

# Lecture 6: Specifications & Testing

(Sections 4.9, 9.5)

CS 1110

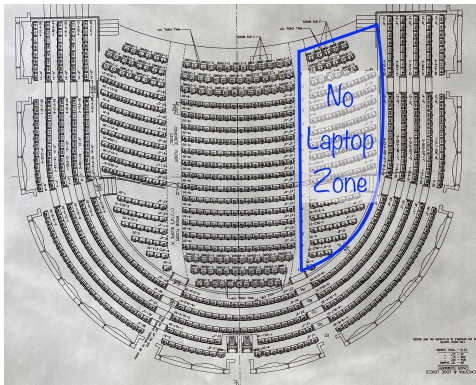
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]



- We strongly encourage you to look at the [last\\_name\\_first function in the Python tutor](#).
- Now try to fix the function implementation!

Welcome!



Please, no cell phones during lecture

## Announcements

- 1-on-1s are happening and they are awesome!
  - Sign up on CMS
- A1 goes out tonight! (many pages, but big figures)
- **Academic Integrity Policy:**
  - You can talk to each other
  - Do not show anyone (except staff) your code
    - Do not post your code to Ed Discussions
    - Do not look at anyone else's code
  - The Full Policy:

<https://www.cs.cornell.edu/courses/cs1110/2022sp/policies/cs1110integrity.html>

## Asking Questions in Lecture

- Raise your hand for a notecard!
- Raise both hands for the catchBox!

## Recall the Python API

<https://docs.python.org/3/library/math.html>

The screenshot shows the Python documentation for `math.sqrt(x)`. Annotations include:

- Function name:** `math.sqrt(x)`
- inputs:** `x`
- What the function evaluates to:** Return the square root of `x`.
- Specification:** This is a **specification** (not an implementation). It describes how to use the function and how to write docstrings.

## Anatomy of a Specification (1)

```
def greet(name):
    """Greet the person called name
    followed by conversation starter.

    <more details could go here>

    name: the person to greet
    Precondition: name is a string"""
    print('Hello '+name+'!')
    print('How are you?')
```

welcome.py

Short description, followed by blank line

As needed, more detail in 1 (or more) paragraphs

Parameter description

Precondition specifies assumptions we make about the arguments

7

## Anatomy of a Specification (2)

```
def get_campus_num(phone_num):
    """Returns the on-campus version
    of a 10-digit phone number.

    Returns: str of form "X-XXXX"

    phone_num: number w/area code
    Precondition: phone_num is a 10
    digit string of only numbers"""
    return phone_num[5]+"-"+phone_num[6:10]
```

campus.py

Short description, followed by blank line

Information about the return value

Parameter description

Precondition specifies assumptions we make about the arguments

8

## A Precondition Is a Contract (1)

```
def get_campus_num(phone_num):
    """Returns: str of form "X-XXXX"
    phone_num: number w/area code
    Precondition: phone_num is a 10
    digit string of only numbers"""
    return phone_num[5]+ "-" + phone_num[6:10]
```

campus.py

If the precondition is met, the function will work!

9

## A Precondition Is a Contract (2)

```
def get_campus_num(phone_num):
    """Returns: str of form "X-XXXX"
    phone_num: number w/area code
    Precondition: phone_num is a 10
    digit string of only numbers"""
    return phone_num[5]+ "-" + phone_num[6:10]
```

If the precondition is **not** met... Sorry, no guarantees!

```
>>> import campus
>>> campus.get_campus_num(6072531234)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/bracy/campus.py", line 7, in
  get_campus_num
    return phone_num[5]+ "-" + phone_num[6:10]
TypeError: 'int' object is not subscriptable
```

Precondition violated: **error message!**

10

## A Precondition Is a Contract (2)

```
def get_campus_num(phone_num):
    """Returns: str of form "X-XXXX"
    phone_num: number w/area code
    Precondition: phone_num is a 10
    digit string of only numbers"""
    return phone_num[5]+ "-" + phone_num[6:10]
```

If the precondition is **not** met... Sorry, no guarantees!

```
>>> import campus
>>> campus.get_campus_num("607-255-4444")
'5-5-44'
```

Precondition violated: **NO error message!**

11

## Software Bugs occur if

- Precondition is not documented properly
  - Easy to be unaware of assumptions we make
- Function use violates the precondition
  - Easy to think we're using a function properly, even if we're not

12

## NASA Mars Climate Orbiter

“NASA lost a \$125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation...”



