Lecture 8

Conditionals & Control Flow
# Announcements For This Lecture

## Readings
- Sections 5.1-5.7 today
- Chapter 4 for Tuesday

## Assignment 2
- Posted Today
  - Written assignment
  - Do while revising A1

## Assignment 1
- Due **TONIGHT**
  - Due *before* midnight
  - Submit something…
  - Can resubmit to Sep. 28
- Grades posted Saturday
- Complete the Survey
  - Must answer individually

9/17/15

Conditionals & Control Flow
# test procedure

def test_last_name_first():
    '''Test procedure for last_name_first(n)'''
    result = name.last_name_first('Walker White')
    cornelltest.assert_equals('White, Walker', result)
    result = name.last_name_first('Walker White')
    cornelltest.assert_equals('White, Walker', result)

# Application code

if __name__ == '__main__':
    test_last_name_first()
    print 'Module name is working correctly'
Types of Testing

Black Box Testing

- Function is “opaque”
  - Test looks at what it does
  - **Fruitful**: what it returns
  - **Procedure**: what changes
- **Example**: Unit tests
- **Problems**:
  - Are the tests everything?
  - What caused the error?

White Box Testing

- Function is “transparent”
  - Tests/debugging takes place inside of function
  - Focuses on where error is
- **Example**: Use of print
- **Problems**:
  - Much harder to do
  - Must remove when done
Finding the Error

- Unit tests cannot find the source of an error
- Idea: “Visualize” the program with print statements

```python
def last_name_first(n):
    """Returns: copy of <n> in form <last>, <first>"""
    end_first = n.find(' ')
    print end_first
    first = n[:end_first]
    print 'first is ' + str(first)
    last = n[end_first+1:]
    print 'last is ' + str(last)
    return last + ', ' + first
```

Print variable after each assignment

Optional: Annotate value to make it easier to identify
Structure vs. Flow

Program Structure

- Way statements are presented
  - Order statements are listed
  - Inside/outside of a function
  - Will see other ways…
- Indicate possibilities over multiple executions

Program Flow

- Order statements are executed
  - Not the same as structure
  - Some statements duplicated
  - Some statements are skipped
- Indicates what really happens in a single execution

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

<table>
<thead>
<tr>
<th>Program Structure</th>
<th>Program Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>def foo():</td>
<td>&gt;&gt;&gt; python foo.py</td>
</tr>
<tr>
<td>print 'Hello'</td>
<td>'Hello'</td>
</tr>
<tr>
<td># Script Code</td>
<td>'Hello'</td>
</tr>
<tr>
<td>if <strong>name</strong> == 'main':</td>
<td>'Hello'</td>
</tr>
<tr>
<td>foo()</td>
<td>Bugs can occur when we get a flow other than one that we were expecting</td>
</tr>
<tr>
<td>foo()</td>
<td>Statement listed once</td>
</tr>
<tr>
<td>foo()</td>
<td>Statement executed 3x</td>
</tr>
</tbody>
</table>

Bugs can occur when we get a flow other than one that we were expecting.
Conditionals: If-Statements

Format

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

Example

```
# 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)
Conditionals: If-Else-Statements

**Format**

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

**Example**

```
# Put max of x, 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

\begin{itemize}
\item \textbf{if} \( b \): \\
\hspace{1cm} \textit{s1} \# statement \\
\hspace{1cm} \textit{s3}
\end{itemize}

\begin{itemize}
\item \textbf{if} \( b \): \\
\hspace{1cm} \textit{s1} \\
\item \textbf{else}: \\
\hspace{1cm} \textit{s2} \\
\hspace{1cm} \textit{s3}
\end{itemize}

\textbf{Flow}
Program only takes one path each execution
def max(x, y):
    """Returns: max of x, y""
    # simple implementation
    1 if x > y:
    2    return x
    3 return y

(max(0, 3):

Frame sequence depends on flow
def max(x, y):
    
    # simple implementation

    # Returns: max of x, y"
    if x > y:
        return x
    return y

max(0, 3):

Frame sequence depends on flow

Skips line 2
def max(x,y):
    """Returns: max of x, y"""
    # simple implementation
    if x > y:
        return x
    return y

max(0,3):

Frame sequence depends on flow

Skips line 2
def max(x, y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return y

• temp is needed for swap
  • x = y loses value of x
  • “Scratch computation”
  • Primary role of local vars

• max(3, 0):

<table>
<thead>
<tr>
<th></th>
<th>max</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
<td>y</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Program Flow vs. Local Variables

```python
def max(x, y):
    """Returns: max of x, y""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return y
```

- `temp` is needed for swap
  - x = y loses value of x
  - “Scratch computation”
  - Primary role of local vars

- `max(3,0)`: 

  ![Diagram](attachment:image.png)
def max(x, y):
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return y

• temp is needed for swap
  ▪ x = y loses value of x
  ▪ “Scratch computation”
  ▪ Primary role of local vars

• max(3,0):

<table>
<thead>
<tr>
<th>max</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
</tr>
<tr>
<td>temp</td>
<td></td>
</tr>
</tbody>
</table>
Program Flow vs. Local Variables

def max(x,y):
    """Returns: max of x, y""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return y

