PyTest

There are several Python unit testing frameworks, but pytest is both the most popular and the easiest to use. It requires no boilerplate, no imports, no API. Just name your files, classes, or methods starting with “test”, type “pytest” into the command line, and observe the results.

Simple, right? And we want it that way. Pytest allows our tests to be uncluttered and easy to read. But as far as the output of those tests goes… we can do better!

This document will show you how to master the pytest CLI (Command-line interface).

Make sure, pytest installed.

All good? Let’s get started!

First create following folder structure to practice pytest commands:

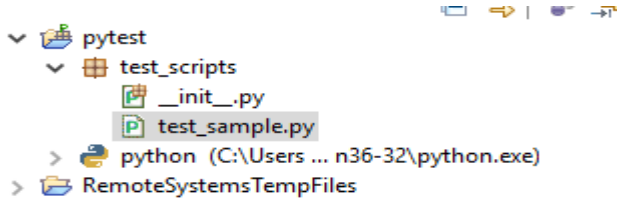
Step 1: Create a project.

Note: If you are creating the project in the new workspace, please don’t forget to change the “change the perspective and add the python preference to the project first”.

Step 2: Create the package.

Step 3: Create the python module.

Step 4: Develop test class and test functions.



test_scripts | test_sample

```
1  def test_demo1():
2      print('Running the demo1')
3
4  def test_demo2():
5      print("Running the demo2")
6
7  class TestSample():
8
9      def test_demo3(self):
10         print('Running the test demo3')
11
12     def test_demo4(self):
13         print('Running the test demo4')
14
```

Open the command prompt:

Verbosity

- Verbose is a general-purpose programming term for produce logging output.
- It's like, asking the program to tell me everything about what you are doing all the time.

Pytest allows the user to control the verbose (logging) information,

To Increase verbosity:

- If programmer wants to see more logging information, we can use the following command,

"Pytest -v"

Output will looks like below if you use the "-v"

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest -v
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1 -- c:\users\priyapra
\python36-32\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.6.3', 'Platform': 'Windows-10-10.0.17134-SP0', 'Packages': {'pyte
ggy': '0.8.1'}, 'Plugins': {'xdist': '1.26.0', 'ordering': '0.6', 'metadata': '1.7.0', 'h
'dependency': '0.4.0', 'cov': '2.5.1', 'allure-pytest': '2.5.4'}, 'JAVA_HOME': 'C:\\Progr
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, dependency-
.5.4
collected 8 items

test_sample.py::test_demo1 PASSED                                        [ 12%]
test_sample.py::test_demo2 PASSED                                        [ 25%]
test_sample.py::TestSample::test_demo3 PASSED                            [ 37%]
test_sample.py::TestSample::test_demo4 PASSED                            [ 50%]
test_sample.py::test_sample1 PASSED                                      [ 62%]
test_sample.py::test_sample2 PASSED                                      [ 75%]
test_sample1.py::test_demo7 PASSED                                       [ 87%]
test_sample1.py::test_demo8 PASSED                                       [100%]

========================== 8 passed in 0.09 seconds ===========================
```

To decrease the verbosity:

Of course, if you want to run all your tests, but just make them run *more quietly*, you can tone down their output like this:

"Pytest -q"

Output will looks like below if you use the "-q"

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest -q
........                                                                 [100%]
8 passed in 0.06 seconds
```

To show output,

- We should tell pytest not to capture & show the output. Use the below commands,

  "—capture=no" or "-s"

Output: The above command will show the output statements on the command prompt.

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --capture=no
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, dep
.5.4
collected 8 items

test_sample.py Running the demo1
.Running the demo2
.Running the test demo3
.Running the test demo4
.Running the test demo3
.Running the test demo4
.
test_sample1.py Running the demo7
.Running the demo8
.

========================== 8 passed in 0.08 seconds ===========================


D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest -s
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, depe
.5.4
collected 8 items

test_sample.py Running the demo1
.Running the demo2
.Running the test demo3
.Running the test demo4
.Running the test demo3
.Running the test demo4
.
test_sample1.py Running the demo7
.Running the demo8
.

