CS 1110:
Introduction to Computing Using Python

Lecture 2

Variables & Assignment

[Andersen, Gries, Lee, Marschner, Van Loan, White]
Announcements

• We want to understand what lab sections are in demand.
• **NO PROMISES.**
• If you are still unable to get into a lab section:
  - Email up to three preferred sections to:
    - Ms. Jenna Edwards: jls478@cornell.edu
  - Use subject:
    - “CS1110 - cannot register, lab preferences”
    - “CS1110 - registered, lab switch preferences”
  - Deadline: Wed. 3pm
Course Website

• www.cs.cornell.edu/courses/cs1110/2017sp/

• LOOK FOR THE SPRING 2017 BAT!!!
Things to Do Before Next Class

Read Textbook

• Chapter 1 (browse)
• Chapter 2 (in detail)
• Chapter 3.1 – 3.4

Lab 1

• Go to your registered section
• Complete lab handout
• Have *one week* to complete
  ▪ Show to TA by end of lab, or:
  ▪ Show in consulting hours up to the day *before* your lab, or:
  ▪ Show to TA *within first 10 minutes* of next week’s 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

- **Office Hours.** Talk to the professors!
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)
Converting From One Type To Another

• Command: `<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)
  - This kind of conversion is also called “casting”

• This is DIFFERENT from `type(<value>)`
  - `type(<value>)` tells you the type
  - `<type>(<value>)` converts the type
Implicit (Automatic) Conversions

- Python sometimes converts types automatically
  - Example: 1/2.0
  - evaluates to a float: 0.5
  - internally:
    - Step 1: Python casts 1 (an int) to 1.0 (a float)
    - Step 2: Python evaluates 1.0/2.0

- Behavior depends on whether the conversion is narrowing or widening
Variable “width”

• Types differ in **how much information** they hold
• Can convert without losing information?
  - `float` to `int` (e.g. 4.7 to 4) **information lost**
  - `int` to `float` (e.g. 4 to 4.0) **seems ok**
• “Wide” = more information capacity
• From narrow to wide: `bool` ⇒ `int` ⇒ `float`
Widening Conversion

• from a *narrower* type to a *wider* type

• Python does *automatically* if needed:
  - **Example:** `1/2.0` evaluates to a *float*: `0.5`
  - **Example:** `True + 1` evaluates to an *int*: `2`
    - True converts to `1`
    - False converts to `0`

• Note: does not work for *string*
  - **Example:** `2 + “ab”` produces an *error*
Narrowing Conversion

• from a *wider* type to a *narrower* type
  ▪ **Example**: `int(2.6)`
• causes information to be lost
• Python *never* does this automatically

• **Note**: you *can* just always cast
  ▪ Instead of `1/2.0`, can write `float(1)/2.0`
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
Operators and Type Conversions

Evaluate this Expression:

False + 1 + 3.0 / 3

A. 3
B. 3.0
C. 1.3333
D. 2
E. 2.0

Operator Precedence

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

Evaluate this Expression:

- False + 1 + 3.0 / 3
- False + 1 + 1.0
- 1 + 1.0
- 2.0

Operator Precedence

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

- An *assignment statement* takes a *value* and stores it in a *variable*

- **Example**: \( x = 5 \)
Executing Assignment Statements

>>> x = 5

• But something did happen!
• Python *assigned* the *value* 5 to the *variable* x
• Internally (and invisible to you):

memory location \( x \)

stored value 5

Press ENTER and…

Hm, looks like nothing happened…
Retrieving Variables

```python
>>> x = 5
```
Retrieving Variables

```python
>>> x = 5
```

Press ENTER and…

```python
>>> x
5
```

Python tells me the stored value

```python
>>> 
```
In More Detail: Variables (Section 2.1)

• A **variable**
  - is a **named** memory location (**box**)
  - contains 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**)

1e2 is a **float**, but e2 is a variable name

Variable names must start with a letter (or _).

The type belongs to the value, not to the variable.
In More Detail: Statements

>>> x = 5

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)
<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><strong>Examples:</strong></td>
<td><strong>Examples:</strong></td>
</tr>
<tr>
<td>▪ 2.3</td>
<td>▪ x = 5</td>
</tr>
<tr>
<td>▪ (3+5)/4</td>
<td></td>
</tr>
</tbody>
</table>
Variables in Expressions

```python
>>> x = 5
>>> x
5
```

This is an *expression*

So Python *evaluates* it
Variables in Expressions

```python
>>> x = 5
>>> x  # This is an expression
5
>>> x + 5  # So Python evaluates it
10
```
Variables in Expressions

>>> x = 5

This is an expression

>>> x

5

So Python evaluates it

>>> x + 5

10

>>> x ** 2 + x - 1

29

>>>
Assignment Statements with Expressions

```python
>>> x = 5
>>> x = x + 2
```

Python evaluates this expression first... … then assigns the result to the variable
Keeping Track of Variables

• Draw boxes on pieces of paper:
  x 5

• If a new variable is declared, write a new box:
  x 5
  y 5

• If a variable is updated, cross it out:
  x ✗ 7
  y 5
Execute the Statement: \( x = x + 2 \)

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

\[
\begin{array}{c}
\text{x} \\
5
\end{array}
\]
Execute the Statement: \( x = x + 2 \)

- Draw variable \( x \) on piece of paper:
  \[
  \begin{array}{c}
  x \\
  5 \\
  \end{array}
  \]

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

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

  \[
  x \quad 5
  \]

- 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 \( x \)
Execute the Statement: \( x = x + 2 \)

- Draw variable \( x \) on piece of paper:
  \[
  \begin{array}{c}
  \text{x} \\
  \hline
  5
  \end{array}
  \]

- 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.
Which One is Closest to Your Answer?

