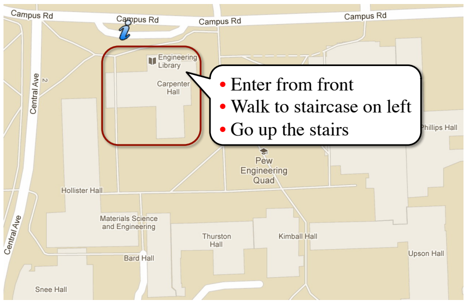
CS 1110 Spring 2013: Lee and Marschner

- **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/cs1110/2013sp/

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**. Participation grade from iClickers
- **Section/labs.** ACCEL Lab, Carpenter 2nd floor
  - Guided exercises with TAs and consultants helping out
  - Please attend the section you registered for
    - Tuesday: 12:20, 1:25, 2:30, 3:35
    - Wednesday: 12:20, 1:25, 2:30, 3:35
  - **Mandatory**. Missing more than 2 lowers your final grade

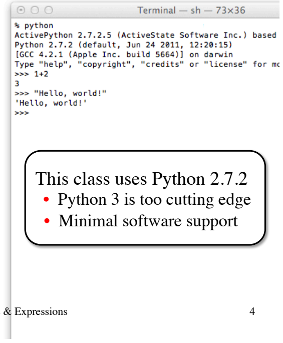
ACCEL Labs



- Enter from front
- Walk to staircase on left
- Go up the stairs

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*



```
% python
ActivePython 2.7.2.5 (ActiveState Software Inc.) based
Python 2.7.2 (default, Jun 24 2011, 12:20:15)
[GCC 4.2.1 (Apple Inc. build 5664)] on darwin
Type "help", "copyright", "credits" or "license" for mc
>>> 1+2
3
>>> "Hello, world!"
'Hello, world!'
>>>
```

This class uses Python 2.7.2
- 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)
  - 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 — Literal
  - (3 * 7 + 2) * 0.1

Expression with three literals and some operators

Statement

- **Does** something
  - Python *executes it*
  - Need not result in a value
- Examples:
  - print “Hello”
  - import sys

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 –

multiply

to power of

- **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)

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 –
    - 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

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 * 2^{-3}

mantissa

exponent

- 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: `" 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**

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: *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