Recall: The Python API

Anatomy of a Specification

```python
def greet(n):
    """Prints a greeting to the name n
    Greeting has format 'Hello <n>!' Followed by conversation starter.
    Parameter n: person to greet
    Precondition: n is a string"
    print 'Hello ' + n + '!
    print 'How are you?'
```

Preconditions

- Precondition is a promise
  - If precondition is true, the function works
  - If precondition is false, no guarantees at all
- Get software bugs when
  - Function precondition is not documented properly
  - Function is used in ways that violates precondition

Test Cases: Finding Errors

Representative Tests

```python
def number_vowels(w):
    """Returns: number of vowels in word w.
    Precondition: w string w/ at least one letter and only letters"
    pass  # nothing here yet!
```

Representative Tests for number_vowels(w)

```python
>>> to_centigrade(32)
0.0
>>> to_centigrade(212)
100.0
>>> to_centigrade('32')
Precondition violated
```
How Many “Different” Tests Are Here?

number_vowels(w)

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hat'</td>
<td>1</td>
</tr>
<tr>
<td>'charm'</td>
<td>1</td>
</tr>
<tr>
<td>'bet'</td>
<td>1</td>
</tr>
<tr>
<td>'beet'</td>
<td>2</td>
</tr>
<tr>
<td>'beetle'</td>
<td>3</td>
</tr>
<tr>
<td>A: 2</td>
<td></td>
</tr>
<tr>
<td>B: 3</td>
<td></td>
</tr>
<tr>
<td>C: 4</td>
<td></td>
</tr>
<tr>
<td>D: 5</td>
<td></td>
</tr>
<tr>
<td>E: I do not know</td>
<td></td>
</tr>
</tbody>
</table>

Running Example

• The following function has a bug:

```
def last_name_first(n):
    """Returns: copy of <n> but in the form <last-name>, <first-name>
    Precondition: <n> is in the form <first-name> <last-name>
    with one or more blanks between the two names""
    end_first = n.find("")
    first = n[:end_first]
    last = n[end_first+1:]
    return last + " first " + first
```

• Representative Tests:
  • `last_name_first(Walker White)` gives 'White, Walker'
  • `last_name_first(Walker White)` gives 'White, Walker'

Unit Test: A Special Kind of Script

• A unit test is a script that tests another module
  • It imports the other module (so it can access it)
  • It imports the `cornell` module (for testing)
  • It defines one or more test cases
    • A representative input
    • The expected output
  • The test cases use the `cornell` function

```
def assert_equals(expected,received):
    """Quit program if expected and received differ""
```

Testing `last_name_first(n)`

```
import cornell # Includes the test procedures
import name # The module we want to test

def test_last_name_first():
    """Test procedure for last_name_first(n)""
    print("Testing function last_name_first()")
    result = name.last_name_first("Walker White")
    cornell.assert_equals("White, Walker", result)
    result = name.last_name_first("Walker White")
    cornell.assert_equals("White, Walker", result)

# Execution of the testing code
if __name__ == '__main__':
    test_last_name_first()
```

Using Test Procedures

• In the real world, we have a lot of test cases
  • I wrote 1000+ test cases for a C++ game library
  • You need a way to cleanly organize them
• Idea: Put test cases inside another procedure
  • Each function tested gets its own procedure
  • Procedure has test cases for that function
  • Also some print statements (to verify tests work)
• Turn tests on/off by calling the test procedure

Test Procedure

```
def test_last_name_first():
    """Test procedure for last_name_first(n)""
    result = name.last_name_first("Walker White")
    cornell.assert_equals("White, Walker", result)
    cornell.assert_equals("White, Walker", result)

# Execution of the testing code
if __name__ == '__main__':
    test_last_name_first()
```

No tests happen if you forget this