

---

# Unit testing

Best practices to unittest your Python software

---

# Basic ideas

A unit test is a smaller test, one that checks that a single component operates in the right way.

A unit test helps you to isolate what is broken in your application and fix it faster.

—

# How to write a good unit test?

1

A testing unit should focus on one tiny bit of functionality and prove it correct.

2

Each test unit must be fully independent.

### 3

Try hard to make tests that run fast.

### 4

Use long and descriptive names for testing functions.

5

Maintain them.

6

Use them.

—

**Okay, but what do I have to do?**

1

Each test class should start with “Test” prefix. This also applies for its methods.

2

Use setUp() to init your class to be tested



### 3

Take a good look on all assert methods that unittest framework already provides ([here](#))

### 4

@mock.patch.object is your friend. Use it whenever you need to mock a dependency

## 5 (bonus)

Use “coverage” package to keep track of your code coverage (and see how can you improve)

## 6 (bonus x2)

Use “pre-commit” package to easily create and manage git-hooks ([here](#))



# Good luck!

For source-code:

<https://github.com/nnelas/python-unit-testing>

## References

<https://realpython.com/python-testing/>

<https://docs.python-guide.org/writing/tests/>

<https://docs.python.org/3/library/unittest.html>