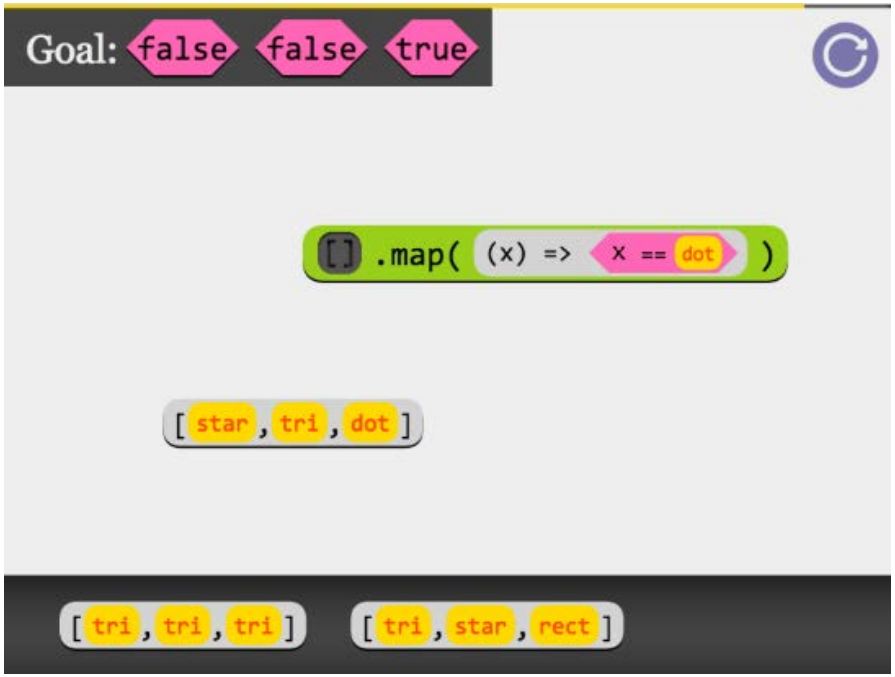
That, for me, sums up the seductive intellectual core of computers and computer programming: here is a magic black box. You can tell it to do whatever you want, within a certain set of rules, and it will do it; within the confines of the box you are more or less God, your powers limited only by your imagination. But the price of that power is strict discipline: you have to *really know* what you want, and you have to be able to express it clearly in a formal, structured way that leaves no room for the fuzzy thinking and ambiguity found everywhere else in life…

The sense of freedom on offer - the ability to make the machine dance to any tune you care to play - is thrilling.

CS1110 can take you places

- Benjamin Van Doren, a CALS student who learned to program as a freshman in CS1110 Spring 2013, helped create the dataset for a paper he co-authored that won Best Paper Award at the Computational Sustainability track at AAAI 2013.
- The paper title was, "Approximate Bayesian Inference for Reconstructing Velocities of Migrating Birds from Weather Radar".
- Van Doren has been a bird lover since third grade and was a finalist in the Intel Science Talent Search.

About Your Instructor: Prof. Andersen



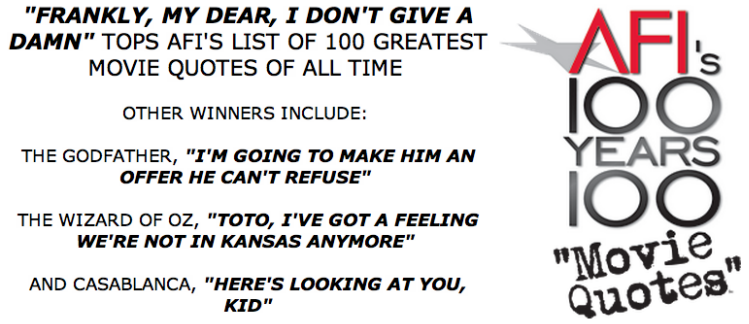
He works on:

- Educational technology
- Games for learning!

About Your Instructor: Prof. Lee

- Fellow of the Association for the Advancement of Artificial Intelligence (AAAI)
- What language distinguishes **memorable movie quotes**?

  —NPR's *All Things Considered*, *The Today Show* (2012)



Why should you take this class?

Why should you take this class?