A: \[ x \times 7 \]

B: \[ x \times 5 \]
\[ x \times 7 \]

C: \[ x \times 7 \]

D: \[ \_(_{-}(ツ))_/-\]
Which One is Closest to Your Answer?

A:

\[ x \times 7 \]

B:

\[ x \quad 5 \]
\[ x \quad 7 \]

C:

\[ x \quad \times \]
\[ x \quad 7 \]

\[ x = x + 2 \]
Execute the Statement: \( x = 3.0 \times x + 1.0 \)

- You have this:

\[
\begin{array}{c}
\text{x} \\
\text{7}
\end{array}
\]
Execute the Statement: $x = 3.0 \times x + 1.0$

• You have this:

\[
\begin{align*}
\text{x} & \hspace{1cm} 7
\end{align*}
\]

• Execute this command:

- Step 1: Evaluate the expression $3.0 \times x + 1.0$
- Step 2: Store its value in x
Execute the Statement: $x = 3.0 \times x + 1.0$

- You have this:
  
  \[
  x \quad \text{7}
  \]

- 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.
Which One is Closest to Your Answer?

A:  
\[ \times \times 22.0 \]

B:  
\[ \times 7 \]
\[ \times 22.0 \]

C:  
\[ \times \times \]
\[ \times 22.0 \]

D:  
\[ (ツ) \]
Which One is Closest to Your Answer?

A:

\[ x \times \times 22.0 \]

✓

B:

\[ x \times 7 \]
\[ x \ 22.0 \]

C:

\[ x \times \times \]
\[ x \ 22.0 \]

\[ x = 3.0 \times x + 1.0 \]
Execute the Statement: \( x = 3.0 \times x + 1.0 \)

- You now have this:
  \[
  x \quad 22.0
  \]

- 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:
  \[ x \times \times 22.0 \quad \text{interestRate} \ 4 \]

- Execute this assignment:
  \[
  \text{interestRate} = x / \text{interestRate}
  \]

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.
Which One is Closest to Your Answer?

A:

x 2x 0 5.5
interestRate 5.5

B:

x 22.0
interestRate
interestRate 5.5

C:

x 22.0
interestRate 5.5

D:

x 22.0
interestRate 5
Which One is Closest to Your Answer?

A:  
\( x \times 22 \times 0 \times 5.5 \)

B:  
\( x \times 22.0 \)

C:  
\( x \times 22.0 \)

interestRate  
\( x \times 5.5 \)

E:  
\( \backslash (ツ)_/ \)

interestRate  
\( x \times 5.5 \)

1/31/17  Variables & Assignments
Which One is Closest to Your Answer?

interestRate = \( \frac{x}{\text{interestRate}} \)

B:

\[
\begin{align*}
x & \neq 22.0 \\
\text{interestRate} & \neq 5.5
\end{align*}
\]

C:

\[
\begin{align*}
x & \neq 22.0 \\
\text{interestRate} & \neq 5.5
\end{align*}
\]

D:

\[
\begin{align*}
x & \neq 22.0 \\
\text{interestRate} & \neq 5
\end{align*}
\]
Exercise: Understanding Assignment

• You now have this:

\[
\text{x} \quad 22.0 \quad \text{interestRate} \quad 5.5
\]

• Execute this assignment:

\[
\text{interestRate} = \text{x} + \text{interestRate}
\]

• Check to see whether you did the same thing as your neighbor, discuss it if you did something different.
Which One is Closest to Your Answer?

A:

\[ x \times 22.0 \]

\[ \text{interestRate } 5.5 \times 27.5 \]

B:

\[ x \times 22.0 \]

\[ \text{interestRate } 5.5 \]

\[ \text{interestRate } 27.5 \]

C:

\[ x \times 20 \times 27.5 \]

\[ \text{interestRate } 5.5 \]

D:

\[ x \times 22.0 \]

\[ \text{interestRate } 5.5 \]

\[ \text{interestRate } 27.5 \]
Which One is Closest to Your Answer?

A:

\[ x \times 22.0 \]

\[ \text{interestRate } \times 5.5 27.5 \]

B:

\[ x \times 22.0 \]

\[ \text{interestRate } \times 5.5 \]

\[ \text{intrestRate } 27.5 \]

\[ \text{intrestRate} = x + \text{interestRate} \]

Spelling mistakes in Python are \textbf{bad}!!
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`
- The following is acceptable in Python:
  ```python
  >>> x = 1
  >>> x = x / 2.0
  ``
  - `x` contains an `int` value
  - `x` now contains a `float` value
- Alternative is a **statically typed language** (e.g. Java)
  - Each variable restricted to values of just one type
More Detail: Testing Types

• Command: `type(<value>)`

• Can test a variable:
  ```
  >>> x = 5
  >>> type(x)
  <type 'int'>
  ```

• Can test a type with a Boolean expression:
  ```
  >>> type(2) == int
  True
  ```