Lecture 7

Conditionals & Control Flow
### Announcements For This Lecture

#### Assignment 1
- Should be working on it
  - Have covered everything
  - Extra testing exercises
  - Credit if you turn in A1
- Due Wednesday at mid.
  - Can work at it during lab
  - But labs are due as normal
- One-on-Ones sti
  - Lots of spaces available

#### Readings
- Thursday: Read 5.1-5.4
- Tuesday: **SKIM** Chap 4
  - Don’t use Swampy

#### AI Quiz
- Sent out e-mails Sunday
- Will start dropping today
Announcements For This Lecture

(Optional) Readings

- Sections 5.1-5.7
- Chapter 4 for Tuesday

Assignment 1

- Due Wednesday
  - Due *before* midnight
  - Can resubmit to Sep. 26

AI Quiz

- Sent e-mails yesterday
- Will start dropping today

9/13/18  Conditionals & Control Flow
# 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
```

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**

9/13/18  Conditionals & Control Flow
## Structure vs. Flow: Example

<table>
<thead>
<tr>
<th>Program Structure</th>
<th>Program Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>def foo():</strong></td>
<td><code>&gt;&gt;&gt; python foo.py</code></td>
</tr>
<tr>
<td><code>print('Hello')</code></td>
<td>'Hello'</td>
</tr>
<tr>
<td></td>
<td>'Hello'</td>
</tr>
<tr>
<td></td>
<td>'Hello'</td>
</tr>
<tr>
<td># Script Code</td>
<td>Statement listed once</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></td>
</tr>
<tr>
<td>foo()</td>
<td></td>
</tr>
<tr>
<td>foo()</td>
<td></td>
</tr>
</tbody>
</table>

9/13/18
Conditionals & Control Flow
## 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

```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

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

### Example

```
# Put max of x, y in z
if x > y:
  z = x
else:
  z = y
```
**Conditionals: “Control Flow” Statements**

- **if** $b$:
  - $s1$ # statement
  - $s3$

- **if** $b$:
  - $s1$
- **else**:
  - $s2$
  - $s3$

**Branch Point:** Evaluate & Choose

**Statement:** Execute

**Flow**
Program only takes one path each execution
def max(x, y):
    """Returns: max of x, y"""
    # simple implementation
    if x > y:
        return x
    return y

max(0, 3):
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):
    # simple implementation
    return x if x > y else y

max(0, 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):

  ![Diagram of max(3,0)](image)
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  2
  x   3   y   0
```

9/13/18
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):
  ```plaintext
<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>3</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  4
  x  0  y  0
temp  3
```

9/13/18
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>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td>temp</td>
<td>3</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

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

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
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

```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!  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 \textit{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
- \textbf{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')
```

9/13/18  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>
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')
**Conditionals: If-Elif-Else-Statements**

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
</table>

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

```
# 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