**WebDriver in selenium**

Selenium WebDriver is a popular web-based automation testing framework that is primarily used for automating tasks related to Web UI testing.

Selenium WebDriver does not interact directly with the web elements on a page. A browser-specific Selenium WebDriver acts as the bridge between the test script and the web browser.

[Selenium locators](https://www.lambdatest.com/learning-hub/selenium-locators) are used for locating elements on the page so that appropriate methods can be used for interacting with the element.

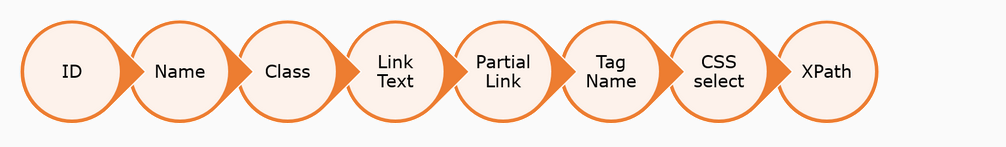
**Methods of selenium webdriver class**

|  |  |  |
| --- | --- | --- |
| **Method** | **Syntax** | **Description** |
| get() | driver.get(url) |
| getCurrentUrl(); | driver.getCurrentUrl() |
| getTitle() | driver.getTitle() |
| findElements() | driver.findElements(By by) |
| findElement() | driver.findElement(By by) |
| [add\_cookie](https://www.geeksforgeeks.org/add_cookie-driver-method-selenium-python/) |  | Adds a cookie to your current session. | |
| [back](https://www.geeksforgeeks.org/back-driver-method-selenium-python/?ref=rp) |  | Goes one step backward in the browser history. | |
| [close](https://www.geeksforgeeks.org/close-driver-method-selenium-python/?ref=rp) | driver.back() | Closes the current window. | |
| [create\_web\_element](https://www.geeksforgeeks.org/create_web_element-driver-method-selenium-python/?ref=rp) | driver.close() | Creates a web element with the specified element\_id. | |
| [delete\_all\_cookies](https://www.geeksforgeeks.org/delete_all_cookies-driver-method-selenium-python/?ref=rp) |  | Delete all cookies in the scope of the session. | |
| [delete\_cookie](https://www.geeksforgeeks.org/delete_cookie-driver-method-selenium-python/?ref=rp) |  | Deletes a single cookie with the given name. | |
| [delete\_cookie](https://www.geeksforgeeks.org/delete_cookie-driver-method-selenium-python/?ref=rp)  [execute\_async\_script](https://www.geeksforgeeks.org/execute_async_script-driver-method-selenium-python/) |  | Deletes a single cookie with the given name.  Asynchronously Executes JavaScript in the current window/frame. | |
|  |
| [execute\_script](https://geeksforgeeks.org/execute_script-driver-method-selenium-python/) |  | Synchronously Executes JavaScript in the current window/frame. | |
| [forward](https://www.geeksforgeeks.org/forward-driver-method-selenium-python/) |  | Goes one step forward in the browser history. | |
| [fullscreen\_window](https://www.geeksforgeeks.org/fullscreen_window-driver-method-selenium-python/) |  | Invokes the window manager-specific ‘full screen’ operation | |
| [get\_cookie](https://www.geeksforgeeks.org/add_cookie-driver-method-selenium-python/?ref=rp) |  | Get a single cookie by name. Returns the cookie if found, None if not. | |
| [get\_cookies](https://www.geeksforgeeks.org/get_cookies-driver-method-selenium-python/?ref=rp) |  | Returns a set of dictionaries, corresponding to cookies visible in the current session. | |
| [get\_log](https://www.geeksforgeeks.org/get_log-driver-method-selenium-python/?ref=rp) |  | Gets the log for a given log type | |
| [get\_screenshot\_as\_base64](https://www.geeksforgeeks.org/get_screenshot_as_base64-driver-method-selenium-python/?ref=rp) | driver.get\_log(log\_type) | Gets the screenshot of the current window as a base64 encoded string which is useful in embedded images in HTML. | |
