Lecture 6

Specifications & Testing
Announcements For This Lecture

Last Call

• Acad. Integrity Quiz
• Take it by tomorrow
• Also remember survey

Assignment 1

• Posted on web page
  ▪ Due Sun, Sep. 18th
  ▪ Due in place of Lab 4
  ▪ Revise until correct

• Can work in pairs
  ▪ One submission for pair
  ▪ Mixer today at 5:30
  ▪ Meet in Gates Atrium
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
Recall: The Python API

Function name: `math.ceil(x)`

Number of arguments: 1

What the function evaluates to: Return the ceiling of x as a float, the smallest integer value greater than or equal to x.
Recall: The Python API

- Function name: math.ceil(x)
- Number of arguments: 1
- What the function evaluates to:
  Return the ceiling of x as a float, the smallest integer value greater than or equal to x.

- 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
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?"""
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?'
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?'
Anatomy of a Specification

```python
def to_centigrade(x):
    """Returns: x converted to centigrade
    Value returned has type float.
    Parameter x: temp in fahrenheit
    Precondition: x is a float"""
    return 5*(x-32)/9.0
```

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
Anatomy of a Specification

```python
def to_centigrade(x):
    
    """Returns: x converted to centigrade
    Value returned has type float.
    Parameter x: temp in fahrenheit
    Precondition: x is a float"

    return 5*(x-32)/9.0
```

"""Returns"" indicates a fruitful function

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
```
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 ...
  TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Precondition violated
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
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')` give 'White, Walker'
  - `last_name_first('Walker White')` gives 'White, Walker'
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(' ') # put ' ' in python code
    first = n[:end_first]
    last = n[end_first+1:]
    return last+', '+first
```

Representative Tests:

- `last_name_first('Walker White')` give 'White, Walker'
- `last_name_first('Walker White')` gives 'White, Walker'

Look at precondition when choosing tests
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'
```
# The module we want to test

```python
import name  # The module we want to test

import cornelltest  # Includes the test procedures
```

# First test case

```python
result = name.last_name_first('Walker White')
cornelltest.assert_equals('White, Walker', result)
```

# Second test case

```python
result = name.last_name_first('Walker            White')
cornelltest.assert_equals('White, Walker', result)
```

print 'Module name is working correctly'

9/8/16 Specifications & Testing
Testing last_name_first(n)

```
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'
```

Quits Python if not equal

Message will print out only if no errors.
Using Test Procedures

• In the real world, we have a lot of test cases
  ▪ I wrote 10000+ 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
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'
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'