- Create folder as (Python Package)
- Create file (test_demo1.py)
- Keep in mind that both methods and test files need the test_ prefix or _test suffix to be recognized as a test file.
- Class name should also start with <Test as TestExample>

```
def test_methodA():
    print('Running method A')


def test_methodB():
    print('Running method B')
```

Чтобы запустить файл: $ py.test

```
(venv)  [pytest_fixtures]$ py.test
============================================================================================== test session starts =====
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0
rootdir: /Users/tolik/PycharmProjects/SauseDemo/pytest_fixtures
collected 2 items

test_file1.py ..

============================================================================================== 2 passed in 0.01s ======
```

$ py.test -v

-v gives the results more verbosity and detail to our tests. We can now see which specific tests have failed or passed.

```
(venv)  [pytest_fixtures]$ py.test -v
============================================================================================== test session starts =====
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo/pytest_fixtures
collected 2 items

test_file1.py::test_methodA PASSED
test_file1.py::test_methodB PASSED

============================================================================================== 2 passed in 0.00s ======
```

$ py.test -v -s
-s prints statements

```
(venv)  [pytest_fixtures]$ py.test -v -s
============================================================================================= test session starts =====
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo/pytest_fixtures
collected 2 items

test_file1.py::test_methodA Running method A
PASSED
test_file1.py::test_methodB Running method B
PASSED

============================================================================================== 2 passed in 0.00s ======
```

- Import the pytest module in all Python files you want to test, as well as in any associated configuration files for the fixtures.

- Function is a fixture with @pytest.fixture. These specific Python decorations let us know that the next method is a pytest fixture.

- @pytest.fixture => Implementing the simplest pytest fixture can just return an object, like an integer.

```
import pytest

@pytest.fixture()
def setUp():
    print("Once before every method")


def test_methodA(setUp):
    print('Running method A')


def test_methodB(setUp):
    print('Running method B')
```

```
(venv)  [pytest_fixtures]$ py.test -v -s
============================================================================================= test session starts ====
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo, configfile: pytest.ini
collected 2 items

test_file1.py::test_methodA
Once before every method
Running method A
PASSED
test_file1.py::test_methodB
Once before every method
Running method B
PASSED

============================================================================================== 2 passed in 0.00s =====
```

```python
import pytest

@pytest.fixture()
def setUp():
    print("\nOnce before every method")


def test_methodA(setUp):
    print('Running method A')


def test_methodB():
    print('Running method B')
```

```
(venv)  [pytest_fixtures]$ py.test -v -s
============================================================================================= test session starts ====
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo, configfile: pytest.ini
collected 2 items

test_file1.py::test_methodA
Once before every method
Running method A
PASSED
test_file1.py::test_methodB Running method B
PASSED

============================================================================================== 2 passed in 0.01s =====
```

- Create one more test file (e.g. test_file2.py)

```python
import pytest


@pytest.fixture()
def set_up():
    print("\nOnce before every method")
    yield
    print('\nOnce after every method')


def test_methodA(set_up):
    print('Running method A')


def test_methodB(set_up):
    print('Running method B')
```

$ py.test -s -v test_file2.py => run one file from the package folder

```
(venv)  [pytest_fixtures]$ py.test -s -v test_file2.py
============================================================================================ test session starts ======
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo, configfile: pytest.ini
collected 2 items

test_file2.py::test_methodA
Once before every method
Running method A
PASSED
Once after every method

test_file2.py::test_methodB
Once before every method
Running method B
PASSED
Once after every method


============================================================================================= 2 passed in 0.00s =======
```

Multiple Ways to Run Test Cases

test_file1.py

```
import pytest

@pytest.fixture()
def set_up():
    print("\nfile1 : Once before every method")


def test_methodA(set_up):
    print('Running method A file1')


def test_methodB(set_up):
    print('Running method B file1')
```

test_file2.py

```
import pytest


@pytest.fixture()
def set_up():
    print("\nfile2: Once before every method")
    yield
    print('\nfile2: Once after every method')


def test_methodA_file2(set_up):
    print('Running method A file2')


def test_methodB_file2(set_up):
    print('Running method B file2')
```

test_file3.py

```
import pytest


@pytest.fixture()
def set_up():
    print("\nfile3: Once before every method")
    yield
    print('\nfile3: Once after every method')


def test_methodA_file3(set_up):
    print('Running method A file3')


def test_methodB_file3(set_up):
    print('Running method B file3')
```

