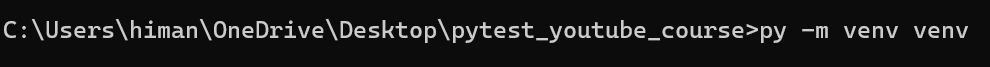
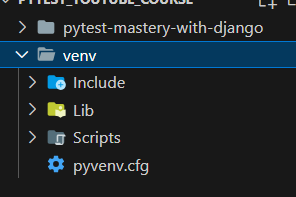


**Introduction**:

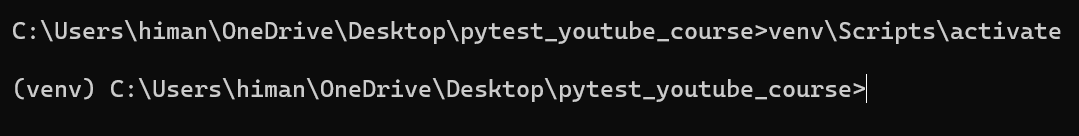
This is a practical natured tutorial where we will test our Django project using pytest testing framework.

Let’s create a virtual environment first  
py -m venv venv

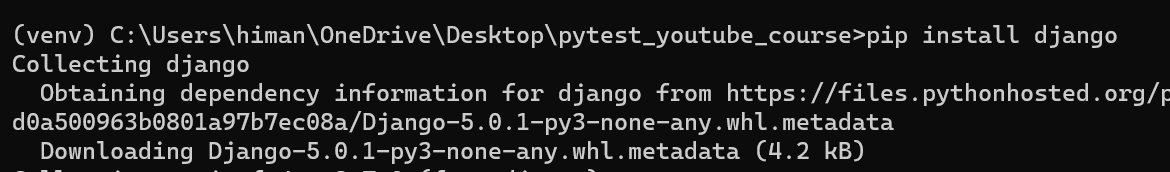


****

Now let’s activate this virtual environment  
venv\Scripts\activate

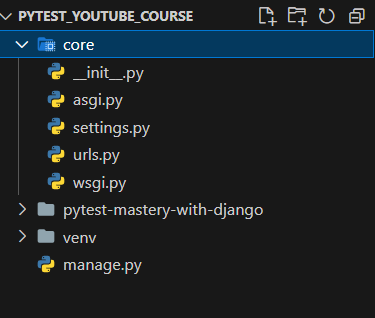


Now we can pip install django once inside this virtual environment.



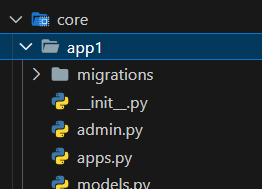
Now we will create a new project called *core*.

django-admin startproject core .



Next we will create a new app called app1 in our project.

So cd into core project and then  
django-admin startapp app1



Also add this app into the list of installed apps in the settings module of our core project.

INSTALLED\_APPS = [

    "django.contrib.admin",

    "django.contrib.auth",

    "django.contrib.contenttypes",

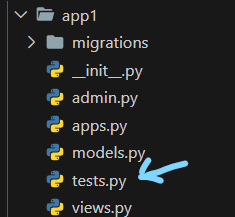
    "django.contrib.sessions",

    "django.contrib.messages",

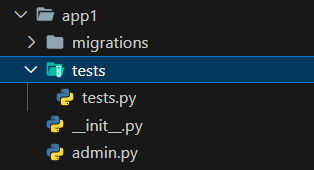
    "django.contrib.staticfiles",

    "app1", 🡪 *Here*

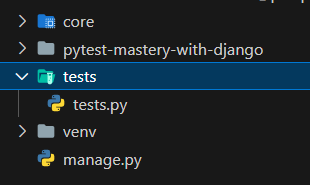
]

This app1 has tests.py module by default…  


But in some structures we create a separate directory called tests inside our app directory and put our individual test files in it.

