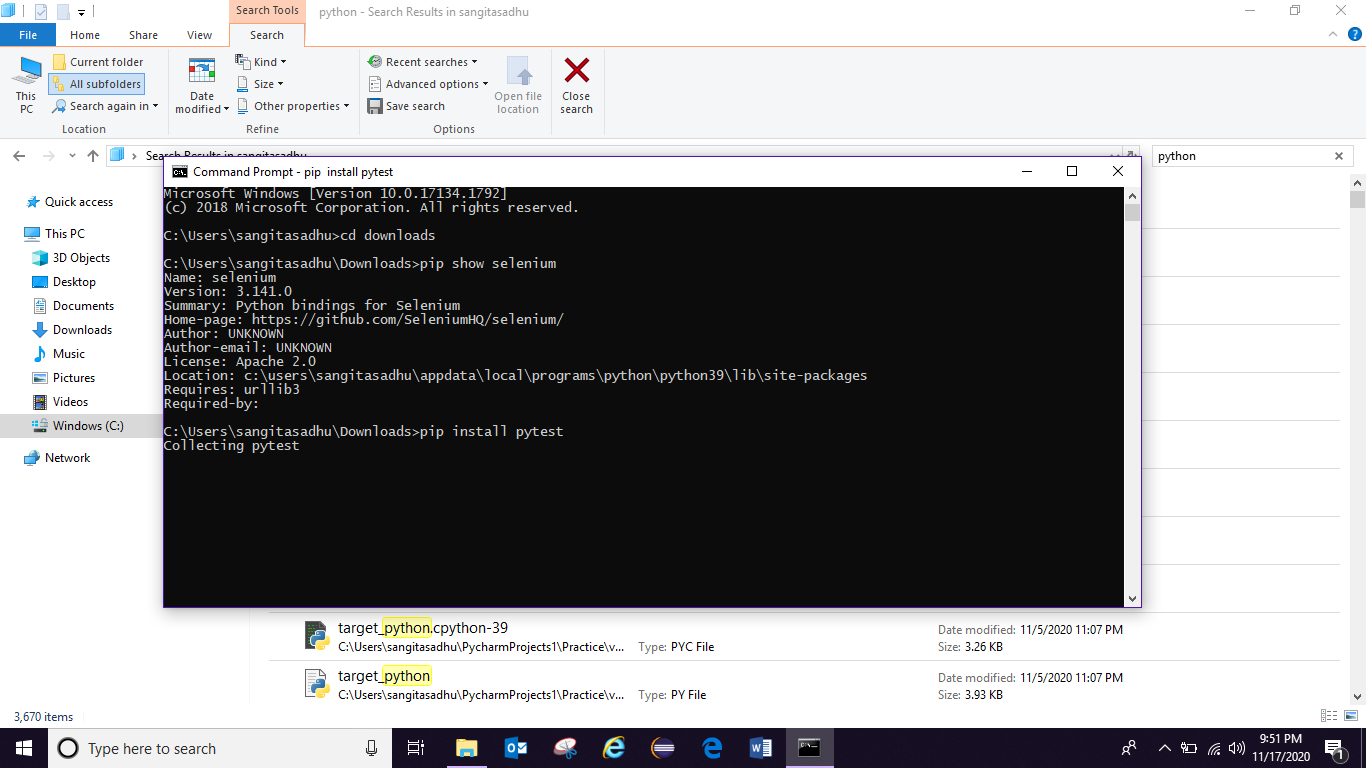
