

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 : R.Deekshith

Enrollment No. : 2303A52104

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 interface with a dark theme. At the top, there are tabs for 'Commands', 'Code', 'Text', and 'Run all'. Below the tabs, a sidebar on the left has icons for file operations like 'New', 'Open', 'Save', and 'Run'. A tree view under 'Task 1 – Even/Odd Number Validator' shows 'Step 1: Test Cases (written first)'. The main code area contains Python test code:

```
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)

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

Below this, another section titled 'Step 2: Implementation (task1.py)' shows the implementation code:

```
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] ✓ 0s
  import unittest

  # Defining the functions directly since 'task2.py' does not exist
  def to_uppercase(s):
      if s is None:
          raise ValueError("Input cannot be None")
      if not isinstance(s, str):
          raise TypeError("Input must be a string")
      return s.upper()

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

  class TestStringCase(unittest.TestCase):

      def test_uppercase_normal(self):
          self.assertEqual(to_uppercase("AI CODING"), "AI CODING")

      def test_lowercase_normal(self):
          self.assertEqual(to_lowercase("TEST"), "test")

      def test_empty_string(self):
          self.assertEqual(to_uppercase(""), "")

      def test_mixed_case(self):
          self.assertEqual(to_lowercase("PyThOn"), "python")

      def test_none_input(self):
          with self.assertRaises(ValueError):
              to_lowercase(None)

      def test_invalid_type(self):
          with self.assertRaises(TypeError):
              to_uppercase(123)

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

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

```
(*) ✓ 1a 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[0]'s first arg is ignored', exit=False for notebook compatibility
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

***
Ran 7 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 – StudentResult Class

Step 1: Test Cases

```
[1] ❶ import unittest  
❷  
❸ # Defining the class directly since 'task4.py' does not exist  
❹ class StudentResult:  
❺     def __init__(self):  
❻         self.marks = []  
❼  
⏵     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[1] first-arg-is-ignored, exit=False for Notebook compatibility  
⏵     unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

Step 2: Implementation (task4.py)

```
[2] ❶ class StudentResult:  
❷     def __init__(self):  
❸         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] 0s ⏪ import unittest

# 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] 0s ⏪ def is_valid_username(username):
    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

