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
```

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

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

Structure vs. Flow: Example

**Program Structure**
```python
def foo():
    print('Hello')
```

**Program Flow**
```bash
>>> python foo.py
'Hello'
'Bugs can occur when we get a flow other than one that we were expecting
```

Conditionals: If-Statements

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if &lt;boolean-expression&gt;:</code>&lt;br&gt;<code>&lt;statement&gt;</code>&lt;br&gt;...&lt;br&gt;<code>&lt;statement&gt;</code></td>
<td># Put x in z if it is positive&lt;br&gt;<code>if x &gt; 0:</code>&lt;br&gt; `</td>
</tr>
</tbody>
</table>

Execution:
if `<boolean-expression>` is true, then execute all of the statements indented directly underneath (until first non-indented statement)

Conditionals: If-Else-Statements

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if &lt;boolean-expression&gt;:</code>&lt;br&gt;<code>&lt;statement&gt;</code>&lt;br&gt;...&lt;br&gt;<code>else:</code>&lt;br&gt;<code>&lt;statement&gt;</code></td>
<td># Put max of x, y in z&lt;br&gt;<code>if x &gt; y:</code>&lt;br&gt; `</td>
</tr>
</tbody>
</table>

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

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

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

**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

```
# 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')
```

**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

**Traces and Functions**

```
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>
| if <boolean-expression>: <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 |