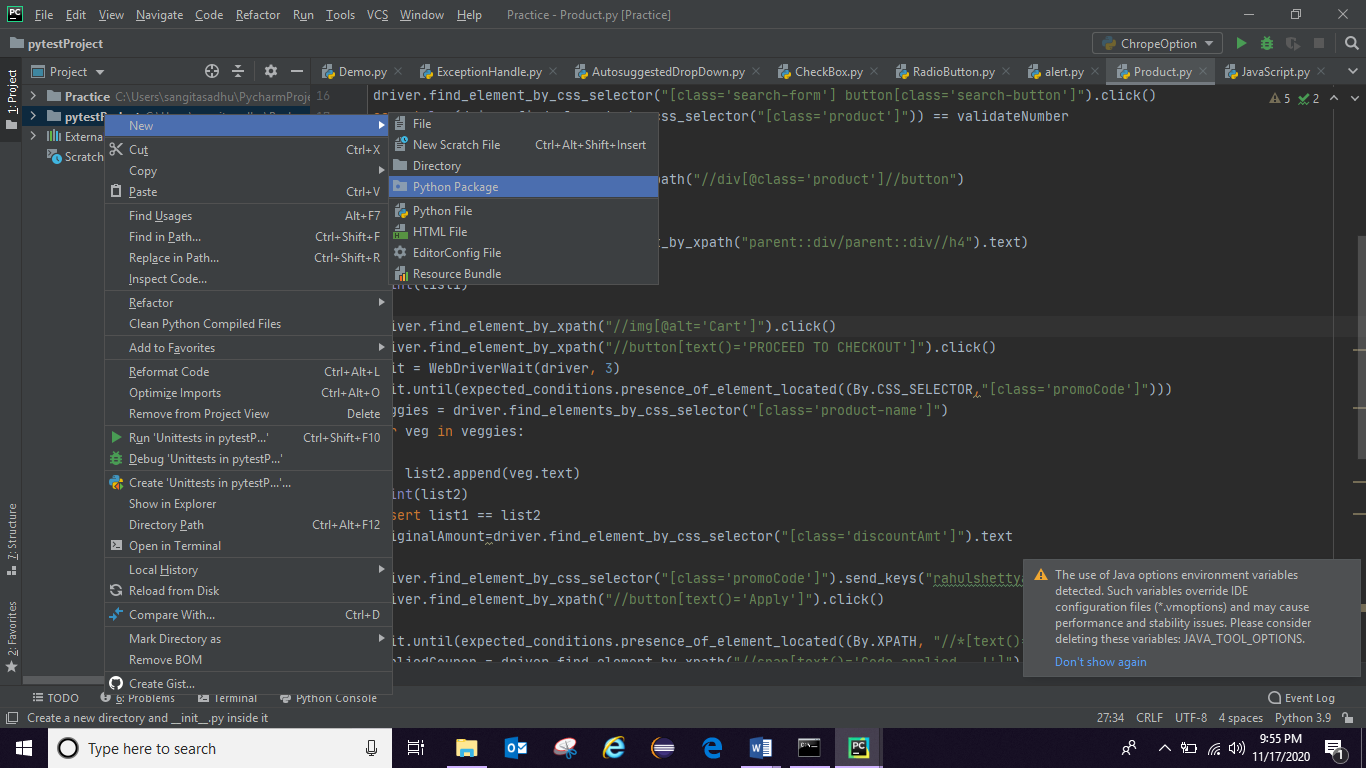
* Install pytest -> pip install pytest

