Lecture 1

Course Overview, Types & Expressions
• **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/](http://www.cs.cornell.edu/courses/cs1110/2012fa/)
Why Programming in 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
Intro Programming Classes Compared

<table>
<thead>
<tr>
<th>CS 1110: Python</th>
<th>CS 1112: Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>• No prior programming experience necessary</td>
<td>• No prior programming experience necessary</td>
</tr>
<tr>
<td>• No calculus</td>
<td>• One semester of calculus</td>
</tr>
<tr>
<td>• Non-numerical problems</td>
<td>• Engineering-type problems</td>
</tr>
<tr>
<td>• More about software design</td>
<td>• Less about software design</td>
</tr>
<tr>
<td>• Focus is on training future computer scientists</td>
<td>• Focus is on training future engineers that compute</td>
</tr>
</tbody>
</table>

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

## CS 1115
- Content of 1112 + GUIs
  - Class still in MatLab
  - But also focus of 1110
  - Requires CS experience
- Great opportunity
  - Smaller (20 students!)
  - Like a discussion section

## CS 1132/1133
- One credit courses
  - Requires CS experience
  - Learn “another” language
  - **1132**: MatLab
  - **1133**: Python
- Ideal for grad students
  - 1110 requires a lot of time
  - Takes away from research
Class Structure

- **Lectures.** Every Tuesday/Thursday
  - Not just slides; interactive demos almost every lecture
  - Please stay in your lecture (no room to move between)
  - **Semi-Mandatory.** 1% Participation grade from iClickers

- **Section/labs.** ACCEL Lab, Carpenter 2nd floor
  - The “overflow sections” are in **Phillips 318**
  - Guided exercises with TAs and consultants helping out
    - Tuesday: 12:20, 1:25, 2:30, 3:35
    - Wednesday: 10:10, 11:15, 12:20, 1:25, 2:30, 3:35, 7:30
  - Contact Molly ([mjt264@cornell.edu](mailto:mjt264@cornell.edu)) for section conflicts
  - **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

• Complete **Quiz: About the Course on CMS**
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/cs1110/2013fa/
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
The Basics

Overview, Types & Expressions

Values
- 42
- 12.345
- “Hello!”

Types
- integer
- float (real number)
- string (of characters)

Expressions
- $34 \times (23 + 14)$
- $1.0 / 3.0$
- "Hello!" + "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
  - (3 * 7 + 2) * 0.1

**Statement**
- **Does** something
  - Python executes it
  - Need not result in a value
- **Examples:**
  - print “Hello”
  - import sys
Type: Set of values and the operations on them

• Type **int** (integer):
  - **values**: \( ... , -3, -2, -1, 0, 1, 2, 3, 4, 5, ... \)
  - **operations**: +, −, *, /, **, unary −

  "Whole" numbers w/o decimals

• **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 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`
• Python stores floats as **binary fractions**
  ▪ Integer mantissa times a power of 2
  ▪ Example: 12.5 is $10 \times 2^{-3}$

• Impossible to write every number this way exactly
  ▪ Similar to problem of writing $\frac{1}{3}$ 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
    - 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 || 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!
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
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