Introduction to Python and programming

Ruth Anderson
UW CSE 160
Spring 2015
1. Python is a calculator

2. A variable is a container

3. Different types cannot be compared

4. A program is a recipe
0. Don’t panic!

• CSE 160 is for beginners to programming
  – (If you know how to program, you don’t belong)
• You can learn to program in 10 weeks
  – You will work hard
  – We will work hard to help you
• Ask questions!
  – This is the best way to learn
1. Python is a calculator
You type *expressions*. Python computes their *values*.

- 5
- 3+4
- 44/2
- 2**3
- 3*4+5*6
  - If precedence is unclear, use parentheses
- (72 – 32) / 9 * 5
An expression is evaluated from the inside out

• How many expressions are in this Python code?

\[(72 - 32) / 9.0 \times 5\]

\[40 / 9.0 \times 5\]

\[4.44 \times 5\]

\[22.2\]
Another evaluation example

\[(72 - 32) / (9.0 \times 5)\]
\[(40) / (9.0 \times 5)\]
\[40 / (9.0 \times 5)\]
\[40 / (45.0)\]
\[40 / 45.0\]
\[.888\]
2. A variable is a container
Variables hold values

• Recall variables from algebra:
  – Let \( x = 2 \) …
  – Let \( y = x \) …

• In Python assign a variable: “\( \text{varname} = \text{expression} \)”
  \[
  \text{pi} = 3.14 \\
  \text{pi} \\
  \text{avogadro} = 6\times10^{23} \\
  \text{avogadro} \\
  22 = x \quad \# \text{Error!}
  \]

• Not all variable names are permitted
Changing existing variables ("re-binding" or "re-assigning")

\[
\begin{align*}
x &= 2 \\
x &= 5 \\
y &= 2 \\
y &= 5 \\
x &= 2 \\
y &= 5 \\
\end{align*}
\]

- "=" in an assignment is not a promise of eternal equality
  - This is different than the mathematical meaning of "="

- Evaluating an expression gives a new (copy of a) number, rather than changing an existing one
How an assignment is executed

1. Evaluate the right-hand side to a value
2. Store that value in the variable

```python
x = 2
print x
y = x
print y
z = x + 1
print z
```

State of the computer:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x: 2</td>
<td>y: 2</td>
<td>z: 3</td>
</tr>
</tbody>
</table>

Printed output:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

To visualize a program’s execution:

http://pythontutor.com
More expressions: Conditionals
(value is True or False)

22 > 4
22 < 4
22 == 4
x = 100  # Assignment, not conditional!
22 = 4  # Error!
x >= 5
x >= 100
x >= 200
not True
not (x >= 200)
3<4 and 5<6
4<3 or 5<6
temp = 72
water_is_liquid = temp > 32 and temp < 212

Numeric operators: +, *, **
Boolean operators: not, and, or
Mixed operators: <, >=, ==
More expressions: strings

A string represents text

'vePython'
myclass = "CSE 160"
"

Empty string is not the same as an unbound variable

Operations:
• Length:
  len(miclass)
• Concatenation:
  "Ruth" + 'Anderson'
• Containment/searching:
  '0' in myclass
  "O" in myclass
3. Different types cannot be compared
Types of values

- Integers (**int**): −22, 0, 44
  - Arithmetic is **exact**
  - Some funny representations: 12345678901L
- Real numbers (**float**, for “floating point”): 2.718, 3.1415
  - Arithmetic is **approximate**, e.g., 6.022*10**23
  - Some funny representations: 6.022e+23
- Strings (**str**): "I love Python", ""
- Truth values (**bool**, for “Boolean”): True, False
Operations behave differently on different types

3.0 + 4.0
3 + 4
3 + 4.0
"3" + "4"
3 + "4"      # Error
3 + True     # Insanity! (Don’t do this.)

Moral: Python sometimes tells you when you do something that does not make sense.
Operations behave differently on different types

\[ 15.0 / 4.0 \]
\[ 15 / 4 \]
\[ 15.0 / 4 \]
\[ 15 / 4.0 \]

# Insanity!

Type conversion:

- `float(15)`
- `int(15.0)`
- `int(15.5)`
- `int("15")`
- `str(15.5)`
- `float(15) / 4`
4. A program is a recipe

**Colvin Run Mill Corn Bread**

1 cup cornmeal
1 cup flour
½ teaspoon salt
4 teaspoons baking powder
3 tablespoons sugar
1 egg
1 cup milk
¼ cup shortening (soft) or vegetable oil

Mix together the dry ingredients. Beat together the egg, milk and shortening/oil. Add the liquids to the dry ingredients. Mix quickly by hand. Pour into greased 8x8 or 9x9 baking pan. Bake at 425 degrees for 20-25 minutes.
What is a program?

- A program is a sequence of instructions.
- The computer executes one after the other, as if they had been typed to the interpreter.
- Saving your work as a program is better than re-typing from scratch.

```python
x = 1
y = 2
x + y
print x + y
print "The sum of", x, "and", y, "is", x+y
```
Interlude: The `print` statement

- The `print` statement always prints one line
  - The next print statement prints below that one
- Write 0 or more expressions after `print`, separated by commas
  - In the output, the values are separated by spaces
- Examples:
  
  ```
  print 3.1415
  print 2.718, 1.618
  print
  print 20 + 2, 7 * 3, 4 * 5
  print "The sum of", x, "and", y, "is", x+y
  ```
Exercise: Convert temperatures

• Make a temperature conversion chart: Fahrenheit to Centigrade, for -40, 0, 32, 68, 98.6, 212, 293, 451
  
  Output:
  
  -40  -40.0
  0    -17.7778
  32   0.0
  68   20.0
  98.6 37.0
  212  100.0
  293  145.0
  451  232.778

• You have created a Python program!
• (It doesn’t have to be this tedious, and it won’t be.)
Expressions, statements, and programs

- An **expression** evaluates to a value
  
  ```
  3 + 4
  pi * r**2
  ```

- A **statement** causes an effect
  
  ```
  pi = 3.14159
  print pi
  ```

- Expressions appear within other expressions and within statements
  
  ```
  (fahr - 32) * (5.0 / 9)
  print pi * r**2
  ```

- A statement may **not** appear within an expression
  
  ```
  3 + print pi  # Error!
  ```

- A **program** is made up of statements
  
  - A program should do something or communicate information
  - Just evaluating an expression does not accomplish either goal
1. Python is a calculator

2. A variable is a container

3. Different types cannot be compared

4. A program is a recipe