## CS 1110 Fall 2012: Walker White

- **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/

## 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**. 1% Participation grade from iClickers
- **Section/labs.** ACCEL Lab, Carpenter 2nd floor
  - Guided exercises with TAs and consultants helping out
  - Register for ANY section, but go to the one you want
    - Tuesday: 12:20, 1:25, 2:30, 3:35
    - Wednesday: 10:10, 11:15, 12:20, 1:25, 2:30, 3:35, 7:20
  - **Mandatory**. Missing more than 2 lowers your final grade

## 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


## Helping You Succeed: Other Resources

- **Consultants.** ACCEL Lab Green Room
  - Daily office hours (see website) with consultants
  - Very useful when working on assignments
- **AEW Workshops**. Additional discussion course
  - Runs parallel to this class – completely optional
  - See website; talk to advisors in Olin 167.
- **Piazza.** Online forum ask and answer questions
  - Go here first before sending question in e-mail
- **Office Hours.** Talk to the professor!
  - Available in Hollister 202 between lectures

## Assignments

- Major portion (40%) of your final grade
  - Larger projects due every two weeks
- First assignment requires **mastery**
  - Submit, get feedback, resubmit, … until correct
  - Everyone eventually scores 10/10
- Later assignments are designed to be fun
  - Examples: graphics, image manipulation
  - Final project is a Breakout game project
- Submitted via **Course Management System** (CMS)
  - Visit cms.csuglab.cornell.edu/ to check you are enrolled

## 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/cs1110/2012fa/

## Expressions vs Statements

| Expression | Statement |
|---|---|
| • **Represents** something<br>▪ Python *evaluates it*<br>▪ End result is a value | • **Does** something<br>▪ Python *executes it*<br>▪ Need not result in a value |
| • Examples:<br>▪ 2.3 (Value)<br>▪ (3+5)/4 (Complex Expression) | • Examples:<br>▪ print "Hello"<br>▪ import sys |

Will see later this is not a clear cut separation

## Type: Set of values and the operations on them

**Memorize this definition!**
**Write it down several times.**

- Type **int**:
  - values: …, –3, –2, –1, 0, 1, 2, 3, 4, 5, … ("Whole" numbers w/o decimals)
  - operations: +, –, *, /, **, unary – (multiply; to power of)
- **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 **floating point** (or **float**):
  - values: fractions and/or real numbers
    - If you add a decimal, Python assumes it is a **float** (e.g. **2.0**)
    - Without a decimal, Python assumes it is an **int** (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) floats
  - 22.51e6 is $-22.51 * 10^{6}$ or –22510000
  - 22.51e–6 is $22.51 * 10^{-6}$ or 0.00002251
  - Must start with an integer: 1e5 is ok, but e5 is not

## Representation Error

- Python stores floats as **binary fractions**
  - Integer mantissa times a power of 2
  - Example: 12.5 is $10 * 2^{-3}$ (mantissa; exponent)
- Impossible to write every number 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: Set of values and the operations on them

- Type boolean or **bool**:
  - values: **True**, **False**
  - 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 comparision: i < j  i <= j  i >= j  i > j
  - Equality, inequality: i == j  i != j (==, not =)

## Casting: Converting Value Types

- Basic form: *type*(*value*)
  - float(2) casts value 2 to type **float** (value now 2.0)
  - int(2.56) casts value 2.56 to type **int** (value is now 2)
- Narrow to wide: **bool ⇒ int ⇒ float**
  - *Widening* Cast. Python automatically if needed
    - **Example**: 1/2.0 evaluates to 0.5 (casts 1 to **float**)
  - *Narrowing* Cast. Python *never* does automatically
    - Narrowing casts cause information to be lost
    - **Example**: float(int(2.56)) evaluates to 2.0