Before final evaluation, following tests have been conducted on the software, and discrepancies have been dealt with accordingly. Although not much changes have been made in the engine, due to it being able to function properly.

TESTS CARRIED OUT:

1. Functional level **setUp() and teardown() method:**

**Carried out at functional level to do away with runtime errors and exceptions, as well as to have a fair idea how well the code is working, or wheather it is working at all.**

1. Unit test:

[**unittest**](http://docs.python.org/library/unittest.html#module-unittest) is the batteries-included test module in the Python standard library. Its API will be familiar to anyone who has used any of the JUnit/nUnit/CppUnit series of tools.

Underlying algorithm:

**import** unittest

**def** fun**(**x**):**

**return** x + 1

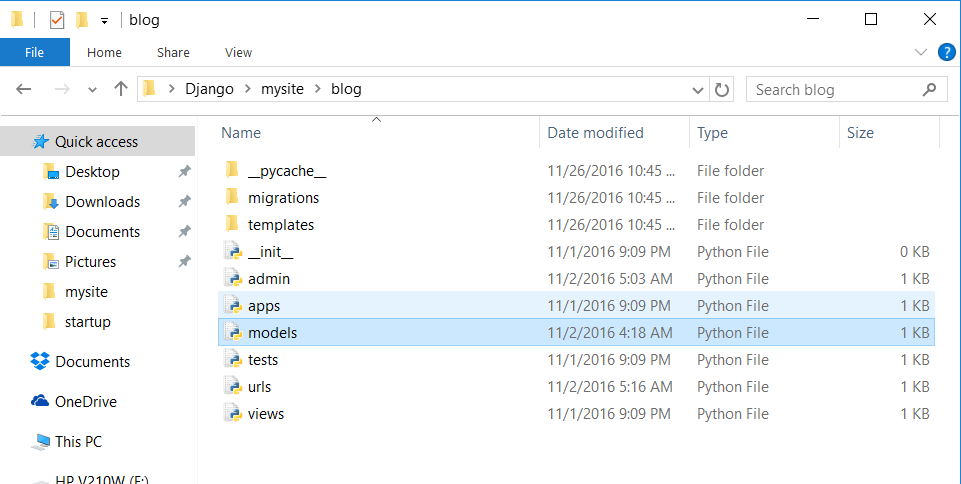
**class** MyTest**(**unittest.TestCase**):**

**def** test**(**self**):**

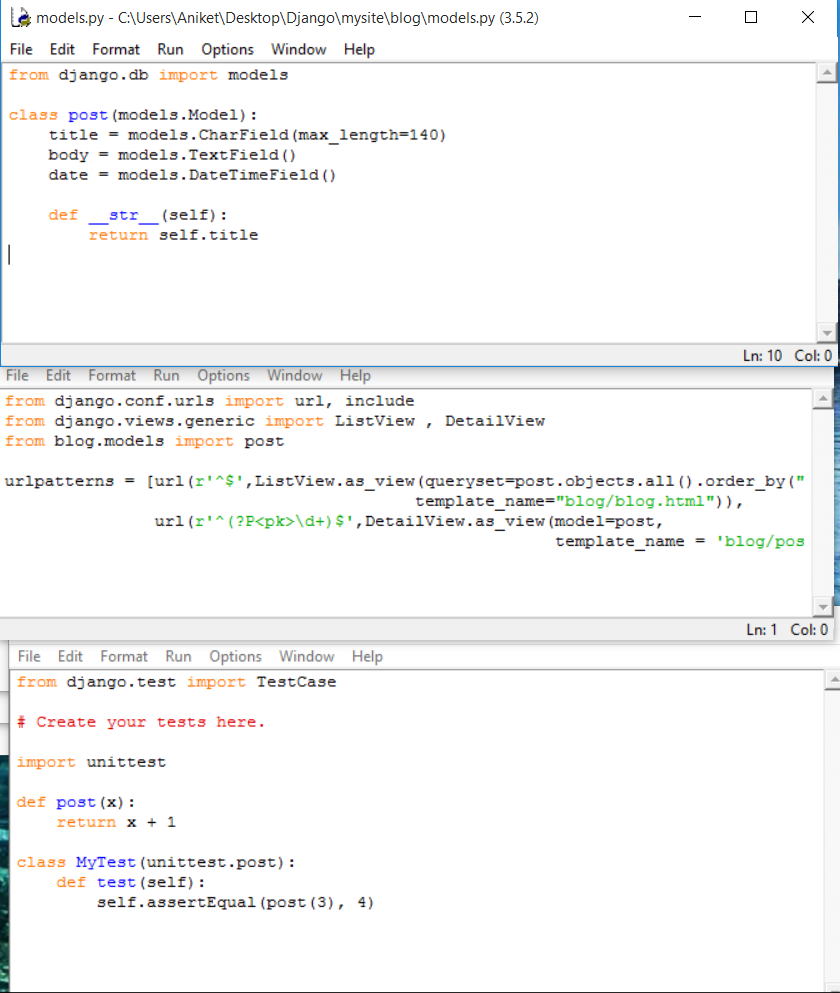
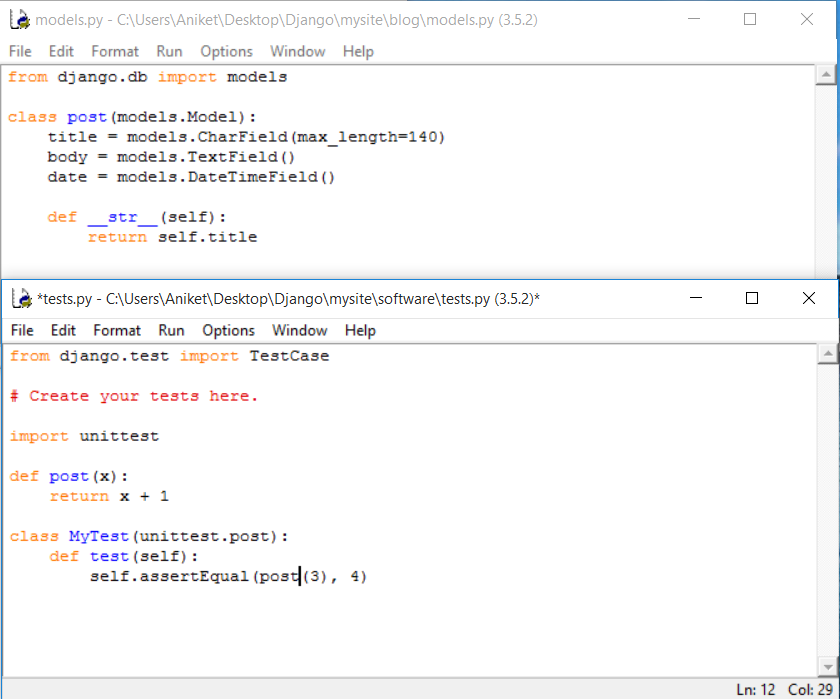
self.assertEqual**(**fun**(**3**),** 4**)**

