CS1110

Lecture 1: Course Overview; Types & Expressions

Announcements

CS1114: Intro to Computing using Matlab and Robotics

- What is CS 1114?
  - An honors-level intro to CS using camera-controlled robots (Sony Aibo, Wowwee Rovio)
  - Meets Tuesday / Thursday 11:15 – 12:05
  - http://www.cs.cornell.edu/courses/cs1114/

Cornell University 1

AEW Workshops

Additional discussion courses

Run parallel to this class—completely optional

See website; talk to advisors in Olin 167.

Bookmark this: www.cs.cornell.edu/courses/cs1110/2013sp/

CS 1110 Spring 2013: Lee and Marschner

- **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/cs1110/2013sp/

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.

Introducing your profs…Prof. Marschner

- Sc.B. Brown ’93, Ph.D. Cornell ’98
- Research area: computer graphics
- Specialty: realistic digital characters (skin, hair, cloth, …)
- Most skin and hair in movies uses his techniques

Technical Oscar (1994) for methods of simulating light scattering in translucent materials

Introducing your profs…Prof. Lee

- A.B. Cornell ’93, Ph.D. Harvard ’97
- Research area: artificial intelligence, specifically “getting computers to understand human language(s)”
    - Can computers **learn how to paraphrase our writing?**

      —*The New York Times* (2003)
    - What kind of language distinguishes **memorable movie quotes**?

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

"FRANKLY, MY DEAR, I DON'T GIVE A DAMN" TOPS AFI'S LIST OF 100 GREATEST MOVIE QUOTES OF ALL TIME

OTHER WINNERS INCLUDE:

THE GODFATHER, ***"I'M GOING TO MAKE HIM AN OFFER HE CAN'T REFUSE"***

THE WIZARD OF OZ, ***"TOTO, I'VE GOT A FEELING WE'RE NOT IN KANSAS ANYMORE"***

AND CASABLANCA, ***"HERE'S LOOKING AT YOU, KID"***

AFI's 100 YEARS 100 "Movie Quotes"

Why Python?

- Python is **easy for beginners**
  - Little to learn before you start "doing"
  - Designed with "rapid prototyping" in mind
- Python is **highly relevant to non-CS majors**
  - NumPy and SciPy heavily used by scientists
- Python is a **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

