## 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-name, first-name' """
    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
```

## How to Use the Results
- **Goal of white box testing is error location**
  - Want to identify the **exact line** with the error
  - Then you look real hard at line to find error
  - What you are doing in lab this week
  - But similar approach to **black box testing**
    - At each line you have **expected** print result
    - Compare it to the **received** print result
    - Line before first mistake is **likely** the error

## Structure vs. Flow

### Program Structure
- Order code is **presented**
- Order statements are listed
- Inside/outside of function
- Will see other ways…
- Defines possibilities over **multiple executions**

### Program Flow
- Order code is **executed**
- Not the same as structure
- Some statements duplicated
- Some statements skipped
- Defines what happens in a **single execution**

## Conditionals: If-Statements

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

**Execution:**
- If `expression` is **True**, execute all statements **indented** underneath

## Conditionals: If-Else-Statements

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

**Execution:**
- If `expression` is **True**, execute all statements indented under if.
- If `expression` is **False**, execute all statements indented under else.
### Conditionals: “Control Flow” Statements

**Format**

```python
if expression:
    statement
    ...
elif 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 first True, skips over all others
- `else` means no `if` or `elif` are true

### Program Flow and Call Frames

#### def max(x,y):

```python
# simple implementation
if x > y:
    return x
else:
    return y
```

#### max(0,3):

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

Frame sequence depends on flow

Skips line 2

### Testing and Code Coverage

- Typically, tests are written from **specification**
  - This is because they should be written first
  - You run these tests while you implement
- But sometimes tests leverage code structure
  - You know the control-flow branches
  - You want to make sure each branch is correct
  - So you explicitly have a test for each branch
- This is called **code coverage**

### 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>Often print/log statement</td>
<td>Often print/log statement</td>
</tr>
<tr>
<td>May have IDE support</td>
<td>May have IDE support</td>
</tr>
<tr>
<td>Looks at <strong>variable value</strong></td>
<td>Looks at program flow</td>
</tr>
<tr>
<td>Anywhere it can change</td>
<td>Anywhere it can change</td>
</tr>
<tr>
<td>Often after assignment</td>
<td>Before/after control</td>
</tr>
</tbody>
</table>

### Traces and Functions

**print('before if')**

```python
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**

- **Watches**
  - Visualization tool
  - Often print/log statement
  - May have IDE support
  - Looks at variable value
  - Anywhere it can change
  - Often after assignment

- **Traces**
  - Visualization tool
  - Often print/log statement
  - May have IDE support
  - Looks at program flow
  - Anywhere it can change
  - Before/after control