

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

>>> to_centigrade(32.0)

■ Function must work

>>> to_centigrade('32')

• If precondition false

Traceback (most recent call last):

■ Function might work

File "<stdin>", line 1, in <module>

Function might not

File "temperature.py", line 19 ...

Assigns responsibility

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

• How to tell fault?

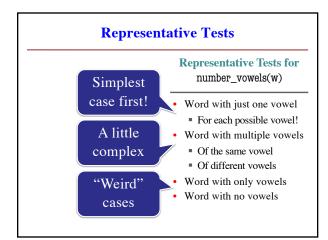
Precondition violated

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 arbritrary 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)



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

Running Example The following function has a bug: 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] Precondition last = n[end_first+1:] forbids a 0th test return last+', '+first • Representative Tests: last_name_first('Walker White') returns 'White, Walker' last_name_first('Walker White') returns 'White, Walker'

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

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

```
Testing last_name_first(n)
import name
                       # The mo
                                     Comment
                       # Includ
import introcs
                                   describing test
# Test one space between names
result = name.last_name_first('Walker White')
introcs.assert\_equals('White, Walker', result)
 Actual Output
                                         Input
# T multiple spaces between names
result = name.last_name_first('Walker
                                         White')
introcs.assert_equals('White, Walker', result)
                           Expected Output
print('Module name passed an
```

#