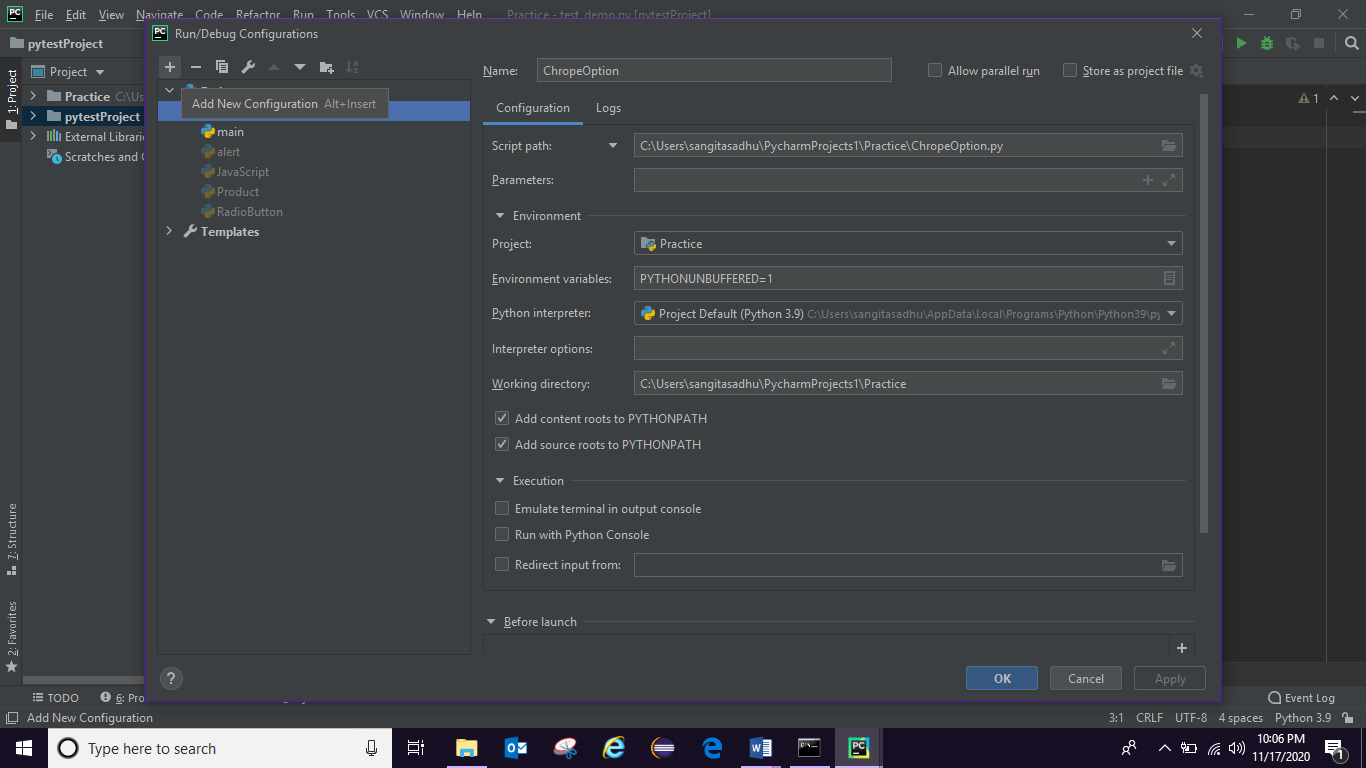


