Anatomy of a Specification

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?')

"""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

Anatomy of a Specification

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

What Makes a Specification “Good”?

• Software development is a business
  § Not just about coding – business processes
  § Processes enable better code development
• Complex projects need multi-person teams
  § Lone programmers do simple contract work
  § Teams must have people working separately
• Processes are about how to break-up the work
  § What pieces to give each team member?
  § How can we fit these pieces back together?

Preconditions are a Promise

• If precondition true
  § Function must work
  >>> to_centigrade(32.0)
  0.0
  >>> 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’
  Precondition violated
  § Function might work
  § Function might not
• Assigns responsibility
  § How to tell fault?

Testing Software

• You are responsible for your function definition
  § You must ensure it meets the specification
  § May even need to prove it to your boss
• Testing: Analyzing & running a program
  § Part of, but not the same as, debugging
  § Finds bugs (errors), but does not remove them
• To test your function, you create a test plan
  § A test plan is made up of several test cases
  § Each is an input (argument), and its expected output

Representative Tests

• We cannot test all possible inputs
  § “Infinite” possibilities (strings arbitrary length)
  § Even if finite, way too many to test
• Limit to tests that are representative
  § Each test is a significantly different input
  § Every possible input is similar to one chosen
• This is an art, not a science
  § If easy, no one would ever have bugs
  § Learn with much practice (and why teach early)
Representative Tests

- **Representative Tests for number_vowels(w)**
  - Word with just one 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 a copy of n in the form 'last-name, first-name'
    Precondition: n is in the form 'first-name last-name'
    with one or more spaces 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')` returns 'White, Walker'
  - `last_name_first('Walker      White')` returns 'White, Walker'

The Rule of Numbers

- When testing the numbers are 1, 2, and 0
  - **Number 1:** The simplest test possible
    - If a complex test fails, what was the problem?
    - **Example:** Word with just one vowels
  - **Number 2:** Add more than was expected
    - **Example:** Multiple vowels (all ways)
  - **Number 0:** Make something missing
    - **Example:** Words with no vowels

Unit Test: An Automated Test Script

- A **unit test** is a script to test a single function
  - Imports the function module (so it can access it)
  - Imports the `introcs` module (for testing)
  - Implements one or more test cases
    - A representative input
    - The expected output
  - The test cases use the `introcs` 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 introcs # Includes the test procedures

# Test one space between names
result = name.last_name_first('Walker White')
introcs.assert_equals('White, Walker', result)

# Test multiple spaces between names
result = name.last_name_first('Walker      White')
introcs.assert_equals('White, Walker', result)
```

Test Procedure

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

test_last_name_first()
```