Mini-Lecture 12

Debugging
Testing `last_name_first(n)`

```python
# test procedure
def test_last_name_first():
    """Test procedure for last_name_first(n)""
    result = name.last_name_first('Walker White')
    introcs.assert_equals('White, Walker', result)
    result = name.last_name_first('Walker White')
    introcs.assert_equals('White, Walker', result)

# Script code
test_last_name_first()
print('Module name is working correctly')
```

Call function on test input

Compare to expected output

Call test procedure to activate the test
# 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 variable after each assignment
    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
```

Optional: Annotate value to make it easier to identify
Conditionals and Debugging

• Must understand which branch 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')
Conditionals and Debugging

• 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 whatever is easiest for you to see the flow

• **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')
```

Traces
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

Example: flow.py

```python
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')
```