Lecture 2:
Variables & Assignments
(Sections 2.1-2.3, 2.5)

CS 1110
Introduction to Computing Using Python

Orange text indicates updates made after lecture

[E. Andersen, A. Bracy, D. Fan, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Lab 1 announcement

• Weren’t able to attend lab? Don’t panic. Do it on your own via link on course website.

• To get credit in the online lab system you need this info:
  • Lab 1 instructions state that if Python gives an error message, you just write “ERROR”—don’t paste in whole error message
  • For the short-answer in the boolean activity, the term for Python’s behavior is “short-circuit evaluation”
  • Secret passwords for the 3 activities that ask for them:
    1
    4
    5
More announcements

• Course website:  
  http://www.cs.cornell.edu/courses/cs1110/2020sp/  
  Make sure it’s spring 2020—look for the whale-sushi logo . We do not use Canvas.

• We will use clickers/Reef polling, but not for credit. Therefore no need to register your clicker.

• Cornell IT working on posting lecture recording. Thanks for your patience.

• Before next lecture, read Sections 3.1-3.3

• Install Anaconda Python 3.7 and Atom editor according to instructions on course website
Helping you succeed in this class

http://www.cs.cornell.edu/courses/cs1110/2020sp/staff/

Consulting Hours. ACCEL Lab Green Room
• Big block of time, multiple consultants (see staff calendar)
• Good for assignment help

TA Office Hours.
• Staff: 1 TA, 1 or two hours at a time (see staff calendar)
• Good for conceptual help

Prof Office Hours.
• After lecture for an hour in Bailey Hall lower lobby
• Prof. Fan has additional drop-in hours (see staff calendar)
• Prof. Lee has additional hours by appointment (use link on course website, Staff/OH → Office Hours)

Piazza. Online forum to ask/answer questions

AEW (ENGRG 1010). “Academic Excellence Workshops”
• Optional discussion course that runs parallel to this class. See website for more info
From last time: **Types**

Type: set of values & operations on them

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

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

**Type bool:**
- Values: true, false
- Ops: not, and, or

*One more type today:*

**Type str:**
- Values: string literals
  - Double quotes: “abc”
  - Single quotes: ‘abc’
- Ops: + (concatenation)
Type: **str** (string) for **text**

**Values:** any sequence of characters  
**Operation(s):** `+` (catenation, or concatenation)  
**Notice:** meaning of operator `+` changes from type to type

**String literal:** sequence of characters in quotes  
- Double quotes: " abcex3$g<&" or "Hello World!"
- Single quotes: 'Hello World!'

Concatenation applies only to strings  
- "ab" + "cd" evaluates to "abcd"  
- "ab" + 2 produces an error
Converting from one type to another
aka “casting”

```python
>>> float(2)
2.0
>>> int(2.6)
2
>>> type(2)
<class 'int'>
```

converts value 2 to type `float`

converts value 2.6 to type `int`

...different from:

```python
>>> type(2)
<class 'int'>
```

which _tells_ you the type
What should Python do?

```python
>>> 1/2.6
```

A. turn 2.6 into the integer 2, then calculate 1/2 \[ \rightarrow 0.5 \]
B. turn 2.6 into the integer 2, then calculate 1//2 \[ \rightarrow 0 \]
C. turn 1 into the float 1.0, then calculate 1.0/2.6 \[ \rightarrow 0.3846... \]
D. Produce a TypeError telling you it cannot do this.
E. Exit Python
Widening Conversion (OK!)

From a **narrower** type to a **wider** type (e.g., int -> float)

Python does it 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 **str**

- Example: `2 + "ab"` produces a TypeError
Narrowing Conversion (OK???)