- **Outcomes:**
  - **Fluency** in (Python) procedural programming
    - Usage of assignments, conditionals, and loops
    - Ability to create Python modules and programs
  - **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

Intro Programming Classes Compared

CS 1110: Python

- No prior programming experience necessary
- **No calculus**
- *Slight* focus on
  - **Software engineering**
  - **Application design**

CS 1112: Matlab

- No prior programming experience necessary
- **One semester of calculus**
- *Slight* focus on
  - **Scientific computation**
  - **Engineering applications**

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

Why *Python*?

- 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

Course Website

- www.cs.cornell.edu/courses/cs1110/2017sp/

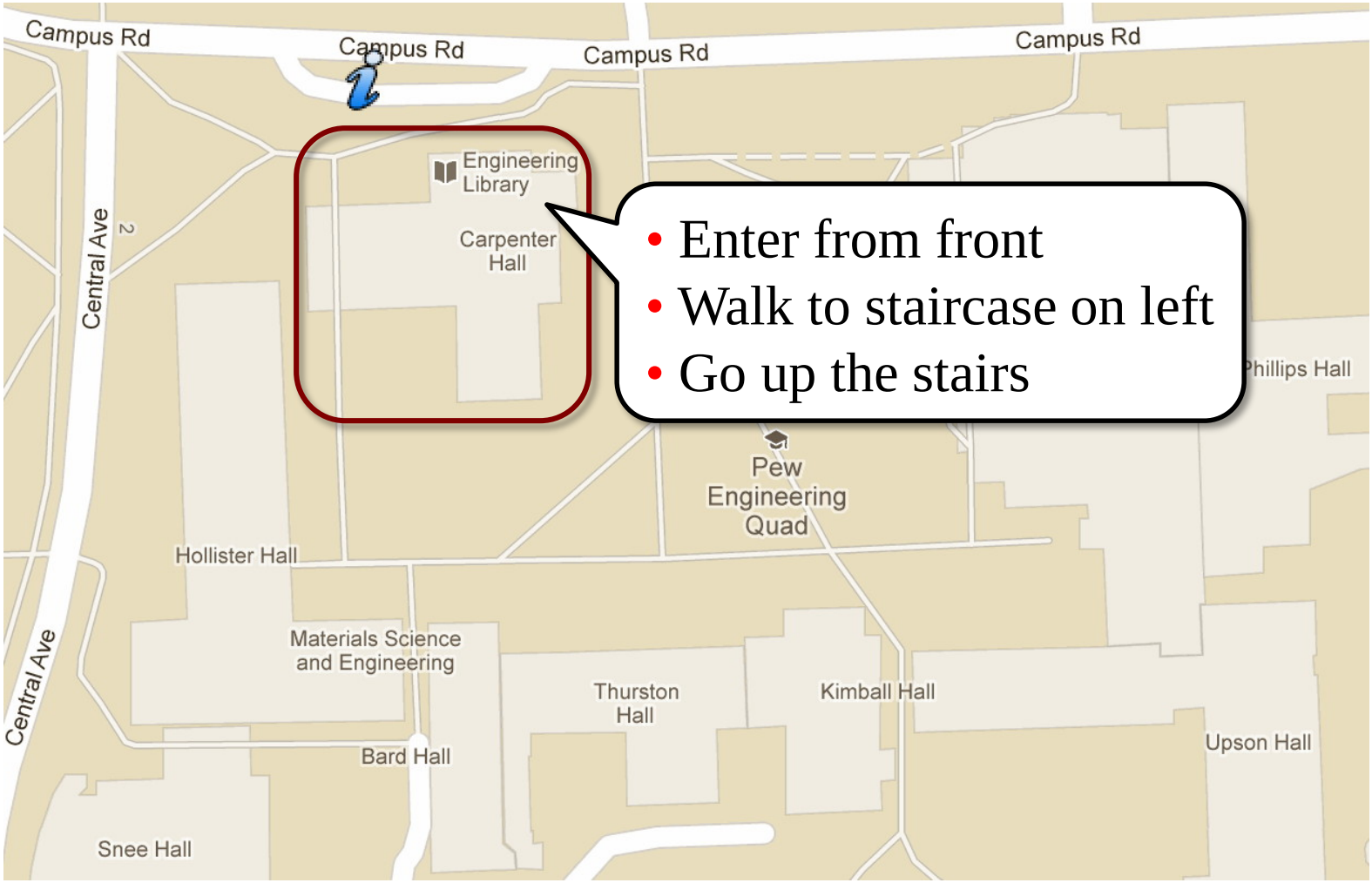
- **LOOK FOR THE SPRING 2017 BAT!!!**

- If no bat, *you are looking at the wrong year*

Communication

- cs-1110profs-L@cornell.edu
  - Includes: two profs, admin assistant
  - **Main correspondence**. Don't email only one prof, or both separately
- cs-1110mgmt-L@cornell.edu
  - Includes: both profs, admin assistant, graduate TAs, head consultants
  - **"Emergency contact number."** nobody at office hours; lab has no printouts
- Email from us: please check your spam filters for mail from ELA63@cornell.edu, LJL2@cornell.edu, or with [CS1110] in the subject line.

ACCEL Labs


Campus Rd
Campus Rd
Campus Rd
Campus Rd
Engineering Library
Carpenter Hall
Central Ave
Central Ave
• Enter from front