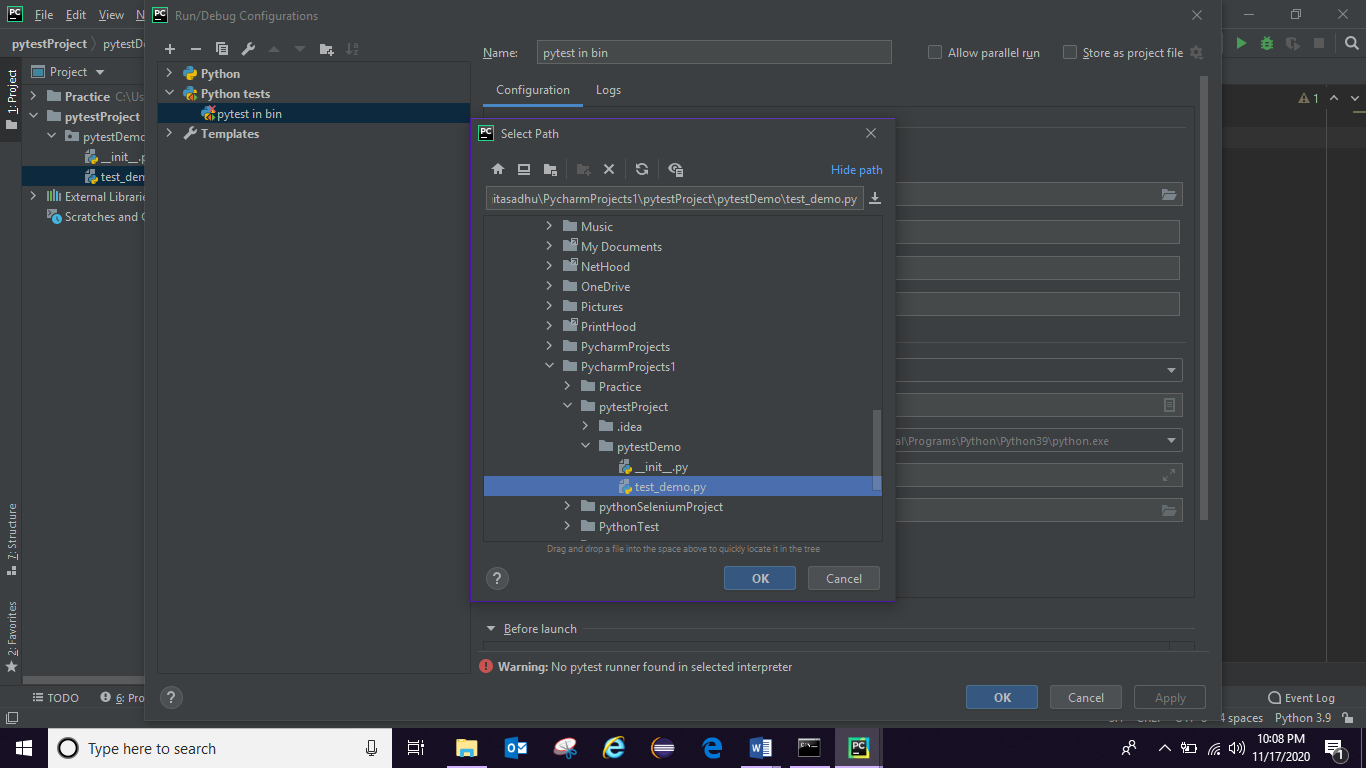
