Unit 2

Expressions and variables; `for` loops

Special thanks to Roy McElmurry, John Kurkowski, Scott Shawcroft, Ryan Tucker, Paul Beck for their work.

Expressions

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

```
>>> 1 + 1
2
>>> 1 + 3 * 4 - 2
11
>>> 7 / 2
3
>>> 7.0 / 2
3.5
>>> 10 ** 6
1000000
```

Variables

- Declaring
  - no type is written; same syntax as assignment
- Operators
  - no `++` or `--` operators (must manually adjust by 1)

Java	Python

`int x = 2;`
`x++;`
`System.out.println(x);`
	`x = 2`
	`x = x + 1`
	`print(x)`

`x = x * 8;`
`System.out.println(x);`
	`x = x * 8`
	`print(x)`

`double d = 3.2;`
`d = d / 2;`
`System.out.println(d);`
	`d = 3.2`
	`d = d / 2`
	`print(d)`

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

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

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 yo yo yo yo yo 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(`**expr**`, `**expr**`)` - prints two items on the same line

```
>>> 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.
```

The `for` Loop

```
for name in range(max):
    statements
```

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

```
>>> for i in range(5):
...     print(i)
0
1
2
3
4
```

`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

```
>>> 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 `+`

```
....1
...2
..3
.4
5
```

Java

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

Python

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

Exercise

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

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

Exercise Solution

```
def bar():
    print "#" + 16 * "=" + "#"

def top():
    for line in range(1, 5):
        # split a long line by ending it with \
        print "|" + (-2 * line + 8) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 8) * " " + "|"

def bottom():
    for line in range(4, 0, -1):
        print "|" + (-2 * line + 8) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 8) * " " + "|"

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

Concatenating Ranges

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

```
>>> 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
```

Exercise Solution 2

```
def bar():
    print "#" + 16 * "=" + "#"

def mirror():
    for line in range(1, 5) + range(4, 0, -1):
        print "|" + (-2 * line + 8) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 8) * " " + "|"

# main
bar()
mirror()
bar()
```

Constants

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

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 Solution 3

```
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) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 2 * SIZE) * " " + "|"

# main
bar()
mirror()
bar()
```