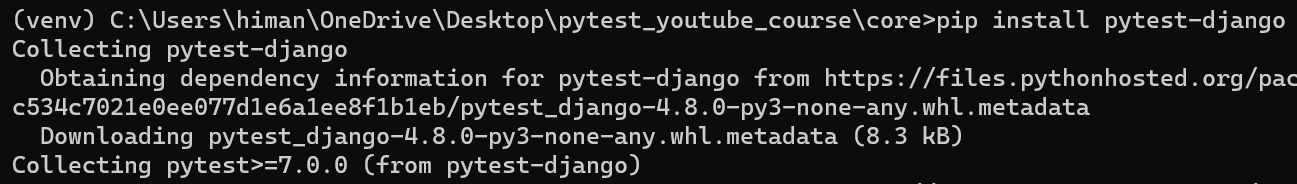


Or we can have a separate folder called tests in the root directory which might be a way of collecting all the tests from all the apps in one place.

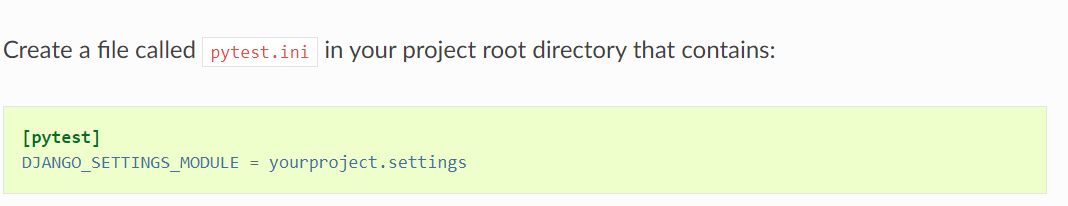
For this tutorial we will keep the tests outside the app or project and in the root directory.  


Next we install *pytest – django*.

pip install pytest-django



Now we need to setup pytest-django in our project,

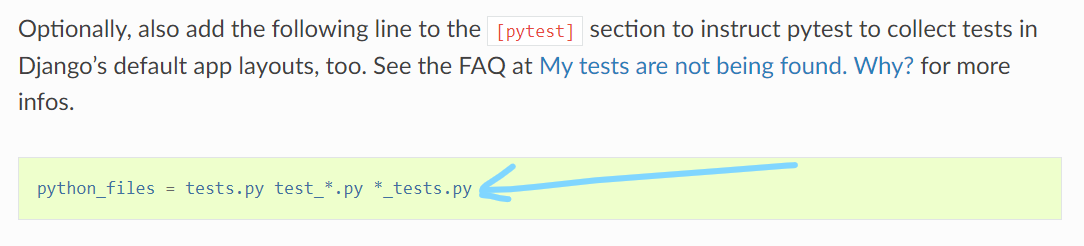


So we will create a pytest.ini file in our root and configure our project setting.

[pytest]

DJANGO\_SETTINGS\_MODULE = core.settings

Next we need to define from where pytest collects tests.



[pytest]

DJANGO\_SETTINGS\_MODULE = core.settings

python\_files = test\_\*.py 🡪 *we will use this convention*

So file names like test\_ex1.py as an example of this.



Let’s write a simple test in this file.

import pytest

def test\_example():

    assert 1 == 1

To run the test from terminal , just run *pytest* command.

Error #1:

ModuleNotFoundError: No module named 'app1'

This is happening because our app1 is inside our core project, so we need to define the path of this app as follows.

In settings module,  
INSTALLED\_APPS = [

    "django.contrib.admin",

    "django.contrib.auth",

    "django.contrib.contenttypes",

    "django.contrib.sessions",

    "django.contrib.messages",

    "django.contrib.staticfiles",

    "core.app1", *🡪 change it here*

]

In apps.py

from django.apps import AppConfig

class App1Config(AppConfig):

