Lecture 2

Variables & Assignment
# Announcements for Today

## If Not Done Already

- Enroll in Piazza
- Sign into CMS
  - Fill out the Survey
  - Complete Quiz 0
- Read the textbook
  - Chapter 1 (browse)
  - Chapter 2 (in detail)

## Lab 1

- Getting started with Python
  - Good time to bring a laptop
  - Help you install the software
- Please stay in your section
  - E-mail conflicts to Molly
    - mjt264@cornell.edu
- Have one week to complete
  - Fill out questions on handout
  - Show to TA before next lab

9/3/13

Variables & Assignment 2
Labs vs. Assignments

**Labs**
- Held every week
- Graded on *completeness*
  - Always S/U
  - Try again if not finished
- Indirect affect on grade
  - Can miss up to 2 labs
  - After that, grade reduced
- Similar to language drills
  - Simple, but take time

**Assignments**
- Every two weeks
  - First one due Sep. 25
- Graded on *correctness*
  - Assign points out of 100
- But *first* one is for *mastery*
  - Resubmit until perfect grade
- 40% of your final grade
- Designed to be more fun
  - graphics, game design
Helping You Succeed in this Class

• **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 to ask and answer questions
  - Go here first **before** sending question in e-mail

• **Office Hours.** Talk to the professor!
  - Available in Carpenter Hall Atrium between lectures
iClickers

• Have you registered your iclicker?
• If not, visit
  ▪ atcsupport.cit.cornell.edu/pollsrvc/
• Instructions on iclickers can be found here:
  ▪ atc.cit.cornell.edu/course/polling/clickers.cfm
• Find these links on the course webpage
  ▪ Click “Texts”
  ▪ Scroll down on the page that opens
Warm-Up: Using Python

How do you plan to use Python?

A. I want to work mainly in the ACCEL lab
B. I want to use my own Windows computer
C. I want to use my own Macintosh computer
D. I want to use my own Linux computer
E. I will use whatever I can get my hands on
Type: Set of values and the operations on them

- **Type int:**
  - **Values:** integers
  - **Ops:** +, –, *, /, %, **

- **Type float:**
  - **Values:** real numbers
  - **Ops:** +, –, *, /, **

- **Type bool:**
  - **Values:** True and False
  - **Ops:** not, and, or

- **Type str:**
  - **Values:** string literals
    - Double quotes: "abc"
    - Single quotes: 'abc'
  - **Ops:** + (concatenation)

Will see more types in a few weeks
Operator Precedence

• What is the difference between the following?
  ▪ 2*(1+3)  add, then multiply
  ▪ 2*1 + 3  multiply, then add

• Operations are performed in a set order
  ▪ Parentheses make the order explicit
  ▪ What happens when there are no parentheses?

• **Operator Precedence**: The *fixed* order Python processes operators in *absence* of parentheses
Precedence of Python Operators

- **Exponentiation**: **
- **Unary operators**: + –
- **Binary arithmetic**: * / %
- **Binary arithmetic**: + –
- **Comparisons**: < > <= >=
- **Equality relations**: == !=
- **Logical not**
- **Logical and**
- **Logical or**

- Precedence goes downwards
  - Parentheses highest
  - Logical ops lowest
- Same line = same precedence
  - Read “ties” left to right
  - Example: 1/2*3 is (1/2)*3

- Section 2.7 in your text
- See website for more info
- Major portion of Lab 1
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 does 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
# Expressions vs Statements

<table>
<thead>
<tr>
<th>Expression</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>Represents</strong> something</td>
<td>• <strong>Does</strong> something</td>
</tr>
<tr>
<td>▪ Python <em>evaluates it</em></td>
<td>▪ Python <em>executes it</em></td>
</tr>
<tr>
<td>▪ End result is a value</td>
<td>▪ Need not result in a value</td>
</tr>
<tr>
<td>• Examples:</td>
<td>• Examples:</td>
</tr>
<tr>
<td>▪ 2.3</td>
<td>▪ <em>print “Hello”</em></td>
</tr>
<tr>
<td>▪ ((3+5)/4)</td>
<td>▪ <em>import sys</em></td>
</tr>
</tbody>
</table>

