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
# test procedure

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

```
>>> python foo.py
'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

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

### Example

```python
# 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 elsec
Conditionals: “Control Flow” Statements

**if** $b$:

- $s1$ # statement
- $s3$

**else:**

- $s1$
- $s2$

**Flow**

Program only takes one path each execution
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):

```
max
max
```

Frame sequence depends on flow
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'''
    # simple implementation
    1 if x > y:
    2     return x
    3         return y

Frame sequence depends on flow

max(0, 3):

Skips line 2
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></th>
<th>max</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
<td>y</td>
</tr>
</tbody>
</table>

9/15/17

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>max</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>0</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   3
  x     3   y   0
  temp  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 y

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

• max(3,0):

  x  0  y  0
  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>x</th>
<th>0</th>
<th>y</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>temp</td>
<td>3</td>
<td></td>
<td></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):

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

9/15/17
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

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
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')
```

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

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

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

print('after if')

Example: flow.py
## Conditionals: If-Elif-Else-Statements

### Format

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

e₁ if bexp else e₂

- e₁ and e₂ 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