

School of Computer Science and Artificial Intelligence

Lab Assignment # 8.2

Program : B. Tech (CSE)

Specialization : AIML

Course Title : AI Assisted

Coding Course Code:

23CS002PC304

Semester : VI

Academic Session : 2025-2026

Name of Student : T.INDRANEEL

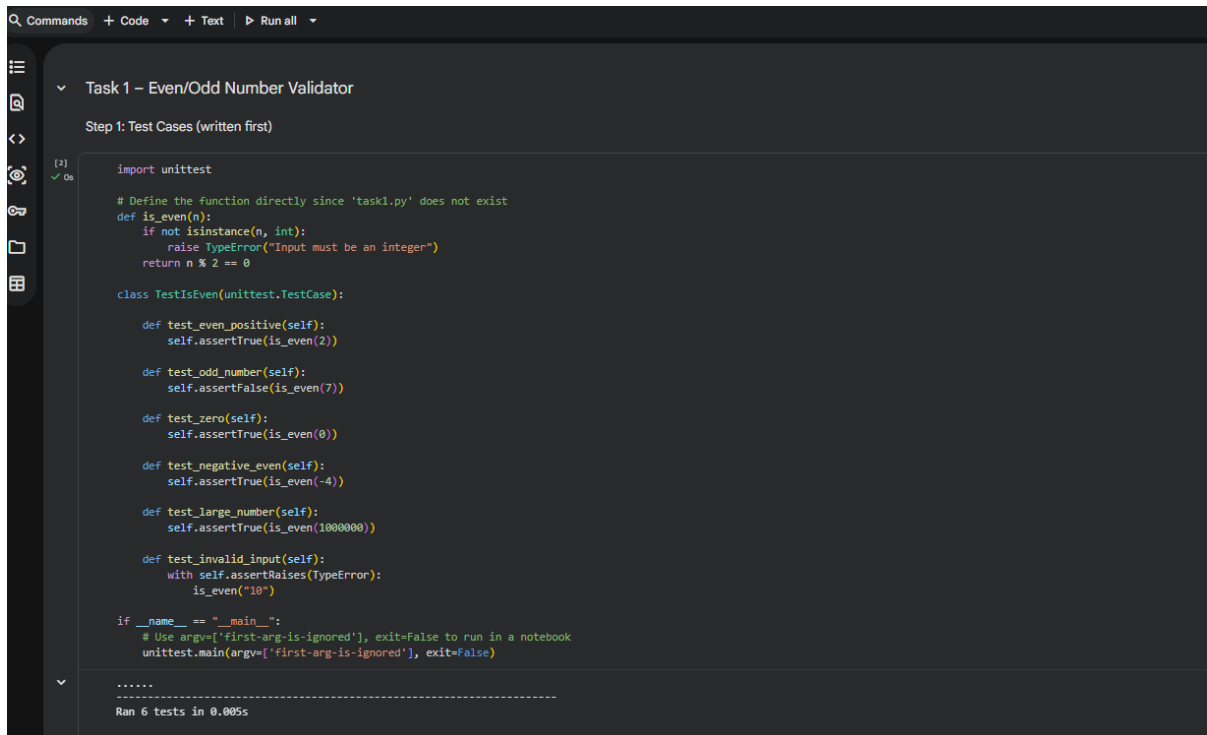
Enrollment No. : 2303A52121

Batch No. : 33

Date : 10/02/26

Task 1 – Even/Odd Number Validator

Step 1: Test Cases (written first)