13

Sources: Wikipedia & CNN

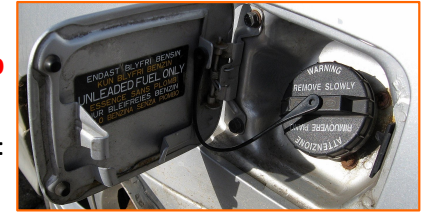
## Preconditions Make Expectations Explicit

In American terms:

**Preconditions help assign blame.**

Something went wrong:

*Engine breaks down.*



Did you give the function a bad argument?

*Did you put the wrong kind of fuel in the car?*

OR

Was the function implemented/specified wrong?

*Did the fuel tank ask for the wrong kind of fuel?*

*Was the engine simply poorly built?*

14

## Basic Terminology

- **Bug:** an error in a program. Expect them!
  - Conceptual & implementation
- **Debugging:** the process of finding bugs and removing them
- **Testing:** the process of *analyzing* and running a 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 its specification  
– even *before* writing the function itself!

15

## Test cases help you find errors

```
def vowel_count(word):
    """Returns: number of vowels in word.

    word: a string with at least one letter & only letters"""
    pass # nothing here yet!
```

### Some Test Cases

- vowel\_count('Bob')  
Expect: 1
- vowel\_count('Aeiuo')  
Expect: 5
- vowel\_count('Grrr')  
Expect: 0

### More Test Cases

- vowel\_count('y')  
Expect: 0? 1?
- vowel\_count('Bobo')  
Expect: 1? 2?

Test Cases can help you find errors in the **specification** as well as the implementation.

16

## 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 vowel\_count(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

17

## Representative Tests Example

```
def last_name_first(full_name):
    """Returns: copy of full_name
    in form <last-name>, <first-name>

    full_name: a string with the form <first-name> <last-name>
    with one or more blanks between the two names"""
    space_index = full_name.index(' ')
    first = full_name[:space_index]
    last = full_name[space_index+1:]
    return last+', '+first
```

name.py

Look at precondition when choosing tests

Representative Tests:

```
>>> import name
>>> name.last_name_first('Katherine Jones')
           Expects: 'Jones, Katherine'
>>> name.last_name_first('Katherine Jones')
           Expects: 'Jones, Katherine'
```

19

## Motivating a Unit Test

- Right now to test a function, we:
  - Start the Python interactive shell
  - Import the module with the function
  - Call the function several times to see if it works right
- Super time consuming! ☹️
  - Quit and re-enter python every time we change module
  - Type and retype...
- What if we wrote a script to do this ?!



20

## cornellasserts module

- Contains useful testing functions
- To use:
  - Download from course website (one of today's lecture files)
  - Put in same folder as the files you wish to test

21

## Unit Test: A Special Kind of Script

- A unit test is a script that tests another module. It:
  - Imports the module to be tested (so it can access it)
  - Imports `cornellasserts` module (supports testing)
  - Defines one or more test cases that each includes:
    - A representative input
    - The expected output
  - Test cases call a `cornellasserts` function:

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

22

## Testing `last_name_first(full_name)`

```
import name # The module we want to test  
import cornellasserts # Module that supports testing  
  
# First test case  
result = name.last_name_first('Katherine Jones')  
cornellasserts.assert_equals('Jones, Katherine', result)  
  
# Second test case  
result = name.last_name_first('Katherine Jones')  
cornellasserts.assert_equals('Jones, Katherine', result)  
  
print('All tests of the function last_name_first passed')
```

