**SWE302 Project Report**

**Unit-Testing Framework**

**Summary:**

This report describes the architecture, components, how to use it, and benefits of a simple Python-based unit-testing framework. The project consists of assertions.py, core.py, runner.py, and test\_samples.py files, which contain sample tests.

**Why we create this code**

* Providing a simple standalone solution for test automation
* Defining and using our own assertion methods without the need for external libraries (pytest/unittest)

**Target audience**

* New developers learning Python
* Those who want to write tests without using external dependencies in small-scale projects

You must create the root directory of the project as like this:

unit-testing/  
├── framework/  
│ ├── core.py # Registers and provides test functions  
│ ├── assertions.py # Custom assertion logic grouped by purpose  
│ └── runner.py # Main test runner  
├── tests/  
│ └── test\_samples.py # Example tests using all assertion types  
└── README.md # Project documentation

follow this steps for creating root directory

open PowerShell or CMD

cd "$HOME\Desktop"

create main Project folder

mkdir unit-testing

cd unit-testing

create sub folders

mkdir framework, tests

create empty “\_\_init\_\_.py” folder

ni framework\\_\_init\_\_.py -ItemType File

ni tests\\_\_init\_\_.py -ItemType File

and create code folders

ni framework\core.py -ItemType File

ni framework\assertions.py -ItemType File

ni framework\runner.py -ItemType File

ni tests\test\_samples.py -ItemType File

How can we run this code?

control the python enviroment

python –version

run the “runner.py” file

python framework/runner.py

Go to the project folder in PowerShell or Terminal

cd "$HOME\Desktop\unit-testing"

The results of the tests will eventually give a summary like this:

>>> Test Summary:

>>> Total Tests: 26

>>> Total Passed Tests: 24

>>> Total Failed Tests: 2

>>> TESTING COMPLETED <<<

File explainations

Framework/assertions.py

Contains customized assertion classes that you will use in tests.

Example:

class Comparison:

    def \_\_init\_\_(self, a, b):

        self.a = a

        self.b = b

    def assertEqual(self):

        if self.a != self.b:

            raise AssertionError(f"{self.a} is not equal to {self.b}")

(small part of this file)

This class contains a simple assertEqual function that checks if two values ​​are equal.

Framework/core.py

@test decorator saves test functions in a list.

These functions are retrieved with get\_tests().

Example:

\_registered\_tests = []

def test(func):

    """

    decorator to register all tests to the private

    \_registered\_tests array

    """

    \_registered\_tests.append(func)

    print("test?")

    return func

def get\_tests():

    """

    returns all the registered tests

    """

    return \_registered\_tests

this way you will be able to run all test functions automatically

framework/runner.py

Example:

def runTests():

    tests = get\_tests()

    total\_tests = len(tests)

    passed, failed = 0,0

    for test in tests:

        try:

            test()

            print(f"{test.\_\_name\_\_}\n>>> PASS <<<")

            passed += 1

        except AssertionError as e:

            print(f"{test.\_\_name\_\_}\n>>> FAIL: {str(e)} <<<")

            failed += 1

All test functions are executed one by one and the PASS/FAIL result is printed

Tests/test\_samples.py

This is where you write tests for example tests like assertEqual, assertNotEqual, assertTrue are written here.

import time

from framework.assertions import Comparison, Truthiness, Identification, Collections, Exceptions, Timing

from framework.core import test

import sys

import os

# Proje dizinini sisteme ekleyin

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

# COMPARISON FAMILY TESTS

@test

def test\_comparison\_equal():

    comparison = Comparison(1, 1)

    comparison.assertEqual()

if this test passes it writes “PASS” to the console if it fails it writes “FAIL”

Expected output is:

test?

test?

test?

test?

