Mini-Lecture 9

Testing
Test Cases: Finding Errors

- **Bug**: Error in a program. (Always expect them!)
- **Debugging**: Process of finding bugs and removing them.
- **Testing**: Process of analyzing, running program, looking for bugs.
- **Test case**: A set of input values, together with the expected output.

Get in the habit of writing test cases for a function from the function’s specification — even *before* writing the function’s body.

```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!
```
Test Cases: Finding Errors

- **Bug**: Error in a program. (Always expect them!)
- **Debugging**: Process of finding bugs and removing them.
- **Testing**: Process of analyzing, running program, looking for bugs.
- **Test case**: A set of input values, together with the expected output.

Get in the habit of writing test cases for a function from the function’s specification—even before writing the function’s body.

Some Test Cases

- `number_vowels('Bob')`
  - Answer should be 1
- `number_vowels('Aeiuo')`
  - Answer should be 5
- `number_vowels('Grrr')`
  - Answer should be 0

```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

• Cannot test all inputs
  ▪ “Infinite” possibilities

• Limit ourselves to tests that are representative
  ▪ Each test is a significantly different input
  ▪ Every possible input is similar to one chosen

• An art, not a science
  ▪ If easy, never have bugs
  ▪ Learn with much practice

Representative Tests for number_vowels(w)

• Word with just one vowel
  ▪ For each possible vowel!

• Word with multiple vowels
  ▪ Of the same vowel
  ▪ Of different vowels

• Word with only vowels
• Word with no vowels
How Many “Different” Tests Are Here?

The function definition is:

\[
\text{number_vowels}(w)
\]

<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>
</tbody>
</table>

A: 2  
B: 3  
C: 4  
D: 5  
E: I do not know
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>
</tbody>
</table>

- If in doubt, just add more tests
- You are never penalized for too many tests

A: 2
B: 3 CORRECT(ISH)
C: 4
D: 5
E: I do not know
The following function has a bug:

```python
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
```

Representative Tests:
- `last_name_first('Walker White')` gives 'White, Walker'
- `last_name_first('Walker White')` gives 'White, Walker'
The following function has a bug:

```python
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
```

Look at precondition when choosing tests

Representative Tests:
- `last_name_first('Walker White')` give 'White, Walker'
- `last_name_first('Walker White')` gives 'White, Walker'
Unit Test: A Special Kind of Script

• Right now to test a function we do the following
  - Start the Python interactive shell
  - Import the module with the function
  - Call the function several times to see if it is okay
• But this is incredibly time consuming!
  - Have to quit Python if we change module
  - Have to retype everything each time
• What if we made a second Python module/script?
  - This module/script tests the first one
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 introcs module (for testing)
  - It defines one or more test cases
    - A representative input
    - The expected output
- The test cases use the introcs function

```python
def assert_equal(expected, received):
    """Quit program if expected and received differ"""
```
import name  # The module we want to test
import introcs  # Includes the test procedures

# First test case
result = name.last_name_first('Walker White')
introcs.assert_equals('White, Walker', result)

# Second test case
result = name.last_name_first('Walker White')
introcs.assert_equals('White, Walker', result)

print('Module name is working correctly')
import name # The module we want to test
import cornell # Includes the test procedures

# First test case
result = name.last_name_first('Walker White')
introcs.assert_equals('White, Walker', result)

# Second test case
result = name.last_name_first('Walker White')
introcs.assert_equals('White, Walker', result)

print('Module name is working correctly')

9/14/18
Testing last_name_first(n)

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

# First test case
result = name.last_name_first('Walker White')
introcs.assert_equals('White, Walker', result)

# Second test case
result = name.last_name_first('Walker White')
introcs.assert_equals('White, Walker', result)

print('Module name is working correctly')

Quits Python if not equal

Message will print out only if no errors.