Annotations:   
- **Actual output**: points to the first `assert_equals` call.  
- **Expected output**: points to the string `'Jones, Katherine'` in the first `assert_equals` call.  
- **Input**: points to the string `'Katherine Jones'` in the first `last_name_first` call.  
- **Quits Python if actual and expected output not equal**: points to the `assert_equals` function.  
- **Prints only if no errors**: points to the `print` statement at the end.

23

## Organizing your Test Cases

- We often have a lot of test cases
  - Common at (good) companies
  - Need a way to cleanly organize them



**Idea:** Bundle all test cases into a single test!

- One **high level test** for each function you test
- High level test performs **all** test cases for function
- Also uses some print statements (for feedback)

24

## One Test to Rule them All

```
import cornellasserts  
import name  
import campus  
  
def test_last_name_first():  
    """Calls all the tests for last_name_first"""  
    print('Testing function last_name_first')  
    # Test Case 1  
    result = name.last_name_first('Katherine Jones')  
    cornellasserts.assert_equals('Jones, Katherine', result)  
    # Test Case 2  
    result = name.last_name_first('Katherine Jones')  
    cornellasserts.assert_equals('Jones, Katherine', result)  
  
# Execution of the testing code  
test_last_name_first()  
print('All tests of the module name passed')
```

Annotations:   
- **name\_campus\_test.py**: points to the filename.  
- **Put all test cases inside one function**: points to the `test_last_name_first` function definition.  
- **No tests happen if you forget to call the function**: points to the `test_last_name_first()` call.

25

## Debugging with Test Cases (Question)

```
def last_name_first(full_name):  
    """Returns: copy of full_name in the form <last-name>, <first-name>
```

```
    full_name: has the form <first-name> <last-name>  
    with one or more blanks between the two names"""
```

```
    #get index of space after first name
```

```
1  space_index = full_name.index(' ')
```

```
    #get first name
```

```
2  first = full_name[:space_index]
```

```
    #get last name
```

```
3  last = full_name[space_index+1:]
```

```
    #return "<last-name>, <first-name>"
```

```
4  return last+', '+first
```

Which line is  
"wrong"?

A: Line 1

B: Line 2

C: Line 3

D: Line 4

E: I don't know

```
last_name_first('Katherine Jones')    gives 'Jones, Katherine'
```

```
last_name_first('Katherine  Jones')   gives ' Jones, Katherine'
```

26



## How to debug

Do not ask:

"Why doesn't my code do what I want it to do?"

Instead, ask:

"What is my code doing?"

Two ways to inspect your code:

1. **Step through your code**, drawing pictures (or use *python tutor if possible*)
2. **Use print statements** to reveal intermediate program states—**variable values**

28

## Take a look in the python tutor!

```
def last_name_first(full_name):  
    # get index of space  
    space_index = full_name.index(' ')  
    # get first name  
    first = full_name[:space_index]  
    # get last name  
    last = full_name[space_index+1:]  
    # return "<last-name>, <first-name>"  
    return last+', '+first
```

```
last_name_first("Katherine Johnson")
```

### Pay attention to:

- Code relevant to the failed test case
- Code you weren't 100% sure of as you wrote it

29

## Using print statement to debug

```
def last_name_first(full_name):  
    # get index of space  
    space_index = full_name.index(' ')  
    print('space_index = '+ str(space_index))  
    # get first name  
    first = full_name[:space_index]  
    print('first = '+ first)  
    # get last name  
    last = full_name[space_index+1:]  
    print('last = '+ last)  
    # return "<last-name>, <first-name>"  
    return last+', '+first
```

Sometimes this is your only option, but it does make a mess of your code, and introduces cut-n-paste errors.

How do I print this?

30