Lecture 1

Course Overview, Python Basics
We Are Very Full!

- Lectures and Labs are at fire-code capacity
  - We cannot add sections or seats to lectures
  - You may have to wait until someone drops
- **No auditors** are allowed this semester
  - All students must do assignments
  - **Graduate students** should take CS 1133
- CS 1112 has plenty of room for students
About Your Instructor: Walker White

- **Director**: GDIAC
  - Game Design Initiative at Cornell
  - Teach game design
- (and CS 1110 in fall)
CS 1110 Fall 2015

• 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

• Website:
  - [www.cs.cornell.edu/courses/cs1110/2015fa/](http://www.cs.cornell.edu/courses/cs1110/2015fa/)
# 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>• <em>Slight</em> focus on</td>
<td>• <em>Slight</em> focus on</td>
</tr>
<tr>
<td>▪ Software engineering</td>
<td>▪ Scientific computation</td>
</tr>
<tr>
<td>▪ Application design</td>
<td>▪ Engineering applications</td>
</tr>
</tbody>
</table>

But either course serves as a pre-requisite to CS 2110
CS 1133: Short Course in Python

- Catalogue lists as “Transition to Python”
  - Says it requires programming experience
  - This is a lie
- 1-credit course in how to use Python
  - All the Python of 1110 without the theory
  - Three assignments; no exams
  - No experience required
- For graduate students who need Python
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
Class Structure

• **Lectures.** Every Tuesday/Thursday
  - Not just slides; interactive demos almost every lecture
  - Because of enrollment, please stay with your section
  - **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, 7:20
  - Contact Jessica (jd648@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 Thursday
  - Will periodically ask questions during lecture
  - Will get credit for answering – even if wrong
  - iClicker App for smartphone is not acceptable
- **Python.** Necessary if you want to use own computer
  - See course website for how to install the software

8/25/15  Overview, Types & Expressions  10
This Class is OS Agnostic

Windows 7 or higher

OS X 10.9 or higher
Not Guaranteed to Support New OSs
Not Guaranteed to Support New OSs
Not Guaranteed to Support New OSs

Windows 10

OS X 10.11

Supreme Court declines to review same-sex marriage cases

FIFA 15: UT

All Apps

Store

Mail
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
   - Complete the Quiz
   - 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/cs1110/2015fa/](http://www.cs.cornell.edu/courses/cs1110/2015fa/)
Academic Integrity

• Every semester we have cases of plagiarism
  ▪ Claiming the work of others as your own
  ▪ This is an Academic Integrity violation

• Protect yourself by citing your sources
  ▪ Just like in writing a paper for freshman seminar
  ▪ Course website covers how and when to cite

• Complete Academic Integrity Quiz on CMS
  ▪ Must complete successfully to stay in class
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 has many “issues”
• Minimal software support
The Basics

Values
- Integer
- Float (real number)
- String (of characters)
- 42
- 12.345
- "Hello!"

Types
- Integer
- Float (real number)
- String (of characters)

Expressions
- 34 * (23 + 14)
- 1.0 / 3.0
- "Hello" + "lo!"
**Python and Expressions**

- An expression **represents** something
  - Python *evaluates it* (turns it into a value)
  - Similar to what a calculator does

- Examples:
  - 2.3
    - Literal (evaluates to self)
  - \[(3 \times 7 + 2) \times 0.1\]
    - An expression with four literals and some operators
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:**
  - **Memorize this definition!**
  - A set of values and the operations on them.
    - Examples of operations: +, -, /, *
    - The meaning of these depends on the type
Example: Type int

- Type **int** represents **integers**
  - **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)**
Example: Type float

- Type **float** (floating point) represents **real numbers**
  - **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
  - \(-22.51\text{e}6\) is \(-22.51 \times 10^6\) or \(-22510000\)
  - \(22.51\text{e}−6\) 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 $5 \times 2^{-2}$

- 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 >>>

8/25/15 Overview, Types & Expressions 23
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!
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**
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