CS 1110, Spring 2014

Readings for today and next week: Ch 1, Ch 2.1-2.9

• Website:
  ▪ http://www.cs.cornell.edu/courses/cs1110/2014sp/

• 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
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
Communication

★ **Piazza:** question-and-answer forum  
  https://piazza.com/cornell/spring2014/cs1110/home

- **Emailing us:** Send a *single* email with *both* llee@cs.cornell.edu and srm@cs.cornell.edu in the To: line.  
  You'll get answered faster, and we'll keep better track.

- **Email from us:** Please check your spam filters for mail from LJL2@cornell.edu, SRM2@cornell.edu, or with [CS1110] in the subject line.
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

This class uses Python 2.7
  • Python 3 is too cutting edge
  • Minimal software support
Representing Values

• **Everything** on a computer reduces to numbers
  - Letters represented by numbers (ASCII codes)
  - Pixel colors are three numbers (red, blue, green)

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Represents</strong> something</td>
<td><strong>Does</strong> something</td>
</tr>
<tr>
<td>- Python <em>evaluates it</em></td>
<td>- Python <em>executes it</em></td>
</tr>
<tr>
<td>- End result is a value</td>
<td>- Need not result in a value</td>
</tr>
<tr>
<td><strong>Examples:</strong></td>
<td><strong>Examples:</strong></td>
</tr>
<tr>
<td>- 2.3</td>
<td>- <code>print “Hello”</code></td>
</tr>
<tr>
<td>- ((3 * 7 + 2) * 0.1)</td>
<td>- <code>import sys</code></td>
</tr>
</tbody>
</table>

*Literal* 

Expression with four literals and some operators
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, the 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
  ▪ \(-22.51 \times 10^6\) is \(-22.51 \times 10^6\) or \(-22510000\)
  ▪ \(22.51 \times 10^{-6}\) is \(22.51 \times 10^{-6}\) 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 $10 \times 2^{-3}$

• 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: " abcex$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

"=" means something else!
Converting Values Between Types

• Basic form: \textit{type(value)}
  - \texttt{float(2)} converts value 2 to type \texttt{float} (value now 2.0)
  - \texttt{int(2.6)} converts value 2.6 to type \texttt{int} (value now 2)
  - Explicit conversion is also called “casting”

• Narrow to wide: \texttt{bool ⇒ int ⇒ float}
  - \textit{Widening}. Python does automatically if needed
    - Example: \(1/2.0\) evaluates to 0.5 (casts 1 to \texttt{float})
  - \textit{Narrowing}. Python \textit{never} does this automatically
    - Narrowing conversions cause information to be lost
    - Example: \texttt{float(int(2.6))} evaluates to 2.0