

# AI ASSISTED CODING

Nasani Sai Pavan

2303A51117

BACTH – 03

17 – 02 – 2026

## ASSIGNMENT – 08

**LAB – 08 :** Test – Driven Development with AI – Generating and Working with Test Cases.

**Task – 01 :** Test – Driven Development for Odd/Even Number Validator.

### Prompt :

Generate unittest test cases for a Python function `is_even(n)` that checks whether a number is even. Handle zero, negative numbers, large integers, and invalid input.

### Code :

```
1 #Generate unittest test cases for a Python function is_even(n) that checks whether a number is even. Handle
2 import unittest
3
4 def is_even(n):
5     if not isinstance(n, int):
6         raise TypeError("Input must be an integer")
7     return n % 2 == 0
8
9 class TestIsEven(unittest.TestCase):
10
11     def test_even_numbers(self):
12         self.assertTrue(is_even(2))
13         self.assertTrue(is_even(0))
14         self.assertTrue(is_even(-4))
15         self.assertTrue(is_even(1000000))
16
17     def test_odd_numbers(self):
18         self.assertFalse(is_even(7))
19         self.assertFalse(is_even(9))
20         self.assertFalse(is_even(-3))
21
22     def test_invalid_input(self):
23         with self.assertRaises(TypeError):
24             is_even("abc")
25         with self.assertRaises(TypeError):
26             is_even(5.5)
27         with self.assertRaises(TypeError):
28             is_even(None)
29
30 if __name__ == "__main__":
31     unittest.main()
32
```

### Output :

```
***
-----
Ran 3 tests in 0.000s

OK
❖ Ran 3 tests in 0.000s
d. No connection Ln 21, Col 1
```

### Explanation :

The function first validates that the input is an integer. It then checks divisibility by 2 using modulus operator. It handles zero, negative, and large integers correctly.

## Task – 02 :

Test – Driven Development for String Case Converter. **Prompt :**

Generate test cases for `to_uppercase(text)` and `to_lowercase(text)` handling empty strings, mixed-case input, and invalid inputs.

### Code :

```
33
34 #Task 02
35 #Generate test cases for to_uppercase(text) and to_lowercase(text) handling empty strings, mixed-case input
36 import unittest
37
38 def to_uppercase(text):
39     if not isinstance(text, str):
40         raise TypeError("Input must be a string")
41     return text.upper()
42
43 def to_lowercase(text):
44     if not isinstance(text, str):
45         raise TypeError("Input must be a string")
46     return text.lower()
47
48 class TestStringCase(unittest.TestCase):
49     def test_uppercase(self):
50         self.assertEqual(to_uppercase("ai coding"), "AI CODING")
51         self.assertEqual(to_uppercase("Hello World"), "HELLO WORLD")
52         self.assertEqual(to_uppercase(""), "")
53     def test_lowercase(self):
54         self.assertEqual(to_lowercase("TEST"), "test")
55         self.assertEqual(to_lowercase("PyThOn"), "python")
56         self.assertEqual(to_lowercase(""), "")
57     def test_invalid_input(self):
58         with self.assertRaises(TypeError):
59             to_uppercase(None)
60         with self.assertRaises(TypeError):
61             to_lowercase(123)
62         with self.assertRaises(TypeError):
63             to_uppercase(5.5)
64
65 if __name__ == "__main__":
66     unittest.main()
```

### Output:

```
...
-----
Ran 3 tests in 0.000s

OK
```

## Explanation :

Both functions validate input type and use built-in string methods `.upper()` and `.lower()` for conversion.

## Task – 03 :

Test – Driven Development for List sum Calculator.

### Prompt :

Generate test cases for `sum_list(numbers)` that handles empty lists, negative numbers, and ignores non-numeric values.