========================== 8 passed in 0.08 seconds ===========================
```

Specifying tests / selecting tests to run

Run tests in a package/directory

- Use the below command to run all the modules under the package which is starting with “test_”.

Syntax: “>pytest package_name/”

Example:

```
D:\Training\PythonSelenium\PyTest_Practice>pytest test_Scripts/
============================= test session starts ================
```

Run test in a module:

- Use the below command to run all the test functions, which is present in the module and test classes.

Syntax: “>pytest package_name/test_module_name.py”

Example:

```
D:\Training\PythonSelenium\PyTest_Practice>pytest test_Scripts/test_sample.py
============================= test session starts ============================
```

Run the single test method:

- Each collected test is assigned a unique name which consist of the module filename followed by specifiers like class names, function names, separated by “::” characters.

To run a specific test within a module:

Syntax: “>pytest test_module_name.py:: test_function_name”

Example:

```
):\Training\PythonSelenium\PyTest_Practice>pytest test_Scripts/test_sample.py::test_demo1
==================== test session starts ==============================
```

To run a specific test within a class in a module:

Syntax: “>pytest test_module_name.py::TestClass:: test_function_name”

Example:

```
D:\Training\PythonSelenium\PyTest_Practice>pytest test_Scripts/test_sample.py::TestSample::test_demo3
============================ test session starts ============================
```

Run tests by keyword expressions

- To run tests which contain names that match the given *string expression*.

Syntax: ">pytest –k keyword"

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest -k demo
============================= test session starts =============================
```

Stop after First Failure

When you're trying to track down a bug, it can help to run all of your tests, so that you can draw conclusions based on what groups of tests are failing.

That said, sometimes you just want to work through failing tests one by one, or there are so many tests in your project that running them all every single time (not to mention sifting through all the output) would be too time-consuming. There's a nice simple switch to take care of this:

```
py.test -x
```

This will stop pytest after the first failure is encountered, saving you oodles of time. Is one just not enough? How about after two failures? No problem!

```
py.test --maxfail=2
```

- The above command stops the execution as soon as the two test methods fails.

Note: You can assign "N" numbers to max fail command, it will stop after that number tests fails.

Changing Defaults

Changing trace back printing:

```
py.test --tb=auto    # (default) 'long' tracebacks for the first and last
                     # entry, but 'short' style for the other entries
py.test --tb=long    # exhaustive, informative traceback formatting
py.test --tb=short   # shorter traceback format
py.test --tb=line    # only one line per failure
py.test --tb=native  # Python standard library formatting
py.test --tb=no      # no traceback at all
```

- When a test fails, pytest will attempt to do a trace back to give you some information on what went wrong. And when I say "some", I mean a lot… often too much. What if you could shorten that output, so that you didn't have to scroll for pages to see the results of all your tests? Well, try this:

"Pytest –tb=short"

Output:

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --tb=short
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, depen
.5.4
collected 8 items

test_sample.py ......                                                    [ 75%]
test_sample1.py ..                                                       [100%]

========================== 8 passed in 0.08 seconds ===========================
```

Using –tb will change pytest's trace back formatting to the format you specify. If "short" is still not short enough, you can do it one better:

"Pytest –tb=line"

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --tb=line
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, depen
.5.4
collected 8 items

test_sample.py ......                                                    [ 75%]
test_sample1.py ..                                                       [100%]

========================== 8 passed in 0.08 seconds ===========================
```

Or if you're looking for something a little more familiar, you can set pytest to use the Python standard trace back formatting, like so:

"Pytest –tb=native"

Output:

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --tb=native
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, depe
.5.4
collected 8 items

test_sample.py ......                                                    [ 75%]
test_sample1.py ..                                                       [100%]

========================== 8 passed in 0.08 seconds ===========================
```

Note: Very important

- Whenever there is a failure, pytest will give you the detailed information about the failure.

Example output:

```
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, depenc
.5.4
collected 8 items

test_sample.py::test_demo1 PASSED                                        [ 12%]
test_sample.py::test_demo2 PASSED                                        [ 25%]
test_sample.py::TestSample::test_demo3 FAILED                            [ 37%]
test_sample.py::TestSample::test_demo4 PASSED                            [ 50%]
test_sample.py::test_sample1 PASSED                                      [ 62%]
test_sample.py::test_sample2 PASSED                                      [ 75%]
test_sample1.py::test_demo7 PASSED                                       [ 87%]
test_sample1.py::test_demo8 PASSED                                       [100%]