• temp is needed for swap
  ▪ x = y loses value of x
  ▪ “Scratch computation”
  ▪ Primary role of local vars

• max(3,0):

```
<table>
<thead>
<tr>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>temp</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
def max(x,y):
    
    """Returns: max of x, y""
    
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return y

• temp is needed for swap
  ▪ x = y loses value of x
  ▪ “Scratch computation”
  ▪ Primary role of local vars

• max(3,0):

```
max  5
x   0  y  3
     temp  3
```
**Program Flow vs. Local Variables**

```python
def max(x,y):
    """Returns: max of x, y""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return y
```

- `temp` is needed for swap
  - `x = y` loses value of `x`
  - “Scratch computation”
  - Primary role of local vars

- `max(3,0)`:

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

- **temp**
  - 3

- **RETURN**
  - 3
Program Flow vs. Local Variables

def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return temp

• Value of max(3,0)?

A: 3
B: 0
C: Error!
D: I do not know
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return temp

• Value of max(3,0)?
  A: 3  CORRECT
  B: 0
  C: Error!
  D: I do not know

• Local variables last until
  ▪ They are deleted or
  ▪ End of the function

• Even if defined inside if
Program Flow vs. Local Variables

```python
def max(x,y):
    """Returns: max of x, y"""
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return temp
```

- Value of max(0,3)?
  A: 3
  B: 0
  C: Error!
  D: I do not know
Program Flow vs. Local Variables

```python
def max(x,y):
    '''Returns: max of x, y'''
    # swap x, y
    # put the larger in y
    if x > y:
        temp = x
        x = y
        y = temp
    return temp
```

• Value of `max(0,3)`?
  
  A: 3  
  B: 0  
  C: Error!  
  D: I do not know

• Variable existence depends on flow
• Understanding flow is important in testing
Program Flow and Testing

- Must understand which flow caused the error
  - Unit test produces error
  - Visualization tools show the current flow for error
- Visualization tools?
  - print statements
  - Advanced tools in IDEs (Integrated Dev. Environ.)

```python
# 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'
```
Program Flow and Testing

• Call these tools **traces**

• No requirements on how to implement your traces
  ▪ Less print statements ok
  ▪ Do not need to word them exactly like we do
  ▪ Do what ever is easiest for you to see the flow

• **Example:** `flow.py`

```python
# 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'
```

9/17/15
Conditionals & Control Flow
## Watches vs. Traces

<table>
<thead>
<tr>
<th><strong>Watch</strong></th>
<th><strong>Trace</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Visualization tool (e.g. print statement)</td>
<td>• Visualization tool (e.g. print statement)</td>
</tr>
<tr>
<td>• Looks at <strong>variable value</strong></td>
<td>• Looks at <strong>program flow</strong></td>
</tr>
<tr>
<td>• Often after an assignment</td>
<td>• Before/after any point where flow can change</td>
</tr>
<tr>
<td>• What you did in lab</td>
<td></td>
</tr>
</tbody>
</table>
```python
def cycle_left(p):
    print 'Start cycle_left()'
    p.x = p.y
    print p.x
    p.y = p.z
    print p.y
    p.z = p.x
    print p.z
    print 'End cycle_left()'
```

**Example:** flow.py
Local Variables Revisited

- Never refer to a variable that might not exist
  - Variable “scope”
    - Block (indented group) where it was first assigned
    - Way to think of variables; not actually part of Python
  - **Rule of Thumb**: Limit variable usage to its scope

```python
def max(x, y):
    """Returns: max of x, y""
    # swap x, y
    # put larger in temp
    if x > y:
        temp = x
        x = y
        y = temp
    return temp
```

First assigned

Outside scope
Local Variables Revisited

• Never refer to a variable that might not exist

• Variable “scope”
  - Block (indented group) where it was first assigned
  - Way to think of variables; not actually part of Python

• Rule of Thumb: Limit variable usage to its scope

```python
def max(x,y):
    """Returns: max of x, y""
    # swap x, y
    # put larger in temp
    temp = y
    if x > y:
        temp = x
    return temp
```

First assigned

Inside scope
def max(x, y):
    """Returns: max of x, y""
    if x > y:
        return x
    else:
        return y

Which is better?
Matter of preference

There are two returns!
But only one is executed
## Conditionals: If-Elif-Else-Statements

### Format

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

### Example

```python
# 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`
  - Can have as many as want
  - Must be between `if`, `else`
- The `else` is always optional
  - `if-elif` by itself is fine
- Booleans checked in order
  - Once it finds a true one, it skips over all the others
  - `else` means `all` are false

9/17/15 Conditionals & Control Flow
# Conditional Expressions

## Format

\[ e_1 \textbf{if } b_{\text{exp}} \textbf{ else } e_2 \]

- \(e_1\) and \(e_2\) are any expression
- \(b_{\text{exp}}\) is a boolean expression
- This is an expression!

## Example

```plaintext
# Put max of x, y in z
z = x \textbf{if } x > y \textbf{ else } y
```

expression, not statement