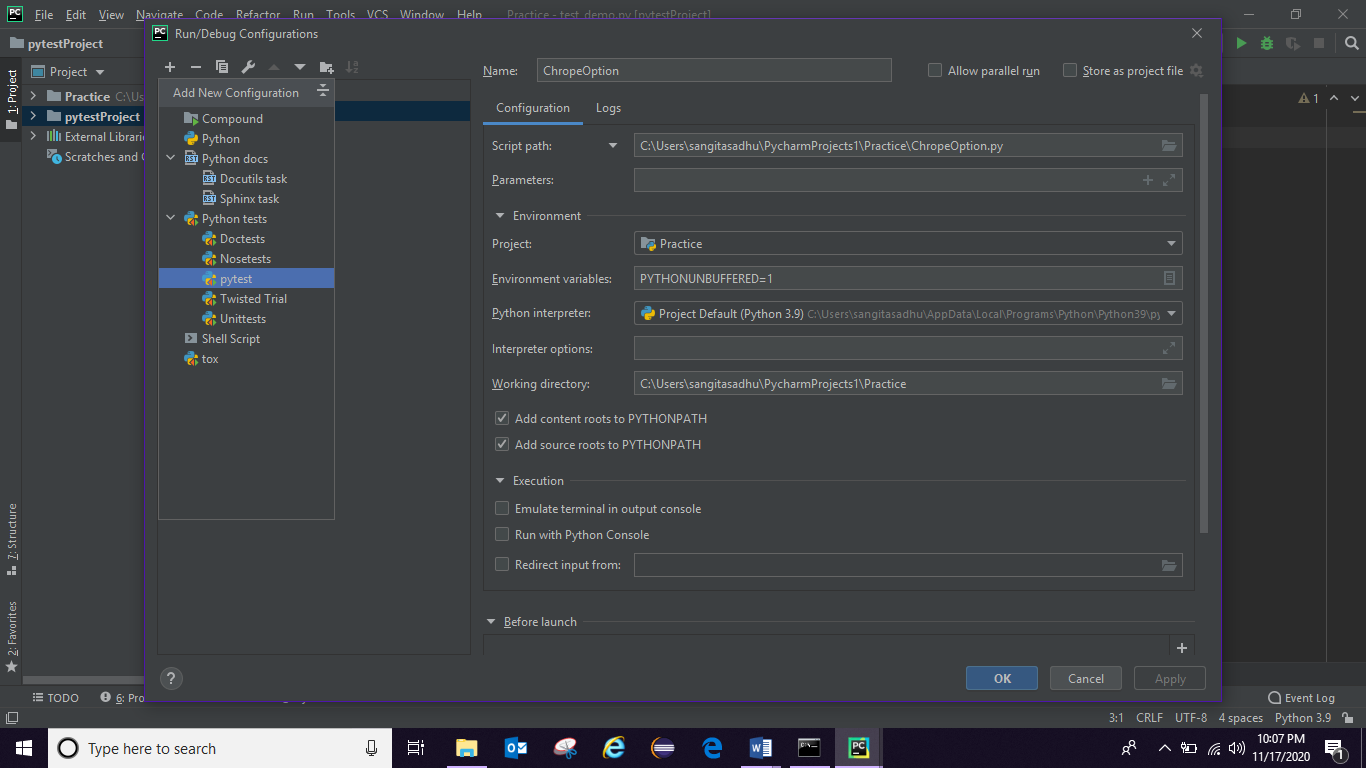
Pytest –version