From a *wider* type to a *narrower* type
(e.g., float $\rightarrow$ int)

• causes information to be lost
• Python *never* does this automatically

What about:

```python
>>> 1/int(2.6)
```

11
Narrowing Conversion (OK???)

From a **wider** type to a **narrower** type (e.g., `float` → `int`)

• causes information to be lost
• Python **never** does this automatically

What about:
```python
>>> 1/int(2.6)
0.5
```

*Python casts the 2.6 to 2 but `/` is a float division, so Python casts 1 to 1.0 and 2 to 2.0*
Types matter!

You Decide:

• What is the right type for my data?
• When is the right time for conversion (if any)?

• Zip Code as an int?
• Grades as an int?
• Lab Grades as a bool?
• Interest level as bool or float?
What is the difference between:

\[2 \times (1+3)\]  \hspace{1cm} \[2 \times 1 + 3\]

- *add, then multiply*
- *multiply, then add*

Operations performed in a set order

- Parentheses make the order explicit

What if there are no parentheses?

→ **Operator Precedence**: fixed order to process operators when no 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 (except for **)  
  - Example: 1/2*3 is (1/2)*3

- Section 2.5 in your text
- See website for more info
- Part of Lab 1
Operators and Type Conversions

Operator Precedence

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

Evaluate this expression:
False + 1 + 3.0 / 3

A. 3
B. 3.0
C. 1.3333
D. 2
E. 2.0
Operators and Type Conversions

**Operator Precedence**

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

Evaluate this expression:

```
False + 1 + 3.0 / 3
```

```
False + 1 + 1.0 + 1.0
```

```
1 + 1.0
```

```
2.0
```
New Tool: Variable Assignment

An assignment statement:
• takes an expression
• evaluates it, and
• stores the value in a variable

Example:

\[ x = 5 \]

Value on right hand side (RHS) is stored in variable named on left hand side (LHS)
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
```

```
5
```

```
stored value
```

>>> terminal time >>>
Retrieving Variables

>>> x = 5

>>> x

5

Interactive mode tells me the value of x

Press ENTER and…

>>> terminal time >>>
In More Detail: Variables (Section 2.1)

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

• Examples:
  Variable \(x\), with value 5 (of type \texttt{int})
  Variable \(\text{area}\), with value 20.1 (of type \texttt{float})

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

Press ENTER and…

Hm, looks like nothing happened…
### Expressions vs. Statements

<table>
<thead>
<tr>
<th><strong>Expression</strong></th>
<th><strong>Statement</strong></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 = 2 + 1</td>
</tr>
<tr>
<td>- (3+5)/4</td>
<td>- x = 5</td>
</tr>
<tr>
<td>- x == 5</td>
<td></td>
</tr>
</tbody>
</table>

*Look so similar but they are not!*
You can assign more than literals

```python
>>> x = 5
>>> x = 3.0 ** 2 + 4 - 1
>>> x = 2 + x
```

“x gets 5”

“x gets the value of this expression”

“x gets 2 plus the current value of x”

The RHS is an expression. An expression includes literals, operators, and variables.
Keeping Track of Variables

• Draw boxes on paper:
  >>>> x = 9

• New variable declared?
  >>>> y = 3

Write a new box.

• Variable updated?
  >>>> x = 5

Cross out old value. Insert new value.
Start with variable $x$ having value 5. Draw it on paper:

**Task:** Execute the statement $x = x + 2$

1. Evaluate the RHS expression, $x + 2$
   - For $x$, use the value in variable $x$
   - Write the expression somewhere on your paper

2. Store the value of the RHS expression in variable named on LHS, $x$
   - Cross off the old value in the box
   - Write the new value in the box for $x$

Did you do the same thing as your neighbor? If not, discuss.
Which one is closest to your answer?

A. \( x \times \frac{3}{5} \times 7 \)

B. \( x \times 5 \times 7 \)

C. \( x \times 5 \times 7 \times 7 \)

D. \[ x = x + 2 \]
And The Correct Answer Is...

A. \( \times \frac{8}{7} \)  
B. \( \times 5 \), \( \times 7 \)  
C. \( \times 6 \), \( \times 7 \)  
D. \( \backslash(つ)\_/\)  

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

Begin with this:

1. **Evaluate** the expression $3.0 \times x + 1.0$

2. **Store** its value in $x$

Did you do the same thing as your neighbor? If not, *discuss*.
Which one is closest to your answer?

A.  
\[ x \times 7 \geq 22.0 \]

B.  
\[ x \times 7 \times 22.0 \]

C.  
\[ x \times 7 \times 22.0 \]

D.  
\[-(ツ)_/\-\]

\[ x = 3.0 \times x + 1.0 \]
And The Correct Answer Is...

A. $x \neq \frac{7}{22.0}$

B. $x \neq 7$

C. $x \neq 7$

D. \/(ツ)/\

$x = 3.0 \times x + 1.0$
Executing an Assignment Statement

The command: \( x = 3.0*x+1.0 \)

“Executing the command”:

1. **Evaluate** right hand side \( 3.0*x+1.0 \)
2. **Store** the value in the variable \( x \)’s box

- Requires both evaluate AND store steps
- Critical mental model for learning Python
Exercise 1: Understanding Assignment

Have variable `x` already from previous

Declare a new variable:

