QAIS_BM SHALAN 20224077

Deniz Öztürk 20210504

HASAN_ASHRAF_HASAN_AZZAM SHUKUKANI 20223100

BADIR ALHAJ_YOUNIS 20215054

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/
├── .env
├── framework/
│   ├── __init__.py
│   ├── core.py # Registers and provides test functions
│   ├── ai_helper.py
│   ├── assertions.py # Custom assertion logic grouped by purpose
│   └── runner.py # Main test runner
├── tests/
│   ├── __init__.py
│   └── test_samples.py # Example tests using all assertion types
└── README.md # Project documentation
```

Create a .env file:

You can create on .txt file but on file name have to be .env not.env.txt
in .env file write gemini API like:

```
GEMINI_API_KEY=”your_real_api_key_here”
```

And import gemini-ai apı:

```
pip install google-generativeai
```

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

Framework/ai_helper.py

```
import os
import google.generativeai as genai
from dotenv import load_dotenv

# upload .env file
load_dotenv()

# Get API key from environment variable
genai.configure(api_key=os.getenv("AIzaSyCjmYx8dAWu4ZXf_s3wVcrwOxFDB36b0iM"))

# Gemini model description
model = genai.GenerativeModel("gemini-1.5-flash")
```

this file set sup integration with gemini1.5 flash, a large language model from Google

- it loads gemini API key securely from a .env file using load_dotenv()
- genai.configure() function sets the API key for use
- generativeModel("gemini-1.5-flash") instance is used to generate AI responses based on given prompt

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
```

```python
# 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/ai_helper.py

```
import os
import google.generativeai as genai
from dotenv import load_dotenv

# upload .env file
```

```
load_dotenv()

# Get API key from environment variable
genai.configure(api_key=os.getenv("Your API key"))

# Gemini model description
model = genai.GenerativeModel("gemini-1.5-flash")

def explain_failure(test_name: str, error_message: str) -> str:
    prompt = f"""
     A unit test in Python has failed.

    Test name: {test_name}
    Erorr message: {error_message}

   Please provide a technical analysis of why this test might have failed and suggest
possible fixes.
    """

    try:
        response = model.generate_content(prompt)
        return response.text.strip()
    except Exception as e:
        return f"[Gemini AI explanation failed: {str(e)}]"
```

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()
```