================================== FAILURES ===================================
_____________________________ TestSample.test_demo3 ______________________________

self = <test_scripts.test_sample.TestSample object at 0x03A628F0>

    def test_demo3(self):
        print('Running the test demo3')
>       assert 1>3
E       assert 1 > 3

test_sample.py:11: AssertionError
---------------------------- Captured stdout call -----------------------------
Running the test demo3
===================== 1 failed, 7 passed in 0.19 seconds ======================
```

Dropping to PDB (Python Debugger) on failures

- Python comes with a built-in Python debugger called PDB. Pytest allows one to drop into the PDB prompt via a command line option:

"Pytest --pdb"

- The above command will open the debugger command prompt if there is any failure in the test method.
- This will invoke the Python debugger on every failure (or Keyboard Interrupt). Often you might only want to do this for the first failing test to understand a certain failure situation:

Note: Adding the failure statement in the script and run with the above command. Output will look like below,

```
1
2  def test_demo1():
3      print('Running the demo1')
4
5  def test_demo2():
6      print("Running the demo2")
7
8  class TestSample():
9
10     def test_demo3(self):
11         print('Running the test demo3')
12         assert 1>3
13
14     def test_demo4(self):
15         print('Running the test demo4')
16
17 def test_sample1():
18         print('Running the test demo3')
19
20 def test_sample2():
21     print('Running the test demo4')
22
23
```

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --pdb
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, dependency-0.
.5.4
collected 8 items

test_sample.py ..F
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> captured stdout >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
Running the test demo3
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> traceback >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

self = <test_scripts.test_sample.TestSample object at 0x045F3DF0>

    def test_demo3(self):
        print('Running the test demo3')
>       assert 1>3
E       assert 1 > 3

test_sample.py:12: AssertionError
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> entering PDB >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
> d:\workspace\automation\python\pytest\pytest\test_scripts\test_sample.py(12)test_demo3()
-> assert 1>3
(Pdb)
```

Note: now, we can debug the script.

- **In order to start debugging in the PDB window, we use the below commands**.

n(next) – step to the next line within the same function

s(step) – step to the next line in this function or called function

b(break) – set up new breakpoints without changing the code

p(print) – evaluate and print the value of an expression

c(continue) – continue execution and only stop when a breakpoint is encountered

q(quit) – quit the debugger/execution

Dropping to PDB (Python Debugger) at the start of a test even though there is no test failure:

Note: ***pytest –pdb*** will open the python debugger window only when there is a test failure. If there is no test failed means, it will not open the debugger window.

Pytest allows one to drop into the PDB prompt immediately at the start of each test via a command line option:

```
Pytest --trace
```

This will invoke the Python debugger at the start of every test.

Profiling test execution duration

To get a list of the slowest 10 test durations:

"Pytest –durations=N"

By default, pytest will not show test durations that are too small (<0.01s) unless **"-vv"** is passed on the command-line.

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --durations=2
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, dep
.5.4
collected 8 items

test_sample.py ......                                                    [ 75%]
test_sample1.py ..                                                       [100%]

========================== slowest 2 test durations ===========================

(0.00 durations hidden.  Use -vv to show these durations.)
========================== 8 passed in 0.17 seconds ===========================


D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest -vv --durations=2
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1 -- c:\users\pri
\python36-32\python.exe
cachedir: .pytest_cache
metadata: {'Python': '3.6.3', 'Platform': 'Windows-10-10.0.17134-SP0', 'Packages': {
ggy': '0.8.1'}, 'Plugins': {'xdist': '1.26.0', 'ordering': '0.6', 'metadata': '1.7.0
'dependency': '0.4.0', 'cov': '2.5.1', 'allure-pytest': '2.5.4'}, 'JAVA_HOME': 'C:\\
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, depend
.5.4
collected 8 items

test_sample.py::test_demo1 PASSED                                        [ 12%]
test_sample.py::test_demo2 PASSED                                        [ 25%]
test_sample.py::TestSample::test_demo3 PASSED                            [ 37%]
test_sample.py::TestSample::test_demo4 PASSED                            [ 50%]
test_sample.py::test_sample1 PASSED                                      [ 62%]
test_sample.py::test_sample2 PASSED                                      [ 75%]
test_sample1.py::test_demo7 PASSED                                       [ 87%]
test_sample1.py::test_demo8 PASSED                                       [100%]

