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

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

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

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

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 $\boxed{5}$ Variable **x**, with value 5 (of type **int**)

  area $\boxed{20.1}$ 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

    the value (3)

    the variable (x)
- 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

    the expression (x + 2)

    the variable (x)

Execute the Statement: `x = x + 2`

- The variable x

  x $\boxed{5}$
- The command:
  - Step 1: **Evaluate** the expression x + 2
  - 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

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

    `type(x) == int`
    `x = float(x)`
    `type(x) == float`
- The following is acceptable in 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 **int** or **float** values
- Use expression `type(<expression>)` to get type
  - `type(2)` evaluates to `<type 'int'>`
  - `type(x)` evaluates to type of contents of x
- Can use in a boolean expression to test type
  - `type('abc') == str` evaluates to **True**