```python
>>> rate = 4
```

Execute this assignment:

```python
>>> rate = x / rate
```

Did you do the same thing as your neighbor? If not, *discuss*. 

Which one is closest to your answer?

A. \( \times 22.0 \ 5.5 \)

rate \( \neq 5.5 \)

B. \( \times \)

\[
\begin{array}{c}
\text{rate} \\
\text{rate} \\
5.5
\end{array}
\]

C. \( \times 22.0 \)

rate \( \neq 5.5 \)

D. \( \times 22.0 \)

rate \( \neq 5 \)

E. \( \\
rate = x / \text{rate} \)

rate = x / rate
And The Correct Answer Is...

A. \( x \times 22.0 \ 5.5 \)
   rate \( \neq \ 5.5 \) \( \checkmark \)

B. \( x \times 22.0 \)
   rate \( \neq \ 5.5 \)
   rate \( \neq \ 5.5 \)

C. \( x \times 22.0 \)
   rate \( \neq \ 5.5 \)
   \( \checkmark \)

D. \( x \times 22.0 \)
   rate \( \neq \ 5.5 \)

rate = x / rate
Dynamic Typing

Python is a **dynamically typed** language

- Variables can hold values of any type
- Variables can hold different types at different times

The following is acceptable in Python:

```
>>> x = 1  # x contains an int value
>>> x = x / 2.0  # x now contains a float value
```

Alternative: a **statically typed** language

- Examples: Java, C
- Each variable restricted to values of just one type
Exercise 2: Understanding Assignment

Begin with:

\[
x \times \ 22.0 \\
rate \ 5.5
\]

Execute this assignment:

>>> rat = x + rate

Did you do the same thing as your neighbor? If not, discuss.
Which one is closest to your answer?

A. $x \times 22.0 \quad 27.5$

rate 5.5

B. $x \times 22.0$

rate 5.5

rat 27.5

C. $x \times 22.0$

rate ≤ 27.5

E. \_(ツ)_/\__

rat = x + rate
And The Correct Answer Is...

A. $x \times \begin{array}{c} 22.0 \ 27.5 \end{array}$

rate \hspace{1cm} 5.5

B. $\begin{array}{c} 22.0 \ 5.5 \ 27.5 \end{array}$

C. $x \times \begin{array}{c} 22.0 \ 27.5 \end{array}$

rate \hspace{1cm} 5.5

D. $\begin{array}{c} 22.0 \ 6.6 \ 27.5 \end{array}$

rat = x + rate

Spelling Matters!
More Detail: Testing Types

May want to track the type in a variable
Command: `type(<expression>)`

Can get the type of a variable:

```python
>>> x = 5
>>> type(x)
<class 'int'>
```

Can test a type with a Boolean expression:

```python
>>> type(2) == int
True
```