Lecture 6:
Specifications & Testing
(Sections 4.9, 9.5)

CS 1110
Introduction to Computing Using Python

Orange text indicates updates made after lecture

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

• No laptop use stage right (your left)

• We will use clickers, but not for credit. Therefore no need to register your clicker.

• To access video of lecture, log in using NetID and password “through Canvas”, but we don’t use Canvas otherwise. Course website is https://www.cs.cornell.edu/courses/cs1110/2020sp/

• Before next lecture, read Chapter 15
More announcements

• Download code from lecture and experiment with it—run, modify, run again, ...

• Assignment 1 will be posted today or Friday
  ▪ Have over a week to do it
  ▪ Can choose to work with one partner and together submit one assignment
  ▪ Can revise and resubmit after getting grading feedback

• Starting next week: **optional 1-on-1** with a staff member to help *just you* with course material. Sign up for a slot on CMS under “SPECIAL: one-on-ones“.
Continue from previous lecture:

String

print vs return
String: Text as a Value

- String are quoted characters
  - 'abc d' (Python prefers)
  - "abc d" (most languages)

- How to write quotes in quotes?
  - Delineate with “other quote”
  - **Example**: " ' " or ' " '
  - What if need both " and '?

- **Solution**: escape characters
  - Format: \ followed by letter (character)
  - Special or invisible chars

<table>
<thead>
<tr>
<th>Char</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\</td>
<td>single quote</td>
</tr>
<tr>
<td>&quot;</td>
<td>double quote</td>
</tr>
<tr>
<td>\n</td>
<td>new line</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
</tbody>
</table>
def greet(n):
    """Prints a greeting to the name n

    Parameter n: name to greet
    Precondition: n is a string"

    print('Hello ' + n + '!
    print('How are you?')

Displays these strings on the screen

No assignments or return (returns None)
print vs. return

- Displays a value on screen
- Used primarily for testing
- Not useful for calculations

• Sends a value from a function call frame back to the caller
• Important for calculations
• Does not display anything

```python
def print_plus(n):
    print(n+1)

>>> print_plus(2)
3

```
Python Interactive Mode

• executes both *statements* and *expressions*

• if *expression:*
  1. *evaluates*
  2. *prints value (if one exists)*

```
>>> 2+2  # evaluates (performs addition)
4

>>> return_plus(2)  # evaluates (makes function call, gets return value)
3
```

```python
def return_plus(x):
    return x + 1
```
def return_plus(n):
    return n + 1

Python Interactive Mode

>>> return_plus(2)  
3

return_plus in action

1. Evaluates: makes function call, evaluates to return value

2. Python interactive mode prints that value
print_plus in action

```
def print_plus(n):
    print(n+1)
```

Python Interactive Mode

```
>>> print_plus(2)
3
```

1. Evaluates : makes function call, evaluates to return value (**NONE**)

2. does not print value b/c it’s **NONE**
**hybrid_plus in action**

```python
def hybrid_plus(n):
    print(n)
    return n+1
```

Python Interactive Mode

```python
>>> print_plus(2)
2
3
```
# Exercise 1

## Module Text

```python
# module.py

def foo(x):
    x = x + 3
    x = 3 * x
```

## Python Interactive Mode

```python
>>> import module
>>> print(module.x)
... What does Python give me?
```

A: 9  
B: 10 
C: 1  
D: None  
E: Error

Code shown in lecture was 1+2. Some students were confused because the argument \( x \) wasn’t used. It wasn’t an error, but we changed the code now to avoid any distraction.
# module.py

```python
def foo(x):
    x = x + 3
    x = 3 * x

>>> import module
>>> print(module.x)
...
```

What does Python give me?

| A: 9 |
| B: 10 |
| C: 1 |
| D: None |
| E: Error | **CORRECT** |

Code shown in lecture was 1+2. Some students were confused because the argument `x` wasn’t used. It wasn’t an error, but we changed the code now to avoid any distraction.
Exercise 2

Module Text

```python
# module.py
def foo(x):
    x = x + 3
    x = 3 * x
y = foo(0)
```

Python Interactive Mode

```python
>>> import module
>>> print(module.y)
...
```

What does Python give me?

- A: 9
- B: 10
- C: 1
- D: None
- E: Error

Code shown in lecture was 1 + 2. Some students were confused because the argument x wasn’t used. It wasn’t an error, but we changed the code now to avoid any distraction.
Exercise 2, Solution

Module Text

```python
# module.py

def foo(x):
    x = x + 3
    x = 3 * x

y = foo(0)
```

Python Interactive Mode

```python
>>> import module
>>> print(module.y)
...
```

What does Python give me?

A: 9
B: 10
C: 1
D: None  CORRECT
E: Error

Code shown in lecture was 1+2. Some students were confused because the argument x wasn’t used. It wasn’t an error, but we changed the code now to avoid any distraction.
Exercise 3

Module Text

```python
# module.py

def foo(x):
    x = x + 3
    x = 3 * x
    return x + 1

y = foo(0)
```

Python Interactive Mode

```python
>>> import module
>>> module.y
... What does Python give me?
```

A: 9
B: 10
C: 1
D: None
E: Error
Exercise 3, Solution

Module Text

```python
# module.py

def foo(x):
    x = x + 3
    x = 3 * x
    return x + 1

y = foo(0)
```

Python Interactive Mode

```python
>>> import module
>>> module.y
...
```

What does Python give me?

