Lecture 8

Conditionals & Control Flow

[Andersen, Gries, Lee, Marschner, Van Loan, White]
Announcements: Assignment 1

• Due *tonight* at 11:59pm.
  ▪ Suggested early submit deadline of 2pm.
• Set CMS notifications to receive automatic emails when a grade is changed.
• First round of feedback should be out by Monday.
• If your A1 is not perfect, your first grade will be a 1.
  ▪ This is a counter for how many times you have submitted.
  ▪ It is not a permanent grade, can resubmit until March 2nd.
• Read section 2.3 of A1 carefully to understand how you can revise.
Announcements

• Please do not post code to Piazza\textsuperscript{1}
• Review the announcements from the end of Lecture 6 for policies:

\url{http://www.cs.cornell.edu/courses/cs1110/2017sp/lectures/02-14-17/presentation-06.pdf}

\textsuperscript{1}actually violating academic integrity rules because you are showing code to others
Methods: Functions Tied to Classes

- **Method**: function tied to class
  - Method call looks like a function call preceded by a variable name:
    \[
    \langle \text{variable} \rangle.\langle \text{method} \rangle(\langle \text{arguments} \rangle)
    \]
  - **Example**: `p.distanceTo(q)`
- Just like we saw for strings
  - `s = \text{'abracadabra'}`
  - `s.index('a')`
- **Are strings objects?** Actually, yes.
Name Resolution

- \langle object \rangle.\langle name \rangle means
  - Go the folder for \textit{object}
  - Look for attribute/method \textit{name}
  - If missing, check \textit{class folder}
- Class folder is a \textit{shared folder}
  - Only one for the whole class
  - Shared by all objects of class
  - Stores common features
  - Typically where methods are

\begin{align*}
\text{id3} & \quad \text{id4} \\
\text{Point3} & & \text{Point3} \\
\begin{array}{c}
x \quad 5.0 \\
y \quad 2.0 \\
z \quad 3.0 \\
\end{array} & & \\
\begin{array}{c}
x \quad 7.4 \\
y \quad 0.0 \\
z \quad 0.0 \\
\end{array}
\end{align*}

\text{x, y, z} \\
\text{distanceTo(other)} \\
\text{abs()}
Structure vs. Flow

Program Structure

• Order in which statements are written in scripts and modules
• Not necessarily the order in which Python executes them

Control Flow

• Order in which statements are actually executed at runtime
  ▪ Statements may be:
    • skipped
    • executed more than once

Have already seen this difference with functions
## Structure vs. Flow: Example

<table>
<thead>
<tr>
<th>Program Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def foo():</code></td>
<td>C:&gt; python foo.py</td>
</tr>
<tr>
<td><code>print 'Hello'</code></td>
<td>'Hello'</td>
</tr>
<tr>
<td># Script Code</td>
<td>'Hello'</td>
</tr>
<tr>
<td>foo()</td>
<td>'Hello'</td>
</tr>
</tbody>
</table>
| foo()             | 'Hello' | **Statement executed 3 times**
| foo()             |        |
| foo()             |        |

This statement listed only once
# Conditionals: If-Statements

## Format

```
if <boolean-expression>:
  <statement>
  ...
  <statement>
```

## Example

```python
# Put x in z if it is positive
if x > 0:
  z = x
```

## Execution:

if `<Boolean-expression>` is true, then execute all of the statements indented directly underneath (until first non-indented statement)
What gets printed?

```
a = 0
print a
```

prints 0
What gets printed?

\[
a = 0 \\
a = a + 1
\]

prints 1

\[
\text{print } a
\]
What gets printed?

```
a = 0
if a == 0:
    a = a + 1
print a
```

prints 1

```
print a
```
What gets printed?

```
a = 0
if a == 1:
    a = a + 1

print a
```
a = 0
if a == 1:
   a = a + 1
print a
a = 0
if a == 0:
  a = a + 1
a = a + 1
print a
prints 2
What gets printed?

```
a = 0
if a == 0:
    a = a + 1
if a == 0:
    a = a + 1
a = a + 1
print a
```

A: 0
B: 1
C: 2  CORRECT
D: 3
E: I do not know
Conditionals: If-Else-Statements

**Format**

```python
if <boolean-expression>:
    <statement>
    ...
else:
    <statement>
    ...
```

**Example**

```python
# Put max of x and y in z
if x > y:
    z = x
else:
    z = y
```

**Execution:**

if \(<Boolean-expression>\) is true, then execute statements indented under if; otherwise execute the statements indented under else
Conditionals: “Control Flow” Statements

\[
\text{if } b : \\
\quad s1 \ # \ statement \\
\quad s3
\]

\[
\text{if } b : \\
\quad s1 \\
\text{else:} \\
\quad s2 \\
\quad s3
\]

