## 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

```
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
```

Print variable after each assignment

**Optional**: Annotate value to make it easier to identify

## 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**

Have already seen this difference with functions

## Conditionals: If-Statements

### Format

```
if expression:
    statement
    ...
    statement
```

Indent

### Example

```
# Put x in z if it is positive
if x > 0:
    z = x
```

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

## Conditionals: If-Else-Statements

### Format

```
if expression:
    statement
    ...
else:
    statement
    ...
```

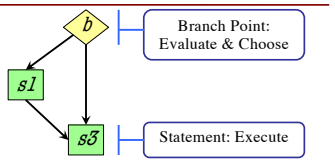
### Example

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

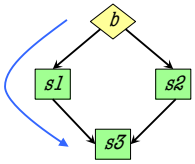
**Execution**:
If *expression* is **True**, execute all statements indented under if.
If *expression* is **False**, execute all statements indented under else.

## Conditionals: "Control Flow" Statements

```
if b:
    s1 # statement
s3
```




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



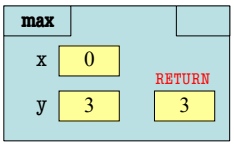

**Flow**
Program only takes one path each execution

## Program Flow and Call Frames

```
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

## 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

### Watch

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

### Trace

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

## Traces and Functions

**Example**: flow.py

```
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

### Format

```
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 **all** are false