| [get\_screenshot\_as\_file](https://www.geeksforgeeks.org/get_screenshot_as_file-driver-method-selenium-python/?ref=rp) |  | Saves a screenshot of the current window to a PNG image file. | |
| [get\_screenshot\_as\_png](https://www.geeksforgeeks.org/get_screenshot_as_png-driver-method-selenium-python/?ref=rp) |  | Gets the screenshot of the current window as a binary data. | |
| [get\_window\_position](https://www.geeksforgeeks.org/get_window_position-driver-method-selenium-python/?ref=rp) |  | Gets the x, y position of the current window. | |
| [get\_window\_rect](https://www.geeksforgeeks.org/get_window_rect-driver-method-selenium-python/?ref=rp) |  | Gets the x, y coordinates of the window as well as height and width of the current window. | |
| [get\_window\_rect](https://www.geeksforgeeks.org/get_window_rect-driver-method-selenium-python/?ref=rp)  [get\_window\_size](https://www.geeksforgeeks.org/get_window_size-driver-method-selenium-python/) |  | Gets the x, y coordinates of the window as well as height and width of the current window.  Gets the width and height of the current window. | |
|  |
| [implicitly\_wait](https://www.geeksforgeeks.org/implicitly_wait-driver-method-selenium-python/?ref=rp) |  | Sets a sticky timeout to implicitly wait for an element to be found, | |
| [maximize\_window](https://www.geeksforgeeks.org/maximize_window-driver-method-selenium-python/?ref=rp) |  | Maximizes the current window that webdriver is using | |
| [minimize\_window](https://www.geeksforgeeks.org/minimize_window-driver-method-selenium-python/?ref=rp) |  | Invokes the window manager-specific ‘minimize’ operation | |
| [quit](https://www.geeksforgeeks.org/quit-driver-method-selenium-python/) |  | Quits the driver and closes every associated window. | |
| [refresh](https://www.geeksforgeeks.org/refresh-driver-method-selenium-python/) | driver.quit() | Refreshes the current page. | |
| [set\_page\_load\_timeout](https://www.geeksforgeeks.org/set_page_load_timeout-driver-method-selenium-python/?ref=rp) |  | Set the amount of time to wait for a page load to complete before throwing an error. | |
| [set\_script\_timeout](https://www.geeksforgeeks.org/set_script_timeout-driver-method-selenium-python/?ref=rp) | driver.set\_page\_load\_timeout(time\_To\_wait) | Set the amount of time that the script should wait during an execute\_async\_script call before throwing an error. | |
| [set\_window\_position](https://geeksforgeeks.org/set_window_position-driver-method-selenium-python/) |  | Sets the x, y position of the current window. (window.moveTo) | |
| [set\_window\_rect](https://geeksforgeeks.org/set_window_rect-driver-method-selenium-python/) |  | Sets the x, y coordinates of the window as well as height and width of the current window. | |
| [current\_url](https://geeksforgeeks.org/current_url-driver-method-selenium-python/) |  | Gets the URL of the current page. | |
| [current\_window\_handle](https://geeksforgeeks.org/current_window_handle-driver-method-selenium-python/) |  | Returns the handle of the current window. | |
| [page\_source](https://geeksforgeeks.org/page_source-driver-method-selenium-python/) |  | Gets the source of the current page. | |
| [title](https://geeksforgeeks.org/title-driver-method-selenium-python/) |  | Returns the title of the current page. | |
|  | driver.title | Returns the title of the current page. | |