• Walk to staircase on left
• Go up the stairs
Phillips Hall
Pew Engineering Quad
Hollister Hall
Materials Science and Engineering
Thurston Hall
Kimball Hall
Upson Hall
Bard Hall
Snee Hall

Class Structure

- **Lectures.** Every Tuesday/Thursday
  - Not just slides; interactive demos almost every lecture
- **Discussion Sections = "Labs".**
  - Guided exercises with TAs and consultants helping out
  - Handouts posted to the website the Monday before
  - **Don't panic if you are not registered yet.**
  - **Go to the lab section you are registered for.**
  - **If not enrolled in a lab section: do the lab on your own. If a lab section opens up, check it in then.**
  - **Mandatory**. Missing more than 2 can lower your final grade.

Class Materials

- **Textbook.** *Think Python* by Allen Downey
  - *Supplemental* text; does not replace lecture
  - Book available for free as PDF or eBook
  - (no hard copy anymore; out of print)

- **iClicker.** Optional but useful.
  - Will periodically ask questions during lecture
  - Not part of the grade at all
- **Python.** Necessary if you want to use own computer
  - See course website for how to install the software

Things to Do Before Next Class

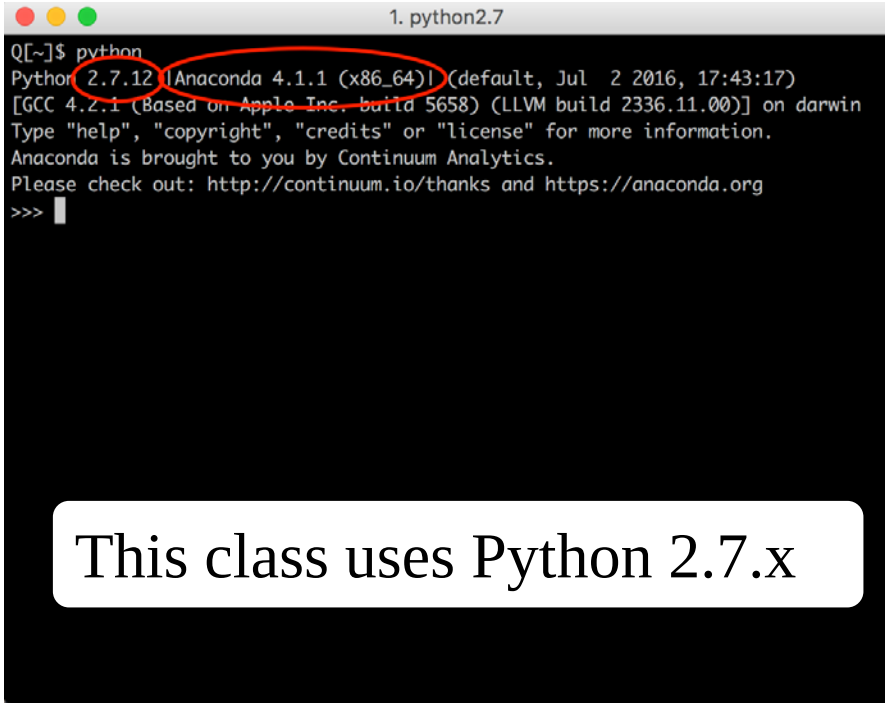
1. Read the textbook
   - Chapter 1 (browse)
   - Chapter 2 (in detail)
2. Install Python **following our instructions:** http://www.cs.cornell.edu/courses/cs1110/2017sp/materials/python.php
3. Look at first lab handout
4. (optional) Piazza: a question-answering forum

- Everything is on website!
  - Piazza instructions
  - Class announcements
  - Consultant calendar
  - Reading schedule
  - Lecture slides
  - Exam dates
- Check it regularly:
  - www.cs.cornell.edu/courses/cs1110/2017sp/

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*

CORRECT:



WRONG:

```
Python 2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Values

- Much of Python is *storing and computing data*
- What data values might we want to work with?

Numbers	Text	Truth
42	“apple”	True
$3.0 * 10^8$	“pineapple”	False
0.00001		

Expressions

- An expression **represents** something
  - Python *evaluates it* (turns it into a value)
  - Similar to a calculator
- Examples:
  - 2.3

Literal
(evaluates to self)

  - (3 * 7 + 2) * 0.1

An expression with four literals and some operators

Types

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

Memorize this definition!

How To Tell The Type of a Value

- Command: type(<value>)
- Example:

```
>>> type(2)
<type 'int'>
```

Example: Type int

- Type **int** represents integers
  - values: …, –3, –2, –1, 0, 1, 2, 3, 4, 5, …
    - Examples
      - 1
      - 45
      - 43028030
    - no commas or periods
  - operations: +, –, *, /, **, unary –, %

multiply

to power of

remainder

Operations on ints

- Operations on **int** values must yield an **int**
  - Examples:
    - 2 + 2
      - result: 4
    - 4 / 7
      - result: **0**
        - This is confusing at first! Then becomes natural.
- Companion to / (*division*) is % (*remainder*)
  - Examples:
    - 1 / 2 = 0
    - 1 % 2 = 1 (remainder when dividing 1 by 2)

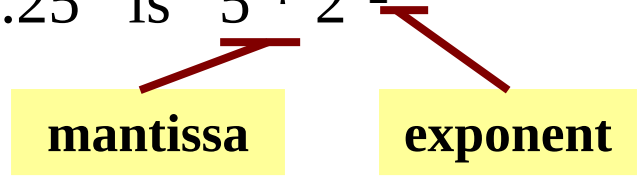
Example: Type float

- Type **float** represents real numbers
  - **float** is short for "floating point"
  - values: distinguished from integers by decimal points
    - In Python a number with a "." is a **float** literal (e.g. 2.0)
    - Without a decimal a number is an **int** literal (e.g. 2)
  - operations: +, –, *, /, **, unary –
    - The meaning for floats differs from that for ints
    - **Example**: 1.0/2.0 evaluates to 0.5
- **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

Floating Point Errors

- Python stores floats as **binary fractions**
  - Integer mantissa times a power of 2
  - Example: 1.25 is $5 * 2^{-2}$



- Impossible to write most real numbers 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 >>>

Example: Type bool

- Type boolean or **bool** represents logical statements
  - values: **True**, **False**
    - Boolean literals are just True and False (have to be capitalized)
  - 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 or c: **True** if b is true or c is true; **False** otherwise
- Often come from comparing **int** or **float** values
  - Order comparison: i < j i <= j i >= j i > j
  - Equality, inequality: i == j i != j

"=" means something else!

Boolean Misconceptions

- Boolean expressions sound similar to English
- However, subtle differences lead to mistakes:
  - In English, "A = B and C" often means "A = B and A = C"
    - **Example**: "Ithaca is cold and snowy"
      - Means: "Ithaca is cold" and "Ithaca is snowy"
      - **Does not mean**: "Ithaca is cold" and…. "snowy"
    - Python requires you to *fully specify* Boolean operations
  - In English, "A or B" often means "A or B ***but not both***"
    - **Example**: "I will take CS 1110 or CS 1112" (but not both)
    - In Python, "A or B" always means "A or B ***or both***"

Example: Type str

- Type String or **str** represents text
  - values: any sequence of characters
  - operation(s): + (catenation, or concatenation)
- **String literal**: sequence of characters 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**