

Finding the Error

- Unit tests cannot find the source of an error
- Idea: “Visualize” the program with print statements

```
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

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

9/15/16

Conditionals & Control Flow

2

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

```
def foo():
    print 'Hello'
```

Statement listed once

```
# Script Code
foo()
foo()
foo()
```

Program Flow

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

Statement executed 3x

Bugs can occur when we get a flow other than one that we where expecting

Conditionals: If-Statements

Format

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

Example

```
# 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

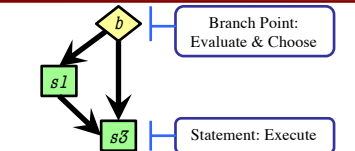
```
# Put max of x, y in z
if x > y:
| z = x
else:
| z = y
```

Execution:

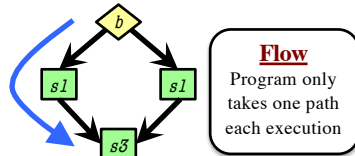
if <boolean-expression> is true, then execute statements indented under if; otherwise execute the statements indented under elsec

Conditionals: "Control Flow" Statements

```
if b:
    | s1 # statement
s2
```



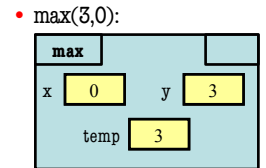
```
if b:
    | s1
else:
    | s2
s3
```



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



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.)

```
# 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

```
def shift(p):
    print 'Start shift()'
    p.x = p.y
    print p.x
    p.y = p.z
    print p.y
    p.z = p.x
    print p.z
    print 'End shift()'
```

Example: flow.py



Conditionals: If-Elif-Else-Statements

Format

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

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
```