################################################################################# Elementslocator in Selenium #

################################################################################

Selenium offers 8 types of locators as follows, which help in uniquely identifying an element in a web page.



#ID locator#

Selenium uses the id attribute of an HTML element for locating a specific element.

The id of an element should be unique within a page, so the id selector is used to select one unique element.

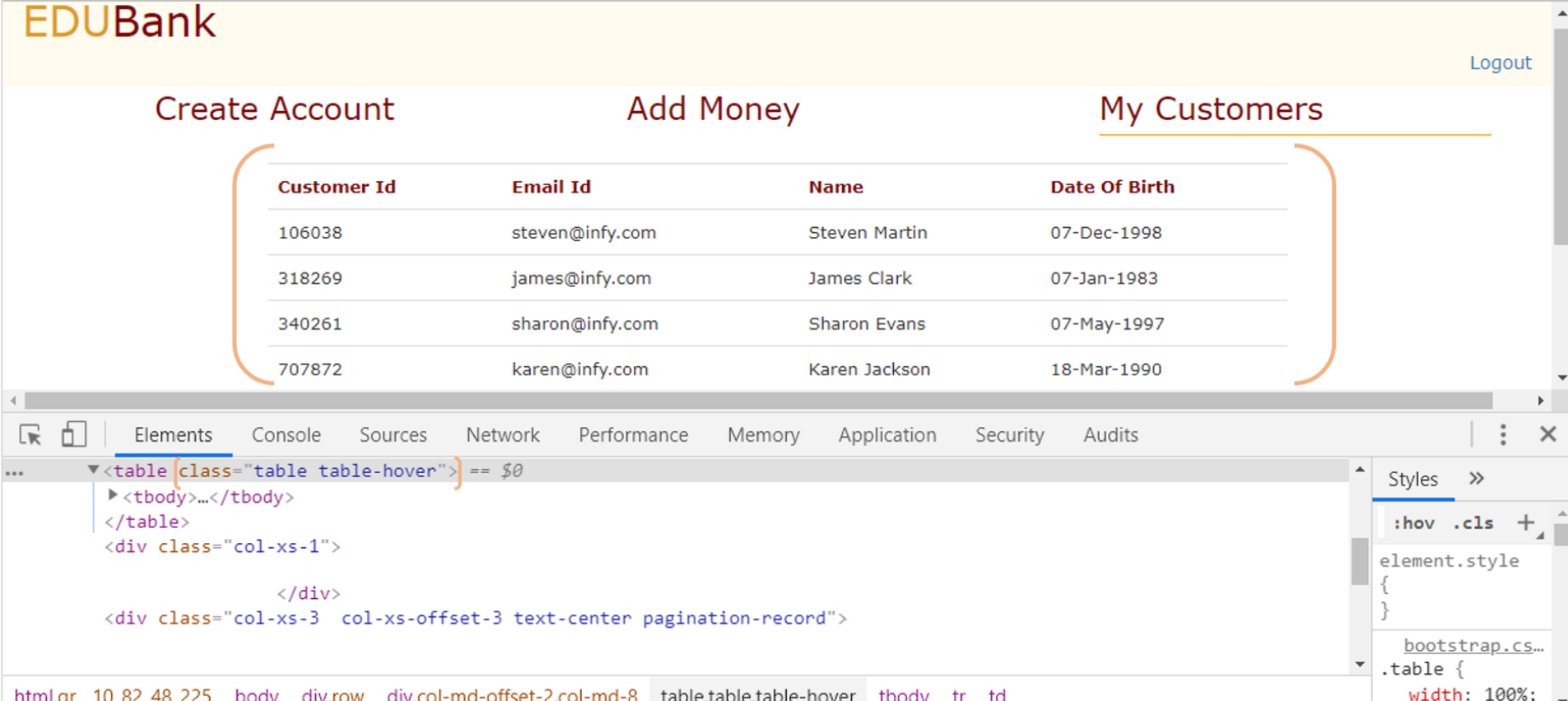
#Name locator#

The name attribute specifies a name for the element. This name attribute can be used to locate the element in a web page.

# Class name #

This locator locates the element which matches the value specified in the attribute name “class”.They basically deal with the styling options added to the element (They might be inbuilt styles or customized styles added by the developer).

In the image given here, class= "table table-hover" is added to the table.

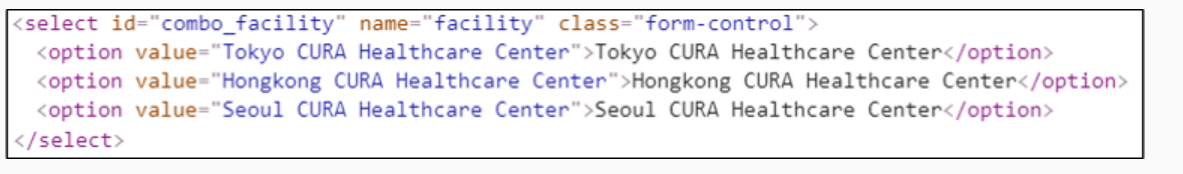


# Tag name #

This locator is used to find the elements matching the specified tag name.

This locator can be **used whenever there is no ID or Class Name or Name.** **Also when there is a need to locate an element and extract the elements nested within them.**

In the below image, suppose you want all the options to be extracted, tag name option can be used.



**# Partial Link text #**

In certain cases, we may need to find links by a portion of the text in a Link Text element. In such situations, we can use Partial Link Text to locate elements.

################################################################################

# WebElements in selenium #

################################################################################

################################################################################# PyTest #

################################################################################

pytest is a framework that makes building simple and scalable tests easy.

pytest will run all files of the form *test\_\*.py or \*\_test.py* in the current directory and its subdirectories if file name is not specified

pytest [options] [file\_or\_dir] [file\_or\_dir] [...] ---------------- Usage

**Example:**

pytest test\_file1.py ---------- Will run all test cases from test\_file1.py.

pytest --------- Will run all test cases from all files of current and sub directories.

**Conventions for Python test discovery**

* If no arguments are specified, then collection starts from testpaths (if configured) or the current directory.
* In those directories, search for test\_\*.py or \*\_test.py files, imported by their test package name.
* From those files, collects test items (test cases):

1. *test prefixed test functions or methods outside of class*
2. *test prefixed test functions or methods inside Test prefixed test classes* (without an \_\_init\_\_ method)

#####################

# Pytest Reporting flags #

#####################

-r

show extra test summary info as specified by chars: **(f)**ailed, **(E**)rror, (**s)**kipped, **(x)**failed, **(X)**passed,**(p)**assed, **(P)**assed with output, **(a)**ll except passed **(p/P),** or **(A)**ll. (w)arnings are enabled by default (see --disable-warnings), 'N' can be used to reset the list. (default: 'fE').