========================== slowest 2 test durations ===========================
0.02s call     test_sample1.py::test_demo8
0.02s call     test_sample.py::test_demo2
========================== 8 passed in 0.11 seconds ===========================
```

Running Broken Tests Only

- Once you've encountered any errors in your tests, you want to focus on the failures and get a better understanding of what's causing the problems as opposed to spending time on running tests that are perfectly fine.
- Following are the commands used to re-run only the failed test functions.

“Pytest --lf” or “Pytest –last-failed”

Note:

1. If there is any failure, above commands will execute only those failed test functions will be re-executed.
2. If there is no failure, all the test functions will be re-executed.

Re-order Tests. Failures First

- Whenever you want to re-arrange the execution, based on the previous execution, I.E running the failed tests first and successful tests after, use the below command.

“Pytest -ff” or “Pytest —failed-first”

Re-order Tests. New test first

- Run tests from new files first, then the rest of the tests sorted by file.

“Pytest --nf” or “Pytest —new-first”

Cache:

- Show cache contents, don't perform collections or tests.

"Pytest —cache-show"

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --cache-show
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, depenc
.5.4
cachedir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts\.pytest_cache
------------------------------- cache values -------------------------------
cache\lastfailed contains:
  {}
cache\nodeids contains:
  ['test_sample.py::test_demo1',
   'test_sample.py::test_demo2',
   'test_sample.py::TestSample::test_demo3',
   'test_sample.py::TestSample::test_demo4',
   'test_sample.py::test_sample1',
   'test_sample.py::test_sample2',
   'test_sample1.py::test_demo7',
   'test_sample1.py::test_demo8']
cache\stepwise contains:
  []

========================== no tests ran in 0.02 seconds ==========================
```

Note: These commands will just collect the information about the tests but never executes.

- Remove all cache contents at start of test run.

"Pytest —cache-clear"

Detailed summary report

The "-r" flag can be used to display a "short test summary info" at the end of the test session, making it easy in large test suites to get a clear picture of all failures, skips, xfails, etc.

The "-r" options accepts a number of characters after it, with a used above meaning "all except passes".

Here is the full list of available characters that can be used:

- f - failed

- E - error
- s - skipped
- x - xfailed
- X - xpassed
- p - passed
- P - passed with output
- a - all except pP

More than one character can be used, so for example to only see failed and skipped tests, you can execute:

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest -rfs
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
```

Collection command:

"Pytest —collect-only" – only collect tests, don't execute them.

Output:

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --collect-only
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, depend
.5.4
collected 8 items
<Package D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>
  <Module test_sample.py>
    <Function test_demo1>
    <Function test_demo2>
    <Class TestSample>
        <Function test_demo3>
        <Function test_demo4>
    <Function test_sample1>
    <Function test_sample2>
  <Module test_sample1.py>
    <Function test_demo7>
    <Function test_demo8>
```

Generating output to files:

Pytest allows us to export the execution results into different format.

Example.

1. To XML format.
2. To HTML format.
3. To JSON format.
4. To TXT format.

Creating JUnitXML format files:

- To create result files to an xml file.

"Pytest –junitxml=path"

Output:

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest -q --junitxml=D:/results.xml
........                                                                 [100%]
-------------------- generated xml file: D:\results.xml ----------------------
8 passed in 0.08 seconds
```

- **Output in the XML will look like below**

```
<?xml version="1.0" encoding="utf-8"?>
    <testsuite errors="0" failures="0" name="pytest" skips="0" tests="8" time="0.078">
        <testcase classname="test_sample" file="test_sample.py" line="0" name="test_demo1" time="0.0">
            <system-out>Running the demo1</system-out>
        </testcase><testcase classname="test_sample" file="test_sample.py" line="3" name="test_demo2" time="0.015636444091796875">
            <system-out>Running the demo2</system-out>
        </testcase><testcase classname="test_sample.TestSample" file="test_sample.py" line="8" name="test_demo3" time="0.0">
            <system-out>Running the test demo3</system-out>
        </testcase><testcase classname="test_sample.TestSample" file="test_sample.py" line="11" name="test_demo4" time="0.0">
            <system-out>Running the test demo4</system-out>
        </testcase><testcase classname="test_sample" file="test_sample.py" line="14" name="test_sample1" time="0.0">
            <system-out>Running the test demo3</system-out>
        </testcase><testcase classname="test_sample" file="test_sample.py" line="17" name="test_sample2" time="0.0">
            <system-out>Running the test demo4</system-out>
        </testcase><testcase classname="test_sample1" file="test_sample1.py" line="5" name="test_demo7" time="0.0">
            <system-out>Running the demo7</system-out>
        </testcase><testcase classname="test_sample1" file="test_sample1.py" line="8" name="test_demo8" time="0.0">
        <system-out>Running the demo8</system-out>
        </testcase>
    </testsuite>
```

Generating the HTML report:

- Pytest results can be exported into HTML file.

- In order to export the results to HTML file,

First, we need to install the "**pytest-html**" plugin.

To install pytest-html,