Will see later this is not a clear cut separation
Variables (Section 2.1)

• A variable is
  ▪ a named memory location (box),
  ▪ a value (in the box)

• Examples
  
  - $x$ 5 Variable $x$, with value 5 (of type int)
  - area 20.1 Variable area, w/ value 20.1 (of type float)

• Variable names must start with a letter
  ▪ So 1e2 is a float, but e2 is a variable name
Variables and Assignment Statements

• Variables are created by assignment statements
  ▪ Create a new variable name and give it a value
    \[ x = 3 \]
  ▪ This is a statement, not an expression
  ▪ Tells the computer to DO something (not give a value)
  ▪ Typing it into >>> gets no response (but it is working)

• Assignment statements can have expressions in them
  ▪ These expressions can even have variables in them
    \[ x = x + 2 \]
Execute the Statement: \( x = x + 2 \)

- Draw variable \( x \) on piece of paper:
  - x

- Step 1: evaluate the expression \( x + 2 \)
  - For \( x \), use the value in variable \( x \)
  - Write the expression somewhere on your paper

- Step 2: Store the value of the expression in \( x \)
  - Cross off the old value in the box
  - Write the new value in the box for \( x \)

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

A: I did it correctly!
B: I drew another box named \( x \)
C: I did something else
D: I did nothing – just watched
Execute the statement: \( x = 3.0 \times x + 1.0 \)

- You have this:
  
  \[ x \quad \text{22.0} \]

- Execute this command:
  
  - Step 1: **Evaluate** the expression \( 3.0 \times x + 1.0 \)
  - Step 2: **Store** its value in \( x \)

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

A: I did it correctly!
B: I drew another box named \( x \)
C: I did something else
D: I did nothing – just watched
Execute the statement: \( x = 3.0 \times x + 1.0 \)

- You now have this:
  \[ x \quad \underline{XX} \quad 22.0 \]

- The command:
  - Step 1: **Evaluate** the expression \( 3.0 \times x + 1.0 \)
  - Step 2: **Store** its value in \( x \)

- This is how you execute an assignment statement
  - Performing it is called **executing the command**
  - Command requires both **evaluate** AND **store** to be correct
  - Important **mental model** for understanding Python
Exercise: Understanding Assignment

• Add another variable, interestRate, to get this:

  x 22.0  interestRate  5.5

• Execute this assignment:

  interestRate = x / interestRate

• Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

  A: I did it correctly!
  B: I drew another box called “interestRate”
  C: I stored the value in the box for x
  D: I thought it would use int division
  E: I did something else (or nothing)
Exercise: Understanding Assignment

• You now have this:

   x ✗✗ 22.0  interestRate ✗ 5.5  intrestRate 27.5

• Execute this assignment:

   \texttt{intrestRate = x + interestRate}

• Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

   A: I did it correctly!
   B: I stored the value in “interestRate”
   C: I stored the value in x
   D: I did something else (or nothing)

Spelling mistakes in Python are bad!!

9/3/13  Variables & Assignment  19
Dynamic Typing

- Python is a *dynamically typed language*
  - Variables can hold values of any type
  - Variables can hold different types at different times
  - Use `type(x)` to find out the type of the value in `x`
  - Use names of types for conversion, comparison
- The following is acceptable in Python:
  ```python
  >>> x = 1
  x contains an `int` value
  >>> x = x / 2.0
  x now contains a `float` value
  ```
- Alternative is a *statically typed language* (e.g. Java)
  - Each variable restricted to values of just one type
Dynamic Typing

• Often want to track the type in a variable
  ▪ What is the result of evaluating \( x / y \)?
  ▪ Depends on whether \( x, y \) are \texttt{int} or \texttt{float} values

• Use expression \texttt{type(<expression>)} to get type
  ▪ \texttt{type(2)} evaluates to \texttt{<type 'int'>}
  ▪ \texttt{type(x)} evaluates to type of contents of \( x \)

• Can use in a boolean expression to test type
  ▪ \texttt{type('abc')} == \texttt{str} evaluates to \texttt{True}