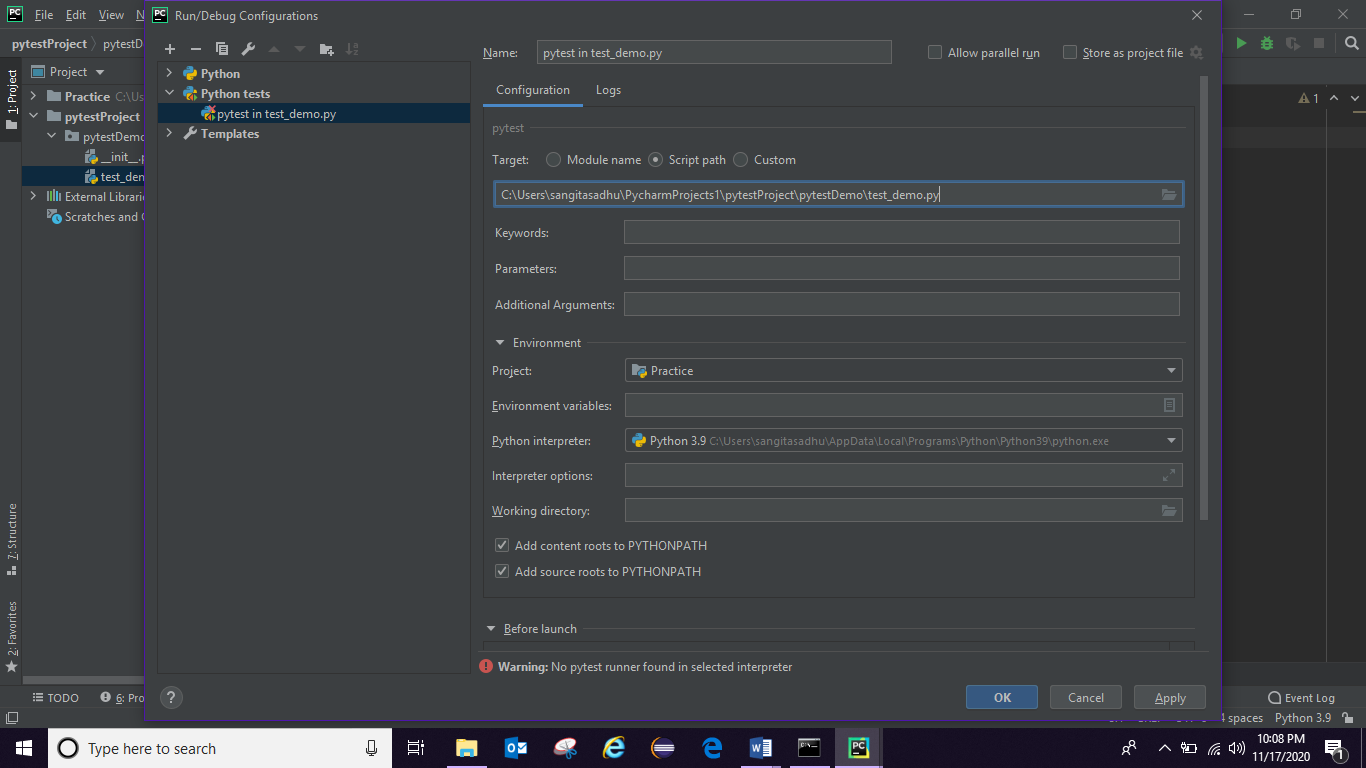
* Create new package pytestDemo



* To Run project in pytest framework mode:







* Any pytest file should start with test\_ or end with \_test
* We have to write all the code inside function whatever code write in pytest testing framework
* Test Method name also should start with test
* We can run from pycharm editor
* And we can run from cmd terminal also for that

Get the project path -> paste it in cmd -> give

py.test -v -s -> to see entire console and run entire package

-s stands for logs in output

-v stands for more info metadata

* In pytest if we have multiple method with same name pytest will print last method.
* To run particular test file from cmd or terminal

py.test <give the file name which want to run> -v -s

* Method name is nothing but Test case name in pyTest
* To run some set of Test Case or How to Regular Expression from cmd:
* If we have multiple Test Method with common name in one module use as regular expression:
* py.test -k <common name Ex: Credit Card > -v -s

k stands for Methods name execution.

* If we have 100 TC and want to run 20 TC as smoke test or regression test how to do in pyTest:
* @pytest.mark.smoke -> have to mark for 20 TC as smoke like group in TestNg

def test\_firstProgram():

print(“aaa”)

@ pytest.mark.smoke

def test\_secondProgram():

print(“bbbb”)

* Run from cmd

Command: py.test -m <Enter the mark ex: smoke> -v -s

m stands for mark

* You can skip tests with @pytest.mark.skip (skip is pre define)

EX:

* @pytest.mark.skip

def test\_firstProgramDemo2():

a = "Hi"

assert a in "Hello ABC"

* Run from cmd: py.test -v -s To run all the files

OutPut:

pytestDemo/test\_Demo2.py::test\_firstProgramDemo2 SKIPPED

pytestDemo/test\_demo.py::test\_firstProgram Hello

PASSED

pytestDemo/test\_demo.py::test\_secondProgram PASSED

* You can run test case as pre-requisite and do not want to show in report as it is failing for something then cannot use skip. We can use xfail: which will not show in report but at backend will execute:

@pytest.mark.xfail

EX:

@pytest.mark.xfail

def test\_firstProgramDemo2():

a = "Hi"

assert a in "Hello ABC"

* @pytest.fixture() :
* will execute before execution of other Test methods.
* In real time inside @pytest.fixture() annotation use to to open browser and to do some setup before executing test methods
* In order to make connection with fixture method to Test method pass fixture Method name into Test Method as an argument.

