CS 1110 Fall 2018

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:

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
  - 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 Jenna (jls478@cornell.edu) for section conflicts
  - Mandatory. Missing more than 2 lowers your final grade

Class Materials

- **Textbook.** *Think Python, 2nd Ed.* by Allen Downey
  - *Optional* text; only used as a reference
  - Book available for free as PDF or eBook
  - Hardbound copies only available online
- **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

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. Complete Lab 0
   - Install (Anaconda) Python
   - Answer online questions
   - Everything is on website!
   - Piazza instructions
   - Class announcements
   - Consultant calendar
   - Reading schedule
   - Lecture slides
   - Exam dates
   - Check it regularly:
     - [www.cs.cornell.edu/courses/cs1110/2017fa/](http://www.cs.cornell.edu/courses/cs1110/2017fa/)

Getting Started with Python

- Designed to be used from the “command line”
  - OS X/Linux: **Terminal**
  - Windows: **PowerShell**
  - 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 3.6

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 * 7 + 2) * 0.1` — An expression with four literals and some operators
Type: Set of values and the operations on them

- **Type int** represents **integers**
  - values: ..., -3, -2, -1, 0, 1, 2, 3, 4, 5, ...
  - Integer literals look like this: 1, 45, 4308030 (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

- **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. 8.0)
    - Without a decimal a number is an int literal (e.g. 2)
  - operations: +, -, *, /, **, unary ~
  - Notice that float has a different division operator
    - Example: 1.0/2.0 evaluates to 0.5
  - **Exponent notation** is useful for large (or small) values
    - \(-22.31e6\) is \(-22.51 \times 10^6\) or \(-2251000\)
    - \(2.51e-6\) 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 \(5 \times 2^{-3}\)
  - 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 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!

Type: Set of values and the operations on them

- **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: " abe@example.com or "Hello World!"
  - Single quotes: 'Hello World!'
  - Concatenation can only apply to strings.
    - 'ab' + 'cd' evaluates to 'abcd'
    - 'ab' + 2 produces an **error**
    - The meaning of + depends on the type

Type: Set of values and the operations on them

- **Type int** represents **integers**
  - values: ..., -3, -2, -1, 0, 1, 2, 3, 4, 5, ...
  - Integer literals look like this: 1, 45, 4308030 (no commas or periods)
  - operations: +, -, *, //, **, unary ~

Type: Set of values and the operations on them

- **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. 8.0)
    - Without a decimal a number is an int literal (e.g. 2)
  - operations: +, -, *, /, **, unary ~
  - Notice that float has a different division operator
    - Example: 1.0/2.0 evaluates to 0.5
  - **Exponent notation** is useful for large (or small) values
    - \(-22.31e6\) is \(-22.51 \times 10^6\) or \(-2251000\)
    - \(2.51e-6\) 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 \(5 \times 2^{-3}\)
  - 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 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!

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

- **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: " abe@example.com or "Hello World!"
  - Single quotes: 'Hello World!'
  - Concatenation can only apply to strings.
    - 'ab' + 'cd' evaluates to 'abcd'
    - 'ab' + 2 produces an **error**
    - The meaning of + depends on the type

**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