A1: The Module urllib2

• Module urllib2 is used to read web pages
  * Function urlopen creates a url object
    
    ```python
    u = urllib2.urlopen('http://www.cornell.edu')
    ```
  * url has a method called read()
    * Returns contents of web page
    * Usage: `s = u.read()` # s is a string

Recall: The Python API

- This is a specification
  * Enough info to use func.
  * But not how to implement
  * Write them as docstrings

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?'
```

- One line description, followed by blank line
- More detail about the function. It may be many paragraphs.
- Parameter description
- Precondition specifies assumptions we make about the arguments

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

```python
>>> to_centigrade(32.0)
0.0
>>> to_centigrade(212)
100.0
>>> to_centigrade('32')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "temperature.py", line 19 ...
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Test Cases: Finding Errors

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

```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
```

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

Running Example

• 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'

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 cornelltest module (for testing)
  * It defines one or more test cases
    * A representative input
    * The expected output
  * The test cases use the cornelltest function

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

Testing last_name_first(n)

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

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

# Second test case
result = name.last_name_first('Walker    White')
cornelltest.assert_equals('White, Walker', result)
print 'Module name is working correctly'
```

Test Procedure

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

# Execution of the testing code
test_last_name_first()
print 'Module name is working correctly'
```