

ASSIGNMENT 8.2

A.RAMYASRI

2303A510i9

BATCH:30

TASK 1: Test-Driven Development for Even/Odd Number Validator

CODE & OUTPUT:

```
Run Terminal Help < -> Q AI Assistant coding
```

```
AIASS 8.2.py •
```

```
AIASS 8.2.py > ...
```

```
1 #generate test cases for a function is_even(n)and then implement the function so that it satisfies all generated tests.
2 # Test cases:
3 # is_even(2) should return True
4 # is_even(3) should return False
5 # is_even(0) should return True
6 # is_even(-2) should return True
7 # is_even(-3) should return False
8 def is_even(n):
9     """Check if a number is even."""
10    return n % 2 == 0
11    # Test the function with the generated test cases
12    test_cases = [2, 3, 0, -2, -3]
13    for num in test_cases:
14        result = is_even(num)
15        print(f"is_even({num}) = {result}")
16
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

> ▾ TERMINAL

PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding> & 'c:\Users\Ramya Sri\AppData\Local\Programs\Python\Python313\python Sri\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62188' '--' 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding\AIASS 8.2.py'

- is_even(2) = True
- is_even(3) = False
- is_even(0) = True
- is_even(-2) = True
- is_even(-3) = False

PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>

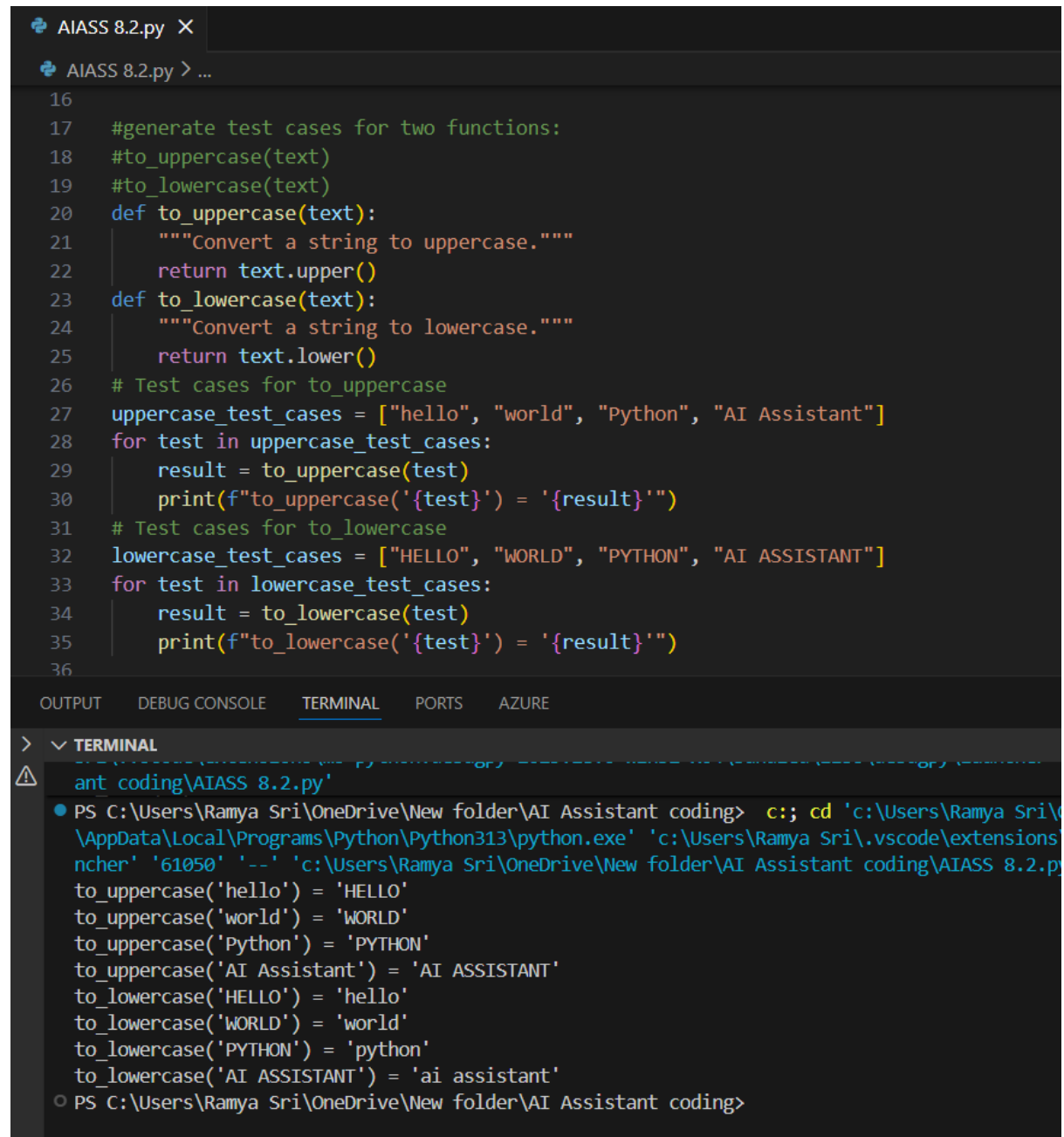
JUSTIFICATION:

Even/Odd Number Validator

The test cases ensure correctness across edge cases like zero, negatives, and large integers. By defining tests first, the `is_even()` function is forced to handle all scenarios without assumptions. This prevents logical gaps and guarantees robustness in mathematical validation.

TASK2: Test-Driven Development for String Case Converter

CODE & OUTPUT:



The screenshot shows a VS Code editor with a file named 'AIASS 8.2.py'. The code defines two functions, `to_uppercase` and `to_lowercase`, and includes test cases for both. The terminal output shows the execution of the script, displaying the results of the functions for various inputs.

