ASSIGNMENT-8

Roll no:2303A52410
Batch-38

Task 1 – Test-Driven Development for Even/Odd Number Validator

Prompt
Write unit test cases for a Python function is_even(n) and then implement the function so that all tests pass.

Code

```python
#Write unit test cases for a Python function is_even(n) and then implement the function so that all tests pass.
import unittest
def is_even(n):
    return n % 2 == 0
class TestIsEven(unittest.TestCase):
    def test_is_even_with_even_number(self):
        self.assertTrue(is_even(4))
    def test_is_even_with_odd_number(self):
        self.assertFalse(is_even(5))
    def test_is_even_with_zero(self):
        self.assertTrue(is_even(0))
    def test_is_even_with_negative_even_number(self):
        self.assertTrue(is_even(-2))
    def test_is_even_with_negative_odd_number(self):
        self.assertFalse(is_even(-3))
if __name__ == '__main__':
    unittest.main()
```

Output

```
y
.....
----------------------------------------------------------------------
Ran 5 tests in 0.002s

OK
```

Explanation
First, unit test cases are created to check different inputs like even, odd, zero, and negative numbers.Then, the is_even() function is written to satisfy all these tests, ensuring correct behavior.

Task 2 – Test-Driven Development for String Case Converter

Prompt

Generate unit test cases for two Python functions: to uppercase text and to lowerca
text.

Code

```python
#Generate unit test cases for two Python functions: to_uppercase(text) and to_lowercase(text).
import unittest
def to_uppercase(text):
    return text.upper()
def to_lowercase(text):
    return text.lower()
class TestStringFunctions(unittest.TestCase):
    def test_to_uppercase(self):
        self.assertEqual(to_uppercase('hello'), 'HELLO')
        self.assertEqual(to_uppercase('World'), 'WORLD')
        self.assertEqual(to_uppercase('Python'), 'PYTHON')
    def test_to_lowercase(self):
        self.assertEqual(to_lowercase('HELLO'), 'hello')
        self.assertEqual(to_lowercase('World'), 'world')
        self.assertEqual(to_lowercase('PYTHON'), 'python')
if __name__ == '__main__':
    unittest.main()
```

Output

```
y
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

Explanation

First, test cases are created to check different string inputs like empty text, mixed c and invalid values.Then, the functions are written to convert text to upper or lower case so that all tests pass.

Task 3 – Test-Driven Development for List Sum Calculator

Prompt

Generate test cases for a Python function sum list numbers that calculates the sum list elements.

Code

```python
#Generate test cases for a Python function sum_list(numbers) that calculates the sum of list elements.
import unittest
def sum_list(numbers):
    return sum(numbers)
class TestSumList(unittest.TestCase):
    def test_sum_list_with_positive_numbers(self):
        self.assertEqual(sum_list([1, 2, 3]), 6)
    def test_sum_list_with_negative_numbers(self):
        self.assertEqual(sum_list([-1, -2, -3]), -6)
    def test_sum_list_with_mixed_numbers(self):
        self.assertEqual(sum_list([-1, 0, 1]), 0)
    def test_sum_list_with_empty_list(self):
        self.assertEqual(sum_list([]), 0)
    def test_sum_list_with_large_numbers(self):
        self.assertEqual(sum_list([1000000, 2000000, 3000000]), 6000000)
if __name__ == '__main__':
    unittest.main()
```

Output

```
y
.....
----------------------------------------------------------------------
Ran 5 tests in 0.002s

OK
```

Explanation

First, test cases are created to check different list inputs like empty lists, negative numbers, and mixed values.

Then, the sum list() function is written to add only numeric elements so that all tests pass.

Task 4 – Test Cases for Student Result Class

Prompt

Generate test cases for a Python class StudentResult with methods add_marks(mark calculate average, and get result.

Code

```python
#Generate test cases for a Python class StudentResult with methods add_marks(mark), calculate_average(), and get_result().
import unittest
class StudentResult:
    def __init__(self):
        self.marks = []
    def add_marks(self, mark):
        self.marks.append(mark)
    def calculate_average(self):
        if not self.marks:
            return 0
        return sum(self.marks) / len(self.marks)
    def get_result(self):
        average = self.calculate_average()
        if average >= 60:
            return 'Pass'
        else:
            return 'Fail'
class TestStudentResult(unittest.TestCase):
    def setUp(self):
        self.student_result = StudentResult()
    def test_add_marks_and_calculate_average(self):
        self.student_result.add_marks(80)
        self.student_result.add_marks(90)
        self.assertEqual(self.student_result.calculate_average(), 85)
    def test_get_result_pass(self):
        self.student_result.add_marks(70)
        self.student_result.add_marks(80)
        self.assertEqual(self.student_result.get_result(), 'Pass')
    def test_get_result_fail(self):
        self.student_result.add_marks(50)
        self.student_result.add_marks(40)
        self.assertEqual(self.student_result.get_result(), 'Fail')
    def test_calculate_average_with_no_marks(self):
        self.assertEqual(self.student_result.calculate_average(), 0)
if __name__ == '__main__':
    unittest.main()
```

Output

```
y
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```

Explanation

First, test cases are created to check valid marks, invalid marks, and different result
outcomes.
Then, the Student class is implemented so that it calculates the average correctly a
returns Pass or Fail.

Task 5 – Test-Driven Development for Username Validator

Prompt

Generate test cases for a Python function is valid username that validates username

Code

```python
#Generate test cases for a Python function is_valid_username(username) that validates usernames.
import unittest
import re
def is_valid_username(username):
    if len(username) < 3 or len(username) > 20:
        return False
    if not re.match("^[A-Za-z0-9_]+$", username):
        return False
    return True
class TestIsValidUsername(unittest.TestCase):
    def test_valid_username(self):
        self.assertTrue(is_valid_username('user_123'))
    def test_username_too_short(self):
        self.assertFalse(is_valid_username('ab'))
    def test_username_too_long(self):
        self.assertFalse(is_valid_username('a' * 21))
    def test_username_with_invalid_characters(self):
        self.assertFalse(is_valid_username('user!@#'))
    def test_username_with_only_letters_and_numbers(self):
        self.assertTrue(is_valid_username('username123'))
    def test_username_with_underscore(self):
        self.assertTrue(is_valid_username('user_name'))
if __name__ == '__main__':
    unittest.main()
```

Output

```
Ran 6 tests in 0.002s

OK
```

Explanation
First, test cases are created to check valid and invalid usernames.
Then, the is valid username() function is written to follow all the validation rules and
pass the tests.