Flow

Program only takes one path each execution
What gets printed?

```python
a = 0
if a == 0:
    a = a + 1
prints 1
else:
    a = a + 1

print a
```
What gets printed?

a = 0
if a == 1:
    a = a + 1
else:
    a = a + 1
print a

prints 1
What gets printed?

```python
a = 0
if a == 1:
    a = a + 1
else:
    a = a + 1
a = a + 1
print a
```

prints 2
What gets printed?

```python
a = 0
if a == 1:
    a = a + 1
else:
    a = a + 1
    a = a + 1
    a = a + 1
a = a + 1
print a
```

prints 3
Program Flow and Call Frames

- **if** can change which statement is executed next

```python
def foo(a):
    if a == 0:
        print "hi"
    print "bye"
foo(0)
```

foo(0)
Program Flow and Call Frames

• if can change which statement is executed next

```python
def foo(a):
    if a == 0:
        print "hi"
    print "bye"
```

foo(1)
def max(x,y):
1    if x > y:
2        return x
3    return y

max(0,3):

<table>
<thead>
<tr>
<th>max</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>
What happens next?

```python
def max(x, y):
    if x > y:
        return x
    return y
```

Current call frame:

<table>
<thead>
<tr>
<th>max</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>

A: max | 2 |
| x    | 0 |
| y    | 3 |

B: max | 3 |
| x    | 0 |
| y    | 3 |

RETURN 0

C: max | 3 |
| x    | 0 |
| y    | 3 |

RETURN 3

D: max | 3 |
| x    | 0 |
| y    | 3 |
Program Flow and Call Frames

def max(x,y):
  if x > y:
    return x
  return y

max(0,3):

Skips line 2

<table>
<thead>
<tr>
<th>max</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>
def max(x,y):
    if x > y:
        return x
    return y

max(0,3):

Skips line 2
Program Flow and Variables

- Variables continue to exist outside of if
  ```python
  a = 0
  if a == 0:
    a = a + 1
  print a
  ```
- Also continue to exist even if *created* in if
Program Flow and Variables

a = 0
if a == 0:
    b = 0
print b

prints 0
Program Flow and Variables

```python
a = 0
if a == 1:
    b = 0
print b
```

Error!
def zero_or_one(b):
    if b:
        a = 0
    else:
        a = 1
    print a

better make sure that ALL if branches create the variable
Program Flow and Testing

- Can use print statements to examine program flow

'before if'

'if x>y'

'after if'

x must have been greater than y

# Put max of x, y in z

print 'before if'

if x > y:
    print 'if x>y'
    z = x
else:
    print 'else x<=y'
    z = y

print 'after if'

Traces
## Conditionals: If-Elif-Else-Statements

### Format

<table>
<thead>
<tr>
<th>if &lt;Boolean expression&gt;:</th>
<th>&lt;statement&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td><strong>elif</strong> &lt;Boolean expression&gt;:</td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td><strong>else</strong>:</td>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

### Example

```
# Put max of x, y, z in w
if x > y and x > z:
    w = x
elif y > z:
    w = y
else:
    w = z
```
**Conditionals: If-Elif-Else-Statements**

**Format**

```python
if <Boolean expression>:
    <statement>
...
elif <Boolean expression>:
    <statement>
    ...
else:
    <statement>
    ...
```

**Notes on Use**

- No limit on number of **elif**
  - Must be between if, else
- **else** is optional
  - if-elif by itself is fine
- Booleans checked in order
  - Once Python finds a true `<Boolean-expression>`, skips over all the others
  - else means all are false
a = 2

if a == 2:
    a = 3
elif a == 3:
    a = 4
print a

What gets printed?

A: 2
B: 3  CORRECT
C: 4
D: I do not know
If-Elif-Else

a = 2

if a == 2:
    a = 3
elif a == 3:
    a = 4
print a

prints 3

prints 4
def test_for_zeros(a, b):
    if a == 0:
        if b == 0:
            print "Both arguments are zero"
        else:
            print "The first argument is zero"
    else:
        if b == 0:
            print "The second argument is zero"
        else:
            print "Neither argument is zero"
## Conditional Expressions

### Format

\[ e_1 \text{ if } bexp \text{ else } e_2 \]

- \( e_1 \) and \( e_2 \) are any expression
- \( bexp \) is a Boolean expression
- This is an expression!

### Example

```
# Put max of x, y in z
z = x if x > y else y
```

expression, not statement