The screenshot shows a code editor with a dark theme. The top bar has tabs for 'Commands', 'Code', 'Text', and 'Run all'. The left sidebar shows a file explorer with a folder icon. The main editor area is titled 'Task 1 – Even/Odd Number Validator' and 'Step 1: Test Cases (written first)'. The code is as follows:

```
import unittest

# Define the function directly since 'task1.py' does not exist
def is_even(n):
    if not isinstance(n, int):
        raise TypeError("Input must be an integer")
    return n % 2 == 0

class TestIsEven(unittest.TestCase):

    def test_even_positive(self):
        self.assertTrue(is_even(2))

    def test_odd_number(self):
        self.assertFalse(is_even(7))

    def test_zero(self):
        self.assertTrue(is_even(0))

    def test_negative_even(self):
        self.assertTrue(is_even(-4))

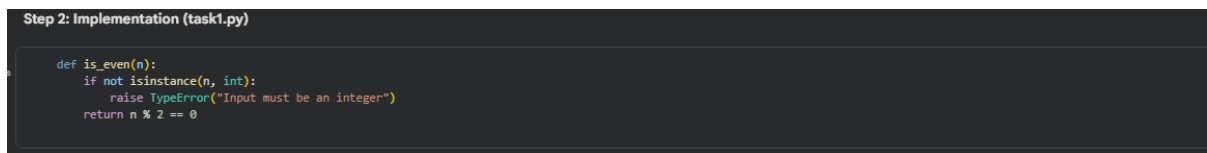
    def test_large_number(self):
        self.assertTrue(is_even(1000000))

    def test_invalid_input(self):
        with self.assertRaises(TypeError):
            is_even("10")

if __name__ == "__main__":
    # Use argv['first-arg-is-ignored'], exit=False to run in a notebook
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

Below the code, the output of the test runner is shown:

```
.....
Ran 6 tests in 0.005s
```



The screenshot shows a code editor with a dark theme. The top bar has tabs for 'Commands', 'Code', 'Text', and 'Run all'. The left sidebar shows a file explorer with a folder icon. The main editor area is titled 'Step 2: Implementation (task1.py)'. The code is as follows:

```
def is_even(n):
    if not isinstance(n, int):
        raise TypeError("Input must be an integer")
    return n % 2 == 0
```

Task 2 – String Case Converter

Step 1: Test Cases

Task 2 – String Case Converter

Step 1: Test Cases

```
(1) 1 import unittest
    2
    3 # Defining the functions directly since 'task2.py' does not exist
    4 def to_uppercase(s):
    5     if not isinstance(s, str):
    6         raise TypeError("Input must be a string")
    7     return s.upper()
    8
    9 def to_lowercase(s):
    10     if s is None:
    11         raise ValueError("Input cannot be None")
    12     if not isinstance(s, str):
    13         raise TypeError("Input must be a string")
    14     return s.lower()
    15
    16 class TestStringCase(unittest.TestCase):
    17
    18     def test_uppercase_normal(self):
    19         self.assertEqual(to_uppercase("ai coding"), "AI CODING")
    20
    21     def test_lowercase_normal(self):
    22         self.assertEqual(to_lowercase("TEST"), "test")
    23
    24     def test_empty_string(self):
    25         self.assertEqual(to_uppercase(""), "")
    26
    27     def test_mixed_case(self):
    28         self.assertEqual(to_lowercase("PyThOn"), "python")
    29
    30     def test_none_input(self):
    31         with self.assertRaises(ValueError):
    32             to_lowercase(None)
    33
    34     def test_invalid_type(self):
    35         with self.assertRaises(TypeError):
    36             to_uppercase(123)
    37
    38 if __name__ == "__main__":
    39     # Use argv['first-arg-is-ignored'], exit=False for notebook compatibility
    40     unittest.main(argv=['first-arg-is-ignored'], exit=False)
    41
    42
    43 Ran 12 tests in 0.012s
```

Step 2: Implementation (task2.py)

```
def to_uppercase(text):
    if text is None:
        raise ValueError("Input cannot be None")
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return text.upper()

def to_lowercase(text):
    if text is None:
        raise ValueError("Input cannot be None")
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return text.lower()
```

Task 3 – List Sum Calculator

Step 1: Test Cases

