

### Week 2

#### 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: <u>http://creativecommons.org/licenses/by-nc-sa/3.0</u>

# 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 (doubles)
  - You may use // for integer division

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



# Variables

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

– no ++ or -- operators (must manually adjust by 1)

Java	Python
<pre>int x = 2; x++; System.out.println(x);</pre>	$\begin{array}{l} x = 2 \\ x = x + 1 \\ print(x) \end{array}$
<pre>x = x * 8; System.out.println(x);</pre>	x = x * 8 print(x)
<pre>double d = 3.2; d = d / 2; System.out.println(d);</pre>	d = 3.2 d = d / 2 print(d)
тм	





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

Value	Java type	Python type
42	int	int
3.14	double	float
"ni!"	String	str

You can find the type of anything using the type() function

```
>>> type(3.3)
<class 'float'>
```



# String Multiplication

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

```
>>> "hello" * 3
"hellohellohello"
>>> print(10 * "yo ")
yo yo
>>> print(2 * 3 * "4")
444444
```



# String Concatenation

- Integers and strings cannot be concatenated in Python.
  - Workarounds:
  - str (value) converts a value into a string
  - print (value, value) prints values, separated by a space

```
>>> x = 4
>>> print("Thou shalt not count to " + x + ".")
TypeError: cannot concatenate 'str' and 'int' objects
>>> print("Thou shalt not count to " + str(x) + ".")
Thou shalt not count to 4.
>>> print(x + 1, "is out of the question.")
5 is out of the question.
```



# **Special Print Options**

 You may define the behavior of the print function as follows: end: string appended after the last value, default \n sep: string inserted between values, default a space

```
>>> print("One", "two", "five", sep="... ")
One... two... five
>>> print("O", "M", "G", sep="... ", end="!!!1")
O... M... G!!!1
```



# The for Loop

- for name in range(max):
- statements
- Repeats for values 0 (inclusive) to max (exclusive)

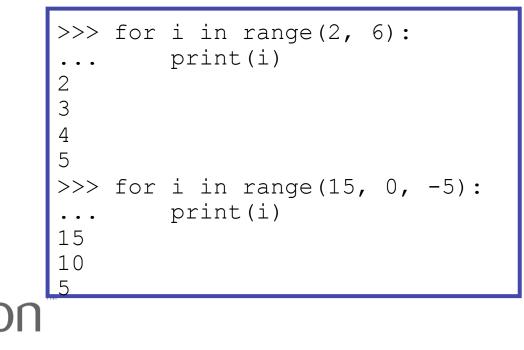


# for Loop Variations

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

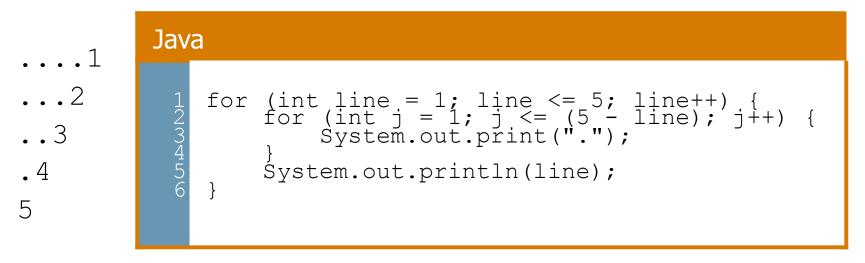
Ż

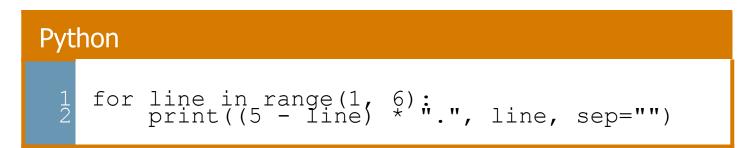
- Can specify a minimum other than 0, and a step other than 1



### **Nested Loops**

• Nested loops are often replaced by string \* and +







#### Constants

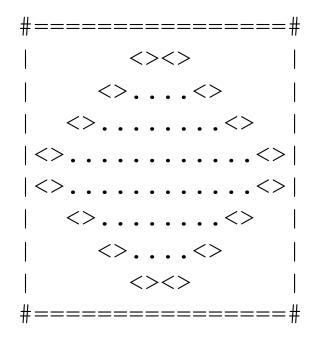
- Python doesn't really have constants.
  - Instead, declare a variable at the top of your code.
  - All methods will be able to use this "constant" value.

#### constant.py

```
MAX_VALUE = 3
def print top():
    for j in range(MAX_VALUE):
        for j in range(i):
            print(j)
    def print bottom():
        for j in range(MAX_VALUE, 0, -1):
            for j in range(i, 0, -1):
                print()
```

#### Exercise

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



- Make the mirror resizable by using a "constant."



```
def bar():
    print("#", 16 * "=", "#")
def draw top half():
    for line in range (1, 5):
        print("|", end=" ")
        print(" " * (-2 * line + 8), end="<>")
        print("." * ( 4 * line - 4), end="<>")
        print(" " * (-2 * line + 8), end=" ")
        print("|", end="n")
def draw bottom half():
    for line in range (4, 0, -1):
        print("|", end=" ")
        print(" " * (-2 * line + 8), end="<>")
        print("." * ( 4 * line - 4), end="<>")
        print(" " * (-2 * line + 8), end=" ")
        print("|", end="\n")
bar()
draw top half()
draw bottom half()
bar()
```



#### **Exercise Solution**

```
# constant
SIZE = 4
def bar():
    print("#", 4 * SIZE * "=", "#")
def draw top half():
    for line in range (1, SIZE + 1):
        print("|", end=" ")
        print(" " * (-2 * line + 2 * SIZE), end="<>")
        print("." * ( 4 * line - 4), end="<>")
        print(" " * (-2 * line + 8), end=" ")
        print("|", end="n")
def draw bottom half():
    for line in range (SIZE, 0, -1):
        print("|", end=" ")
        print(" " * (-2 * line + 2 * SIZE), end="<>")
        print("." * ( 4 * line - 4), end="<>")
        print(" " * (-2 * line + 2 * SIZE), end=" ")
        print("|", end="\n")
bar()
draw top half()
draw bottom half()
bar()
```

# **Concatenating Ranges**

- Ranges can be concatenated with +
  - However, you must use the "list()" command
  - Can be used to loop over a disjoint range of numbers



### **Exercise Solution 2**

```
SIZE = 4
```

```
def bar():
    print("#", 4 * SIZE * "=", "#")
def mirror():
    for line in list(range(1, SIZE + 1)) + list(range(SIZE, 0, -1)):
        print(" |", end=" ")
        print(" " * (-2 * line + 2 * SIZE), end="<>")
        print(" " * (-2 * line + 4), end="<>")
        print(" " * (-2 * line + 4), end="<>")
        print(" " * (-2 * line + 8), end=" ")
        print(" |", end="\n")
# main
bar()
mirror()
bar()
```