EX:

import pytest

@pytest.fixture()

def setup():

print(" Execute first")

def test\_DemoFixture(setup):

print("Execute")

* Post Test Execution like @afterMethod annotation we can mention the same in @pytest.fixture() method itself using yield statement
* “yield” means whatever will write after yield will execute after Test Methods
* Fixtures are used as setup and tear down methods for test cases
* Ex:

import pytest  
  
  
@pytest.fixture()  
def setup():  
 print(" Execute first")  
 yield  
 print("I will execute at last")

def test\_DemoFixture1(setup):  
  
 print("Execute11")

def test\_DemoFixture2(setup):  
  
 print("Execute22")

output:

Execute first

Execute11

I will execute at last

Execute first

Execute22

I will execute at last

* conftest.py file is used to generalize fixture and make it available to all test cases
* We can connect fixture with all the Test Cases by passing fixture method(setup) name trough Test Cases as an argument
* When we have n number of Test Cases we no need to write fixture method for each Test Cases we can optimize this by mentioning fixture as class level:
* @pytest.mark.usefixtures(“setup”)
* Ex using class:
* @pytest.mark.usefixtures("setup")  
  class TestExample:  
    
   def test\_DemoFixture1(self):  
   print("Execute")  
    
   def test\_DemoFixture2(self):  
   print("Execute")  
    
   def test\_DemoFixture3(self):  
    
   print("Execute")
* fixture file(setup and teardown)will execute before execution of each and every Test Cases
* We can make the scope of fixture file as class level like @beforeTest and @afterTest annotation ,which will execute only once before and after executing of all the Test Cases present in class
* At conftest file inside @pytest.fixture(scope=”class”) so fixture file will execute before executing of each Test Method
* @pytest.fixture(scope="class")  
  def setup():  
   print(" Execute first")  
   yield  
   print("I will execute at last")
* @pytest.mark.usefixtures("setup")  
  class TestExample:  
    
   def test\_DemoFixture1(self):  
   print("Execute1")  
    
   def test\_DemoFixture2(self):  
   print("Execute2")
* Fixture will help to load data also
* You can have data set up in a fixture in conftest file and in the run time you can bring all the data and can use all the data in web Automation as required.
* How we have to drive data in Test Case
* conftest.py
* @pytest.fixture(scope="class")  
  def dataload():  
   print(" data load method execute")  
   return ["sangita","sadhu","sangitasadhu345@gmail.com"]
* @pytest.mark.usefixtures("dataload")  
  class TestExpample:  
    
   def test\_first(self,dataload):  
   print(dataload)  
   print(dataload[0])  
   print(dataload[2])
* Output:

['sangita', 'sadhu', 'sangitasadhu345@gmail.com']

sangita

[sangitasadhu345@gmail.com](mailto:sangitasadhu345@gmail.com)

* Normally we can pass fixture method in globally but when we have to drive data setup have to return fixture name to the method also.
* We are passing data as Tuple in fixture method and can use it.
* Parameterizing test with multiple data sets using fixture

Running test with multiple data set

1. request is a instance of fixture using this all the parameter should call
2. return request.param

def test\_crossBrowser(self,crossBrowser):

* print(crossBrowser)

* conftest file
* @pytest.fixture(params=["chrome","Firefox","IE"])  
  def crossBrowser(request):  
   return request.param
* You can pass multiple value one single run:

@pytest.fixture(params=[("chrome","Sangita","AAA"),("Firefox","Sadhus","BBB"),("IE","SS","TT")])  
def crossBrowser(request):  
  
 return request.param

def test\_crossBrowser(crossBrowser):  
 print(crossBrowser)  
 print(crossBrowser[1])  
 print(crossBrowser[2])

output:

('chrome', 'Sangita', 'AAA')

Sangita

AAA

('Firefox', 'Sadhus', 'BBB')

Sadhus

BBB

('IE', 'SS', 'TT')

SS

TT

* Datadriven and parameterization can be done with return statements in tuple format
* When you define fixture scope to class only,it will run once before class is initiated and at the end after all the class executed
* PyTest HTML Report:
* <https://pypi.org/project/pytest-html/> go to link
* pip install pytest-html
* pytest --html=report.html -> get report what we run
* Now to run py.test -- html=report.html to get the report and report will generate also
* Logs: