CS1110

Lecture 1: Course Overview; Types & Expressions

Announcements

Two handouts: announcements and slides. The slides are a note-taking guide, *not* a complete copy of this presentation (which you can find on the website).

Watch your email. Info about labs (discussions sections), etc. continues to be sent. Check your spam folders...

Don't worry if the reading is hard. Doing the reading first and *then* seeing lecture will help.

AEW Workshops

Additional discussion courses

They run parallel to this class—completely optional

See website; talk to advisors in Olin 167.

Slides by D. Gries, L. Lee, S. Marschner, W. 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.

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
- In 2013, named a Fellow of the Association for the Advancement of Artificial Intelligence (AAAI)
  - 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

CS 1112: Matlab

- No prior programming experience necessary
- One semester of calculus
- Engineering-type problems
- Less about software design

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

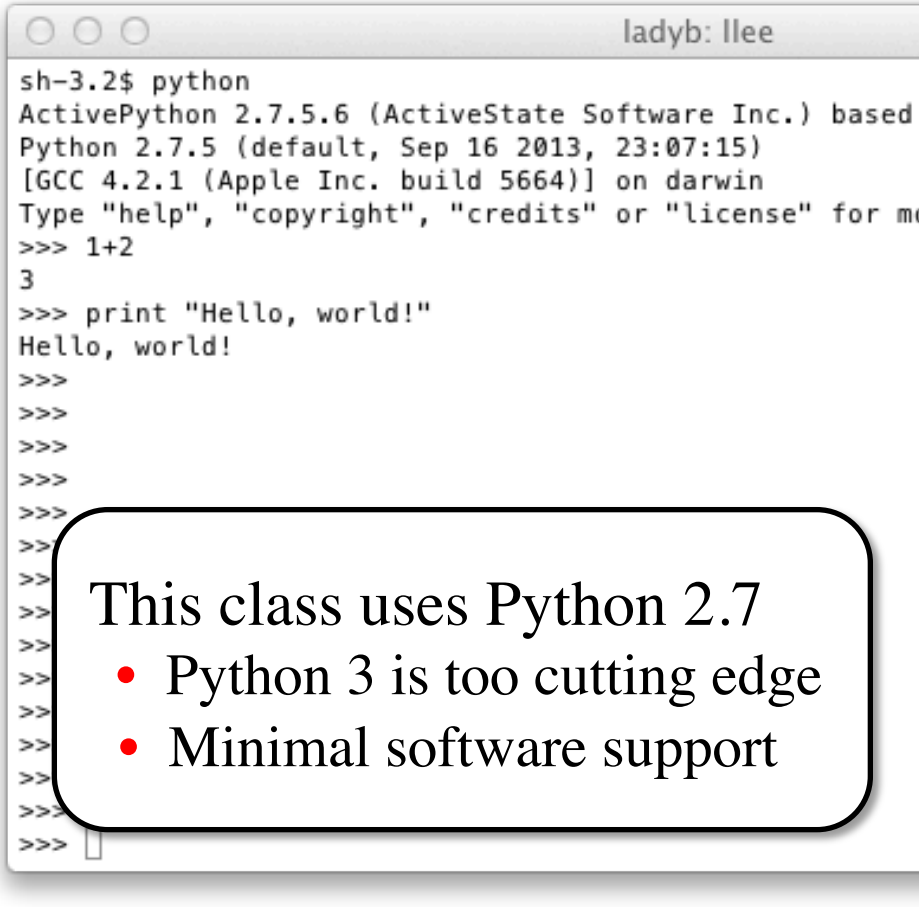
Class Structure

- **Lectures.** Every Tuesday/Thursday
  - Not just slides; interactive demos almost every lecture.
    Handouts posted to the website the afternoon before class.
    Slides and code posted to the website after class.
  - You may attend *either* lecture time (9:05 or 11:15).
  - **Semi-Mandatory**: Participation grade from **iClickers**.
    The 11:15 lecture is recorded by **VideoNote**.
- **Labs (aka "discussion sections")**
  - Guided exercises with TAs and consultants helping out.
  - Handouts posted to the website the Monday before.
  - **Mandatory**: Missing more than 2 lowers your final grade

