AI ASSISTED CODING
LAB ASSIGNMENT 8.2 – TEST DRIVEN DEVELOPMENT WITH AI

2303A51207

MD SUBHANI

BATCH 04

Task 1 — Even/Odd Number Validator Question
Generate test cases for a function is_even(n) and implement it to pass all tests.

PROMPT
Use AI to generate test cases for an even/odd validator that handles zero, negative numbers, and large integers. Implement the function accordingly.

CODE

```
def is_even(n):
    return n % 2 == 0


print(is_even(2))
print(is_even(7))
print(is_even(0))
print(is_even(-4))
print(is_even(9))


assert is_even(2) ==
True assert is_even(7)
== False assert
is_even(0) == True
assert is_even(-4) ==
True assert is_even(9)
== False
```

print("All tests passed")

OUTPUT

```
PS D:\3-2 SEM\AI ASSISTED> python -u "d:\3-2 SEM\AI
True
False
True
True
False
All tests passed
PS D:\3-2 SEM\AI ASSISTED>
```

EXPLANATION
The function checks divisibility by 2 using modulus operator. If remainder is zero, the number is even. Test cases ensure handling of zero, negatives, and large integers.

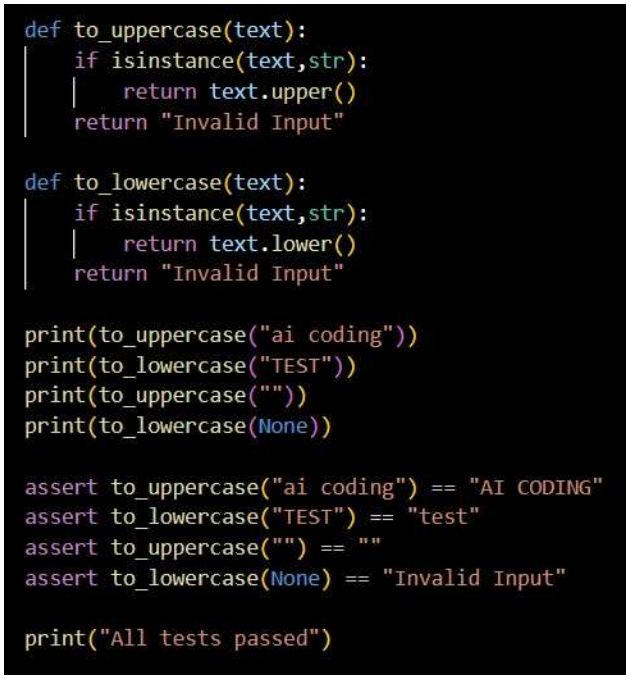
Task 2 — String Case Converter QUESTION
Generate test cases for to_uppercase(text) and to_lowercase(text).

PROMPT
Create functions to convert strings to uppercase and lowercase while handling empty strings and invalid inputs safely.

CODE

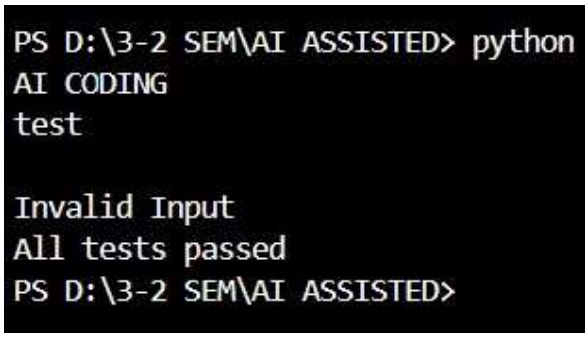
```
def to_uppercase(text):
    if isinstance(text,str):
return text.upper()
return "Invalid Input"
def to_lowercase(text):
if isinstance(text,str):
return text.lower()
return "Invalid Input"
print(to_uppercase("ai
coding"))
```

print(to_lowercase("TES T")) print(to_uppercase("")) print(to_lowercase(None)) assert to_uppercase("ai coding") == "AI CODING" assert to_lowercase("TEST") == "test" assert to_uppercase("") == "" assert to_lowercase(None) == "Invalid Input" print("All tests passed")



```
def to_uppercase(text):
    if isinstance(text,str):
        return text.upper()
    return "Invalid Input"

def to_lowercase(text):
    if isinstance(text,str):
        return text.lower()
    return "Invalid Input"

print(to_uppercase("ai coding"))
print(to_lowercase("TEST"))
print(to_uppercase(""))
print(to_lowercase(None))

assert to_uppercase("ai coding") == "AI CODING"
assert to_lowercase("TEST") == "test"
assert to_uppercase("") == ""
assert to_lowercase(None) == "Invalid Input"

print("All tests passed")
```

OUTPUT



EXPLANATION

Functions check if input is string before conversion to avoid runtime errors. This ensures safe handling of invalid inputs.