- To run only one method from specific file:

  $ py.test test_file3.py::test_methodA_file3 -s -v

```
(venv)  [pytest_fixtures]$ py.test test_file3.py::test_methodA_file3 -s -v
============================================================================================== test session starts ===
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo, configfile: pytest.ini
collected 1 item

test_file3.py::test_methodA_file3
file3: Once before every method
Running method A file3
PASSED
file3: Once after every method


=============================================================================================== 1 passed in 0.00s ====
```

conftest.py => Common Fixtures to Multiple Modules

Создаем файл conftest.py

```
import pytest


@pytest.fixture()
```

```
def set_up():
    print("\nRunning conftest demo method setup")
    yield
    print('\nRunning conftest demo method teardown')
```

Из файла test_file2.py удаляем функцию def set_up():

```
import pytest


def test_methodA_file2(set_up):
    print('Running method A file2')


def test_methodB_file2(set_up):
    print('Running method B file2')
```

$ py.test -v -s test_file*.py

```
(venv)  [pytest_fixtures]$ py.test -v -s test_file*.py
============================================================================================= test session starts ==
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo, configfile: pytest.ini
collected 6 items

test_file1.py::test_methodA
file1 : Once before every method
Running method A file1
PASSED
test_file1.py::test_methodB
file1 : Once before every method
Running method B file1
PASSED
test_file2.py::test_methodA_file2
Running conftest demo method setup
Running method A file2
PASSED
Running conftest demo method teardown

test_file2.py::test_methodB_file2
Running conftest demo method setup
Running method B file2
PASSED
```

```
test_file3.py::test_methodA_file3
file3: Once before every method
Running method A file3
PASSED
file3: Once after every method

test_file3.py::test_methodB_file3
file3: Once before every method
Running method B file3
PASSED
file3: Once after every method


========================================================================================== 6 passed in 0.01s ==
```

Добавляем еще одну функцию в conftest file

```
import pytest


@pytest.fixture()
def set_up():
    print("\nRunning conftest demo method setup")
    yield
    print('\nRunning conftest demo method teardown')


@pytest.fixture()
def oneTimeSetup(scope='module'):
    print("\nRunning conftest demo one time setup")
    yield
    print('\nRunning conftest demo one time teardown')
```

By default, scope set as a function, and it applies to every function in the module.

```
import pytest


def test_methodA_file2(set_up, oneTimeSetup):
    print('Running method A file2')


def test_methodB_file2(set_up, oneTimeSetup):
    print('Running method B file2')
```

```
(venv)  [pytest_fixtures]$ py.test -s -v
============================================================================================= test session starts ===
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo, configfile: pytest.ini
collected 6 items

test_file1.py::test_methodA
file1 : Once before every method
Running method A file1
PASSED
test_file1.py::test_methodB
file1 : Once before every method
Running method B file1
PASSED
test_file2.py::test_methodA_file2
Running conftest demo method setup

Running conftest demo one time setup
Running method A file2
PASSED
Running conftest demo one time teardown

Running conftest demo method teardown

test_file2.py::test_methodB_file2
Running conftest demo method setup

Running conftest demo one time setup
Running method B file2
PASSED
Running conftest demo one time teardown

Running conftest demo method teardown

test_file3.py::test_methodA_file3
file3: Once before every method
Running method A file3
PASSED
file3: Once after every method

test_file3.py::test_methodB_file3
file3: Once before every method
Running method B file3
PASSED
file3: Once after every method


============================================================================================== 6 passed in 0.01s ==
```

