One-on-One Sessions

- Starts tomorrow: 1/2-hour one-on-one sessions
  - To help prepare you for the assignment
  - **Primarily for students with little experience**
- There are still some spots available
  - Sign up for a slot in CMS
- Will keep running after **September 19**
  - Will open additional slots after the due date
  - Will help students revise Assignment 1

A1: The Module urllib2

- Module urllib2 is used to read web pages
  - Function urlopen creates a url object
    - `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

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

```
>>> to_centigrade(32)
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.
- **Testing**: Process of analyzing, running program, looking for bugs.
- **Test case**: A set of input values, together with the expected output.

```
def number_vowels(w):
    """Returns: number of vowels in word w.
    Parameter 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

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
  """
import name
import cornelltest
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)
print 'Module name is working correctly'
```

- 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""
    
    """Quit program if expected and received differ"
    ```

Testing last_name_first(n)

- The tests run:
  ```python
import name
import cornelltest

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

# 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'
```

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

Testing last_name_first(n)

- The tests run:
  ```python
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')
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'
```