Testing

CSC148, Introduction to Computer Science
Fall 2016
Diane Horton
A beginner’s way to test a function

- Write calls in the shell.
- Read the results and judge whether correct.
- What are the disadvantages of this?
Using doctest

• We have been giving examples in our docstrings.

• A module called doctest lets us run these!

• To run all the doctest examples in your module:

```python
if __name__ == '__main__':
    import doctest
doctest.testmod()
```
Thorough testing

- A function must work on *any* valid call. We can’t test them all.
- But we can make a good argument if we
  - Divide all possible inputs into meaningful categories.
  - Choose a representative test case from each.
- That might lead to 5 or 10 or 20 test cases.
- With doctest, our docstrings would be too long.
- With module `unittest`, code and its tests are in separate modules.
Using unittest

• Define a separate module with the structure used in our examples.

• For each test case:
  – Define a method whose name begins with `test`.
  – In it, set up any variables you need.
  – Call the function you are testing.
  – Use an `assert` to state what must be true if the function ran correctly.
# Common types of assert

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>assertEqual(a, b)</code></td>
<td><code>a == b</code></td>
</tr>
<tr>
<td><code>assertNotEqual(a, b)</code></td>
<td><code>a != b</code></td>
</tr>
<tr>
<td><code>assertTrue(x)</code></td>
<td><code>bool(x)</code> is True</td>
</tr>
<tr>
<td><code>assertFalse(x)</code></td>
<td><code>bool(x)</code> is False</td>
</tr>
<tr>
<td><code>assertIs(a, b)</code></td>
<td><code>a is b</code></td>
</tr>
<tr>
<td><code>assertIsNot(a, b)</code></td>
<td><code>a is not b</code></td>
</tr>
<tr>
<td><code>assertIn(a, b)</code></td>
<td><code>a in b</code></td>
</tr>
<tr>
<td><code>assertNotIn(a, b)</code></td>
<td><code>a not in b</code></td>
</tr>
</tbody>
</table>

A complete list:  
[https://docs.python.org/3/library/unittest.html](https://docs.python.org/3/library/unittest.html)
Thoughts on testing

• Designing test cases before writing code is a best practice in industry.

• It is part of test-driven development.

• When you test code, you must try to break it.
Fixing a bug

• When your testing reveals a bug, what to do?

• Beginners often:
  – Try some “typical” changes, e.g., change “>” to “>=“.
  – Add print statements.

• A rarely done but better strategy:
  – Trace the code on paper.
  – Why is this better?

• A professional strategy:
  – Use the debugger to trace it for you.
  – Use what you learn to hypothesize a fix.
Checking your fix

• Reap the benefit of having defined a thorough test suite with unittest:

  Check your fix with the press of a button!
Professionalism

• We have seen two practices that are expected of any professional:
  – Test-driven development.
  – Using a debugger to find and fix bugs.

• You will hone these skills throughout the course.

• Professionalism is a theme we will revisit.