Lecture 2

Expressions, Types & Variables

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
- Go to section that you want
  - Tue: 12:20, 1:25, 2:30, 3:35
  - Wed: 10:10, 11:15, 12:20, 1:25, 2:30, 3:35, 7:30
- Have one week to complete
  - Fill out questions on handout
  - Show to TA before next lab

Fix Your E-Mails (The Following Bounce)

- rjc362@cornell.edu
- stephen.markham@sjprephawks.org
- ahanson@hotchkiss.org
- adtumuluri12@gmail.com

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

Expressions vs Statements

Expression

- **Represents** something
  - Python *evaluates it*
  - End result is a value
- Examples:
  - `2.3` — Value
  - `(3+5)/4` — Complex Expression

Statement

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

Will see later this is not a clear cut separation

Types

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
- 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

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 $\underbrace{10}_{\text{mantissa}} * 2^{\overbrace{-3}^{\text{exponent}}}$

Do not need details

Just understand "floats are not exact"

- 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 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

==, not =

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

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

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

    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

Exercise: Understanding Assignment

A: I did it correctly!
B: I drew another box named x
C: I did something else
D: I did nothing –just watched

- Draw variable x on piece of pa

  x ~~5~~ 7

- 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.

Exercise: Understanding Assignment

A: I did it correctly!
B: I drew another box named x
C: I did something else
D: I did nothing –just watched

- You have this:

  x $\boxed{\cancel{5}\ \cancel{7}\ 22}$

- Execute this command:
  - Step 1: **Evaluate** the expression 3 * x + 1
  - 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.

Exercise: Understanding Assignment

- You now have this:

  x ~~5~~ ~~7~~ 22

- The command:
  - Step 1: **Evaluate** the expression 3 * x + 1
  - 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

- Put another variable y on your paper to get this:

x ~~3~~ ~~7~~ 22 variable ~~3~~ 7

- Execute this assignment:

variable = x / variable

- 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 "variable"
C: I stored the value in the box for x
D: I forgot about division in **int** types
E: I did something else (or nothing)

Exercise: Understanding Assignment

- You now have this:

x	~~5~~ ~~7~~ 22	variable	~~3~~ 7	varable	29

- Execute this assignment:

```
varable =  x + variable
```

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

Spelling mistakes in Python are bad!!

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