1. Testing blog module:

Test directory



Test 1



Outcome: successful

### 2) Doctest:

The **[doctest](http://docs.python.org/library/doctest.html" \l "module-doctest" \o "(in Python v2.7))** module searches for pieces of text that look like interactive Python sessions in doc strings, and then executes those sessions to verify that they work exactly as shown.

Doctests have a different use case than proper unit tests: they are usually less detailed and don’t catch special cases or obscure regression bugs. They are useful as an expressive documentation of the main use cases of a module and its components.

ALGORITHM FOR DOCTEST:

**def** square**(**x**):**

*"""Return the square of x.*

*>>> square(2)*

*4*

*>>> square(-2)*

*4*

*"""*

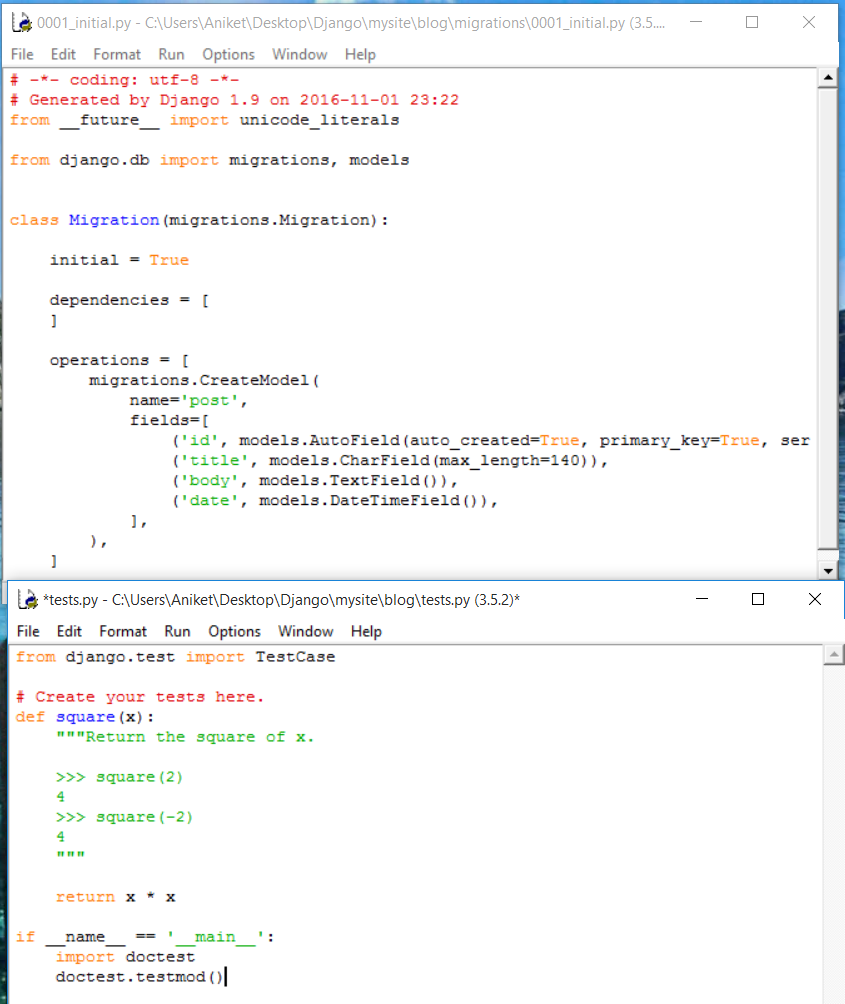
**return** x \* x

**if** \_\_name\_\_ == '\_\_main\_\_'**:**

**import** doctest

doctest.testmod**()**

Test 1 # For the migration code:



OUTCOME: successful

1. Py Test:

Is an alternate to python’s standard unit test module.

Testing algorithm :

**def** func**(**x**):**

**return** x + 1

**def** test\_answer**():**

**assert** func**(**3**)** == 5

OUTCOME: unsuccesfull

RECOMMENDED UPDATES IN DATA BASE APPLIED.

Added:

url embedding , to database file directly

Result:

Quicker Response Speed

urlpatterns = [

url(r'^admin/', admin.site.urls),

url(r'^', include('software.urls')),

url(r'^login/auth/', include('user.urls')),

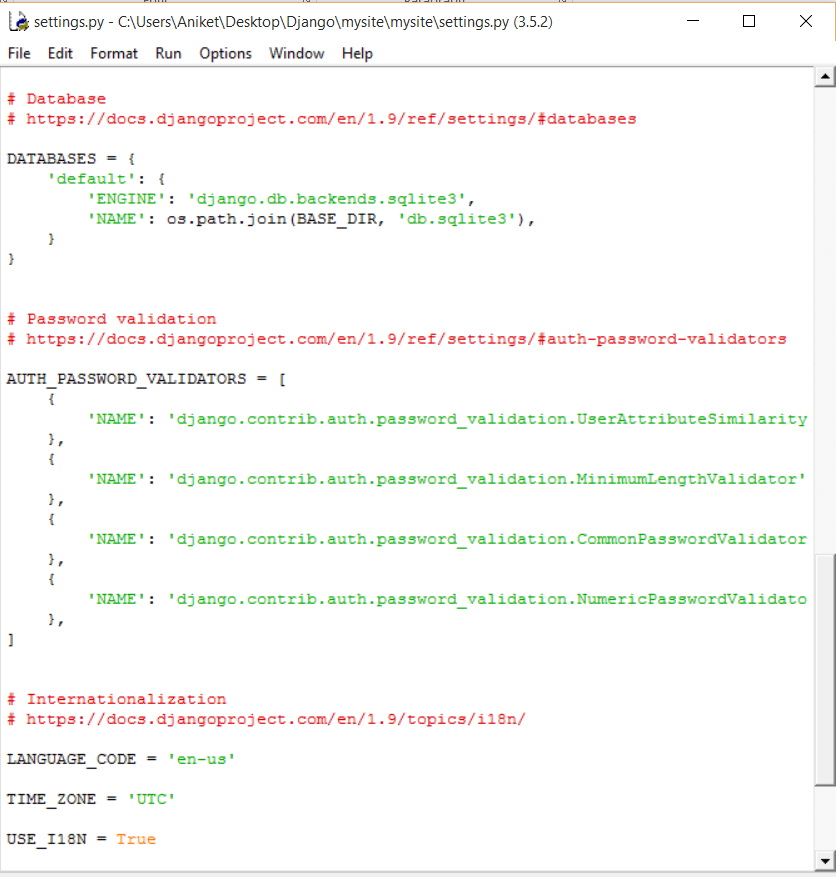
url(r'^profile/', include('Profile.urls')),

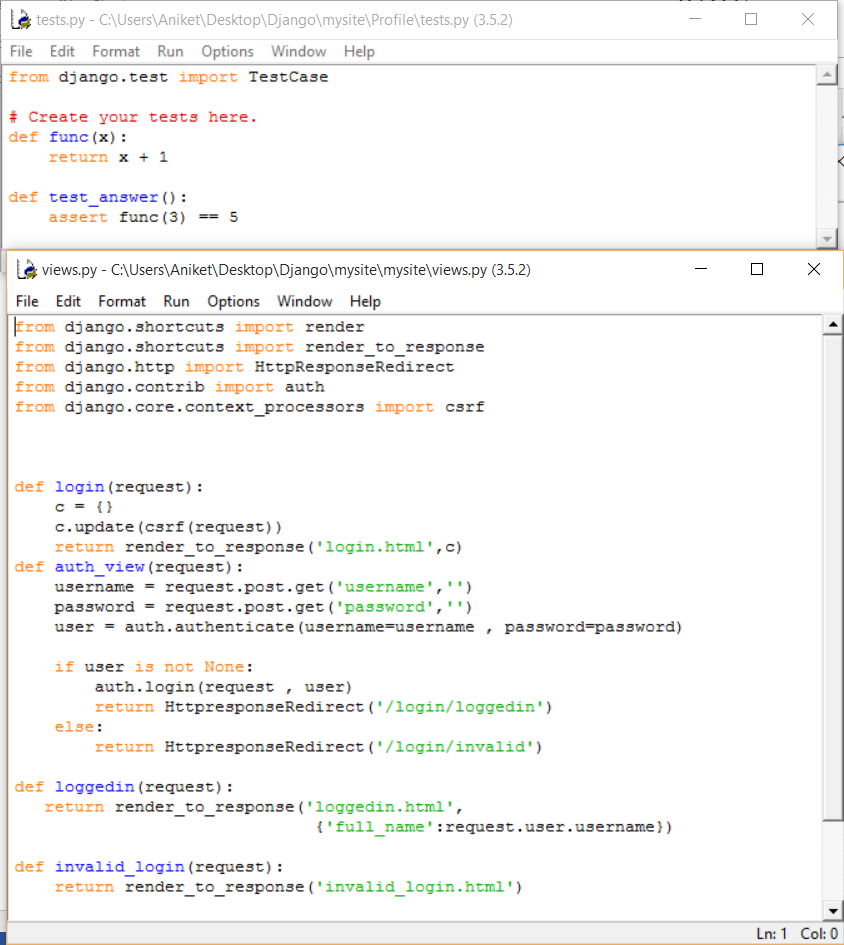
url(r'^startup/', include('startup.urls')),