## Code :

```
65 #Task 03
66 #Generate test cases for sum_list(numbers) that handles empty lists, negative numbers, and ignores non-nume
67 import unittest
68
69 def sum_list(numbers):
70     if not isinstance(numbers, list):
71         raise TypeError("Input must be a list")
72     total = 0
73     for num in numbers:
74         if isinstance(num, (int, float)):
75             total += num
76     return total
77
78 class TestSumList(unittest.TestCase):
79
80     def test_normal_list(self):
81         self.assertEqual(sum_list([1, 2, 3]), 6)
82     def test_empty_list(self):
83         self.assertEqual(sum_list([]), 0)
84     def test_negative_numbers(self):
85         self.assertEqual(sum_list([-1, 5, -4]), 0)
86     def test_mixed_values(self):
87         self.assertEqual(sum_list([2, "a", 3]), 5)
88
89     def test_invalid_input(self):
90         with self.assertRaises(TypeError):
91             sum_list("123")
92
93 if __name__ == "__main__":
94     unittest.main()
```

## Output :

```
...
-----
❖ Ran 3 tests in 0.000s

OK
```

## Explanation :

The function iterates through the list and adds only numeric values. It safely ignores non-numeric elements and returns 0 for empty lists.

## Task – 04 :

Test Cases for Student Result Class.

## Prompt :

Generate test cases for a StudentResult class with methods: add\_marks, calculate\_average, get\_result. Marks must be between 0 and 100.

## Code :

```

96 #Task 04
97 #Generate test cases for a StudentResult class with methods: add_marks, calculate_average, get_result. Marks must be between 0 and 100.
98 import unittest
99
100 class StudentResult:
101     def __init__(self):
102         self.marks = []
103     def add_marks(self, mark):
104         if not isinstance(mark, (int, float)) or mark < 0 or mark > 100:
105             raise ValueError("Marks must be between 0 and 100")
106         self.marks.append(mark)
107     def calculate_average(self):
108         if not self.marks:
109             return 0
110         return sum(self.marks) / len(self.marks)
111     def get_result(self):
112         return "Pass" if self.calculate_average() >= 40 else "Fail"
113 class TestStudentResult(unittest.TestCase):
114     def test_pass_case(self):
115         student = StudentResult()
116         student.add_marks(60)
117         student.add_marks(70)
118         student.add_marks(80)
119         self.assertEqual(student.calculate_average(), 70)
120         self.assertEqual(student.get_result(), "Pass")
121     def test_fail_case(self):
122         student = StudentResult()
123         student.add_marks(10)
124         student.add_marks(15)
125         student.add_marks(40)
126         self.assertEqual(student.calculate_average(), 35)
127         self.assertEqual(student.get_result(), "Fail")
128     def test_invalid_marks(self):
129         student = StudentResult()
130         with self.assertRaises(ValueError):
131             student.add_marks(-10)
132         with self.assertRaises(ValueError):
133             student.add_marks(120)
134         with self.assertRaises(ValueError):
135             student.add_marks("A")
136 if __name__ == "__main__":
137     unittest.main()
138

```

## Output:

```

...
-----
Ran 3 tests in 0.000s

OK

```

## Explanation :

The class validates marks before storing them. It calculates average dynamically and determines result based on 40% threshold.

## Task – 05 :

Test – Driven Development for username Validator.

## Prompt :

Generate test cases for `is_valid_username(username)` with minimum 5 characters, no spaces, and only alphanumeric characters.

## Code :

```

139 #Task 05
140 #Generate test cases for is_valid_username(username) with minimum 5 characters, no spaces, and only alphanu
141 import unittest
142
143 def is_valid_username(username):
144     if not isinstance(username, str):
145         return False
146     if len(username) < 5:
147         return False
148     if " " in username:
149         return False
150     if not username.isalnum():
151         return False
152     return True
153
154 class TestUsernameValidator(unittest.TestCase):
155     def test_valid_username(self):
156         self.assertTrue(is_valid_username("user01"))
157         self.assertTrue(is_valid_username("abcde"))
158     def test_short_username(self):
159         self.assertFalse(is_valid_username("ai"))
160         self.assertFalse(is_valid_username("usr"))
161     def test_space_in_username(self):
162         self.assertFalse(is_valid_username("user name"))
163     def test_special_characters(self):
164         self.assertFalse(is_valid_username("user@123"))
165         self.assertFalse(is_valid_username("user#1"))
166     def test_invalid_type(self):
167         self.assertFalse(is_valid_username(None))
168         self.assertFalse(is_valid_username(12345))
169
170 if __name__ == "__main__":
171     unittest.main()

```

## Output :

```

.....
-----
Ran 5 tests in 0.000s

OK

```

## Explanation :

The function checks length, space restriction, and alphanumeric condition using built-in string validation methods.