Изменяем порядок аргументов в функции. И первым аргументом будет, oneTimeSetup, а потом set_up

```
import pytest


def test_methodA_file2(oneTimeSetup, set_up ):
    print('Running method A file2')


def test_methodB_file2(oneTimeSetup, set_up):
    print('Running method B file2')
```

```
test_file2.py::test_methodA_file2
Running conftest demo one time setup

Running conftest demo method setup
Running method A file2
PASSED
Running conftest demo method teardown

Running conftest demo one time teardown

test_file2.py::test_methodB_file2
Running conftest demo one time setup

Running conftest demo method setup
Running method B file2
PASSED
Running conftest demo method teardown

Running conftest demo one time teardown
```

How to maintain run order of tests

Install new package pytest-ordering

Create file test_run_ordering.py

conftest.py

```
import pytest


@pytest.fixture()
def set_up():
    print("\n*** Running method level setup ***")
    yield
    print('\n*** Running method level teardown ***')
```

```
@pytest.fixture()
def oneTimeSetup(scope='module'):
    print("\n< === > Running conftest demo one time setup < === >")
    yield
    print('\n< === > Running conftest demo one time teardown < === >')
```

test_run_ordering.py

```
import pytest


def test_order_methodA(oneTimeSetup, set_up):
    print('Running method A')


def test_order_methodB(oneTimeSetup, set_up):
    print('Running method B')


def test_order_methodC(oneTimeSetup, set_up):
    print('Running method C')


def test_order_methodD(oneTimeSetup, set_up):
    print('Running method D')


def test_order_methodE(oneTimeSetup, set_up):
    print('Running method E')


def test_order_methodF(oneTimeSetup, set_up):
    print('Running method F')
```

```
test_run_order.py::test_order_methodA
< === > Running conftest demo one time setup < === >

*** Running method level setup ***
Running method A
PASSED
*** Running method level teardown ***

< === > Running conftest demo one time teardown < === >

test_run_order.py::test_order_methodB
< === > Running conftest demo one time setup < === >

*** Running method level setup ***
Running method B
PASSED
*** Running method level teardown ***

< === > Running conftest demo one time teardown < === >
```

```python
import pytest


@pytest.mark.run(order=2)
def test_order_methodA(oneTimeSetup, set_up):
    print('Running method A')


@pytest.mark.run(order=1)
def test_order_methodB(oneTimeSetup, set_up):
    print('Running method B')

@pytest.mark.run(order=4)
def test_order_methodC(oneTimeSetup, set_up):
    print('Running method C')


@pytest.mark.run(order=6)
def test_order_methodD(oneTimeSetup, set_up):
    print('Running method D')


@pytest.mark.run(order=3)
def test_order_methodE(oneTimeSetup, set_up):
    print('Running method E')
```

```
@pytest.mark.run(order=5)
def test_order_methodF(oneTimeSetup, set_up):
    print('Running method F')
```

Install new package pytest-order
Create new file test_run_order.py

https://pypi.org/project/pytest-order/

```
import pytest


@pytest.mark.order(2)
def test_order_methodA(oneTimeSetup, set_up):
    print('Running method A')


@pytest.mark.order(1)
def test_order_methodB(oneTimeSetup, set_up):
    print('Running method B')

@pytest.mark.order(4)
def test_order_methodC(oneTimeSetup, set_up):
    print('Running method C')


@pytest.mark.order(3)
def test_order_methodD(oneTimeSetup, set_up):
    print('Running method D')


@pytest.mark.order(6)
def test_order_methodE(oneTimeSetup, set_up):
    print('Running method E')


@pytest.mark.order(5)
```

```
def test_order_methodF(oneTimeSetup, set_up):
    print('Running method F')
```

```
test_run_order.py::test_order_methodB
< === > Running conftest demo one time setup < === >

*** Running method level setup ***
Running method B
PASSED
*** Running method level teardown ***

< === > Running conftest demo one time teardown < === >

test_run_order.py::test_order_methodA
< === > Running conftest demo one time setup < === >

*** Running method level setup ***
Running method A
PASSED
*** Running method level teardown ***

< === > Running conftest demo one time teardown < === >

test_run_order.py::test_order_methodD
< === > Running conftest demo one time setup < === >
```

