Exercise 47: Automated Testing

Having to type commands into your game over and over to make sure it's working is annoying. Wouldn't it be better to write little pieces of code that test your code? Then when you make a change, or add a new thing to your program, you just "run your tests" and the tests make sure things are still working. These automated tests won't catch all your bugs, but they will cut down on the time you spend repeatedly typing and running your code.

Every exercise after this one will not have a *What You Should See* section, but instead will have a *What You Should Test* section. You will be writing automated tests for all of your code starting now, and this will hopefully make you an even better programmer.

I won't try to explain why you should write automated tests. I will only say that you are trying to be a programmer, and programmers automate boring and tedious tasks. Testing a piece of software is definitely boring and tedious, so you might as well write a little bit of code to do it for you.

That should be all the explanation you need because *your* reason for writing unit tests is to make your brain stronger. You have gone through this book writing code to do things. Now you are going to take the next leap and write code that knows about other code you have written. This process of writing a test that runs some code you have written *forces* you to understand clearly what you have just written. It solidifies in your brain exactly what it does and why it works and gives you a new level of attention to detail.

Writing a Test Case

We're going to take a very simple piece of code and write one simple test. We're going to base this little test on a new project from your project skeleton.

First, make a `ex47` project from your project skeleton. Here are the steps you would take. I'm going to give these instructions in English rather than show you how to type them so that *you* have to figure it out.

1. Copy `skeleton` to `ex47`.
2. Rename everything with `NAME` to `ex47`.
3. Change the word `NAME` in all the files to `ex47`.
4. Finally, remove all the `*.pyc` files to make sure you're clean.

Refer back to Exercise 46 if you get stuck, and if you can't do this easily then maybe practice it a few times.

> 🪄 **NOTE**
>
> Remember that you run the command `nosetests` to run the tests. You can run them with `python ex47_tests.py` but it won't work as

easily and you'll have to do it for each test file.

Next, create a simple file `ex47/game.py` where you can put the code to test.
This will be a very silly little class that we want to test with this code in it:

```
 1   class Room(object):
 2
 3       def __init__(self, name, description):
 4           self.name = name
 5           self.description = description
 6           self.paths = {}
 7
 8       def go(self, direction):
 9           return self.paths.get(direction, None)
10
11       def add_paths(self, paths):
12           self.paths.update(paths)
```

Once you have that file, change the unit test skeleton to this:

```
 1   from nose.tools import *
 2   from ex47.game import Room
 3
 4
 5   def test_room():
 6       gold = Room("GoldRoom",
 7                   """This room has gold in it you can grab. There's a
 8                   door to the north.""")
 9       assert_equal(gold.name, "GoldRoom")
10       assert_equal(gold.paths, {})
11
12   def test_room_paths():
13       center = Room("Center", "Test room in the center.")
14       north = Room("North", "Test room in the north.")
15       south = Room("South", "Test room in the south.")
16
17       center.add_paths({'north': north, 'south': south})
18       assert_equal(center.go('north'), north)
19       assert_equal(center.go('south'), south)
20
21   def test_map():
22       start = Room("Start", "You can go west and down a hole.")
23       west = Room("Trees", "There are trees here, you can go east.")
24       down = Room("Dungeon", "It's dark down here, you can go up.")
25
26       start.add_paths({'west': west, 'down': down})
27       west.add_paths({'east': start})
28       down.add_paths({'up': start})
29
30       assert_equal(start.go('west'), west)
31       assert_equal(start.go('west').go('east'), start)
32       assert_equal(start.go('down').go('up'), start)
```

This file imports the `Room` class you made in the `ex47.game` module so that
you can do tests on it. There is then a set of tests that are functions starting
with `test_`. Inside each test case there's a bit of code that makes a room or
a set of rooms, and then makes sure the rooms work the way you expect
them to work. It tests out the basic room features, then the paths, then tries
out a whole map.

The important functions here are `assert_equal` which makes sure that
variables you have set or paths you have built in a `Room` are actually what

you think they are. If you get the wrong result, then `nosetests` will print out an error message so you can go figure it out.

Testing Guidelines

Follow this general loose set of guidelines when making your tests:

1. Test files go in `tests/` and are named `BLAH_tests.py` otherwise `nosetests` won't run them. This also keeps your tests from clashing with your other code.
2. Write one test file for each module you make.
3. Keep your test cases (functions) short, but do not worry if they are a bit messy. Test cases are usually kind of messy.
4. Even though test cases are messy, try to keep them clean and remove any repetitive code you can. Create helper functions that get rid of duplicate code. You will thank me later when you make a change and then have to change your tests. Duplicated code will make changing your tests more difficult.
5. Finally, do not get too attached to your tests. Sometimes, the best way to redesign something is to just delete it and start over.

What You Should See

```
$ nosetests
...
----------------------------------------------------------------------
Ran 3 tests in 0.008s

OK
```

That's what you should see if everything is working right. Try causing an error to see what that looks like and then fix it.

Study Drills

1. Go read about nosetests more, and also read about alternatives.
2. Learn about Python's "doc tests" and see if you like them better.
3. Make your room more advanced, and then use it to rebuild your game yet again but this time, unit test as you go.

Common Student Questions

Q: I get a syntax error when I run `nosetests`.

If you get that then look at what the error says and fix that line of code or the ones above it. Tools like `nosetests` are running your code and the test code, so they will find syntax errors the same as running Python will.

Q: I can't import `ex47.game`?

Make sure you create the `ex47/__init__.py` file. Refer to Exercise 46 again to see how it's done. If that's not the problem then do this on OSX/Linux:

```
export PYTHONPATH=.
```

And on Windows:

```
$env:PYTHONPATH = "$env:PYTHONPATH;."
```

Finally, make sure you're running ther tests with `nosetests` not with just Python.

Q: I get `UserWarning` when I run `nosetests`.

You probably have two versions of Python installed or you aren't using distribute. Go back and install `distribute` or `pip` as I describe in Exercise 46.

Video

Purchase The Videos For $29.59

For just $29.59 you can get access to all the videos for Learn Python The Hard Way, **plus** a PDF of the book and no more popups all in this one location. For $29.59 you get:

- All 52 videos, 1 per exercise, almost 2G of video.
- A PDF of the book.
- Email help from the author.
- See a list of everything you get before you buy.

When you buy the videos they will immediately show up **right here** without any hassles.

Already Paid? Reactivate Your Purchase Right Now!

Buying Is Easy

Buying is easy. Just fill out the form below and we'll get started.

Full Name
Full Name

Email Address
Email Address

◉ VISA MasterCard AMERICAN EXPRESS   ○ PayPal Use
Pay With Credit Card   your PayPal™
(by Stripe™)   account.

Buy Learn Python The Hard Way, 3rd Edition

Zed Shaw	Amazon	Amazon	B&N	B&N
PDF + Videos + Updates	Paper + DVD	Kindle	Paper + DVD	Nook (No Video)
$29.95	$22.74	$17.27	$22.96	$17.27
	InformIT	Interested In Ruby?		
	eBook + Paper	Ruby is also a great language.		
	$43.19	Learn Ruby The Hard Way		