expressions, variables, for loops

Special thanks to Scott Shawcroft, Ryan Tucker, Paul Beck, Hélène Martin, Kim Todd, John Kurkowski, and Marty Stepp for their work on these slides.

Except where otherwise noted, this work is licensed under:
http://creativecommons.org/licenses/by-nc-sa/3.0
Who uses Python?

"Python is fast enough for our site and allows us to produce maintainable features in record times, with a minimum of developers"

-Cuong Do, Software Architect, YouTube.com
Expressions

- Arithmetic is very similar to Java
  - Operators: + - * / % (plus ** for exponentiation)
  - Precedence: () before ** before * / % before + -
  - Integers vs. real numbers

```python
>>> 1 + 1
2
>>> 1 + 3 * 4 - 2
11
>>> 7 / 2
3
>>> 7.0 / 2
3.5
```
**Variables**

- **Declaring**
  - no type is written; same syntax as assignment

- **Operators**
  - no `++` or `--` operators (must manually adjust by 1)

<table>
<thead>
<tr>
<th>Java</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int x = 2;</code></td>
<td><code>x = 2</code></td>
</tr>
<tr>
<td><code>x++;</code></td>
<td><code>x = x + 1</code></td>
</tr>
<tr>
<td><code>System.out.println(x);</code></td>
<td><code>print x</code></td>
</tr>
<tr>
<td><code>x = x * 8;</code></td>
<td><code>x = x * 8</code></td>
</tr>
<tr>
<td><code>System.out.println(x);</code></td>
<td><code>print x</code></td>
</tr>
<tr>
<td><strong>double d = 3.2;</strong></td>
<td><strong>d = 3.2</strong></td>
</tr>
<tr>
<td><code>d = d / 2;</code></td>
<td><code>d = d / 2</code></td>
</tr>
<tr>
<td><code>System.out.println(d);</code></td>
<td><code>print d</code></td>
</tr>
</tbody>
</table>
Types

- Python is looser about types than Java
  - Variables' types do not need to be declared
  - Variables can change types as a program is running

<table>
<thead>
<tr>
<th>Value</th>
<th>Java type</th>
<th>Python type</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>3.14</td>
<td>double</td>
<td>float</td>
</tr>
<tr>
<td>&quot;ni!&quot;</td>
<td>String</td>
<td>str</td>
</tr>
</tbody>
</table>
String Multiplication

- Python strings can be multiplied by an integer.
  - The result is many copies of the string concatenated together.

```python
>>> "hello" * 3
"hellohellohello"

>>> print 10 * "yo 
yo yo yo yo yo yo yo yo yo

>>> print 2 * 3 * "4"
444444
```
String Concatenation

• Integers and strings cannot be concatenated in Python.
  – Workarounds:

  \texttt{\textbf{str}(value)} \quad - \text{converts a value into a string}

  \texttt{print value, value} \quad - \text{prints value twice, separated by a space}

>>> x = 4
>>> print "Thou shalt not count to " + x + "."
\texttt{TypeError: cannot concatenate 'str' and 'int' objects}

>>> print "Thou shalt not count to " + \texttt{str(x)} + "."
Thou shalt not count to 4.

>>> print x + 1, "is out of the question."
5 is out of the question.
for **name** in range(**max**):
    **statements**

- Repeats for values 0 (inclusive) to **max** (exclusive)
for Loop Variations

for name in range(min, max):
    statements

for name in range(min, max, step):
    statements

– Can specify a minimum other than 0, and a step other than 1

```python
>>> for i in range(2, 6):
    ...     print i
2
3
4
5
>>> for i in range(15, 0, -5):
    ...     print i
15
10
5
```
Nested Loops

- Nested loops are often replaced by string * and +

Java

```java
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (5 - line); j++) {
        System.out.print(".");
    }
    System.out.println(line);
}
```

Python

```python
for line in range(1, 6):
    print((5 - line) * "." + str(line))
```
Constants

- Python doesn't really have constants.
  - Instead, declare a variable at the top of your code.
  - All methods will be able to use this "constant" value.

```python
constant.py

MAX_VALUE = 3

def printTop():
    for i in range(MAX_VALUE):
        for j in range(i):
            print j
        print

def printBottom():
    for i in range(MAX_VALUE, 0, -1):
        for j in range(i, 0, -1):
            print MAX_VALUE
        print
```
Exercise

• Rewrite the Mirror lecture program in Python. Its output:

```python
#==============#
|  <><>       |
| <>      .. <>|
| <>     ......<>
| <>   ..........<>
| <> .         . <>|
| <> .         . <>|
| <> .         . <>|
| <>     ......<>
| <>      .. <>|
|  <><>       |
#==============#
```

– Make the mirror resizable by using a "constant."
SIZE = 4

```python
def bar():
    print "#" + 4 * SIZE * "=" + "#"

def top():
    for line in range(1, SIZE + 1):
        print "|" + (-2 * line + 2 * SIZE) * " " + "<>
        print "|" + (-2 * line + 2 * SIZE) * " " + "" + 
        print "|" + (-2 * line + 2 * SIZE) * " " + "|"

    for line in range(SIZE, 0, -1):
        print "|" + (-2 * line + 2 * SIZE) * " " + "<>
        print "|" + (-2 * line + 2 * SIZE) * " " + "|"

# main
bar()
top()
bottom()
bar()
```
Concatenating Ranges

- Ranges can be concatenated with +
  - Can be used to loop over a disjoint range of numbers

```python
>>> range(1, 5) + range(10, 15)
[1, 2, 3, 4, 10, 11, 12, 13, 14]

>>> for i in range(4) + range(10, 7, -1):
...     print i
0
1
2
3
10
9
8
```
SIZE = 4

def bar():
    print "#" + 4 * SIZE * "=" + "#"

def mirror():
    for line in range(1, SIZE + 1) + range(SIZE, 0, -1):
        print "|" + (-2 * line + 2 * SIZE) * " " + ">
        print "<>" + (4 * line - 4) * "." + "<>" + ">
    (-2 * line + 2 * SIZE) * " " + "|"

# main
bar()
mirror()
bar()