

Python Testing - Nose2

Using the Nose2 framework

What is nose2

What is nose2?

nose

is nicer testing for python

- Nose2 is a Python Testing Framework that is positioned to be an extension of unittest.
- Much like the relationship of JavaScript to TypeScript. Nose2 is also a superset of unittest library with new features of its own

Key Features of nose2

- Discovery of test modules and functions.
- Support for plugins to extend functionality.
- 100% interop + integration with existing unittest tests.
- Additional tools for more efficient testing.
- If default/non-loaded plugins don't have what you need, can install external ones

nose2 vs unittest

- **Ease of Use:** This category goes to Nose2 this is mostly because features like auto-test discovery allow for a much more user friendly experience
- **Extensibility:** Since nose2 introduces the concept of plugins which can be used in test, it allows for much more extensibility than unittest.
- **Test Reports:** Nose2 allows for extensive types of reporting through plugins or innate features. For example code coverage

Installation

Overview

This guide will assume a beginner with none of the dependencies installed and will walk you through the process:

1. Installing Python
2. Installing virtualenv (optional but recommended)
3. Installing nose2
4. Installing nose2 plugins

Python for Windows

Firstly open this webpage in any browser of your choice:

<https://www.python.org/downloads/windows/>

From there select the latest stable release of python Download the appropriate installer and go through the installation steps

Make sure to select the Add python.exe to PATH checkbox

For optional features make sure to select all of them

Finally you can follow this link to set the ENV variables needed:

<https://docs.python.org/3/using/windows.html>

To verify python installation run: **python --version** or **python3 --version**



Python for MacOS

1. The first way is if you have the macos package manager “homebrew” installed. If so you can simply run:

brew install python

2. Second method is to do it manually. For the step go to: <https://www.python.org/downloads/macos/>
Select the latest stable release.

Open the installer and follow the steps and you should be good to go

To verify python installation run: **python --version** or **python3 --version**

Virtualenv (OPTIONAL)

Virtualenv is a way to setup python environments specific to only your given project. This means that you do not need to install packages globally on your computer, but rather it is in an isolated environment from project to project.

1. Install: `pip install virtualenv`
2. Usage:
 - a. `cd /my/python/project/directory`
 - b. `virtualenv venv` (initial setup of env)
 - c. `source venv/bin/activate`
 - d. `pip install <package_name>`

Installing nose2

Once you have successfully completed and verified the installation of Python.

Installing nose2 will be simple

1. Run the command: **pip install nose2** or `pip3 install nose2`
2. Restart your terminal / command prompt
3. Verify installation: **nose2 --version**

nose2 plugins

nose2 has a plethora of plugins you can install. But for this tutorial these are the only plugin(s) you will need to install

1. **pip install nose2[coverage_plugin] or pip3 install nose2[coverage_plugin]**

- a. Sometimes you will get errors saying plugin not found. If this occurs just wrap the nose2[coverage_plugin] in single quotes like this: pip install **'nose2[coverage_plugin]'**

Basic Usage

- Create test cases as subclasses of `unittest.TestCase`.
- Run tests using the command: `nose2`.
- Test discovery based on `unittest`'s conventions.

Test Discovery Rules

- Naming conventions (e.g., files starting with test).
- Directory structure (tests in tests/ directory or similar).

Using Plugins

- Coverage report plugin.
- Failure detail plugin.
- Profiling and randomization plugins.

Configuring nose2

- Using unittest.cfg for custom configurations.
- Command-line options for on-the-fly configurations.


Best Practices

- Organizing tests logically.
- Keeping tests independent and isolated.
- Writing clear and concise test cases.

Code Demo

Step 1:

Create any set of new directories as you please to store your test files



```

  ✓ test_code
    | test_mod.py
  ✓ test_code_2/test_code_3
    | test_mod_2.py

```

The screenshot shows a file explorer interface with a dark background. It displays a directory tree. The root directory is 'test_code', which contains a file 'test_mod.py'. Below it is a subdirectory 'test_code_2/test_code_3', which contains a file 'test_mod_2.py'. The file 'test_mod_2.py' has a red dot next to it, indicating it is selected or has a change. The number '5' is visible next to the file name.

Code Demo

Step 2:

- Create a new file in the testing directory, the file must start with 'test_'
- Create a simple test case

```
class TestTicTacToeTwo(unittest.TestCase):  
    def setUp(self):  
        # Set up an empty board for each test  
        self.board = [" "] * 10 # assuming the first index [0] is not used  
  
    def test_place_mark(self):  
        self.board = [" "] * 10  
        self.avail = [" "] * 10  
        for i in range(1, 10):  
            self.avail[i] = str(i)  
            place_marker(self.board, self.avail, "X", 1)  
        assert self.board[1] == "X"
```

Code Demo

Step 3:

- To run the tests, you must run the command “nose2” in the terminal
- This works as the command nose2 runs all testing files that start with ‘test_’

```
● > nose2
```

```
.....
```

```
-----  
Ran 7 tests in 0.000s
```

```
OK
```

```
○  🍏 > 📁 ~/Doc/Co/F23/T/4150groupproject > 🐙 🦉 zaid/uz1 = !1
```

Code Demo

Step 4:

Nose2 will then produce the results of the tests in terminal. You can also use the `--use-coverage` flag to show the overall code coverage

```
● > nose2 --with-coverage
```

```
.....
```

```
-----  
Ran 7 tests in 0.000s
```

```
OK
```

Name	Stmts	Miss	Cover
test_code/test_mod.py	22	1	95%
test_code_2/test_code_3/test_mod_2.py	28	1	96%
tic_tac_toe_ai.py	176	151	14%
TOTAL	226	153	32%