-rA ---- Shows test summary for all test cases

-rf ----- Shows test summary for failed

--maxfail=num

exit after first num failures or errors.

--duration=N

how N slowest setup/test durations (N=0 for all).

--no-header disable header

-- no-summary disable summary

-q, -quite decrease verbosity.

########################

# Pytest general parameters #

########################

-k expression

only run tests which match the given substring expression. An expression is a python evaluable expression where all names are substring-matched against test names and their parent classes.

**Example 1**:

-k 'test\_method or test\_other' matches all test functions and classes whose name contains 'test\_method' or 'test\_other'

**Example 2:**

-*k 'not test\_method'* matches those that don't contain 'test\_method' in their names. -k 'not test\_method and not test\_other' will eliminate the matches.

-x

exit instantly on first error or failed test.

-marker

show markers (builtin, plugin and per-project ones).

-lf, -last-failed

rerun only the *tests that failed at the last run (or all if none failed)*

-v

Increase verbosity

####################################

# Generating pytest report in html format #

####################################

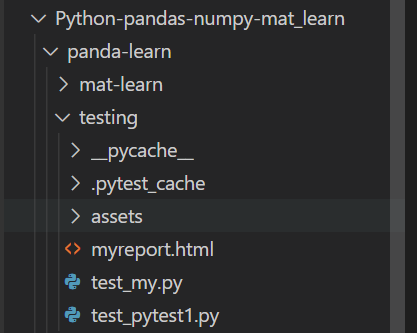
For generating report into html format then we need to use third party package 'pytest-html'.

For its usage check documentation.

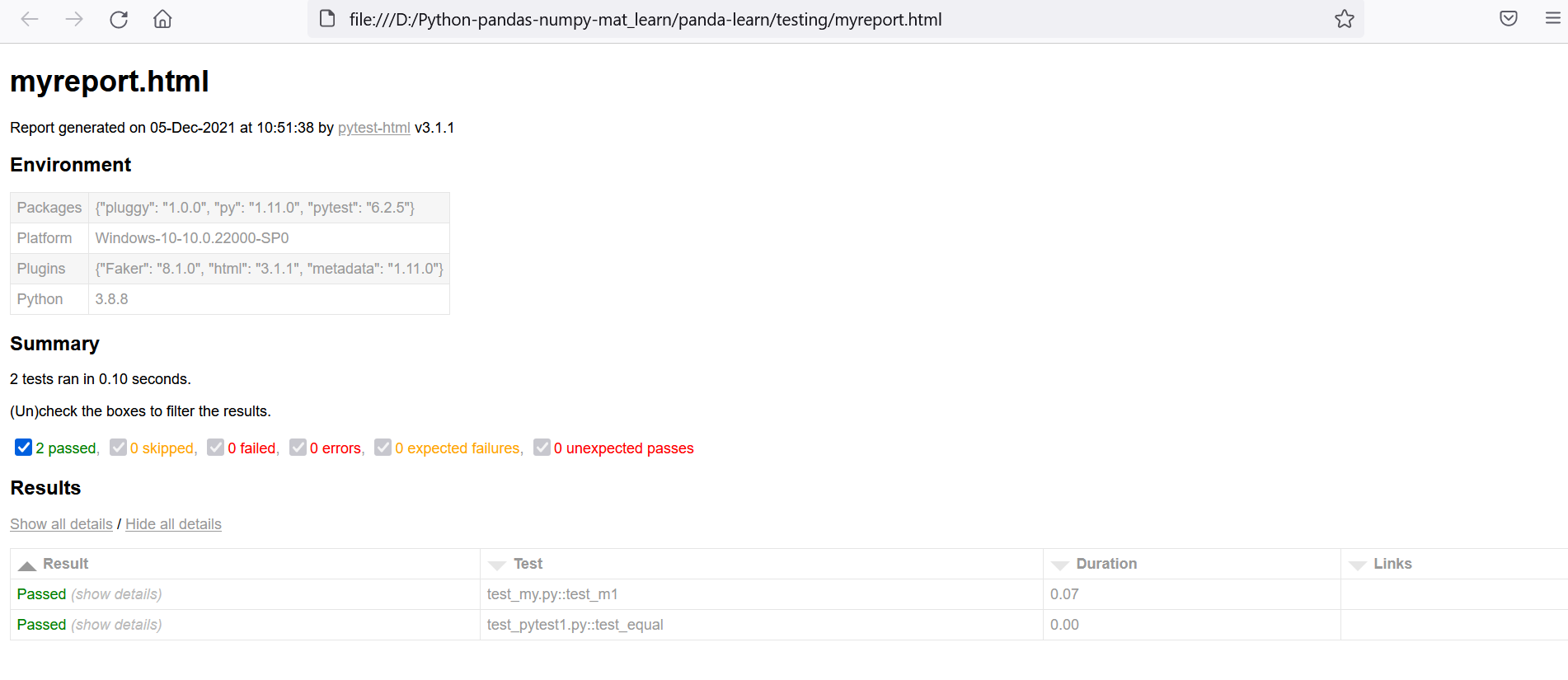
pytest [running options] –html='report\_file\_path.html' ----------- Generating report.