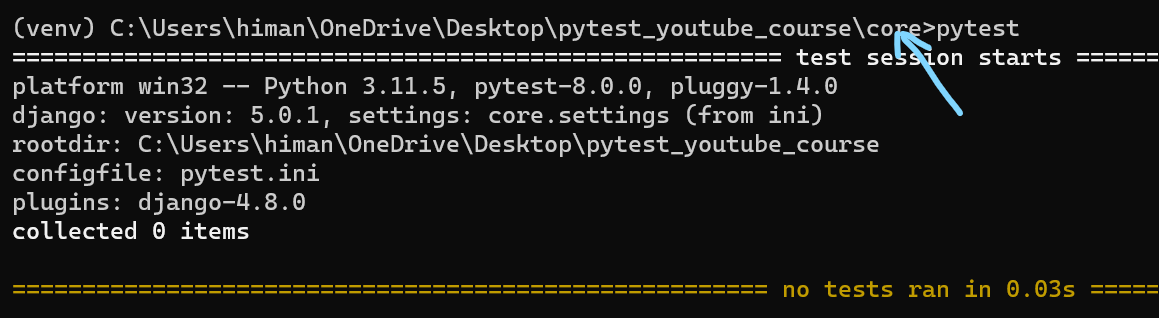
    default\_auto\_field = "django.db.models.BigAutoField"

    name = "core.app1" *🡪 Here*

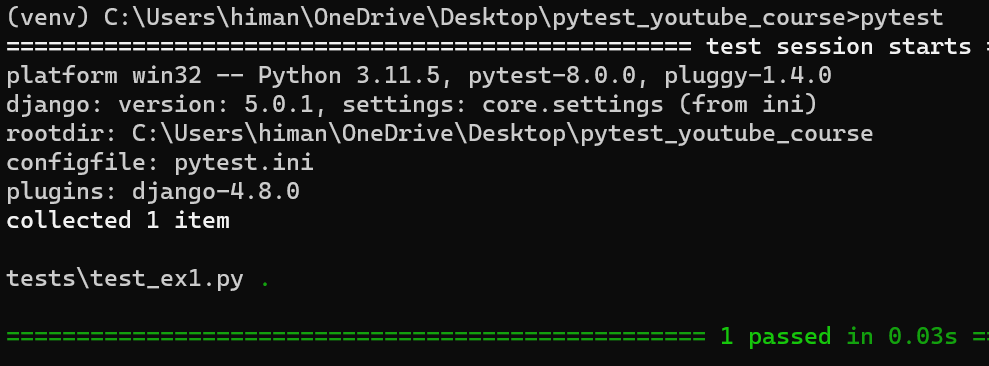
We solved this error, next

Error #2: no tests ran

This is happening because we are trying to run tests from wrong directory.



We need to come out of this directory and move inside root directory.



Finally our tests run and pass.

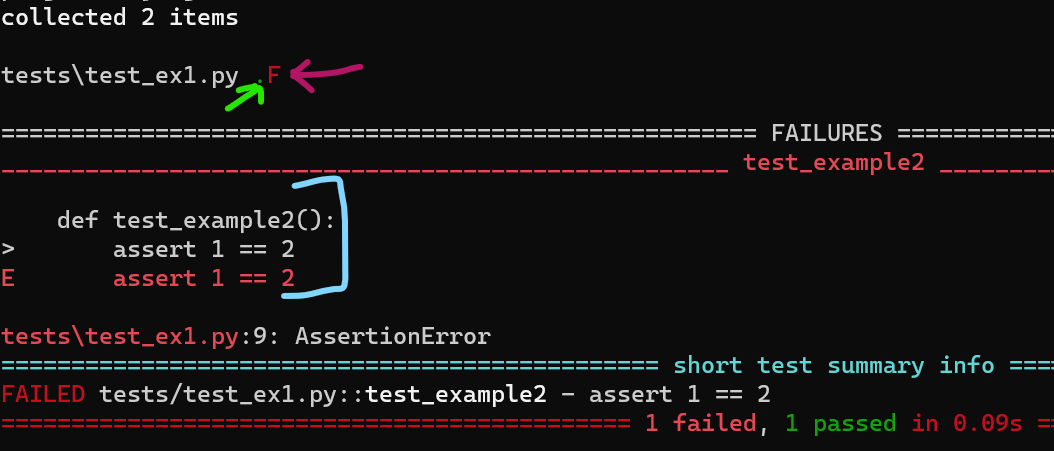
def test\_example():