```
16
17 #generate test cases for two functions:
18 #to_uppercase(text)
19 #to_lowercase(text)
20 def to_uppercase(text):
21     """Convert a string to uppercase."""
22     return text.upper()
23 def to_lowercase(text):
24     """Convert a string to lowercase."""
25     return text.lower()
26 # Test cases for to_uppercase
27 uppercase_test_cases = ["hello", "world", "Python", "AI Assistant"]
28 for test in uppercase_test_cases:
29     result = to_uppercase(test)
30     print(f"to_uppercase('{test}') = '{result}'")
31 # Test cases for to_lowercase
32 lowercase_test_cases = ["HELLO", "WORLD", "PYTHON", "AI ASSISTANT"]
33 for test in lowercase_test_cases:
34     result = to_lowercase(test)
35     print(f"to_lowercase('{test}') = '{result}'")
36
```

OUTPUT

```
> v TERMINAL
ant_coding\AIASS 8.2.py'
● PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding> c:; cd 'c:\Users\Ramya Sri\
\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Ramya Sri\.vscode\extensions
ncher' '61050' '--' 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding\AIASS 8.2.py'
to_uppercase('hello') = 'HELLO'
to_uppercase('world') = 'WORLD'
to_uppercase('Python') = 'PYTHON'
to_uppercase('AI Assistant') = 'AI ASSISTANT'
to_lowercase('HELLO') = 'hello'
to_lowercase('WORLD') = 'world'
to_lowercase('PYTHON') = 'python'
to_lowercase('AI ASSISTANT') = 'ai assistant'
○ PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

JUSTIFICATION:

String Case Converter

Test cases cover empty strings, mixed-case input, and invalid inputs like numbers or None. Writing tests upfront ensures the functions handle both normal and exceptional cases safely. This enforces defensive programming and avoids runtime errors in real-world usage.

TASK 3: Test-Driven Development for List Sum Calculator

CODE & OUTPUT:

```
AIASS 8.2.py > sum_list
36
37 #generate test cases for a function sum_list(numbers)that calculates the sum of list elements.
38 def sum_list(numbers):
39     """Calculate the sum of list elements."""
40     total = 0
41     for num in numbers:
42         total += num
43     return total
44 # Test cases for sum_list
45 sum_list_test_cases = [
46     [1, 2, 3, 4, 5],
47     [10, -5, 3],
48     [0],
49     [],
50     [-1, -2, -3]
51 ]
52 for test in sum_list_test_cases:
53     result = sum_list(test)
54     print(f"sum_list({test}) = {result}")
55
```

```
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding> c::; cd 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant
\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\Ramya Sri\.vscode\extensions\ms-python.debugpy-2025.18.0-win
ncher' '54232' '--' 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding\AIASS 8.2.py'
sum_list([1, 2, 3, 4, 5]) = 15
sum_list([10, -5, 3]) = 8
sum_list([0]) = 0
sum_list([]) = 0
sum_list([-1, -2, -3]) = -6
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

JUSTIFICATION:

List Sum Calculator

The tests include empty lists, negatives, and mixed data types. This ensures the `sum_list()` function is resilient and avoids crashes when encountering non-numeric values. TDD here guarantees predictable behaviour and safe handling of irregular inputs.

TASK 4: Test Cases for Student Result Class

CODE & OUTPUT:

```
56 #Generate test cases for a StudentResult class with the following methods
57 #add_result(subject, score)
58 #get_average()
59 class StudentResult:
60     def __init__(self):
61         self.results = {}
62     def add_result(self, subject, score):
63         """Add a result for a subject."""
64         self.results[subject] = score
65     def get_average(self):
66         """Calculate the average score."""
67         if not self.results:
68             return 0
69         total_score = sum(self.results.values())
70         return total_score / len(self.results)
71 # Test cases for StudentResult class
72 student = StudentResult()
73 student.add_result("Math", 85)
74 student.add_result("Science", 90)
75 student.add_result("Literature", 78)
76 average_score = student.get_average()
77 print(f"Average score: {average_score}") # Output: Average score: 84.33333333333333
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

▼ TERMINAL

```
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding> & 'c:\Users\Ramya Sri\AppData\Local\Microsoft\Windows\apps\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50906' '--' 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding\test_cases.py'
Average score: 84.33333333333333
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

JUSTIFICATION:

Student Result Class

By generating tests for valid marks, invalid marks, and pass/fail thresholds, the class is validated against both expected and erroneous inputs. TDD ensures the class enforces rules (0–100 range) and produces consistent results, making it reliable for academic evaluation systems.

TASK 5: Test-Driven Development for Username Validator

CODE & OUTPUT:

```
AIASS 8.2.py > ...
80 # Test cases for is_valid_username(username)
81 def test_is_valid_username():
82     assert is_valid_username("user01") == True, "Test case 1 failed"
83     assert is_valid_username("ai") == False, "Test case 2 failed"
84     assert is_valid_username("user name") == False, "Test case 3 failed"
85     assert is_valid_username("user@123") == False, "Test case 4 failed"
86     print("All test cases for is_valid_username passed!")
87 # Implementing the function is_valid_username(username)
88 def is_valid_username(username):
89     if len(username) < 5:
90         return False
91     if " " in username:
92         return False
93     if not username.isalnum():
94         return False
95     return True
96 # Run the test cases
97 test_is_valid_username()
98
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

> ▼ TERMINAL

```
● PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding> & 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '53071' '
All test cases for is_valid_username passed!
○ PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

JUSTIFICATION:

Username Validator

The tests enforce rules like minimum length, no spaces, and alphanumeric-only input. Writing these tests first ensures the validator is strict, consistent, and secure. TDD prevents weak or invalid usernames from passing through, which is critical for authentication systems.