**Example**:

pytest --html="myreport.html"



Report



#################

# 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 error happened while executing tests

**Exit code 4:** pytest command line usage error

**Exit code 5:** No tests were collected

###############

# PyTest Marker #

###############

Markers can be used to apply meta data to test functions (but not fixtures), which can then be accessed by fixtures or plugins. There are two types of markers in python.

Markers are used to set various features/attributes to test functions. Pytest provides many inbuilt markers such as xfail, skip and parametrize. Apart from that, users can create their own marker names

1. In built markers
2. Custom markers

###################

# How to use marker #

###################

We can use it with function (test function name) name.

@pytest.mark.marker\_name ----- for custom and in built both

def test\_testname():

#

#####################

# Pytest inbuilt markers #

#####################

We have below inbuilt marker.

pytest.mark.filterwarnings(filter)

filter ----- String , A warning specification string, which is composed of contents of the tuple (action, message, category, module, lineno)

pytest.mark.skip(reason)

reason ------- String, Unconditional skipping and test function.

pytest.mark.skipif(comdtion,reason)

condition ---- conditional statement to decide skipping of test

reason ----- String, some message

This is for skipping a test function when some condition arrives.

pytest.mark.usefixtures(\*names)

name ------ name of fixture to use as string.

pytest.mark.xfail(condition=None, \*, reason=None, raises=None, run=True, strict=False)

*condition (bool or str)* ---- Condition for marking the test function as xfail

*reason (str)* ------ Reason why the test function is marked as xfail.

*raises (Type[Exception])* ----- Exception subclass expected to be raised by the test function

*run (bool)* ----- If the test function should actually be executed.If False, the function will always xfail and will not be executed (useful if a function is segfaulting)

pytest.mark.parametarize(param1,param2..paramn,[(val1,val2..valn),…(val1,val2..valn)])

This is used for passing some values to test case. Test cases will execute for all sets of values.

param1,param2..paramn ----- These are parameters

[(val1,val2..valn),…(val1,val2..valn)] ---- These are list of tupple. It’s list of values for each param.

################

# Custom marker #

################

If we are using any custom marker then we need to register it first. Steps to register

1. Create pytest.ini file in current path or project
2. Add below entries into it

[pytest]

markers =

maker1 : “some description of marker 1”

maker2 : “some description of marker 2”

.

.

maker n : “some description of marker n”

**description of marker are optional.**

**Running tests with marker name**

Use -m flag to run any specific marker test function.

pytest -m “mymarker” ------ Run only that test fuction which have mymarker

**Example:**

pytest.init file

[pytest]

markers =

    smoke: 'this is smoke case'

    login:'this is login marker'

import py

import pytest

@pytest.mark.smoke #Custom smoke marker, it will run

def test\_m1():

    print('this is 1')

    assert 1==1

@pytest.mark.skip #Uncoditional skip, it will be skipped

def test\_login():

    print('logging in')

    assert 1==2

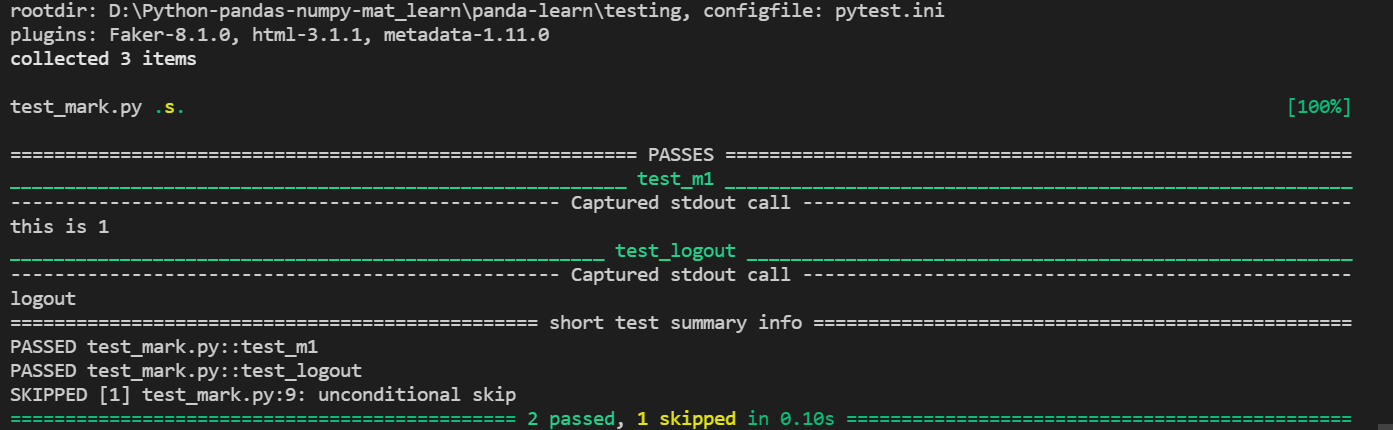
@pytest.mark.skipif(3<2,reason='1<2, skkiping it') #coditional skip, 3!<2 so will run

def test\_logout():

    print('logout')

    assert 2==2

Result:



############

# pytest.ini #

############

Pytest.ini is configuration file, using this file we can do lot of configurations. Check below link for all available configuration options-

<https://docs.pytest.org/en/6.2.x/reference.html#ini-options-ref>

Some common are-

**markers**

this is used for registering custom markers.

[pytest]

addopts = --strict-markers

*markers =*

*slow*

*serial*

**minversion**

Specifies a minimal pytest version required for running tests.

# content of pytest.ini

[pytest]

*minversion* = 3.0 # will fail if we run with pytest-2.8

**adopts**

Add the specified OPTS (*Space separated value*) to the set of command line arguments (pytest –help --- to check all command line argument) as if they had been specified by the user.

[pytest]

addopts = --maxfail=2 -rf # exit after 2 failures, report fail info

**filterwarnings**

Sets a list of filters and actions that should be taken for matched warnings. By default all warnings emitted during the test session will be displayed in a summary at the end of the test session.

[pytest]

filterwarnings =

error

ignore::DeprecationWarning

################################################################################

Fixtures in pytest

################################################################################

Fixtures are nothing but a function (any python function) decorated with @pytest.fixtures decorator.

Fixtures are *used to feed some data to the tests* such as database connections, URLs to test and some sort of input data.

Advantage of fixture is that it provides code reusability

Fixtures are of two types ---- 1.) Built-in fixtures 2.) Custom fixtures

###############

# Custom fixture #

###############

Custom fixture is nothing but a python function decorated with @pytest.fixture decorator.

@pytest.fixture

def login():

#

**Note:**

1. Here name of function doesn’t need to be in test\_\* or \*\_test format.
2. We can return data from fixture also

#################

#How to use fixture#

#################

Use the fixture name as argument in test function. e.g-

**def test\_login(login):** #login is fixture

#

Important points:

1. A fixture can use/request another fixture
2. A test function can take/accept/request any number of fixture
3. Fixtures are reusable
4. Fixtures can be request more than once per test (return values are cached)
5. autouse fixture don’t need to requested by any function.

######################

# Creating custom fixture#

######################

Let’s take a look on all argument that a function can take while creating fixture.

@**fixture**(fixture\_function: \_FixtureFunction, \*, scope: Union[\_Scope, Callable[[str, Config], \_Scope]] = 'function', params: Optional[Iterable[object]] = None, autouse: bool = False, ids: Optional[Union[Iterable[Union[None, str, float, int, bool]], Callable[[Any], Optional[object]]]] = None, name: Optional[str] = None)

@fixture(fixture\_function: None = None, \*, scope: Union[\_Scope, Callable[[str, Config], \_Scope]] = 'function', params: Optional[Iterable[object]] = None, autouse: bool = False, ids: Optional[Union[Iterable[Union[None, str, float, int, bool]], Callable[[Any], Optional[object]]]] = None, name: Optional[str] = 'None')

Description of parameters are –

* scope

The scope for which this fixture is shared; one of "function" (default), "class", "module", "package" or "session".

* Params

n optional list of parameters which will cause multiple invocations of the fixture function and all of the tests using it. The current parameter is available in request.param.

* autouse

If True, the fixture func is activated for all tests that can see it. If False (the default), an explicit reference is needed to activate the fixture.

* name

The name of the fixture. This defaults to the name of the decorated function.

* ids

List of string ids each corresponding to the params so that they are part of the test id. If no ids are provided they will be generated automatically from the params.

################################################################################

Pytest Function

################################################################################

In pytest we have some function which are generally used in testing purpose. We can check them all on below link.

<https://docs.pytest.org/en/6.2.x/reference.html#functions>

approx(expected, rel=None, abs=None, nan\_ok: bool = False)

Asserts that two numbers (or two sets of numbers) are equal within some tolerance.