```
s>pip install pytest-html
```

Use the below command to export the result to HTML.

"Pytest –html=path"

```
D:\Workspace\Automation\Python\PyTest\pytest\test_scripts>pytest --html=D:/report.html
============================= test session starts =============================
platform win32 -- Python 3.6.3, pytest-4.1.1, py-1.5.3, pluggy-0.8.1
rootdir: D:\Workspace\Automation\Python\PyTest\pytest\test_scripts, inifile:
plugins: xdist-1.26.0, ordering-0.6, metadata-1.7.0, html-1.20.0, forked-0.2, dependency
.5.4
collected 8 items

test_sample.py ......                                                    [ 75%]
test_sample1.py ..                                                       [100%]

--------------------- generated html file: D:\report.html ---------------------
========================== 8 passed in 0.28 seconds ===========================
```

Test Report

File | D:/report.html

report.html

Report generated on 03-Apr-2019 at 23:33:06 by pytest-html v1.20.0

Environment

JAVA_HOME	C:\Program Files\Java\jdk1.8.0_181
Packages	{'pytest': '4.1.1', 'py': '1.5.3', 'pluggy': '0.8.1'}
Platform	Windows-10-10.0.17134-SP0
Plugins	{'xdist': '1.26.0', 'ordering': '0.6', 'metadata': '1.7.0', 'html': '1.20.0', 'forked': '0.2', 'dependency': '0.4.0', 'cov': '2.5.1', 'allure-pytest': '2.5.4'}
Python	3.6.3

Summary

8 tests ran in 0.09 seconds.

(Un)check the boxes to filter the results.

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

Results

Show all details / Hide all details

Result	Test	Duration
Passed *(show details)*	test_sample.py::test_demo1	0.00
Passed *(show details)*	test_sample.py::test_demo2	0.00
Passed *(show details)*	test_sample.py::TestSample::test_demo3	0.00
Passed *(show details)*	test_sample.py::TestSample::test_demo4	0.00
Passed *(show details)*	test_sample.py::test_sample1	0.00
Passed *(show details)*	test_sample.py::test_sample2	0.00
Passed *(show details)*	test_sample1.py::test_demo7	0.00
Passed *(show details)*	test_sample1.py::test_demo8	0.02

Generating the JSON report:

- In order to generate the execution results in JSON format, we need to use the "pytest reportlog" plugin.

  Install pytest reportlog:

- Open the command prompt and type the below command, which will install the pytest reportlog.

```
C:\Users>pip install pytest-reportlog
```

After installing the pytest reportlog, run the project using the below command.

Syntax:

- Pytest –report-log=path/filename.log

Example:

```
C:\Users>pytest --report-log=D:/report.json
```

Generating the JSON report:

- Make sure pytest-reportlog has been installed in the system. If it is not installed, install pytest-reportlog plugin first and run the pytest project.

Syntax:

- Pytest –report-log=path/filename.log

Example:

```
C:\Users>pytest --report-log=D:/report.txt
```

Sending test report to online pastebin service

Creating a URL for each test execution for all the test.

```
D:\Weekend_Selenium\pytest_practice>pytest --pastebin=all
```

When you execute the above command, it will generate the URL of the current test execution.

```
D:\Weekend_Selenium\pytest_practice>pytest --pastebin=all
============================= test session starts =============================
platform win32 -- Python 3.8.1, pytest-5.3.5, py-1.8.1, pluggy-0.13.1 -- c:\us
ers\jayapriyapramod\appdata\local\programs\python\python38-32\python.exe
rootdir: D:\Weekend_Selenium\pytest_practice
plugins: allure-pytest-2.8.10, dependency-0.5.1, forked-1.1.3, html-2.0.1, met
adata-1.8.0, ordering-0.6, reportlog-0.1.0, xdist-1.31.0
collected 8 items

test_demo\test_sample.py ........                                        [100%]

============================== 8 passed in 0.34s ==============================
------------------- Sending information to Paste Service --------------------
pastebin session-log: https://bpaste.net/show/BCNA
```

Now copy and paste the URL in any of the browser.

Possible pytest exit codes

Running pytest can result in six different exit codes:

Exit code 0: All tests were collected and passed successfully.

Exit code 1: Tests were collected and run but some of the tests failed.

Exit code 2: Test execution was interrupted by the user.

Exit code 3: Internal/system error happened while executing tests.

Exit code 4: pytest command line usage error.

Exit code 5: No tests were collected.