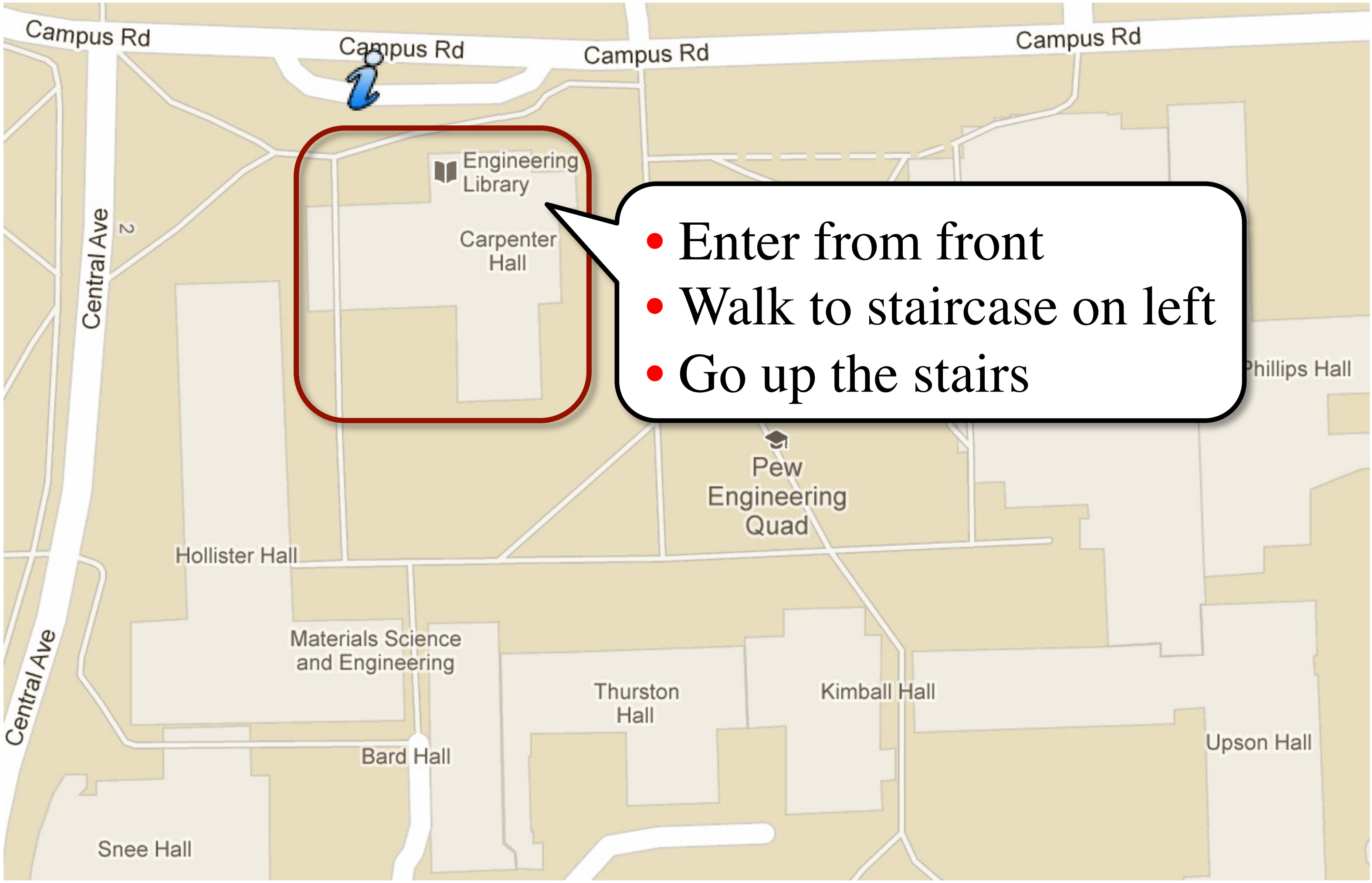
- No prior programming experience necessary
- One semester of calculus
- Engineering-type problems
- Less about software design
- Focus is on training future **engineers that compute**

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**. Participation grade from iClickers
- **Section/labs.** ACCEL Lab, Carpenter 2nd floor
    - Guided exercises with TAs and consultants helping out
    - Please attend the section you registered for
        - Tuesday: 12:20, 1:25, 2:30, 3:35
        - Wednesday: 12:20, 1:25, 2:30, 3:35
    - **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

