Day 2: Basic Syntax


Chapter 6: The Dynamic Typing Interlude
Chapter 10: Introducing Python Statements
Chapter 11: Assignment, Expressions, and Prints
Chapter 12: if Tests and Syntax Rules
Chapter 13: `while` and `for` Loops
Turn In Homework
Housekeeping

• If you have not enrolled:
  – Please consider enrolling or auditing
  – You may attend regardless
  – I cannot provide help (homework, office hours, …)
  – I can add you to the mailing list (email me)
Office Hours

Mondays, 3–4 p.m.
Thursdays, 3–4 p.m.

Computer Sciences 4265

*Other times are OK!*
Always best to email first
Write code.
At least a little.
Every day.
Play around!
Basic Python Syntax (cont’d)
From Interactive To Scripts

• Numbers/strings and operations form *expressions*
• Python computes the *value* of an expression

\[ \text{'Answer: ' + str(6 * 7)} \]
\[ \downarrow \]
\[ \text{'Answer: 42'} \]

• Interactive Python displays values automatically
• Scripted Python does not
print

print expression

- Prints a value (*to standard output*)
- Separate items with comma (prints space between)
- Prints newline at end
- Suppress newline with trailing comma

print 5
print 'Result:', 5.0 / 2.5, 'm/s'
print 'Result of complex calculation:',
print round(532.2 * (4.2 + 1.2), 1)
#!/usr/bin/python

# This line is a comment
print 'Hello, world!'

# Continued line (put nothing after \)
print '2pi = ' + \
    str(2.0 * 3.14159)  # 2 * pi
Variables
Variables

variable_name = value
other_variable = variable_name + 1

- Create name by assigning a value
- Must create name before using it
- Subsequent assignment changes value
- When using name, Python substitutes current value

my_bucket = 0
my_bucket = my_bucket + 1
my_bucket = 'Tim'
greeting = 'Hello, ' + my_bucket
Values? Variables?
Python Object Model

- All variables refer to objects
- Objects = Data \textit{(in memory)} + Operations
- Assignment binds a variable name to an object
- Types live in objects, \textbf{not} variables
- No more references? Python can remove object

\begin{tabular}{|c|c|c|c|}
\hline
\textbf{x} = 42 & \textbf{y} = \textbf{x} & \textbf{x} = 'hi' & \textbf{y} = \textbf{x} \\
\hline
\textbf{x} & \textbf{x} & \textbf{y} & \textbf{y} \\
\hline
\textbf{int} & \textbf{int} & \textbf{str} & \textbf{int} \\
\hline
42 & 42 & \textbf{hi} & 42 \\
\hline
\end{tabular}
Types

`type(object)`

- Python can tell you type of object
- Example of *introspection*

```
 type(42)        =>  <type 'int'>
type(3.141)     =>  <type 'float'>
type('hi')      =>  <type 'str'>

x = 5.0 / 2
 type(x)         =>  <type 'float'>
type(type(42))  =>  <type 'type'>
```
None

- Special object which means “no value”
- In other languages: undef, nil, null, …
- In Python, still an object…

```python
>>> x = None
>>> x
None
>>> print x
None
>>> type(x)
<type 'NoneType'>
```
Built-In Help I

`dir(object or type)`

- Lists all operations for that object or type
- For now, ignore everything that starts with __
- Use as `object.operation(...)`

```python
>>> dir(str)
[...,'capitalize','center','count','decode','encode','endswith','expandtabs','find','index','isalnum','isalpha','isdigit','islower','isspace','istitle','isupper','join','ljust','lower','lstrip','replace','rfind','rindex','rjust','rsplit','rstrip','split','splitlines','startswith','strip','swapcase','title','translate','upper','zfill']
```
Built-In Help II

```
help(something)
```

- Shows built-in documentation
- Works on objects, types, and their operations

```python
>>> help(str.lower)
...
lower(...)
    S.lower() -> string

Return a copy of the string S converted to lowercase.
```
Back to Variables
Assignment

\begin{align*}
    a &= 42 \\
    a &= a + 1 \quad \# \quad a &= 43 \\
    a &= a - 3 \quad \# \quad a &= 40 \\
    a &= a \times 2 \quad \# \quad a &= 80 \\
    a &= a / 8 \quad \# \quad a &= 10 \\
    \ldots
\end{align*}

- \texttt{a += 1} is slightly more efficient than \texttt{a = a + 1}
- \texttt{+=} and \texttt{*=} work on strings, too

\begin{align*}
    a &= b = c = 0
\end{align*}

- OK but not recommended
Basic Input

- Gets input from user
- Strips trailing newline automatically
- Result is always a string object — convert if needed

```python
a = raw_input()
a = raw_input('Enter a number: ')

name = raw_input('Enter your name: ')
print 'Hello, %s!' % name

age = raw_input('Enter your age: ')
print 'Age next year: %d' % (int(age) + 1)
```
Comparisons
Booleans

| True | False |

```python
cartwright@university:~> x = True
cartwright@university:~> x
True
cartwright@university:~> print x
True
cartwright@university:~> print False
False
cartwright@university:~> type(False)
<type 'bool'>
```
## Boolean Operations

<table>
<thead>
<tr>
<th></th>
<th>not x</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x or y</th>
<th>x and y</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
**Comparison Operators**

- same values \(==\)
- not same values \(!=\)
- same object \(is\)
- not same object \(is\ not\)
- less than \(<\)
- less than or equal to \(<=\)
- greater than \(>\)
- greater than or equal to \(>=\)

- All comparison operations yield a Boolean value
- Use **is/is not** with **None**, **True**, and **False**
- Can chain inequalities: \(1 < x <= 4\)
Conditionals

```python
if condition:
    # do when condition is True
elif other-condition:
    # do when condition is False
    # and other-condition is True
else:
    # do when all conditions are False
```

```python
name = raw_input('Name? ')  
if name == 'Tim Cartwright':
    print 'Instructor'
else:
    print 'Student'
student_count += 1
```
Indentation

• Blocks of code must be indented consistently
• *Strongly* suggest using 4 spaces…
  (see PEP 8: http://www.python.org/dev/peps/pep-0008/)

```python
user_input = raw_input('Number: ')
user_num = int(user_input)

if user_num > 0:
    print 'Non-negative'
    if user_num > 999:
        print 'But too large'
    else:
        print 'Just right'
else:
    print 'Negative'
```
Basic Loop

```python
while condition:
    # do when condition is True
    # then return to top and re-evaluate
    if condition-a:
        continue  # return to top now
    if condition-b:
        break    # exits loop
    # more stuff

count = 0
while count < 10:
    print count
    count += 1
```
You Made It!
Other Scripting Languages

• The cellphone metaphor…

• Check for different or additional:
  – **Literals** ("/", true/false, null/nil/undef, 1_234)
  – **Operators** (===, =~)
  – **Conditionals** (elsif vs. elseif vs. else if; unless)
  – **Loops** (do … while, unless, foreach)
  – **Block syntax** ({…} vs. do…end vs. indentation)
  – **Object syntax** (Perl…)
Homework

• Simple number-guessing game
  – *You* pick the number & the *computer* guesses
  – Seek a straightforward solution

• **BE SURE TO LABEL YOUR PRINTOUT!!!**

```python
#!/usr/bin/python

"""Homework for CS 368-4 (2012 Fall)
Assigned on Day 02, 2012-10-25
Written by <Your Name>
"""
```