Running Tests Based on Command Line Arguments

conftest.py

```
import pytest


@pytest.fixture()
def setUp():
    print("Running method level setUp")
    yield
    print("Running method level tearDown")


@pytest.fixture(scope="module")
def oneTimeSetUp(browser, osType):
    print("Running one time setUp")
```

```
    if browser == 'firefox':
        print("Running tests on FF")
    else:
        print("Running tests on chrome")
    yield
    print("Running one time tearDown")


def pytest_addoption(parser):
    parser.addoption("--browser")
    parser.addoption("--osType", help="Type of operating system")


@pytest.fixture(scope="session")
def browser(request):
    return request.config.getoption("--browser")


@pytest.fixture(scope="session")
def osType(request):
    return request.config.getoption("--osType")
```

test_command_line.py

```
import pytest


def test_command_line_methodA(oneTimeSetUp, setUp):
    print("Running method A")


def test_command_line_methodB(oneTimeSetUp, setUp):
    print("Running method B")
```

```
$ py.test -s -v test_command_line.py --browser firefox
```

$ py.test -s -v test_command_line.py --browser chrome

```
(venv)  [pytest_fixtures]$ py.test -s -v test_command_line.py --browser firefox
============================================================================================= test session starts ====
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
rootdir: /Users/tolik/PycharmProjects/SauseDemo, configfile: pytest.ini
plugins: ordering-0.6, order-1.0.1
collected 2 items

test_command_line.py::test_command_line_methodA Running one time setUp
Running tests on FF
Running method level setUp
Running method A
PASSEDRunning method level tearDown

test_command_line.py::test_command_line_methodB Running method level setUp
Running method B
PASSEDRunning method level tearDown
Running one time tearDown
```

Structure Tests in a Test Class

- How to use test class to wrap methods under one class
- Learn about the autouse keywords in fixtures
- Assert the result to create a real test scenario
- @pytest.fixture(scope="class") in the conftest.py file change the scope to be equal to class

Create file < class_to_test.py >

```
class SomeClassToTest:

    def __init__(self, value):
        self.value = value

    def sumNumbers(self, a, b):
        return a + b + self.value
```

conftest.py file

```
import pytest


@pytest.fixture()
def set_up():
    print("Running method level setUp")
    yield
    print("Running method level tearDown")


@pytest.fixture(scope="class")
def oneTimeSetUp(browser, osType):
    print("Running one time setUp")
    if browser == 'firefox':
        print("Running tests on FF")
    else:
        print("Running tests on chrome")
    yield
    print("Running one time tearDown")


def pytest_addoption(parser):
    parser.addoption("--browser")
    parser.addoption("--osType", help="Type of operating system")


@pytest.fixture(scope="session")
def browser(request):
    return request.config.getoption("--browser")


@pytest.fixture(scope="session")
def osType(request):
    return request.config.getoption("--osType")
```

- Create the file < test_class.py >

- ```
  import pytest
  from pytest_fixtures.class_to_test import SomeClassToTest


  class TestClaseDemo:

      def test_methodA(self):
          print('Running method A')

      def test_methodB(self):
          print('Running method B')
  ```

Импортируем pytest, также импортируем из файла class_to_test => class SomeClassToTest

Если мы хотим импортировать некоторые декораторы из fixtures и у нас этих методов около сотни, то должен ли я прописывать их всех в качестве аргументов?

```
def test_methodA(self, oneTimeSetup, set_up, method1, method2, args):
        print('Running method A')
```

Легче добавить fixture и перечислить все те функции, которые мы хотим использовать из conftest файла. Это добавляется над строкой класс и указываем, какие функции мы хотим использовать. Таким образом все эти функции доступны для нашего класса.