It can be used with many data types --- dictionary, list, numpy array etc

Due to the intricacies of floating-point arithmetic, numbers that we would intuitively expect to be equal are not always, e.g ---- .1+.2==.3 #false in python

**with scalar/values-**

0.1 + 0.2 == approx(0.3)

**Sequence of values/data**

(0.1 + 0.2, 0.2 + 0.4) == approx((0.3, 0.6))

**Dictionary values**

{'a': 0.1 + 0.2, 'b': 0.2 + 0.4} == approx({'a': 0.3, 'b': 0.6})

**numpy array**

np.array([0.1, 0.2]) + np.array([0.2, 0.4]) == approx(np.array([0.3, 0.6]))

**numpy array against scalar**

np.array([0.1, 0.2]) + np.array([0.2, 0.1]) == approx(0.3)

fail(msg: str = '', pytrace: bool = True)

Explicitly fail an executing test with the given message.

**msg (str)** – The message to show the user as reason for the failure.

**pytrace (bool)** – If False, msg represents the full failure information and no python traceback will be reported.

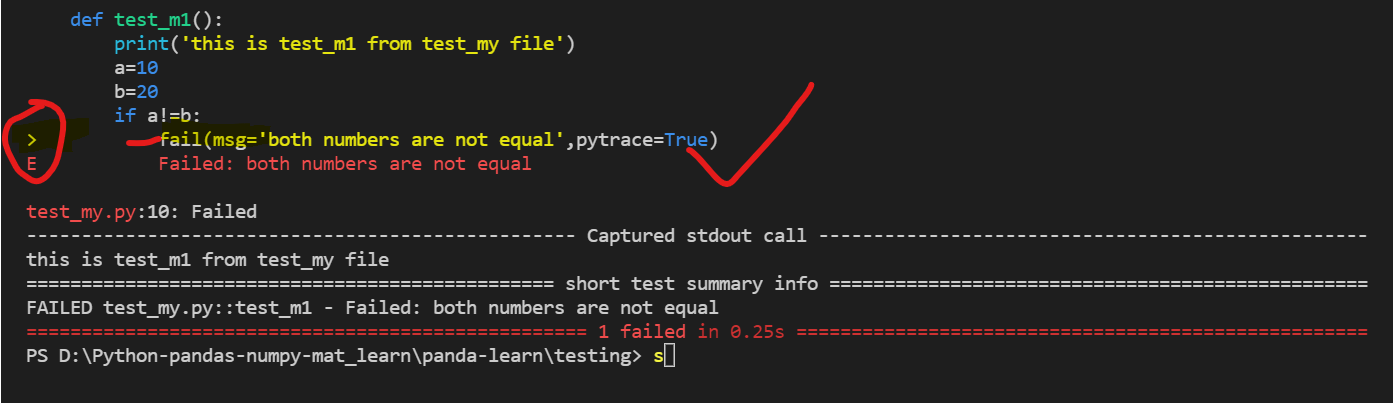
def test\_m1():

    a=10

    b=20

    if a!=b:

        fail(msg='both numbers are not equal',pytrace=True)



skip(msg[, allow\_module\_level=False])

Skip an executing test with the given message.

This function should be called only **during testing (setup, call or teardown)** or during collection by using the allow\_module\_level flag.

This function can be called in doctests as well.

**allow\_module\_level:** (bool) – Allows this function to be called at module level, skipping the rest of the module. Defaults to False.

**msg :** Message to give, str

importorskip(modname: str, minversion: Optional[str] = None, reason: Optional[str] = None)

Import and return the requested module modname, or skip the current test if the module cannot be imported.

**modname**: (str) – The name of the module to import.

**minversion**: (str) – If given, the imported module’s \_\_version\_\_ attribute must be at least this minimal version, otherwise the test is still skipped

**reason**: (str) – If given, this reason is shown as the message when the module cannot be imported.

xfail(reason: str = '')

Imperatively xfail an executing test or setup function with the given reason.

exit(msg: str, returncode: Optional[int] = None)

Exit testing process.

#########################################

# Creating test class #

#########################################

We can create any test class same as python class additional below naming convention:-

1. Name of class with start with ‘TEST’
2. All methods (for testing purpose) will start or end with ‘test’

e.g-

class TestClass:

        def test\_tc1(self): #self we need to pass as this will be passed by pytest

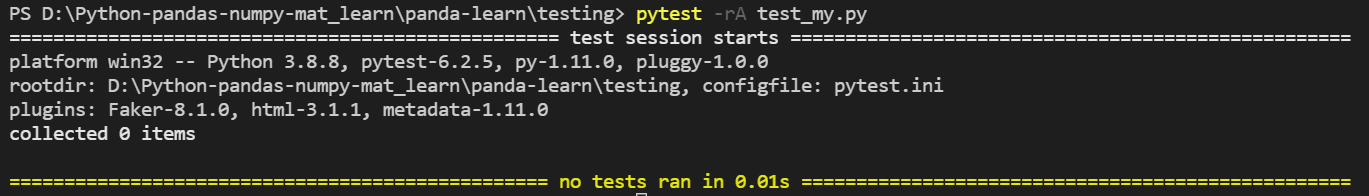
        assert 1==1 #self will be passed by default by pytest