url(r'^blog/', include('blog.urls')),

url(r'^info/', include('interview.urls')),

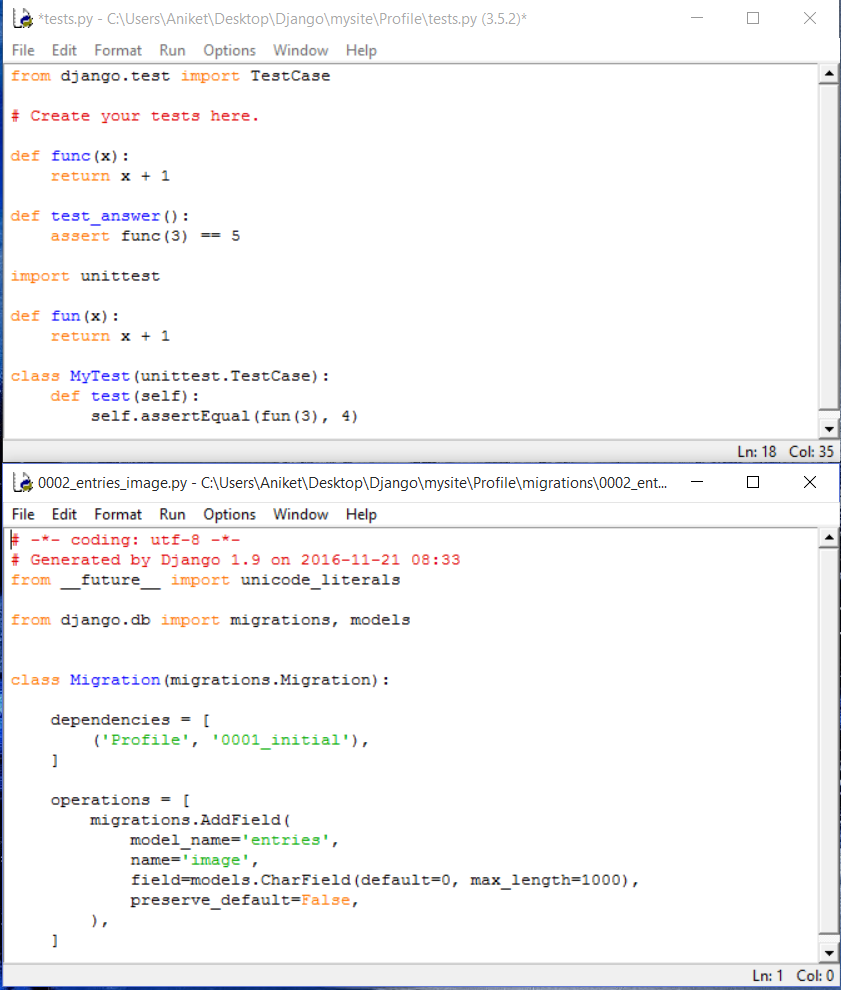
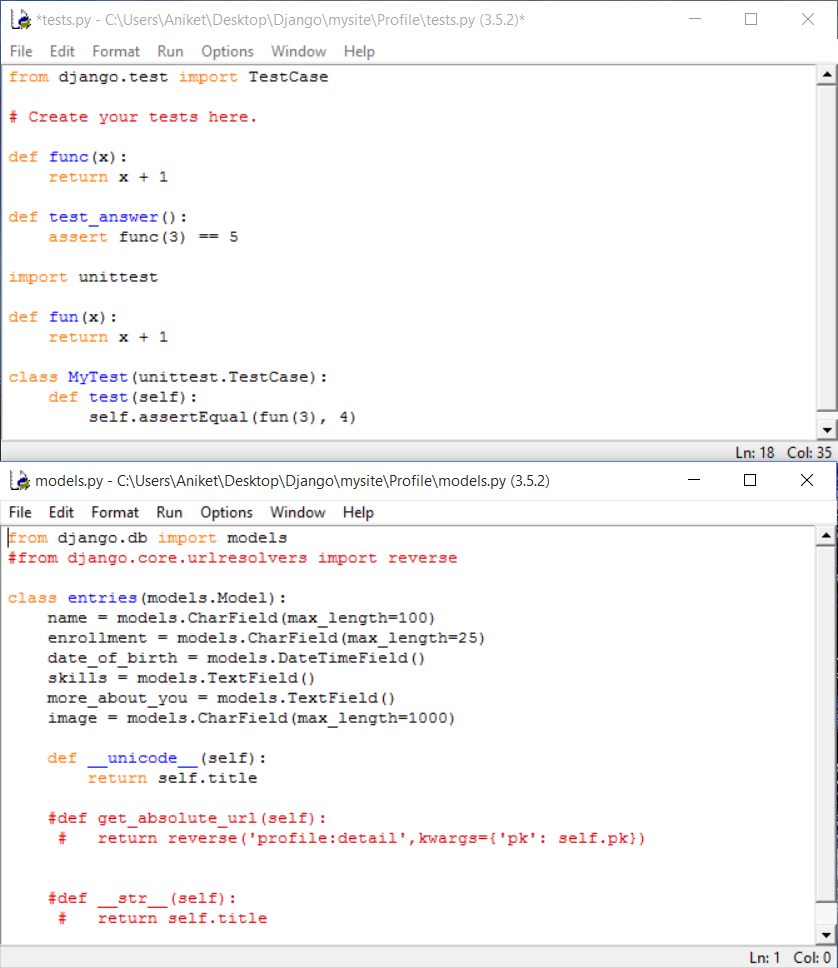
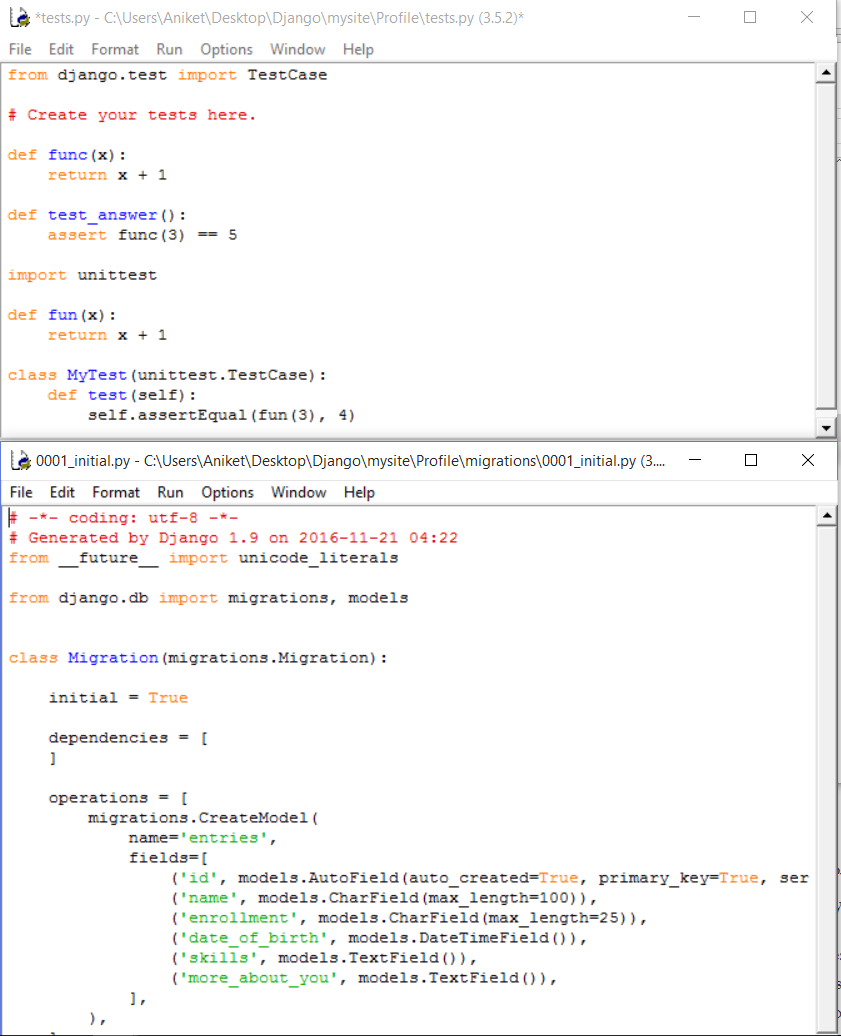
url(r'^login/', include('login.urls')),





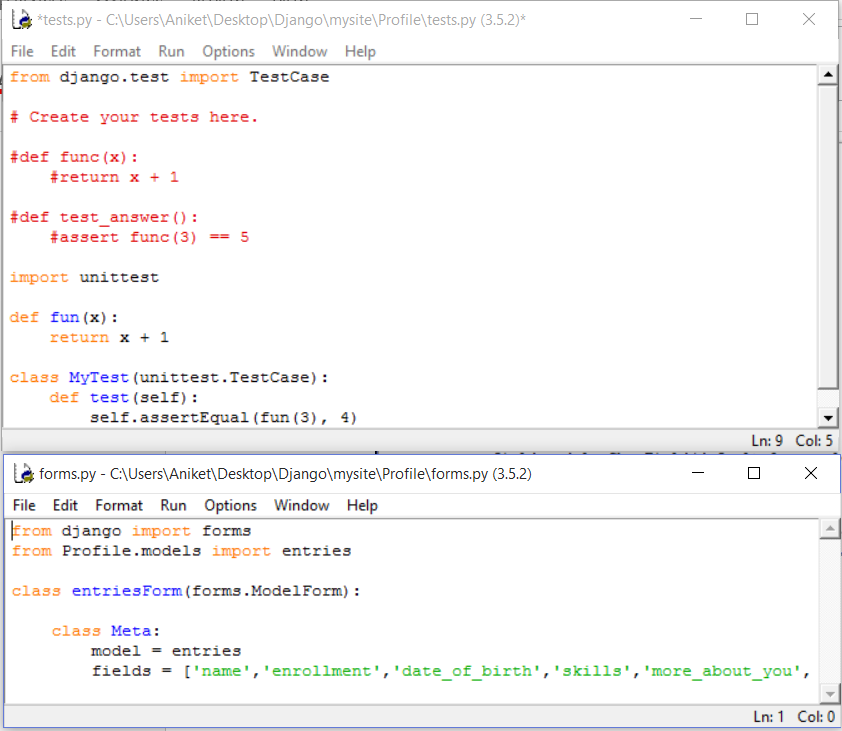
**Additional Testing of Peripheral Modules:**

1. **Testing Profile Module:**
2. **Tests applied : Unit test and Py test:**



1. **Test applied for forms specific input field: Unit Test only**

To avoid the larger probability of runtime errors arising specifically out of the executional stack of processes, it is highly expected that a py test would in any condition have a negative outcome , since the list of names (stack) may underflow upon de-queuing the first name/profile .



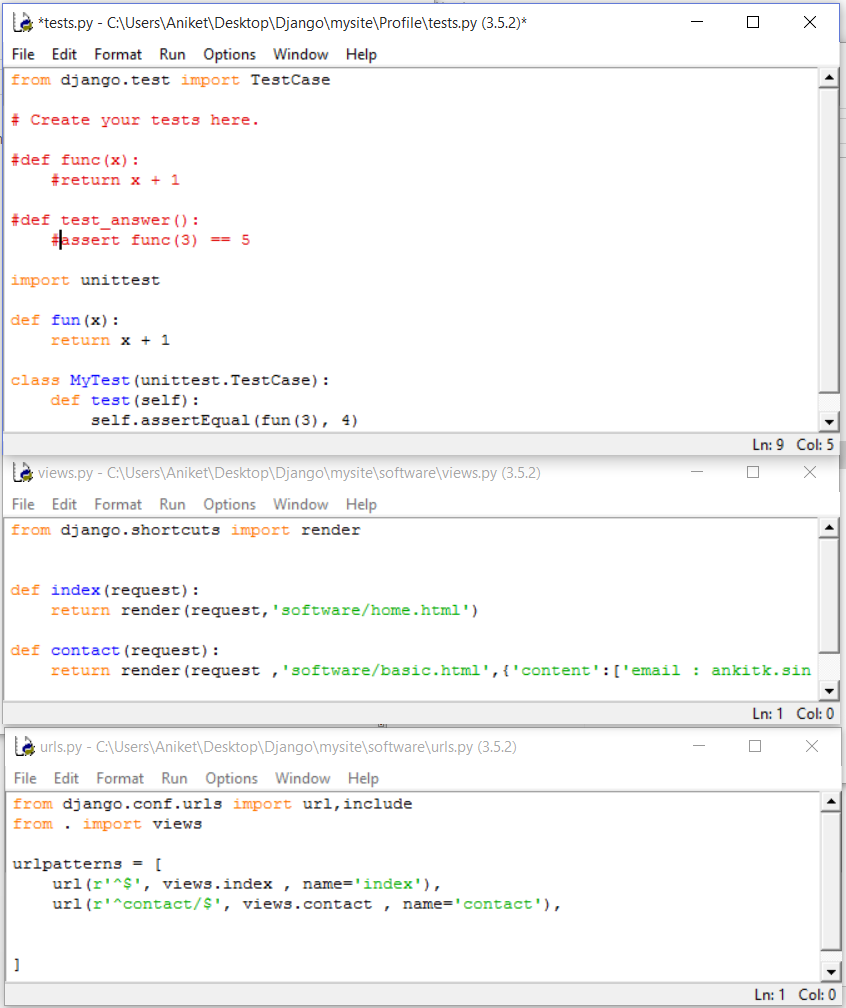
**TEST OUTCOME:**

**All Successful.**

**2.) Testing Software’s module:**

**a.) Tests applied: Unit test only.**

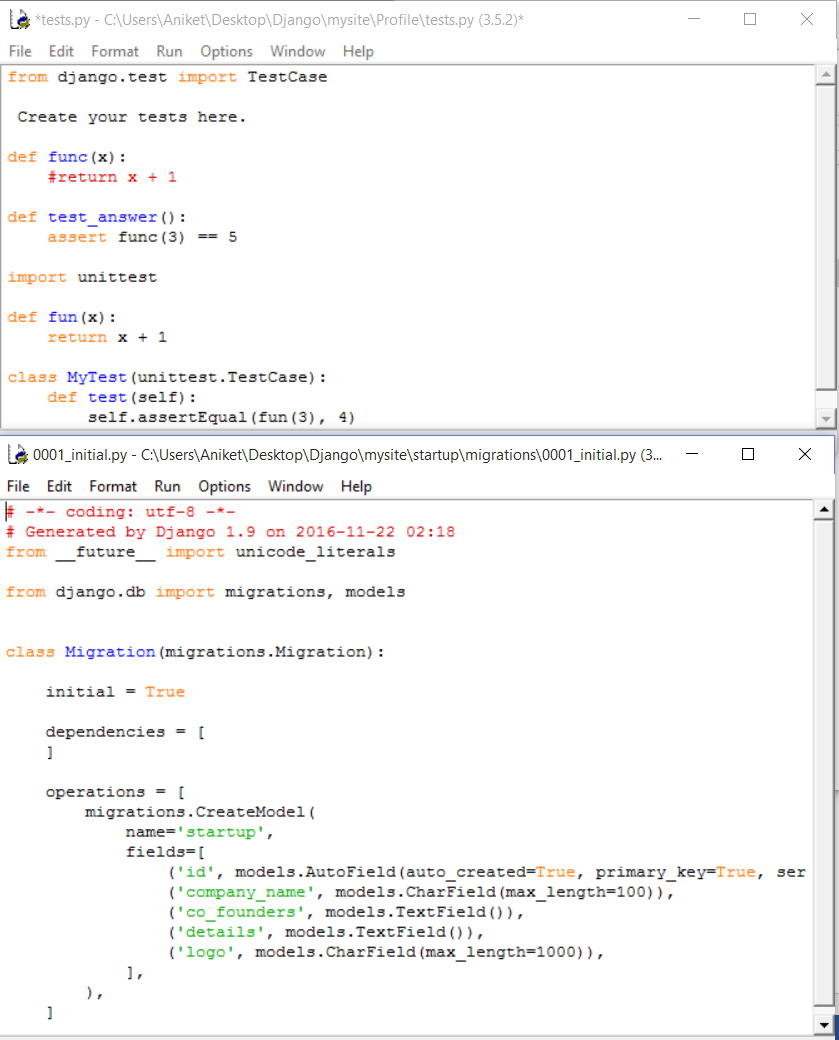
Depending upon the size of the code that is to be checked.



