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

CS 1110
Introduction to Computing Using Python
CS 1110, Lecture 2 Announcements

Sections

- Start this week! Yay!
- Please go only to the Section you are enrolled in
- Need to Change your Section or your Lecture?
  See our Section Swapping Station on Piazza:
  https://piazza.com/class/jckqwmqflaz6i?cid=10

Enrollment

- Lots of turnover in the first week. Don’t give up!
- Perhaps another class meets your needs?
Things to do this week

Read textbook
- Chapter 2.1-2.3, 2.5
- Chapter 3.1-3.3

Lab 1:
- Go to your registered section
- Complete lab handout
- You have **one week** to show your work:
  - to TA by end of lab, or:
  - in consulting hours up to 1 day before your lab, or:
  - in TA (not professor) office hours
    (but student questions take precedence over this)
  - 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

**ENGRG 1010: AEW Workshops.** Additional discussion course open to **ALL** students
- Runs parallel to this class – optional
- See website

**Piazza.** Online forum to ask/answer questions
- Go here first before e-mailing questions

**Office Hours.** Talk to the professors!
- Olin 128 between lectures
iClickers

• Fun! Interactive!
• Not for a grade; no need to register.
How do you Plan to use Python?

A. I want to work mainly in the computer labs
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
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: integers
- 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>)`

```python
>>> float(2)
2.0
```
converts value `2` to type `float`

```python
>>> int(2.6)
2
```
converts value `2.6` to type `int`

This kind of conversion is also called “casting”

**Different from type(<value>)**

`type(<value>)` *tells* you the type

`<type>(<value>)` *converts* the type
Implicit (Automatic) Conversions

Python sometimes converts types automatically.

Example: \( \frac{1}{2.0} \)

- evaluates to a \textbf{float}: 0.5
- internally:
  - Step 1: Python casts 1 (an \textbf{int}) to 1.0 (a \textbf{float})
  - Step 2: Python evaluates \( 1.0/2.0 \)

Behavior depends on whether the conversion is \textbf{narrowing} or \textbf{widening}.
Variable “width”

Types differ in how much information held

Convert without losing information?

• **float** → **int** (e.g., 4.7 to 4)  \(\leftarrow\) info lost
• **int** → **float** (e.g., 4 to 4.0)  \(\leftarrow\) seems ok

“Wide” = more information capacity

From narrow to wide: **bool** \(\Rightarrow\) **int** \(\Rightarrow\) **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 **str**

- Example: `2 + “ab”` produces an error
Narrowing Conversion

From a \textbf{wider} type to a \textbf{narrower} type

- Example: \texttt{int(2.6)}
- causes information to be lost
- Python \texttt{never} does this automatically

Note: you can just always cast

- Instead of \texttt{1/2.0}, can write \texttt{float(1)/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?

What are your goals:
Accuracy? Clarity? Fairness?
Operator Precedence

What is the difference between:

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

*add, then multiply* \hspace{2cm} *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 processes 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
  - Example: 1/2*3 is (1/2)*3

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

Evaluate this expression:

\[ \text{False} + 1 + 3.0 / 3 \]

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

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 + 1.0

2.0
New Tool: Variable Assignment

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

Example: \(\text{= x = 5}\) (read right to left)

variable
expression
equals sign
(just one!)
evaluates to 5
Executing Assignment Statements

>>> x = 5

Press ENTER and…

>>> Hmm, looks like nothing happened…

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

  memory location \( x \)

  stored value 5
Retrieving Variables

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

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

Press ENTER and…

Python tells me the stored value
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 `int`)
  - 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)

Press ENTER and…

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Represents</strong> something</td>
<td></td>
</tr>
<tr>
<td>▪ Python <em>evaluates it</em></td>
<td></td>
</tr>
<tr>
<td>▪ End result is a value</td>
<td></td>
</tr>
<tr>
<td><strong>Examples:</strong></td>
<td>▪ Does something</td>
</tr>
<tr>
<td>▪ 2.3</td>
<td>▪ Python <em>executes it</em></td>
</tr>
<tr>
<td>▪ (3+5)/4</td>
<td>▪ Need not result in a value</td>
</tr>
<tr>
<td>▪ x == 5</td>
<td>▪ Examples:</td>
</tr>
<tr>
<td></td>
<td>▪ x = 2 + 1</td>
</tr>
<tr>
<td></td>
<td>▪ x = 5</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”
Keeping Track of Variables

• Draw boxes on paper:
  >>>> x = 5

• New variable declared?
  >>>> y = 3
  Write a new box.

• Variable updated?
  >>>> x = 7
  Cross out old value. Insert new value.
Execute the Statement: \( x = x + 2 \)

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

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

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 \( x \)

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

A. \[ x \times \begin{array}{c} \text{	extcolor{red}{3}} \\ \text{7} \end{array} \]

B. \[ x \times \begin{array}{c} 5 \\ 7 \end{array} \]

C. \[ x \times \begin{array}{c} \text{	extcolor{red}{3}} \\ \text{7} \end{array} \]

D. \[ \_\_\_ \_\_ \( \text{ツ} \) \_\_\_ \]

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

A. \[ x \times 7 \]

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

C. [Image of a question with a red mark]

D. [Image of a question with a red mark]

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

Begin with this:

\[
\begin{array}{c}
x \\ \leq 7
\end{array}
\]

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. \[ x \times 22.0 \]

B. \[ x \times \frac{7}{22.0} \]

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

D. \[ \_\_\_(ツ)_\_/\_ \]

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

\[ x = 3.0 \times x + 1.0 \]
Executing an Assignment Statement

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

“Executing the command”:

1. **Evaluate** right hand side \( 3.0 \times x + 1.0 \)
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

Begin with:

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. \( x \div \text{rate} = 5.5 \)

rate \( = 5.5 \)

B. \( x \div \text{rate} \)

rate \( \div 5.5 \)

C. \( x \div \text{rate} = 22.0 \)

rate \( = 5.5 \)

D. \( x \div \text{rate} = 22.0 \)

rate \( = 5 \)

E. \( \backslash(ツ)_/\) \( \text{rate} = x / \text{rate} \)
And The Correct Answer Is…

A. $x \cdot \frac{5.5}{rate} = 5.5$

B. $x \cdot \frac{22.0}{rate} = 5.5$

C. $x \cdot \frac{22.0}{rate} = 5.5$

D. $x \cdot \frac{22.0}{rate} = 5$

$rate = \frac{x}{rate}$
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:

```python
>>> x = 1
>>> 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
More Detail: Testing Types

Command: `type(<value>)`

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

Can test a type with a Boolean expression:
```python
>>> type(2) == int
True
```
Exercise 2: Understanding Assignment

Begin with:

\[
\begin{align*}
x & \leq 22.0 \\
rate & \geq 5.5
\end{align*}
\]

Execute this assignment:

\[
\texttt{>>> rat = x + rate}
\]

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

A. \[ x \leq 22.0 \leq 27.5 \]
   rate \[ \leq 5.5 \]

B. \[ x \leq 22.0 \]
   rate \[ \leq 5.5 \]
   rat \[ 27.5 \]

C. \[ x \leq 22.0 \]
   rate \[ \leq 27.5 \]

D. \[ x \leq 22.0 \]
   rate \[ \leq 5.5 \]
   rat \[ 27.5 \]

E. \( \_-(ツ)_/\)  

rat = x + rate
And The Correct Answer Is…

**Spelling Matters!**

\[ \text{rat} = x + \text{rate} \]