    def test\_tc2(self):

        assert 2==1

**Note:**

1. For each method we need to use self as argument b/c pytest will pass one argument for those test method (test methods are instance method)
2. All of our test method should be instance methods else pytest will not capture them and will not be executed
3. class TestClass:
5. @classmethod
6. def test\_m3(cls): #defining test function to be class method
7. print('method m4')
8. assert 4==4
9. TestClass.test\_m3()



Test method (declared as class method) is not executed as we can see.

##############################

# Test setup and teardown #

##############################

We can do setup and teardown at class level, module level, method level.

**Module level setup/teardown**

If you have multiple test functions and test classes in a single module you can optionally implement the following fixture methods *which will usually be called once for all the functions:*

(gets call only once when file executes)

def setup\_module(module):

""" setup any state specific to the execution of the given module."""

def teardown\_module(module):

"""teardown any state that was previously setup with a setup\_module

method. At end of file execution

"""

**Class level setup/teardown**

Following methods are called at class level before and after all test methods of the class are called:

*Before each test method execution and after execution of class.*

@classmethod

def setup\_class(cls):

"""setup any state specific to the execution of the given class (which

usually contains tests).

"""

@classmethod

def teardown\_class(cls):

"""teardown any state that was previously setup with a call to

setup\_class.

"""

class TestClass:

    @classmethod

    def setup\_class(cls): #Should be called before file execution start

        print('steup\_class')

    @classmethod #should be called end of file execution

    def teardown\_class(cls):

        print('teardown\_class')

    def setup\_method(self, method): #should be called before of each method

        print('setup\_method')

    def teardown\_method(self, method): #should be called after of each method

        print('teardown\_method')

    def test\_tc1(self):

        print('tc1')

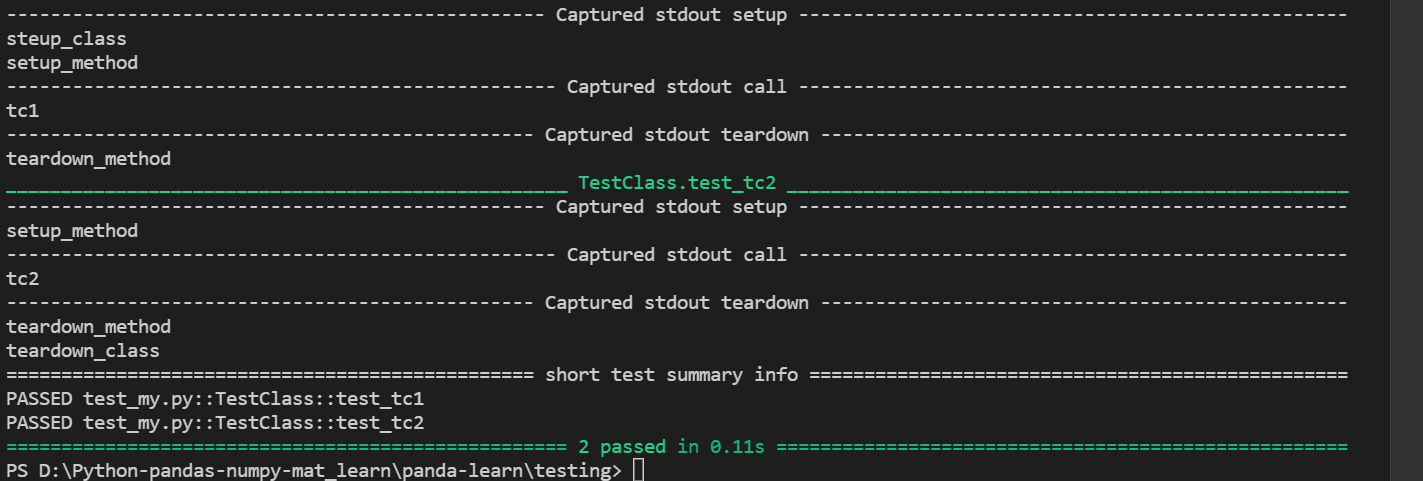
        assert 1==1

    def test\_tc2(self):

        print('tc2')

        assert 2==2

Output:



**Method and function level setup/teardown**

def setup\_method(self, method):

"""setup any state tied to the execution of the given method in a

class. setup\_method is invoked for every test method of a class.

"""

def teardown\_method(self, method):

"""teardown any state that was previously setup with a setup\_method

call.