***OUTCOME: Test Successfull***

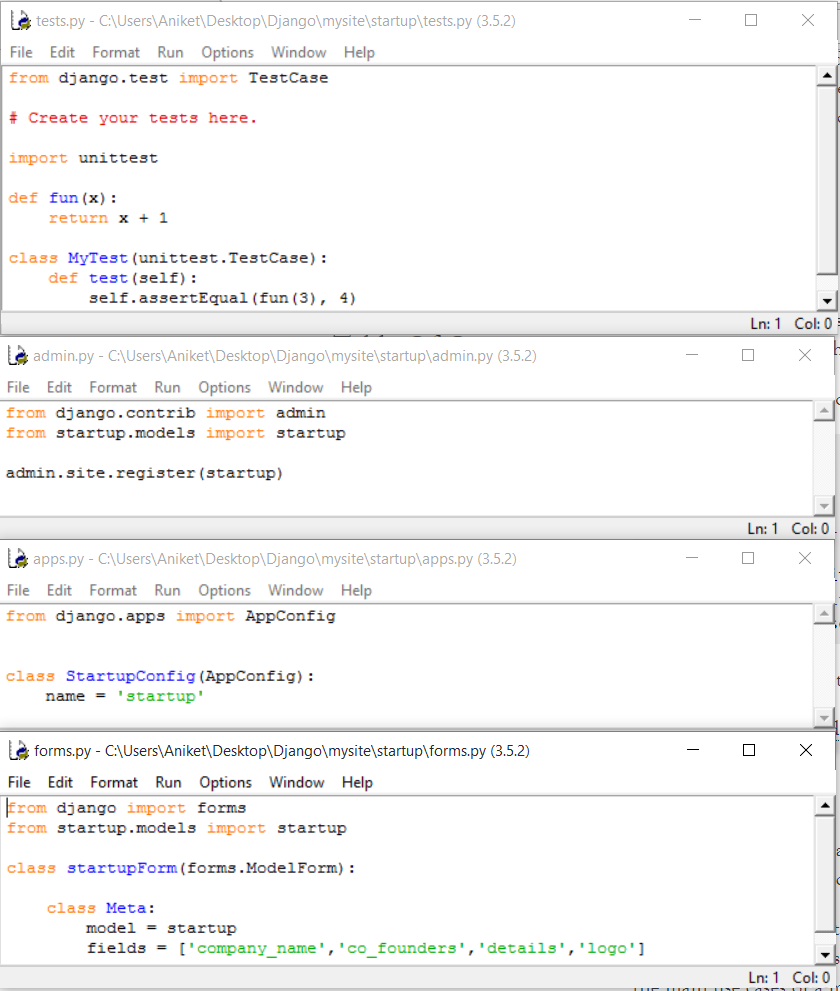
**3.) Testing Startup module:**

**a. tests applied: py test and unit test for migrations:**



O***UTCOME: Successful***

1. **tests applied: UNIT test only:**



***OUTCOME: Successful***

**4.) Testing User Module:**

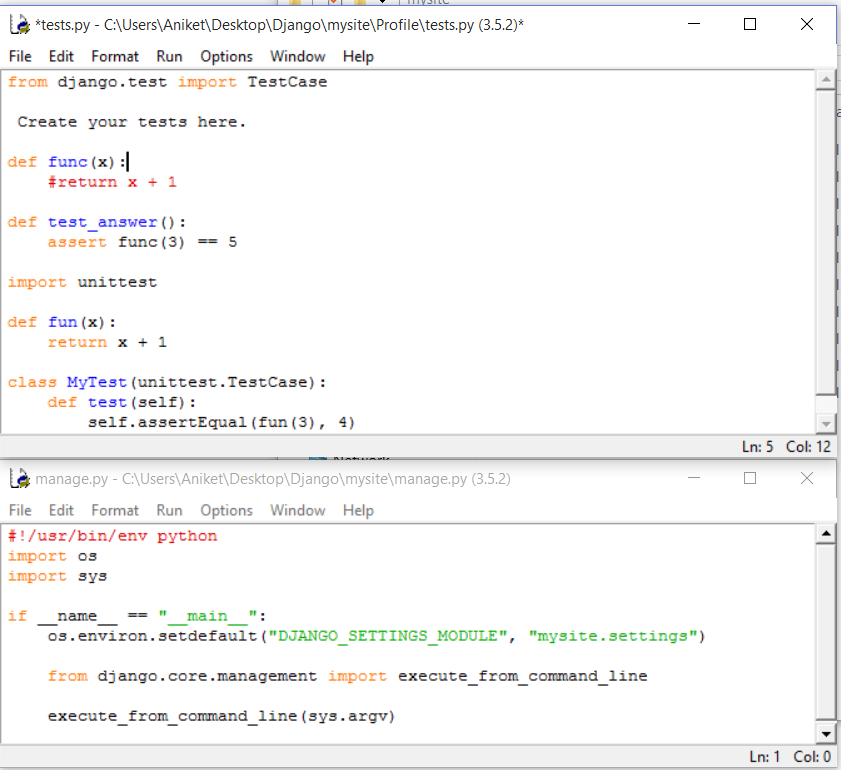
**a. testing method applied: Unit test only.**



***OUTCOME: Successful***

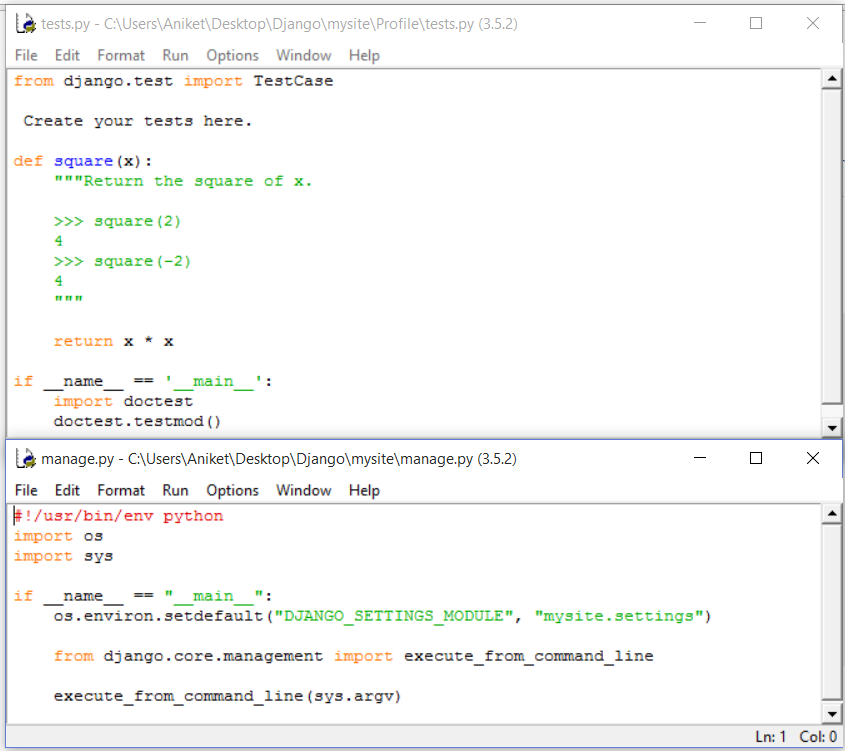
**5.) Testing the Site control managing algorithm:**

**a. tests applied: PY test and unit test:**



***OUTCOME: Successful***

**b. test applied: Doctest**



***OUTCOME: Successful.***

**UPDATES IN the MAIN software:**

As a result of testing and a thorough revision of the software, the following few modules have been added, to make things better!

