CS 130R: Programming in Python

# 2: Types & Variables

Reading: Chapter 2
A few notes about CS130R

- Final exam: Dec. 17, 3:00pm to 5:30pm
- Pass: more than F
- Lab session begins from next Friday (Sept. 12)
- Textbook:
Outline

• Variables

• Standard Data Types
  - Number
  - String

• Simple I/O
Variables

• Let’s see the Greeter Program

```python
# Greeter
# Demonstrates the use of a variable

name = "Michael"

print name

print "Welcome to CA13OR"," , name

Running result:

>>> Michael
Welcome to CA13OR, Michael
>>> |
```
Creating Variables

- **Variables** are nothing but reserved memory locations to store values.
- Variables provide a way to label and access information.
- Before using a variable, create it as follows:
  ```python
  name = "Michael"   # assignment statement
  ```
- It creates a variable called `name` and assigns it a value "Michael".
- Actually, Python doesn’t directly store `name`;
  `name` is only a refer to "Michael".
Using Variables

• Once a variable has been created, it **refers to some value**
• Variables can be used just like the value to which it refers

```python
name = "Michael"
print name
print "Welcome to CA130R, ", name
```
Naming Variables

• **Rules** to create legal variable names
  1. A variable name can **contain only numbers, letters, and underscores**
     
         1-9, a-z, A-Z, _
     
     e.g. highScore, high_score, highScore100

  2. A variable name **can’t start with a number**

     crossed: 100highScore
Name Variables

• Guidelines to follow:
  1. Choose descriptive names
  2. Be consistent
  3. Follow traditions of the language
     Start with a lower case letter
     Avoid starting with an underscore
     (Beginning with an underscore have special meaning in Python)
  4. Keep the length in track
     e.g. personal_checking_account_balance
     (Hard to read, more chance of a typo)
     Keep variable names under 15 characters
Standard data types

#!/usr/bin/python

counter = 100          # An integer assignment

miles = 1287.95        # A floating point
name = "Michael"      # A string

print counter
print miles
print name
Standard data types

• Numbers
• String
• List
• Tuple
• Dictionary
Standard data types

- Numbers
- String
  - List
  - Tuple
  - Dictionary
Numbers

• Integers

    counter = 100  # An integer assignment

• Floating-point numbers

    miles = 1287.95  # A floating point
Using Mathematical Operators
For Python 3.0 and more advanced versions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Evaluates to</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>7 + 3</td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>7 – 3</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>7 * 3</td>
<td>21</td>
</tr>
<tr>
<td>/</td>
<td>Division (True)</td>
<td>7 / 3</td>
<td>2.3333333333333335</td>
</tr>
<tr>
<td>//</td>
<td>Division(Integer)</td>
<td>7 // 3</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>7 % 3</td>
<td>1</td>
</tr>
</tbody>
</table>

1. In Python 3.*, “//” represents integer division
2. While true division is pretty accurate, it’s not exact. Results are fine for most purposes, but you should be aware of this when using floats.
Using Mathematical Operators

For Python 2.* versions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Evaluates to</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>7 + 3</td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>7 – 3</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>7 * 3</td>
<td>21</td>
</tr>
<tr>
<td>/ or //</td>
<td>Division (Integer)</td>
<td>7 / 3</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>7 % 3</td>
<td>1</td>
</tr>
</tbody>
</table>

Integers

Our class programming follows Python 2.*, and the textbook follows Python 3.*.
Using Mathematical Operators

For Python 2.* versions

Floating-point numbers

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Evaluates to</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>7.0 + 3.0</td>
<td>10.0</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>7.0 – 3.0</td>
<td>4.0</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>7.0* 3.0</td>
<td>21.0</td>
</tr>
<tr>
<td>/ or //</td>
<td>Division (Integer)</td>
<td>7.0 / 3.0</td>
<td>2.3333333333333335</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>7.0 % 3.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

e.g. If you want to know the floating-point result of "7 / 3", you can write (7*1.0)/3 or 7.0/3
String

• Strings in Python are identified as a contiguous set of characters in between quotation marks
• Python allows for either pairs of single or double quotes
• Two basic operators for strings: “*”, “+”

#!/usr/bin/python

# strings
# Demonstrates the use of a string
# Python allows for either pairs of single or double quotes

str = "Hello, Python! "  # Double quotation mark
strl = 'Hello, Python! '  # Single quotation mark

print str * 3  # Repeating strings three times
print strl + 'I love you!'  # Concatenating strings
## String methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper()</td>
<td>Returns the uppercase version of the string.</td>
</tr>
<tr>
<td>lower()</td>
<td>Returns the lowercase version of the string.</td>
</tr>
<tr>
<td>swapcase()</td>
<td>Returns a new string where the case of each letter is switched. Uppercase becomes lowercase and lowercase becomes uppercase.</td>
</tr>
<tr>
<td>capitalize()</td>
<td>Returns a new string where the first letter is capitalized and the rest are lowercase.</td>
</tr>
<tr>
<td>title()</td>
<td>Returns a new string where the first letter of each word is capitalized and all others are lowercase.</td>
</tr>
<tr>
<td>strip()</td>
<td>Returns a string where all the white space (tabs, spaces, and newlines) at the beginning and end is removed.</td>
</tr>
<tr>
<td>replace(old, new [,max])</td>
<td>Returns a new string where occurrences of the string old are replaced with the string new. The optional max limits the number of replacements.</td>
</tr>
</tbody>
</table>
String methods

quote = "I think there is a world market for maybe five computers"

print "Original quote:
print quote

print "\n In lowercase:
print quote.lower()

print "\n As a title:
print quote.title()

print "\nWith a minor replacement"
print quote.replace("five", "millions of")

print "\nOriginal quote is still:
print quote

print "\nThe swapcase of quote is:
print quote.swapcase()
String methods

quotel = quote.title()

print quotel.capitalize()

quote2 = "I think there is a world market for maybe five computers"

print "\nOriginal quote2 is:"
print quote2

print "\nRemoving all the white space at beginning and end of quote2"
print quote2.strip()
Simple I/O

• Use quotes inside strings
• Printing multiple values
• Using escaping sequences with strings
Use quotes inside strings

• If you use double quotes to bookend your string, you can use as many single quotes inside the string as you want.

• If you use single quotes to bookend your string, you can use as many double quotes inside the string as you want.

print "Hello, 'Python'!"

print 'Hello, "Python"!'
Printing multiple values

• You can print multiple values with a single call to the “print” function

```python
print "Same", "message", "as before"
print "Same",\ 
    "message",\ 
    "as before"
```
Using escaping sequences with strings

• Escape sequences allow you to put special characters into your strings.
• Made up of two characters:
  a backslash + another character

  e.g. \n, \', \", \t
Using escaping sequences with strings

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\</code></td>
<td>Backslash. Prints one backslash.</td>
</tr>
<tr>
<td><code>'</code></td>
<td>Single quote. Prints a single quote.</td>
</tr>
<tr>
<td><code>&quot;</code></td>
<td>Double quote. Prints a double quote.</td>
</tr>
<tr>
<td><code>\a</code></td>
<td>Bell. Sounds the system bell.</td>
</tr>
<tr>
<td><code>\b</code></td>
<td>Backspace. Moves cursor back one space.</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>Newline. Moves cursor to beginning of next line.</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>Horizontal tab. Moves cursor forward one tab stop.</td>
</tr>
</tbody>
</table>