

## Nose 2 Tutorial

### **Background:**

Nose2 is a python library used for testing. Like it's predecessor Nose, Nose2 is at it's core just unittest with plugins. Nose2's purpose is to extend unittest to make testing nicer and easier to understand.

### **When to Use:**

The major competitor to nose2 is pytest. A major advantage of nose 2 is that it allows tests to be regular functions. We don't need to create a class and inherit it from any base class. As long as the function starts with the word test, it is considered a test and executed. On top of this, nose2 comes with many plugins by default and even allows users to download third-party plugins.

### **Installation:**

To get started with nose2 like many other libraries it is best to use the package manager pip and run the command:

```
pip install nose2
```

Your output will look similar to this:

```
C:\Users\tanma>pip install nose2
Collecting nose2
  Downloading nose2-0.9.2-py2.py3-none-any.whl (137 kB)
    | 137 kB 2.2 MB/s
Requirement already satisfied: coverage>=4.4.1 in c:\users\tanma\appdata\local\programs\python\python38-32\lib\site-packages (from nose2) (5.3)
Requirement already satisfied: six>=1.7 in c:\users\tanma\appdata\local\programs\python\python38-32\lib\site-packages (from nose2) (1.15.0)
Installing collected packages: nose2
Successfully installed nose2-0.9.2
```

### **Running Tests:**

To run tests make use of the nose2 command inside of a directory with unit tests or test files that have lowercase "test" at the front of the name. The command to run all unit tests and individual test files is:

```
nose2
```

### **Example:**

Through this step-by-step tutorial, we will help you familiarize yourself with nose2 and understand which scenarios to use it, and where there are better alternatives.

One of the helpful plugins provided is parameterization. Using nose2, you can make a test function or method parameterized by parameters.

For this example, begin by creating a python file called "test\_params"

The top of your file should include these two import statements:

```
import unittest

from nose2.tools import params
```

Next, we will start by creating a simple parameterized test. In your file, put “@params(x,y,z)”  
These variables will serve as the parameters for our test. The next line should have a method,  
let’s use a simple one that tells us if a number is even.

```
def test_even(num):
    assert (num % 2) == 0
```

Now, change the parameters to be whichever numbers you wish to test, and test\_even will test all 3. Go to your source directory and use the “nose 2” command and you should get an output like this:

```
C:\Users\tanma\Documents\GitHub\nose2tutorial>nose2
...
-----
Ran 3 tests in 0.001s

OK
```

If you change your parameters to be odd, the tests will fail, and your output will be similar to this:

```
-----
Ran 1 test in 0.001s

FAILED (errors=1)

C:\Users\tanma\Documents\GitHub\nose2tutorial>nose2
F.F...
=====
FAIL: test_params.test_even:1
1
-----
Traceback (most recent call last):
  File "C:\Users\tanma\Documents\GitHub\nose2tutorial\test_params.py", line 7, in test_even
    assert (num % 2) == 0
AssertionError

=====
FAIL: test_params.test_even:3
3
-----
Traceback (most recent call last):
  File "C:\Users\tanma\Documents\GitHub\nose2tutorial\test_params.py", line 7, in test_even
    assert (num % 2) == 0
AssertionError
```

Parameters can also be used with methods, such as the example below. Parameters can be tuples at most, but you can have as many parameters as you wish.

```
class Test(unittest.TestCase):

    @params((1, 2), (2, 3), (4, 5))
    def test_less_than(self, a, b):
        assert a < b
```

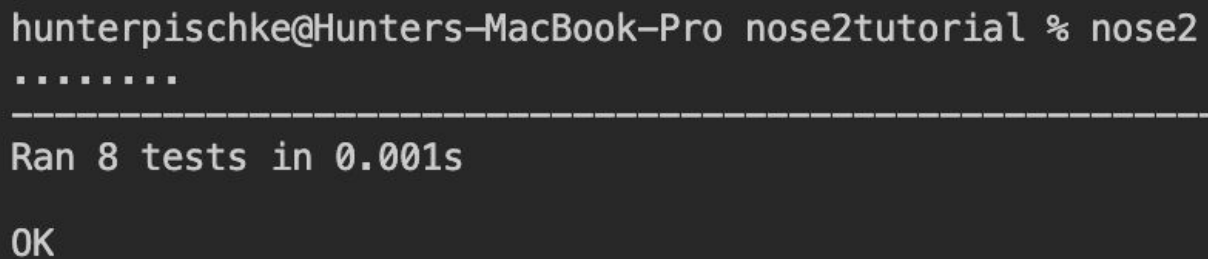
Add the code mentioned above to your file and run the nose2 command. You should be running 6 total tests now.

Another helpful plugin of nose2 is test generation. Looking at the test\_generator.py file we can see that you can write separate test methods and use the yield command to run another function.

```
def test_odds():
    For i in range(2, 10):
        yield check_odd, i

def check_odd(n):
    assert (n*n)-1 % 2 != 0
```

This feature allows for modularity. Looking at the results of running nose2 in a directory with only the test\_generator.py file we see the tests pass after it is run.



```
hunterpischke@Hunters-MacBook-Pro nose2tutorial % nose2
.....
-----
Ran 8 tests in 0.001s

OK
```

#### Source Files:

<https://github.com/HunterPischke/nose2tutorial>