Lecture 8

Conditionals & Control Flow
Announcements For This Lecture

<table>
<thead>
<tr>
<th>Readings</th>
<th>Assignment 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sections 5.1-5.7 today</td>
<td>• Due <strong>SUNDAY</strong></td>
</tr>
<tr>
<td>• Chapter 4 for Tuesday</td>
<td>▪ Due <em>before</em> midnight</td>
</tr>
<tr>
<td></td>
<td>▪ Submit something…</td>
</tr>
<tr>
<td></td>
<td>▪ Can resubmit to Sep. 28</td>
</tr>
<tr>
<td></td>
<td>• Grades posted Tuesday</td>
</tr>
<tr>
<td></td>
<td>• Complete the Survey</td>
</tr>
<tr>
<td></td>
<td>▪ Must answer individually</td>
</tr>
</tbody>
</table>
# 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)

# Script code
test_last_name_first()
print 'Module name is working correctly'

Call function on test input
Compare to expected output
Call test procedure to activate the test
## Types of Testing

<table>
<thead>
<tr>
<th>Black Box Testing</th>
<th>White Box Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function is “opaque”</strong></td>
<td><strong>Function is “transparent”</strong></td>
</tr>
<tr>
<td>- Test looks at what it does</td>
<td>- Tests/debugging takes place inside of function</td>
</tr>
<tr>
<td>- Fruitful: what it returns</td>
<td>- Focuses on where error is</td>
</tr>
<tr>
<td>- Procedure: what changes</td>
<td>- <strong>Example</strong>: Use of print</td>
</tr>
<tr>
<td><strong>Example</strong>: Unit tests</td>
<td><strong>Example</strong>: Use of print</td>
</tr>
<tr>
<td><strong>Problems:</strong></td>
<td><strong>Problems:</strong></td>
</tr>
<tr>
<td>- Are the tests everything?</td>
<td>- Much harder to do</td>
</tr>
<tr>
<td>- What caused the error?</td>
<td>- Must remove when done</td>
</tr>
</tbody>
</table>
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

Program Structure

```python
def foo():
    print 'Hello'

# Script Code
foo()
foo()
foo()

```

Program Flow

```bash
>>> python foo.py
'Hello'
'Hello'
'Hello'

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

### Format

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

Format

\[
\text{if } <\text{boolean-expression}>: \\
\quad <\text{statement}> \\
\quad \ldots \\
\text{else: } \\
\quad <\text{statement}> \\
\quad \ldots 
\]

Example

\[
# \text{Put max of } x, y \text{ in } z \\
\text{if } x > y: \\
\quad z = x \\
\text{else: } \\
\quad z = y 
\]

Execution:

if <boolean-expression> is true, then execute statements indented under if; otherwise execute the statements indented under else.
**Conditionals: “Control Flow” Statements**

If $b$:
- $s1$ # statement
- $s3$

If $b$:
- $s1$
Else:
- $s2$
- $s3$

**Flow**
Program only takes one path each execution

9/15/16
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
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"""
    # 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>max</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>0</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](image)

  - `max = 2`
  - `x = 3`
  - `y = 0`
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):

```python
max 3
x 3  y 0
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):**

```
+---+---+---+
| max| 4 |
+---+---+---+
| x  | 0 | y  | 0 |
| temp | 3 |
+---+---+---+
```

9/15/16

Conditionals & Control Flow
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>x</th>
<th>y</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

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

```python
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):**

```
max
x | 0 | y | 3
```

```
temp | 3
```

```
RETURN | 3
```
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
**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(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

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

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! CORRECT
  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 whatever 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'
```

Traces
## Watches vs. Traces

<table>
<thead>
<tr>
<th>Watch</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visualization tool (e.g. <code>print</code> statement)</td>
<td>Visualization tool (e.g. <code>print</code> 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>

9/15/16

Conditionals & Control Flow
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()'
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

```
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
- Booleanas checked in order
  - Once it finds a true one, it skips over all the others
  - else means all are false
# Conditional Expressions

## Format

The format of conditional expressions is:

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

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

## Example

Example of using conditional expressions:

\[
\# \text{ Put max of } x, y \text{ in } z \\
z = x \textbf{if} x > y \textbf{else} y
\]

This is not a statement, but an expression.