A: 9  
B: 10  CORRECT  
C: 1  
D: None  
E: Error

Code shown in lecture was 1+2. Some students were confused because the argument x wasn’t used. It wasn’t an error, but we changed the code now to avoid any distraction.
Exercise 4

Function Definition

```python
def foo(a,b):
    x = a
    y = b
    return x*y+y
```

Function Call

```python
>>> x = 2
>>> foo(3,4)
>>> x
```

What does Python give me?

**A:** 2  
**B:** 3  
**C:** 16  
**D:** None  
**E:** I do not know
## Exercise 4, Solution

### Function Definition

```python
def foo(a, b):
    x = a
    y = b
    return x*y + y
```

### Function Call

```python
>>> x = 2
>>> foo(3, 4)
>>> x
...
```

What does Python give me?

A: 2  **CORRECT**
B: 3
C: 16
D: None
E: I do not know

[Link to tutor](http://cs1110.cs.cornell.edu/tutor/#mode=edit)
Specifications & Testing
Recall the Python API

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

<table>
<thead>
<tr>
<th>Function name</th>
<th>Possible arguments</th>
<th>What the function evaluates to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>math.ceil(x)</code></td>
<td></td>
<td>Return the ceiling of x, the smallest integer greater than or equal to x.</td>
</tr>
</tbody>
</table>

- This is a **specification**
  - How to use the function
  - Not how to implement it
- Write them as **docstrings**
def greet(name):
    """Prints a greeting to person name followed by conversation starter.
    """
    print('Hello ' + name + '!
    print('How are you?')
    print('How are you?

    name: the person to greet
    Precondition: name is a string"""

    Precondition specifies assumptions we make about the arguments
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"

A Precondition Is a Contract

- Precondition is met: The function will work!
- Precondition not met? Sorry, no guarantees...

**Software bugs** occur if:
- Precondition is not documented properly
- Function use violates the precondition

```python
>>> get_campus_num("6072554444")
'5-4444'

>>> get_campus_num("6072531234")
'3-1234'

>>> get_campus_num(6072531234)
Traceback (most recent call last):
  File "/Users/bracy/cornell_phone.py", line 12, in get_campus_num
TypeError: 'int' object is not subscriptable

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

Precondition violated: **error message!**
Precondition violated: **no error message!**
“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...”

Sources: Wikipedia & CNN
In American terms:
Preconditions help assign blame.

Something went wrong.

Did you use the function wrong?

OR

Was the function implemented/specified wrong?
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!
def vowel_count(word):
    """Returns: number of vowels in word.

    word: a string with at least one letter and 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.
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
def last_name_first(full_name):
    """Returns: copy of full_name in form <last-name>, <first-name>
    full_name: has the form <first-name> <last-name>
    with one or more blanks between the two names""
    end_first = full_name.find(' ')  
    first = full_name[:end_first] 
    last = full_name[end_first+1:] 
    return last+', '+first

Representative Tests:
- last_name_first('Katherine Johnson')  Expects: ‘Johnson, Katherine'
- last_name_first('Katherine       Johnson')  Expects: ‘Johnson, Katherine'
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
    space_index = full_name.find(' ')
    #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') gives 'Johnson, Katherine'
last_name_first('Katherine Johnson') gives ' Johnson, Katherine'

Which line is “wrong”?
A: Line 1
B: Line 2
C: Line 3
D: Line 4
E: I do not know
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
    space_index = full_name.find(' ')  
    # 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') gives 'Johnson, Katherine'
- last_name_first('Katherine     Johnson') gives ' Johnson, Katherine'

Which line is “wrong”?
A: Line 1
B: Line 2
C: Line 3  CORRECT
D: Line 4
E: I do not know
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 ?!
**testcase module**

- Contains useful testing functions

- To use:
  - Download from “Schedule” link of course website (one of today’s lecture files)
  - Put in same folder as the files you wish to test
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 `testcase` module (for testing)
  ▪ Defines one or more test cases that each includes:
    • A representative input
    • The expected output
  ▪ Test cases use the `testcase` function:

```python
def assert_equals(expected, received):
    """Quit program if `expected` and `received` differ"""
```
import name # The module we want to test
import testcase # Includes the tests

# First test case
result = name.last_name_first('Katherine Johnson')
testcase.assert_equals('Johnson, Katherine', result)

# Second test case
result = name.last_name_first('Katherine Johnson')
testcase.assert_equals('Johnson, Katherine', result)

print('All tests of the function last_name_first passed')
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)
def test_last_name_first():
    
    """Calls all the tests for last_name_first"""
    print('Testing function last_name_first')
    # Test 1
    result = name.last_name_first('Katherine Johnson')
    testcase.assert_equals('Johnson, Katherine', result)
    # Test 2
    result = name.last_name_first('Katherine Johnson')
    testcase.assert_equals('Johnson, Katherine', result)

    # Execution of the testing code
    test_last_name_first()
    print('All tests of the function last_name_first passed')

Still need to import modules name, testcase

Put all tests inside one function

No tests happen if you forget to call the function