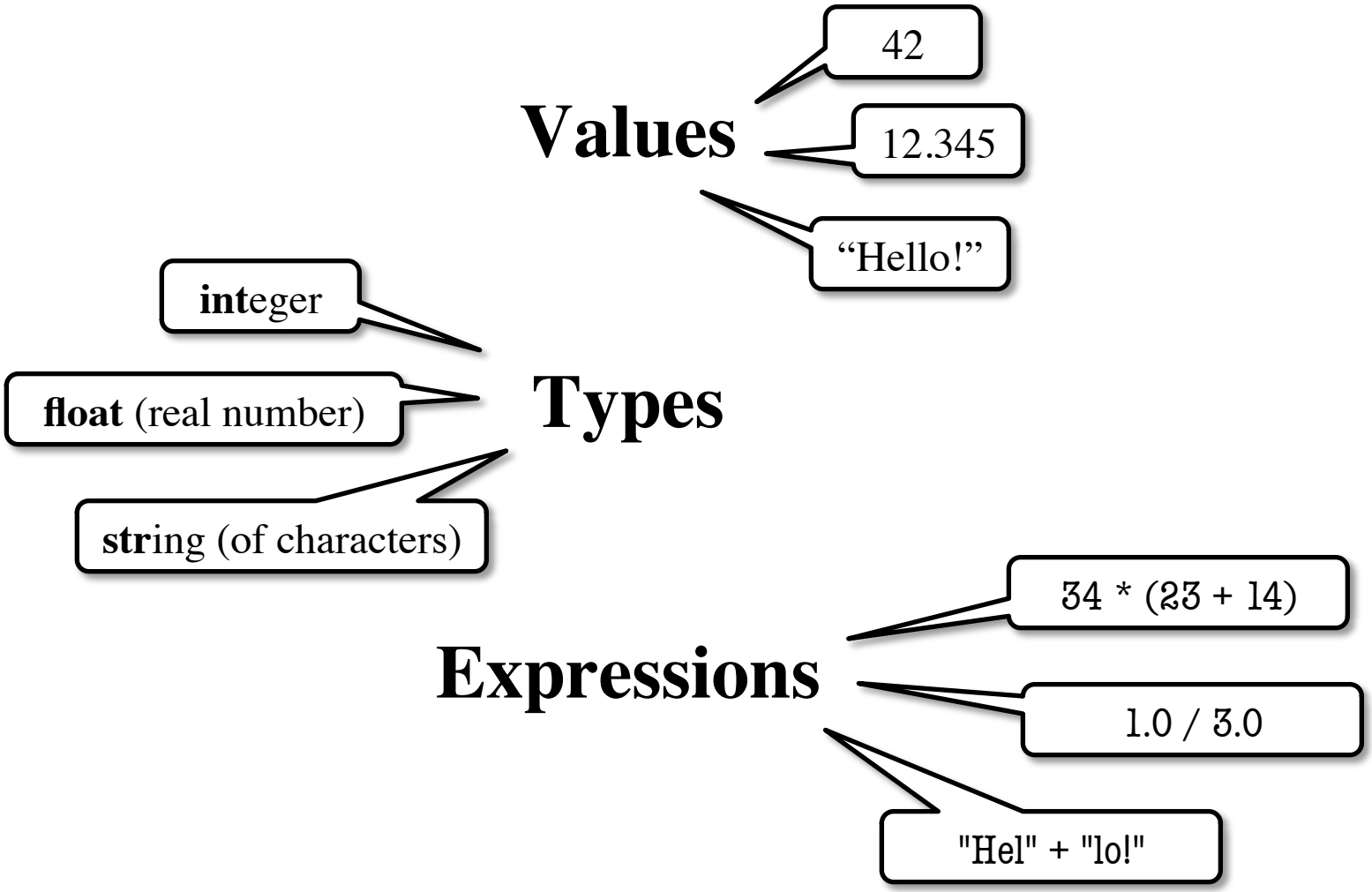
Getting Started with Python

- 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 the *interactive mode*
  - Type commands at >>>
- First experiments:

  evaluate *expressions*



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?

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

Literal

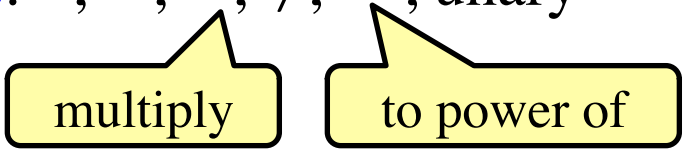
An expression with four literals and some operators

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 `-`



- **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)

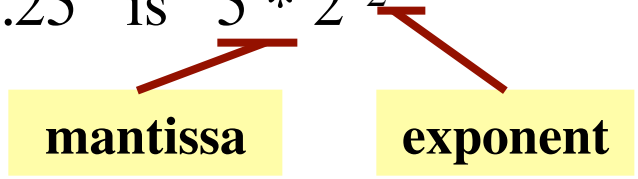
Type: float

- 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 `-`
    - 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

Floats Have Finite Precision

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



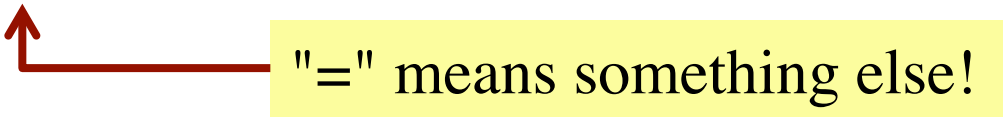
- Impossible to write most real numbers this way exactly
  - Similar to problem of writing 1/3 with decimals
  - Python chooses the closest binary fraction it can
- This approximation results in **representation error**
  - When combined in expressions, the error can get worse
  - **Example**: type `0.1 + 0.2` at the prompt >>>

Type: str

- Type String or **str**:
  - 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 boolean or **bool**:
  - 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



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