Task 3 — List Sum Calculator QUESTION

Generate test cases for sum_list(numbers) to calculate sum safely.

PROMPT

Implement list summation function handling empty lists, negatives, and ignoring non-numeric values.

CODE

```
def sum_list(numbers):
    total = 0
for n in
numbers:
        if isinstance(n,(int,float)):
            total += n
return total
print(sum_list([1,2,3]))
print(sum_list([]))
print(sum_list([-1,3,-
4]))
print(sum_list([2,"a",3
```

])) assert

sum_list([1,2,3]) == 6

assert sum_list([]) ==

0 assert sum_list([-

1,3,-4]) == -2 assert

sum_list([2,"a",3]) ==

5 print("All tests

passed")

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

print(sum_list([1,2,3]))
print(sum_list([]))
print(sum_list([-1,3,-4]))
print(sum_list([2,"a",3]))

assert sum_list([1,2,3]) == 6
assert sum_list([]) == 0
assert sum_list([-1,3,-4]) == -2
assert sum_list([2,"a",3]) == 5

print("All tests passed")
```

OUTPUT

```
PS D:\3-2 SEM\AI ASSISTED> python
6
0
-2
5
All tests passed
PS D:\3-2 SEM\AI ASSISTED>
```

EXPLANATION

Loop iterates through list and adds only numeric values. This prevents errors caused by nonnumeric elements.

Task 4 — Student Result Class QUESTION
Generate test cases for StudentResult class with methods add_marks(), calculate_average(), get_result().

PROMPT
Implement class that calculates average and determines pass/fail based on average ≥ 40.

CODE

```
class StudentResult:
def __init__(self):
self.marks = []     def
add_marks(self,mark):
if 0 <= mark <= 100:
            self.marks.append(mark)
        else:
            return "Invalid
Marks"     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" s = StudentResult()
s.add_marks(60)
s.add_marks(70)
s.add_marks(80)
print(s.calculate_avera
```

ge())

print(s.get_result())

assert s.get_result() ==

"Pass" print("All tests

passed")

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

    def add_marks(self,mark):
        if 0 <= mark <= 100:
            self.marks.append(mark)
        else:
            return "Invalid Marks"

    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"

s = StudentResult()
s.add_marks(60)
s.add_marks(70)
s.add_marks(80)

print(s.calculate_average())
print(s.get_result())

assert s.get_result() == "Pass"

print("All tests passed")
```

OUTPUT

```
PS D:\3-2 SEM\AI ASSISTED> python
70.0
Pass
All tests passed
PS D:\3-2 SEM\AI ASSISTED>
```

EXPLANATION

Marks are validated between 0 and 100. Average is calculated and result determined based on threshold rule.

Task 5 — Username Validator QUESTION

Generate test cases for is_valid_username(username).

PROMPT

Create validation function ensuring minimum length 5, no spaces, and only alphanumeric characters.

CODE

```
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
print(is_valid_username("user0
1"))
print(is_valid_username("ai"))
print(is_valid_username("user
name"))
```

print(is_valid_username("user@ 123")) assert is_valid_username("user01") == True assert is_valid_username("ai") == False assert is_valid_username("user name") == False assert is_valid_username("user@123") == False print("All tests passed")

```
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

print(is_valid_username("user01"))
print(is_valid_username("ai"))
print(is_valid_username("user name"))
print(is_valid_username("user@123"))

assert is_valid_username("user01") == True
assert is_valid_username("ai") == False
assert is_valid_username("user name") == False
assert is_valid_username("user@123") == False

print("All tests passed")
```

OUTPUT

```
PS D:\3-2 SEM\AI ASSISTED> python
True
False
False
False
All tests passed
PS D:\3-2 SEM\AI ASSISTED>
```

EXPALANATION

Function checks length, spaces, and allowed characters using built-in string methods. Ensures robust validation logic.