    assert 1 == 1

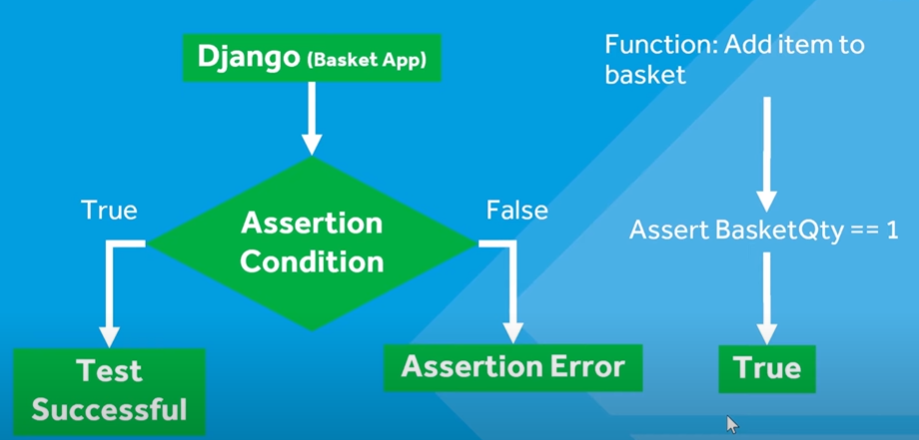
def test\_example2():

    assert 1 == 2

We create one more test that will fail.



Let’s learn a little bit about the *assert* statement here.



Python has a built in assert statement to use assertion conditions. *Pytest is just extending the output from assertion error and providing us more information to help us understand where the test are actually failing*.

*Some useful command line arguments*:

1. If we want pytest to stop test execution as soon as a test fails.

pytest -x

1. If we want to see any print statements in the test

pytest -rP

def test\_example():

    print("test1")

    assert 1 == 2



1. To run a specific test out of all tests,

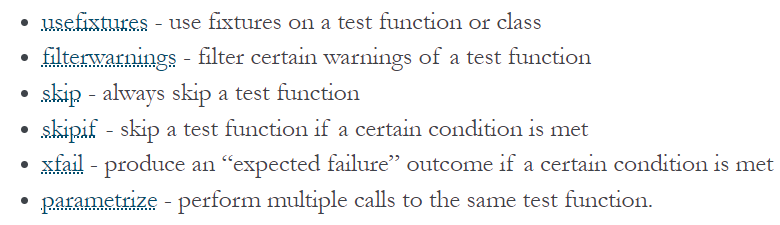
pytest .\tests\test\_ex1.py::test\_example2

Locate the test directory 🡪 test file 🡪 test name

*What are pytest marks*?

A mark is basically a way of adding meta data to a test. Pytest has some built in marks which we can utilize or we can build our own.

[How to mark test functions with attributes — pytest documentation](https://docs.pytest.org/en/stable/how-to/mark.html#mark)



These are built – in marks.

Let’s use the *skip* mark,

import pytest

@pytest.mark.skip

def test\_example():

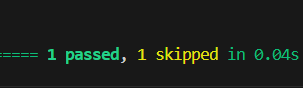
    print("test1")

    assert 1 == 2

def test\_example2():

    assert 1 == 1

We can see one test is skipped



We can run our own markers in the pytest.ini file

[pytest]

DJANGO\_SETTINGS\_MODULE = core.settings

python\_files = test\_\*.py

markers =

        slow: slow running test 🡪 *Here*

Now we have an option with a pytest to identify these markers and only run these particular tests.

@pytest.mark.slow

def test\_example():

    print("test1")

    assert 1 == 2

def test\_example2():

    assert 1 == 1

pytest -m ‘slow’

