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.
Fridays, 11 a.m. – noon

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

```python
'Answer: ' + str(6 * 7)
```

'Answer: 42'

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

print \texttt{expression}

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

\begin{verbatim}
print 5
print 'Result:', 5.0 / 2.5, 'm/s'
print 'Result of complex calculation:',
print round(532.2 * (4.2 + 1.2), 1)
\end{verbatim}
#!/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

```python
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

```python
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 *(in memory)* + Operations
- Assignment binds a variable name to an object
- Types live in objects, *not* variables
- No more references? Python can remove object

<table>
<thead>
<tr>
<th>x = 42</th>
<th>y = x</th>
<th>x = 'hi'</th>
<th>y = x</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>str</td>
<td>int</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>hi</td>
<td>42</td>
</tr>
</tbody>
</table>

2012 Spring
Types

\textbf{type(}object\textbf{)}

- Python can tell you the type of object
- Example of \textit{introspection}

\begin{align*}
\text{type(42)} & \implies \text{\textless type 'int'\textgreater} \\
\text{type(3.141)} & \implies \text{\textless type 'float'\textgreater} \\
\text{type('hi')} & \implies \text{\textless type 'str'\textgreater} \\
\text{x = 5.0 / 2} \\
\text{type(x)} & \implies \text{\textless type 'float'\textgreater} \\
\text{type(type(42))} & \implies \text{\textless type 'type'\textgreater}
\end{align*}
None

\[ x = \text{None} \]

- 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

\textbf{dir(\textit{object or type})}

- Lists \textit{all} operations for that object or type
- For now, ignore everything that starts with \texttt{__}
- Use as \texttt{object.operation(...)}

\begin{verbatim}
>>> 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']
\end{verbatim}
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

```
a = 42
a += 1    # a = 43
a -= 3    # a = 40
a *= 2    # a = 80
a /= 8    # a = 10
...
```

- `a += 1` is slightly more efficient than `a = a + 1`
- `+=` and `*=` work on strings, too

```
a = b = c = 0
```

- OK but not recommended
Basic Input

```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)
```

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

<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
</tr>
</thead>
</table>

```python
>>> x = True
>>> x
True
>>> print x
True
>>> print False
False
>>> type(False)
<type 'bool'>
```
**Boolean Operations**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>( x )</th>
<th>( \text{not } x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td></td>
<td></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

name = raw_input('Name? ')
if name == 'Tim Cartwright':
    print 'Instructor'
else:
    print 'Student'
```

\[\text{opt.} \quad 0-n \text{ times}\]
Indentation

- Blocks must be indented consistently

```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

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 368-2 (2012 Spring)
Assigned on Day 02, 2012-03-15
Written by <Your Name>
"""
```