Пример:

```
@pytest.mark.usefixtures("метод1", "метод2", "метод3", ..args)
class TestClaseDemo:
```

```
import pytest
from pytest_fixtures.class_to_test import SomeClassToTest


@pytest.mark.usefixtures("oneTimeSetUp", "set_up")
class TestClaseDemo:
```
```

```
    @pytest.fixture(autouse=True)
    def classSetup(self):
        self.abc = SomeClassToTest(10)

    def test_methodA(self):
        result = self.abc.sumNumbers(2, 8)
        assert result == 20
        print("Running method A")

    def test_methodB(self):
        print("Running method B")
```

How to Generate HTML Test Report

Install package pytest-html

$ pip3 install pytest-html

$ py.test -s -v test_class.py --browser firefox --html=html_report.html

Если вы хотите получить репорт то в конце строки команды введите --html=имя_файла.html

```
(venv)  [pytest_fixtures]$ py.test -s -v test_class.py --browser firefox --html=html_report.html
============================================================ test session starts ============================================================
platform darwin -- Python 3.10.4, pytest-7.2.0, pluggy-1.0.0 -- /Users/tolik/PycharmProjects/SauseDemo/venv/bin/python
cachedir: .pytest_cache
metadata: {'Python': '3.10.4', 'Platform': 'macOS-12.6-x86_64-i386-64bit', 'Packages': {'pytest': '7.2.0', 'pluggy': '1.0.0'}, 'Plugins': {'html': '3.2.0', 'ordering': '0.6', 'order': '1.0.1', 'metadata': '2.0.4'},
 'JAVA_HOME': '/Library/Java/JavaVirtualMachines/jdk1.8.0_341.jdk/Contents/Home'}
rootdir: /Users/tolik/PycharmProjects/SauseDemo, configfile: pytest.ini
plugins: html-3.2.0, ordering-0.6, order-1.0.1, metadata-2.0.4
collected 2 items

test_class.py::TestClaseDemo::test_methodA Running one time setUp
Running tests on FF
Running method level setUp
Running method A
PASSEDRunning method level tearDown

test_class.py::TestClaseDemo::test_methodB Running method level setUp
Running method B
PASSEDRunning method level tearDown
Running one time tearDown


---------- generated html file: file:///Users/tolik/PycharmProjects/SauseDemo/pytest_fixtures/html_report.html ----------
============================================================ 2 passed in 0.02s =============================================================
```

html_report.html

Report generated on 13-Nov-2022 at 19:20:03 by pytest-html v3.2.0

Environment

JAVA_HOME	/Library/Java/JavaVirtualMachines/jdk1.8.0_341.jdk/Contents/Home
Packages	{"pluggy": "1.0.0", "pytest": "7.2.0"}
Platform	macOS-12.6-x86_64-i386-64bit
Plugins	{"html": "3.2.0", "metadata": "2.0.4", "order": "1.0.1", "ordering": "0.6"}
Python	3.10.4

Summary

2 tests ran in 0.02 seconds.

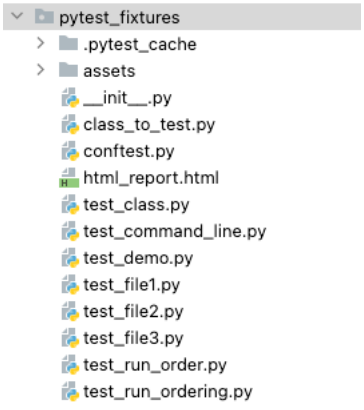
(Un)check the boxes to filter the results.

2 passed, 0 skipped, 0 failed, 0 errors, 0 expected failures, 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration	Links
Passed *(show details)*	pytest_fixtures/test_class.py::TestClaseDemo::test_methodA	0.00	
Passed *(show details)*	pytest_fixtures/test_class.py::TestClaseDemo::test_methodB	0.00	



Taking a screenshot method

```
def screenShot(self, resultMessage):
    """
    Takes screenshot of the current open web page
    """
    fileName = resultMessage + "." + str(round(time.time() * 1000)) + ".png"
    screenshotDirectory = "../screenshots/"
    relativeFileName = screenshotDirectory + fileName
    currentDirectory = os.path.dirname(__file__)
    destinationFile = os.path.join(currentDirectory, relativeFileName)
    destinationDirectory = os.path.join(currentDirectory, screenshotDirectory)

    try:
        if not os.path.exists(destinationDirectory):
            os.makedirs(destinationDirectory)
        self.driver.save_screenshot(destinationFile)
        self.log.info("Screenshot save to directory: " + destinationFile)
    except:
        self.log.error("### Exception Occurred when taking screenshot")
        print_stack()
```