Academic Integrity

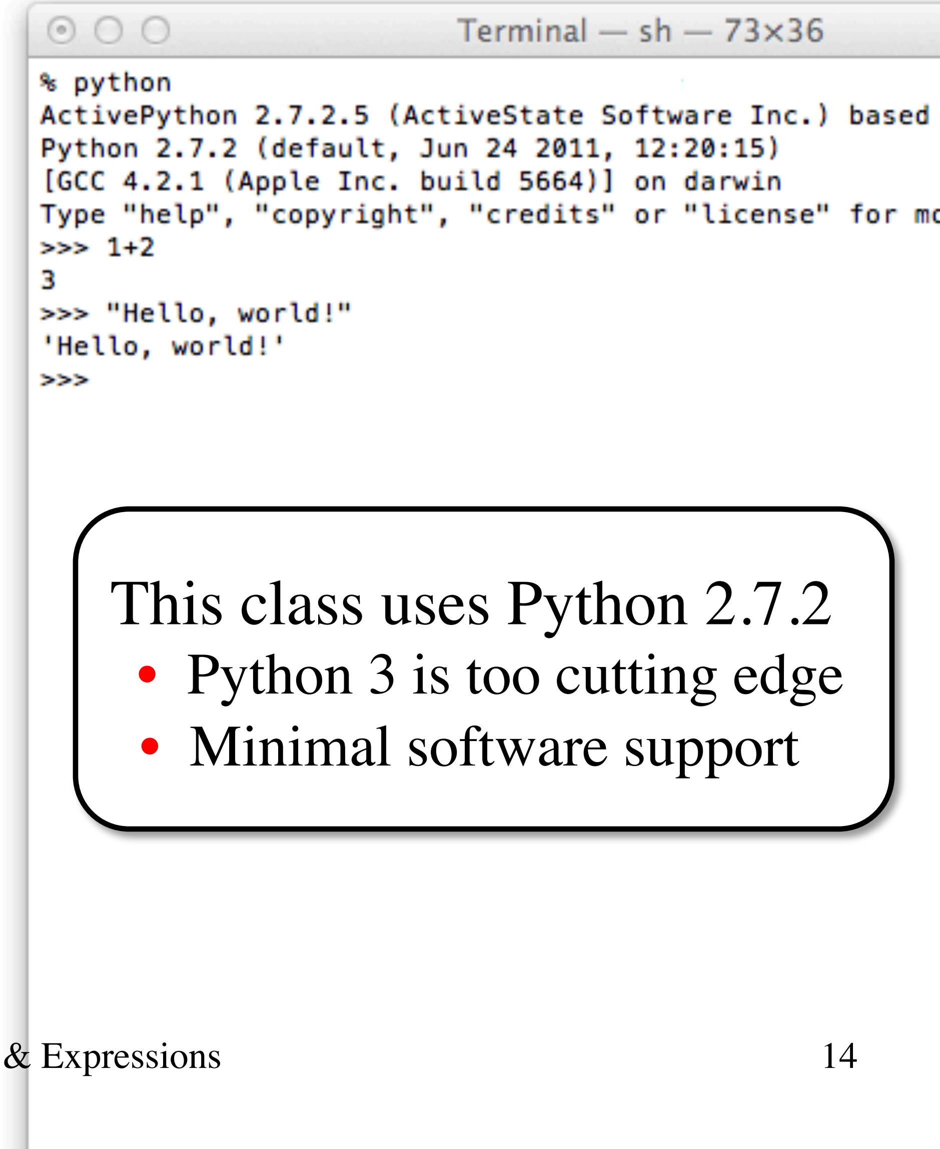
- **Do not cheat**, in any way, shape, or form
  - Will be very explicit about this throughout course
  - Pay attention to all assignment instructions
- In return, we try to be fair about amount of work, grading the work, and giving you a course grade
- See website for more information

CS 1110: A Work in Progress

- **Switched from Java to Python last semester**
- **First semester Python is (still) new to us**
  - We are (still) learning what students find easy/hard
  - We might "overshoot" or "undershoot" this semester
- **Treat all assignments as a dialogue**
  - If something seems too hard, tell someone! (instructor, TA, consultant)
  - We may adjust assignments, labs, lectures to adapt
- **We want you to succeed, not drop out**

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*

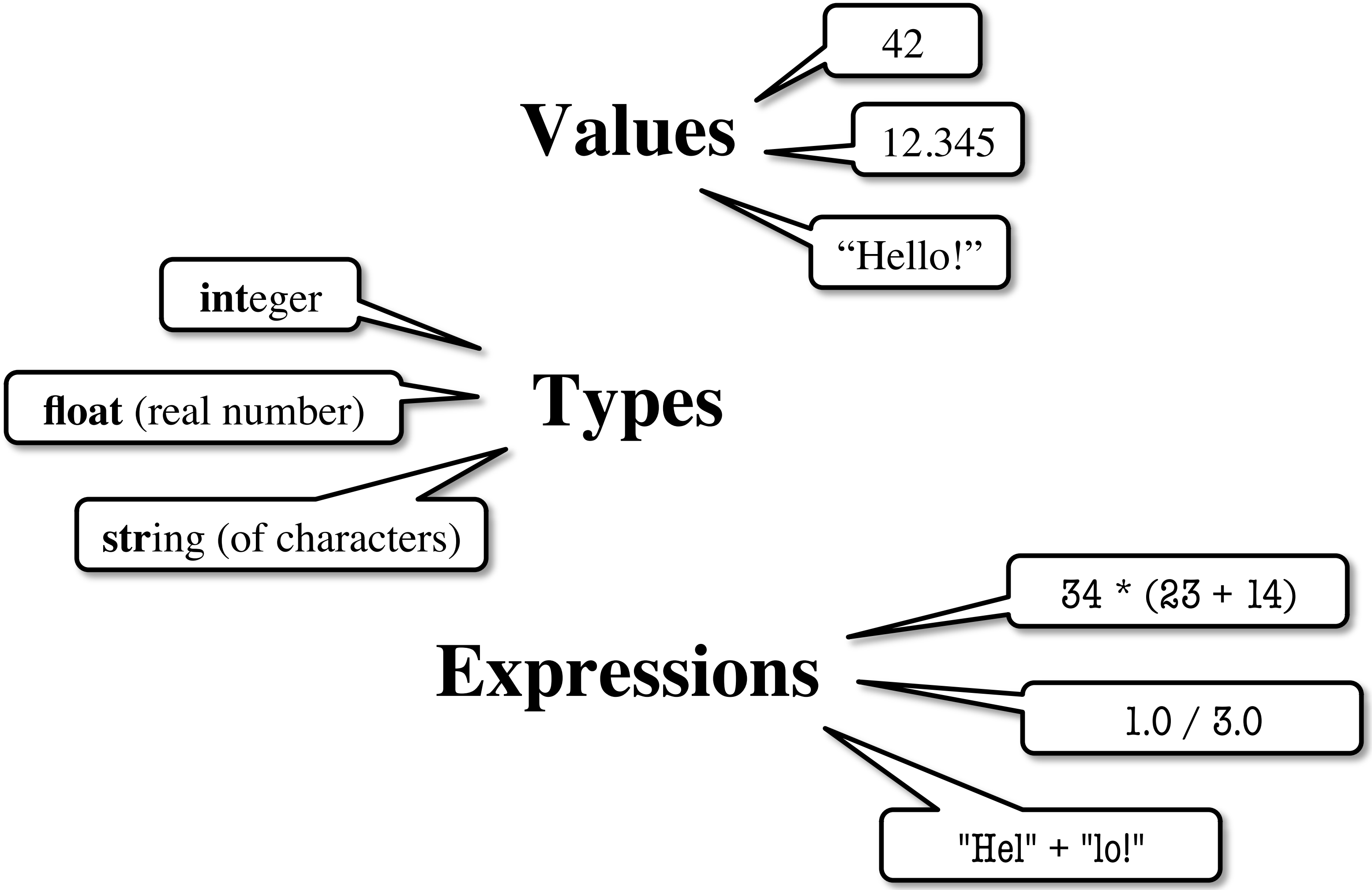


```
% python
ActivePython 2.7.2.5 (ActiveState Software Inc.) based
Python 2.7.2 (default, Jun 24 2011, 12:20:15)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for mo
>>> 1+2
3
>>> "Hello, world!"
'Hello, world!'
>>>
```

This class uses Python 2.7.2

- Python 3 is too cutting edge
- Minimal software support

The Basics



Values

42

12.345

“Hello!”

Types

integer

float (real number)

string (of characters)

Expressions

34 * (23 + 14)

1.0 / 3.0

"Hel" + "lo!"

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

Expressions vs. Statements

Expression

- **Represents** something
  - Python *evaluates it*
  - End result is a value
- Examples:
  - `2.3`

    Literal

  - `(3 * 7 + 2) * 0.1`

    An expression with four literals and some operators

Statement

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

Type: int

- Type **int** (integer):
  - values: …, –3, –2, –1, 0, 1, 2, 3, 4, 5, …
    - Integer literals look like this: `1`, `45`, `43028030` (no commas or periods)
  - operations: `+`, `–`, `*`, `/`, `**`, unary `–`

multiply

to power of

- **Principle**: 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: float

- Type **float** (floating point):
  - values: (approximations of) real numbers
    - 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 `–`
    - But meaning is different for floats
    - **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

Floats Have Finite Precision

- Python stores floats as **binary fractions**
  - Integer mantissa times a power of 2
  - Example: 1.25 is $\underbrace{10}_{\text{mantissa}} * 2^{\underbrace{-3}_{\text{exponent}}}$
- 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 >>>

Type: str

- Type **str** (string of characters):
  - 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**

Type: bool

- Type **bool** (Boolean logical value):
  - 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!

Converting Values Between Types

- Basic form: *type*(*value*)
  - `float(2)` converts value 2 to type **float** (value now 2.0)
  - `int(2.6)` converts value 2.6 to type **int** (value now 2)
  - Explicit conversion is also called "casting"
- Narrow to wide: **bool** ⇒ **int** ⇒ **float**
  - *Widening*. Python does automatically if needed
    - **Example**: `1/2.0` evaluates to `0.5` (casts 1 to **float**)
  - *Narrowing*. Python *never* does this automatically
    - Narrowing conversions cause information to be lost
    - **Example**: `float(int(2.6))` evaluates to 2.0