```
Task 3 – List Sum Calculator

Step 1: Test Cases

[4]
✓ On
import unittest

# Defining the function directly since 'task3.py' does not exist
def sum_list(items):
    if not isinstance(items, list):
        raise TypeError("Input must be a list")
    total = 0
    for item in items:
        if isinstance(item, (int, float)):
            total += item
    return total

class TestSumList(unittest.TestCase):

    def test_normal_list(self):
        self.assertEqual(sum_list([1, 2, 3]), 6)

    def test_empty_list(self):
        self.assertEqual(sum_list([]), 0)

    def test_negative_numbers(self):
        self.assertEqual(sum_list([-1, 5, -4]), 0)

    def test_with_non_numeric(self):
        self.assertEqual(sum_list([2, "a", 3]), 5)

    def test_invalid_input(self):
        with self.assertRaises(TypeError):
            sum_list("123")

if __name__ == "__main__":
    # Use argv['first-arg-is-ignored'], exit=False for notebook compatibility
    unittest.main(argv=['first-arg-is-ignored'], exit=False)

--
.....
Run 17 tests in 0.018s
OK
```

```
Step 2: Implementation (task3.py)

def sum_list(numbers):
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")

    total = 0
    for num in numbers:
        if isinstance(num, (int, float)):
            total += num
    return total
```

Task 4 – Student Result Class

Step 1: Test Cases

```
Task 4 – Student Result Class
```

Step 1: Test Cases

```
import unittest

# Defining the class directly since 'task4.py' does not exist
class StudentResult:
    def __init__(self):
        self.marks = []

    def add_marks(self, mark):
        if mark < 0 or mark > 100:
            raise ValueError("Mark must be between 0 and 100")
        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):
        avg = self.calculate_average()
        return "Pass" if avg >= 40 else "Fail"

class TestStudentResult(unittest.TestCase):

    def test_pass_result(self):
        s = StudentResult()
        s.add_marks(60)
        s.add_marks(70)
        s.add_marks(80)
        self.assertEqual(s.calculate_average(), 70)
        self.assertEqual(s.get_result(), "Pass")

    def test_fail_result(self):
        s = StudentResult()
        s.add_marks(30)
        s.add_marks(35)
        s.add_marks(40)
        self.assertEqual(s.get_result(), "Fail")

    def test_invalid_mark(self):
        s = StudentResult()
        with self.assertRaises(ValueError):
            s.add_marks(-10)

    def test_empty_marks(self):
        s = StudentResult()
        self.assertEqual(s.calculate_average(), 0)

if __name__ == "__main__":
    # Use argv["first-arg-is-ignored"], exit=False for notebook compatibility
    unittest.main(argv=["first-arg-is-ignored"], exit=False)
```

Ran 21 tests in 0.028s

Step 2: Implementation (task4.py)

```
class StudentResult:
    def __init__(self):
        self.marks = []

    def add_marks(self, mark):
        if mark < 0 or mark > 100:
            raise ValueError("Marks must be between 0 and 100")
        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):
        avg = self.calculate_average()
        return "Pass" if avg >= 40 else "Fail"
```

Task 5 – Username Validator

Step 1: Test Cases

Task 5 – Username Validator

Step 1: Test Cases

```
[14] import unittest
✓ Os
# Defining the function directly since 'task5.py' does not exist
def is_valid_username(username):
    if not isinstance(username, str):
        return False
    if len(username) < 3:
        return False
    if not username.isalnum():
        return False
    return True

class TestUsername(unittest.TestCase):

    def test_valid_username(self):
        self.assertTrue(is_valid_username("user01"))

    def test_short_username(self):
        self.assertFalse(is_valid_username("ai"))

    def test_space_in_username(self):
        self.assertFalse(is_valid_username("user name"))

    def test_special_characters(self):
        self.assertFalse(is_valid_username("user@123"))

    def test_non_string(self):
        self.assertFalse(is_valid_username(12345))

if __name__ == "__main__":
    # Use argv=['first-arg-is-ignored'], exit=False for notebook compatibility
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```


Ran 26 tests in 0.027s
OK

Step 2: Implementation (task5.py)

```
[15] def is_valid_username(username):
✓ Os
    if not isinstance(username, str):
        return False
    if len(username) < 5:
        return False
    if " " in username:
        return False
    if not username.isalnum():
        return False
    return True
```

Lab Outcomes Covered

- Test cases written first (TDD style)
- Input validation & error handling
- Edge cases: empty, None, negative, large values
- unittest usage
- Clean and reliable implementations

