Testing Functions that Mutate Values

In this lecture, we'll explore how to test functions that mutate objects.
We'll start with a `doctest` example, and then translate it to `unittest`.

The problem: given two lists, remove the items from the first list that appear in the second list.
The algorithm: for each item in the second list, if that item appears in the first list, remove it.
Here is the implementation:

```
def remove_shared(L1, L2):
    """ (list, list) -> NoneType

    Remove items from L1 that are in both L1 and L2.

    >>> list_1 = [1, 2, 3, 4, 5, 6]
    >>> list_2 = [2, 4, 5, 7]
    >>> remove_shared(list_1, list_2)
    >>> list_1
    [1, 3, 6]
    >>> list_2
    [2, 4, 5, 7]
    """

    for v in L2:
        if v in L1:
            L1.remove(v)
```

`doctest`

In the `doctest`, we create two lists that variables `list_1` and `list_2` refer to. Now we call `remove_shared(list1, list2)`.

The function has no return statement; it produces `None`. This means that when we call the function there is no useful return value to examine in our test. Instead, we examine `list_1` and then `list_2` to make sure that the list that `list_1` refers to has been mutated properly, and that the list that `list_2` refers to has not been mutated.

For thorough testing, it's important to verify that any mutable objects passed to a function are not mutated, unless the function description specifies that they should be.

`unittest` version

Each `unittest` starts by importing:

- `unittest`, and
- the module containing the function to be tested.

We need to:

1. create a subclass based on `unittest.TestCase` and name it based on the function we're testing,
2. add a brief `docstring` describing the test case, and
3. name every `unittest` test method with a name starting with "test".

Because our test involves two lists where there are some shared items and some non-shared items, we'll choose `test_general_case` for our name. The `docstring` should describe the test. Remember that if the test fails then this `docstring` is part of the report, so try to make it clear and helpful.

We can start creating variables that refer to the lists that will be used as arguments:

```
list_1 = [1, 2, 3, 4, 5, 6]
list_2 = [2, 4, 5, 7]
```

We specify what we expect the two variables to refer to after the function call has executed:

```
list_1_expected = [1, 3, 6]
list_2_expected = [2, 4, 5, 7]
```

We call the function:

```
duplicates.remove_shared(list_1, list_2)
```

Finally, we assert that the lists refer to the values we expect:

```
self.assertEqual(list_1, list_1_expected)
self.assertEqual(list_2, list_2_expected)
```

The full code appears as follows:

```
import unittest
import duplicates

class TestRemoveShared(unittest.TestCase):
    """Tests for function duplicates.remove_shared."""

    def test_general_case(self):
        """
        Test remove_shared where there are items that
        appear in both lists, and items that appear in
        only one or the other list.
        """

        list_1 = [1, 2, 3, 4, 5, 6]
        list_2 = [2, 4, 5, 7]
        list_1_expected = [1, 3, 6]
        list_2_expected = [2, 4, 5, 7]

        duplicates.remove_shared(list_1, list_2)

        self.assertEqual(list_1, list_1_expected)
        self.assertEqual(list_2, list_2_expected)

if __name__ == '__main__':
    unittest.main(exit=False)
```

This test class contains only one method (the one from the docstring). To thouroghly test `remove_shared`, you would need to choose a set of test cases and write one method for each test.

Jennifer Campbell • Paul Gries
University of Toronto
