Lecture 6

Conditionals & Control Flow
Announcements For This Lecture

Readings

• Sections 5.1-5.7 today
• Chapter 10 for Monday

Assignments

• Assignment 1 is posted
  ▪ Due next week
  ▪ Can revise until good
  ▪ …But ideally 3 tries
• Assignment 2 is the last
  ▪ Making some additions
  ▪ Will have all we need
Testing last_name_first(n)

# 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'
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

**Program Structure**

```python
def foo():
    print 'Hello'
# Script Code
foo()
foo()
foo()
```

**Statement listed once**

**Program Flow**

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

**Statement executed 3x**

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

### Format

<table>
<thead>
<tr>
<th>if &lt;boolean-expression&gt;:</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;statement&gt;</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>&lt;statement&gt;</td>
</tr>
</tbody>
</table>

### Example

# Put x in z if it is positive

```python
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

```python
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

```
if b:
    s1 # statement
s3
```

Branch Point: Evaluate & Choose

Statement: Execute

```
if b:
    s1
else:
    s2
s3
```

Flow
Program only takes one path each execution

9/15/16
Program Flow and Call Frames

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
Program Flow and Call Frames

```python
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):

```
+----+----+----+
| max| 1  |
+----+----+----+
| x 3 | y 0 |
```

max(3, 0):
9/15/16
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):

```
x    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

  temp  3

  x  3  y  0

  return y
```
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):**

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>temp</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
```

9/15/16
Conditionals & Control Flow
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 | 5
---+---
x  | 0  | y  | 3
    |    | temp | 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 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>
<tr>
<td>RETURN</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 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 \text{ 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
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

## Watch
- Visualization tool (e.g. `print` statement)
- Looks at **variable value**
- Often after an assignment
- What you did in lab

## Trace
- Visualization tool (e.g. `print` statement)
- Looks at **program flow**
- Before/after any point where flow can change
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

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>if</strong> <code>&lt;boolean-expression&gt;</code>:</td>
<td><code># Put max of x, y, z in w</code></td>
</tr>
<tr>
<td><code>&lt;statement&gt;</code></td>
<td><code>if x &gt; y and x &gt; z:</code></td>
</tr>
<tr>
<td>...</td>
<td><code>w = x</code></td>
</tr>
<tr>
<td><strong>elif</strong> <code>&lt;boolean-expression&gt;</code>:</td>
<td><code>elif y &gt; z:</code></td>
</tr>
<tr>
<td><code>&lt;statement&gt;</code></td>
<td><code>w = y</code></td>
</tr>
<tr>
<td>...</td>
<td><code>else:</code></td>
</tr>
<tr>
<td><strong>else:</strong></td>
<td><code>w = z</code></td>
</tr>
<tr>
<td><code>&lt;statement&gt;</code></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
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
Conditional Expressions

Format

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

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

Example

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

expression, not statement