**Testing the Updates:**

1. **accounts module testing**

**Code = Migration code for accounts module**

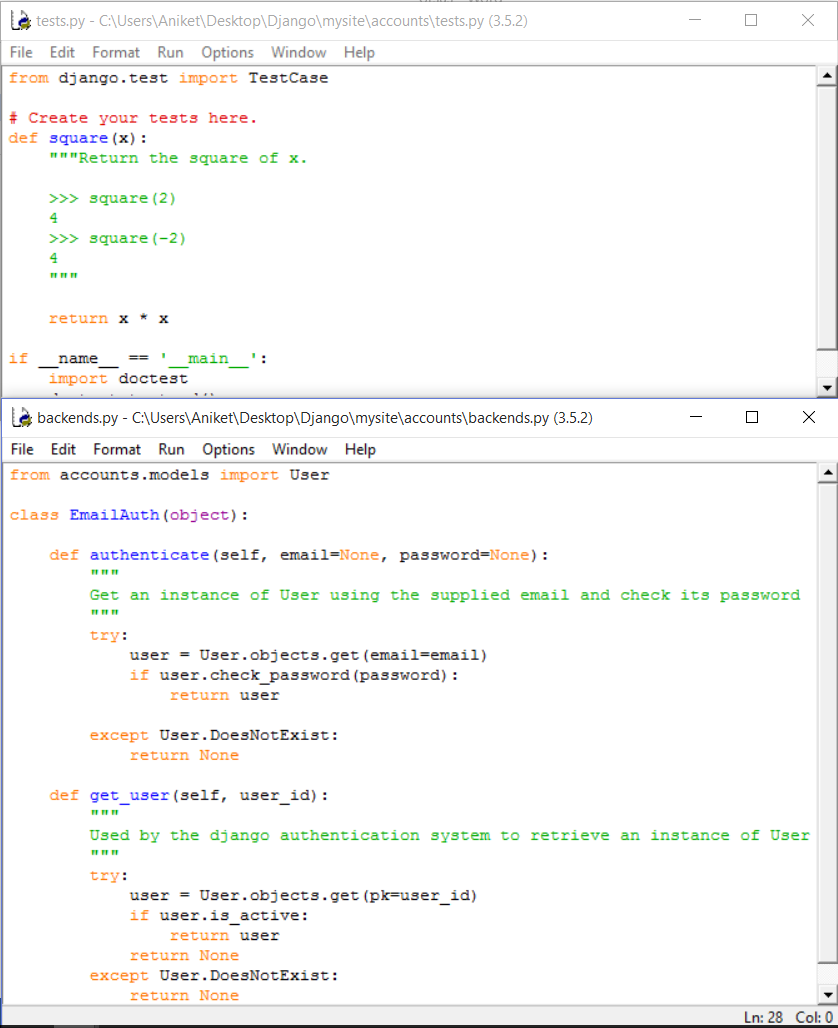
**Method= Doctest**



Outcome= Successful

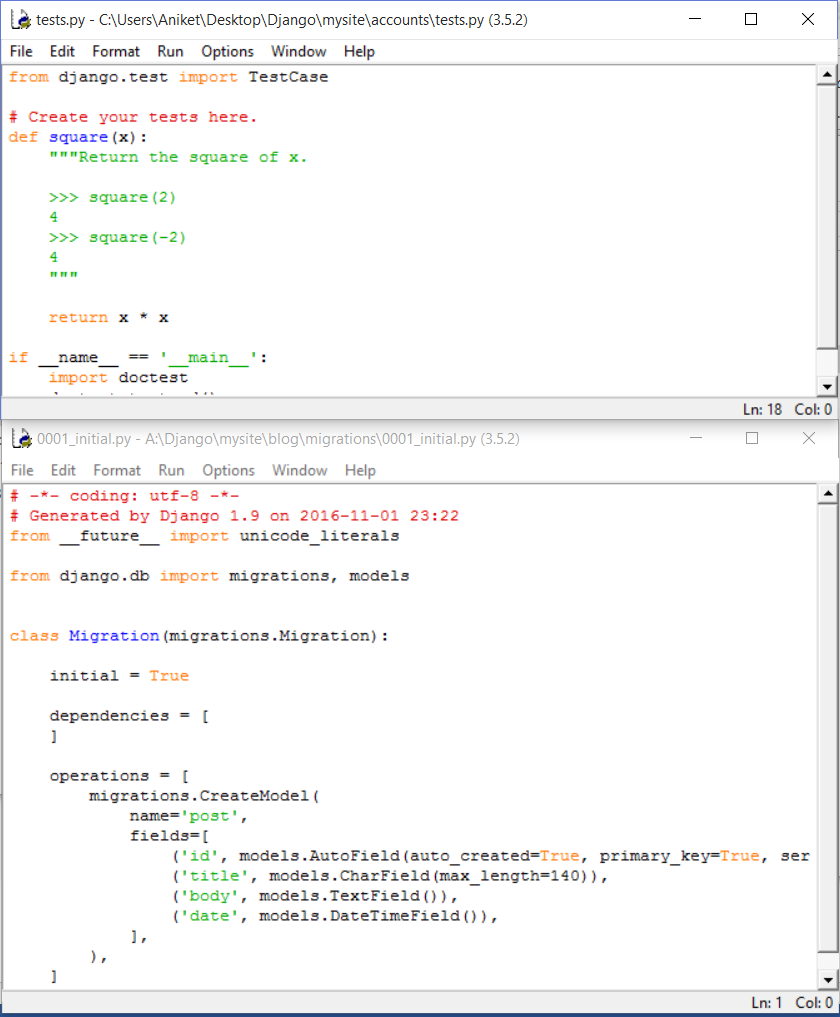
b. Testing Backends.py

Method used=Doctest



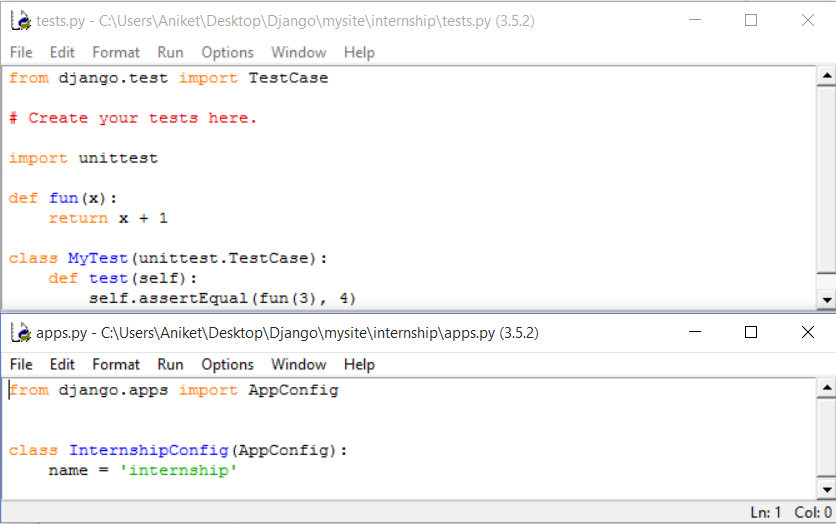
OUTCOME=Successful.

1. Retesting for Blog Migrations:



OUTCOME= Successful

1. Re Testing the internships module:
2. Test applied: Unit Test. (with objective of retesting already tested and updated service).



***OUTCOME=Succesfull.***

**Conclusion:**

With all critical , and most peripheral/auxiliary components working as required, the final verdict of conducting tests on the codes is –

Software passes the testing is every aspect.

Software is ready for market.