test\_comparison\_equal

>>> PASS <<<

test\_comparison\_not\_equal

>>> PASS <<<

test\_comparison\_greater

>>> PASS <<<

test\_comparison\_greater\_equal

>>> PASS <<<

test\_comparison\_less

>>> PASS <<<

test\_comparison\_less\_equal

>>> PASS <<<

test\_truthiness\_true

>>> PASS <<<

test\_truthiness\_false

>>> PASS <<<

test\_truthiness\_none

>>> PASS <<<

test\_truthiness\_not\_none

>>> PASS <<<

test\_identification\_is

>>> PASS <<<

test\_identification\_is\_not

>>> PASS <<<

test\_identification\_instance\_of

>>> PASS <<<

test\_identification\_not\_instance\_of

>>> PASS <<<

test\_collections\_in

>>> PASS <<<

test\_collections\_not\_in

>>> PASS <<<

test\_collections\_count\_equal

>>> PASS <<<

test\_collections\_list\_equal

>>> PASS <<<

test\_collections\_tuple\_equal

>>> PASS <<<

test\_collections\_set\_equal

>>> PASS <<<

test\_collections\_dict\_equal

>>> PASS <<<

test\_exceptions\_raises

>>> PASS <<<

test\_exceptions\_raises\_no\_error

>>> FAIL: no error was raised <<<

test\_timing\_runs\_under

>>> PASS <<<

test\_timing\_runs\_over

>>> FAIL: The function took longer than 1 seconds to execute <<<

test\_timing\_takes\_at\_least

>>> PASS <<<

test\_timing\_takes\_less\_than

>>> FAIL: The function took less than 1 seconds to execute <<<

>>> Test Summary:

>>> Total Tests: 27

>>> Total Passed Tests: 24

>>> Total Failed Tests: 3

**Use cases**

* Writing quick tests for small projects
* A standalone testing environment that does not require external libraries
* For educational purposes, to show how the “assert” mechanism Works

**Advantages and Disadventages**

**Adventages:**

* Not external dependencies
* Small and clear architecture
* Easily extensible assertion methods

**Disadventages:**

* No parametric testing, no parallel work
* No advanced fixture/configuration support
* Limited reporting format (console only)

**ALL CODES**

Framework/assertions.py

import time

# COMPARISON FAMILY

class Comparison:

    def \_\_init\_\_(self, a, b):

        self.a = a

        self.b = b

    def assertEqual(self):

        if self.a != self.b:

            raise AssertionError(f"{self.a} is not equal to {self.b}")

    def assertNotEqual(self):

        if self.a == self.b:

            raise AssertionError(f"{self.a} is equal to {self.b}")

    def assertGreater(self):

        if self.a <= self.b:

            raise AssertionError(f"{self.b} is greater than or equal to {self.a}")

    def assertGreaterEqual(self):

        if self.a < self.b:

            raise AssertionError(f"{self.b} is greater than {self.a}")

    def assertLess(self):

        if self.a >= self.b:

            raise AssertionError(f"{self.b} is less than or equal to {self.a}")

    def assertLessEqual(self):

        if self.a > self.b:

            raise AssertionError(f"{self.b} is less than {self.a}")

# TRUTHINESS FAMILY

class Truthiness:

    def \_\_init\_\_(self, value):

        self.value = value

    def assertTrue(self):

        if not self.value:

            raise AssertionError(f"{self.value} is not a True boolean value")

    def assertFalse(self):

        if self.value:

            raise AssertionError(f"{self.value} is a True boolean value")

    def assertNone(self):

        if not (self.value is None):

            raise AssertionError(f"{self.value} is not None")

    def assertNotNone(self):

        if self.value is None:

            raise AssertionError(f"{self.value} is None")

# IDENTIFICATION FAMILY

class Identification:

    def \_\_init\_\_(self, a, b):

        self.a = a

        self.b = b

    def assertIs(self):

        if not (self.a is self.b):

            raise AssertionError(f"the objects are not equal to each other")

    def assertIsNot(self):

        if (self.a is self.b):

            raise AssertionError(f"the objects are equal to each other")

    def assertIsInstanceOf(self):

        if not (isinstance(self.a, self.b)):

            raise AssertionError(f"{self.a} is not an instance of class {self.b}")

    def assertIsNotInstanceOf(self):

        if (isinstance(self.a, self.b)):

            raise AssertionError(f"{self.a} is an instance of class {self.b}")

# COLLECTIONS FAMILY

class Collections:

    def \_\_init\_\_(self, a, b):

        self.a = a

        self.b = b

    def assertIn(self):

        if not (self.a in self.b):

            raise AssertionError(f"{self.a} is not in {self.b}")

    def assertNotIn(self):

        if (self.a in self.b):

            raise AssertionError(f"{self.a} is in {self.b}")

    def countEqual(self):

        if len(self.a) != len(self.b):

            raise AssertionError(f"length of {self.a} is not equal to the length of {self.b}")

        for elem in set(self.a + self.b):

            if elem not in self.a or elem not in self.b:

                raise AssertionError(f"The collections are not equal")

            if self.a.count(elem) != self.b.count(elem):

                raise AssertionError(f"The collections are not equal")

    def assertListEqual(self):

        if self.a != self.b:

            raise AssertionError("the lists are not equal!")

    def assertTupleEqual(self):

        if self.a != self.b:

            raise AssertionError("the tuples are not equal!")

    def assertSetEqual(self):

            if self.a != self.b:

                raise AssertionError("the sets are not equal!")

    def assertDictEqual(self):

                if self.a != self.b:

                    raise AssertionError("the dictionaries are not equal!")

# EXCEPTIONS FAMILY

class Exceptions:

    def \_\_init\_\_(self, e, func, \*args, \*\*kwargs):

        self.e = e

        self.func = func

        self.args = args

        self.kwargs = kwargs

    def assertRaises(self):

        try:

            self.func(\*self.args, \*\*self.kwargs)

        except Exception as e:

            if not isinstance(e, self.e):

                raise AssertionError(f"expected: {self.e}, got {type(e)}")

        else:

            raise AssertionError("no error was raised")

# TIMING FAMILY

class Timing:

    def \_\_init\_\_(self, seconds, func, \*args, \*\*kwargs):

        self.seconds = seconds

        self.func = func

        self.args = args

        self.kwargs = kwargs

    def assertRunsUnder(self):

        start = time.perf\_counter()

        self.func(\*self.args, \*\*self.kwargs)

        end = time.perf\_counter()

        duration = end - start

        if duration > self.seconds:

            raise AssertionError(f"The function took longer than {self.seconds} seconds to execute")

    def assertTakesAtleast(self):

        start = time.perf\_counter()

        self.func(\*self.args, \*\*self.kwargs)

        end = time.perf\_counter()

        duration = end - start

        if duration < self.seconds:

            raise AssertionError(f"The function took less than {self.seconds} seconds to execute")

Framework/core.py

\_registered\_tests = []

def test(func):

    """

    decorator to register all tests to the private

    \_registered\_tests array

    """

    \_registered\_tests.append(func)

    print("test?")

    return func

def get\_tests():

    """

    returns all the registered tests

    """

    return \_registered\_tests

framework/runner.py

def runTests():

    tests = get\_tests()

    total\_tests = len(tests)

    passed, failed = 0,0

    for test in tests:

        try:

            test()

            print(f"{test.\_\_name\_\_}\n>>> PASS <<<")

            passed += 1

        except AssertionError as e:

            print(f"{test.\_\_name\_\_}\n>>> FAIL: {str(e)} <<<")

            failed += 1

    testSummary(total\_tests, passed, failed)

    return ">>> TESTING COMPLETED <<<"

if \_\_name\_\_ == "\_\_main\_\_":

    runTests()

Tests/test\_samples.py

import time

from framework.assertions import Comparison, Truthiness, Identification, Collections, Exceptions, Timing

from framework.core import test

import sys

import os

# Proje dizinini sisteme ekleyin

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

# COMPARISON FAMILY TESTS

@test

def test\_comparison\_equal():

    comparison = Comparison(1, 1)

    comparison.assertEqual()

@test

def test\_comparison\_not\_equal():

    comparison = Comparison(1, 2)

    comparison.assertNotEqual()

@test

def test\_comparison\_greater():

    comparison = Comparison(3, 2)

    comparison.assertGreater()

@test

def test\_comparison\_greater\_equal():

    comparison = Comparison(3, 3)

    comparison.assertGreaterEqual()

@test

def test\_comparison\_less():

    comparison = Comparison(2, 3)

    comparison.assertLess()

@test

def test\_comparison\_less\_equal():

    comparison = Comparison(2, 3)

    comparison.assertLessEqual()

# TRUTHINESS FAMILY TESTS

@test

def test\_truthiness\_true():

    truthiness = Truthiness(True)

    truthiness.assertTrue()

@test

def test\_truthiness\_false():

    truthiness = Truthiness(False)

    truthiness.assertFalse()

@test

def test\_truthiness\_none():

    truthiness = Truthiness(None)

    truthiness.assertNone()

@test

def test\_truthiness\_not\_none():

    truthiness = Truthiness(1)

    truthiness.assertNotNone()

# IDENTIFICATION FAMILY TESTS

@test

def test\_identification\_is():

    a = [1]

    b = a

    identification = Identification(a, b)

    identification.assertIs()

@test

def test\_identification\_is\_not():

    a = [1]

    b = [1]

    identification = Identification(a, b)

    identification.assertIsNot()

@test

def test\_identification\_instance\_of():

    identification = Identification("hello", str)

    identification.assertIsInstanceOf()

@test

def test\_identification\_not\_instance\_of():

    identification = Identification(1, str)

    identification.assertIsNotInstanceOf()

# COLLECTIONS FAMILY TESTS

@test

def test\_collections\_in():

    collections = Collections(1, [1, 2, 3])

    collections.assertIn()

@test

def test\_collections\_not\_in():

    collections = Collections(4, [1, 2, 3])

    collections.assertNotIn()

@test

def test\_collections\_count\_equal():

    collections = Collections([1, 2, 3], [3, 2, 1])

    collections.countEqual()

@test

def test\_collections\_list\_equal():

    collections = Collections([1, 2, 3], [1, 2, 3])

    collections.assertListEqual()

@test

def test\_collections\_tuple\_equal():

    collections = Collections((1, 2, 3), (1, 2, 3))

    collections.assertTupleEqual()

@test

def test\_collections\_set\_equal():

    collections = Collections({1, 2, 3}, {1, 2, 3})

    collections.assertSetEqual()

@test

def test\_collections\_dict\_equal():

    collections = Collections({"a": 1}, {"a": 1})

    collections.assertDictEqual()

# EXCEPTIONS FAMILY TESTS

@test

def test\_exceptions\_raises():

    def test\_func():

        raise ValueError("An error occurred")

    exceptions = Exceptions(ValueError, test\_func)

    exceptions.assertRaises()

@test

def test\_exceptions\_raises\_no\_error():

    def test\_func():

        pass

    exceptions = Exceptions(ValueError, test\_func)

    exceptions.assertRaises()

# TIMING FAMILY TESTS

@test

def test\_timing\_runs\_under():

    def slow\_func():

        time.sleep(0.5)

    timing = Timing(1, slow\_func)

    timing.assertRunsUnder()

@test

def test\_timing\_runs\_over():

    def slow\_func():

        time.sleep(1.5)

    timing = Timing(1, slow\_func)

    timing.assertRunsUnder()

@test

def test\_timing\_takes\_at\_least():

    def slow\_func():

        time.sleep(2)

    timing = Timing(1, slow\_func)

    timing.assertTakesAtleast()

@test

def test\_timing\_takes\_less\_than():

    def slow\_func():

        time.sleep(0.5)

    timing = Timing(1, slow\_func)

    timing.assertTakesAtleast()