

END SEMESTER LAB EXAM

NAME: CHANDA HARINI

BATCH NO.: 05

ROLL NO.: 2403A510E1

Q1: Generate unit test cases

- Task 1: Use AI to create boundary test cases.

PROMPT:

```
1  Generate boundary test cases for a function that takes marks from 0 to 100.
2  Output table format with columns:
3  Test Case ID, Input, Expected Output, Reason (boundary selection).
4  Pass if marks >= 35 else Fail.
5  Include valid, invalid and out-of-range boundaries.
6  | Test Case ID | Input | Expected Output | Reason (boundary selection) |
```

CODE:

```
> Users > Praneeth Cheekati > Downloads > sunny > exam.py > ...
1  def result(marks):
2      if marks < 0 or marks > 100:
3          return "Error"
4      return "Pass" if marks >= 35 else "Fail"
5
6  # Run Boundary Test Cases
7  test_cases = [-1, 0, 34, 35, 36, 100, 101]
8  for tc in test_cases:
9      print(f"Input: {tc} → Output: {result(tc)}")
0
```

OUTPUT:

```
PS C:\WINDOWS\System32\WindowsPowerShell\v1.0> & "C:/Users/Praneeth Cheekati/AppData/Local/Microsoft/Windows/Praneeth Cheekati/Downloads/sunny/exam.py"
Input: -1 → Output: Error
Input: 0 → Output: Fail
Input: 34 → Output: Fail
Input: 35 → Output: Pass
Input: 36 → Output: Pass
Input: 100 → Output: Pass
Input: 101 → Output: Error
PS C:\WINDOWS\System32\WindowsPowerShell\v1.0>
```

OBSERVATION:

All boundary test cases produced the expected results – values below 0 and above 100 returned “Error”, marks below 35 returned “Fail”, and marks 35 and above returned “Pass”.
This confirms that the function correctly handles both valid and invalid boundary limits.

• Task 2: Implement tests in Python/Java.

PROMPT:

Write a program to test the function `result(marks)` using unit test cases.
Use boundary inputs: -1, 0, 34, 35, 36, 100, 101.
Provide output and ensure pass/fail validation.
Give short code and clean formatted output

CODE:

```
1 def result(marks):
2     if marks < 0 or marks > 100:
3         return "Error"
4     return "Pass" if marks >= 35 else "Fail"
5
6 # Test cases
7 test_data = {
8     -1: "Error",
9     0: "Fail",
10    34: "Fail",
11    35: "Pass",
12    36: "Pass",
13    100: "Pass",
14    101: "Error"
15 }
16
17 print("Running Tests...\n")
18 for marks, expected in test_data.items():
19     output = result(marks)
20     status = "PASS" if output == expected else "FAIL"
21     print(f"Input: {marks} | Expected: {expected} | Got: {output} | Result: {status}")
22
```

OUTPUT:

```
Running Tests...

Input: -1 | Expected: Error | Got: Error | Result: PASS
Input: 0 | Expected: Fail | Got: Fail | Result: PASS
Input: 34 | Expected: Fail | Got: Fail | Result: PASS
Input: 35 | Expected: Pass | Got: Pass | Result: PASS
Input: 36 | Expected: Pass | Got: Pass | Result: PASS
Input: 100 | Expected: Pass | Got: Pass | Result: PASS
Input: 101 | Expected: Error | Got: Error | Result: PASS
PS C:\WINDOWS\System32\WindowsPowerShell\v1.0>
```

OBSERVATION:

All test inputs returned the correct expected results in both Python and Java. The function accurately handles boundary values, valid marks, and invalid out-of-range inputs, proving that the implementation is logically correct and robust.

Q2: Validate grading logic

- Task 1: Use AI to simulate failing tests.

PROMPT:

```
Simulate failing test cases for a grading function that assigns:
1 A for 90-100
2 B for 80-89
3 C for 70-79
4 D for 60-69
5 F for below 60
6 Generate inputs where the expected output does NOT match the function result in order to create failing tests.
7 Return test case table with Input Marks, Expected Output, and Actual Output produced by code.
8
9 |
```

CODE:

```
exam.py 9+  exam1.py 9+  exam2.py  X  = Simulate failing test cases for a grading system - Untitled-
C: > Users > Praneeth Cheekati > Downloads > sunny > exam2.py > ...
1  # ❌ Buggy grading logic (will cause failing test)
2  def grade(marks):
3      if marks > 90:                # Incorrect: excludes 90 from grade A
4          return "A"
5      elif marks >= 80:
6          return "B"
7      elif marks >= 70:
8          return "C"
9      elif marks >= 60:
10         return "D"
11     else:
12         return "F"
13
14 # Test cases to simulate failure
15 test_data = {
16     90: "A",
17     95: "A",
18     89: "B",
19     79: "C",
20     60: "D",
21
15 test_data = {
16     90: "A",
17     95: "A",
18     89: "B",
19     79: "C",
20     60: "D",
21     59: "F"
22 }
23
24 print("Running Failing Tests...\n")
25 for marks, expected in test_data.items():
26     output = grade(marks)
27     status = "PASS" if output == expected else "FAIL"
28     print(f"Input: {marks} | Expected: {expected} | Got: {output} | Result: {status}")
29
```

OUTPUT:

```
Running Failing Tests...

Input: 90 | Expected: A | Got: B | Result: FAIL
Input: 95 | Expected: A | Got: A | Result: PASS
Input: 89 | Expected: B | Got: B | Result: PASS
Input: 79 | Expected: C | Got: C | Result: PASS
Input: 60 | Expected: D | Got: D | Result: PASS
Input: 59 | Expected: F | Got: F | Result: PASS
```

OBSERVATION:

The test failed for marks = 90, meaning the grading logic incorrectly excluded 90 from Grade A.
AI successfully simulated failing tests that revealed the defect.

- Task 2: Correct code until all tests pass

PROMPT:

```
Fix the grading code so that all unit tests pass.  
Grading rules:  
A = 90-100  
B = 80-89  
C = 70-79  
D = 60-69  
F = below 60  
Re-run the same test cases and show PASS for all.  
# -----
```

CODE:

```
C: > Users > Praneeth Cheekati > Downloads > sunny > exam 3.py > ...  
1 # -----  
2 # Test Data (must be declared first)  
3 # -----  
4 test_data = {  
5     90: "A",  
6     95: "A",  
7     89: "B",  
8     79: "C",  
9     60: "D",  
10    59: "F"  
11 }  
12  
13 # -----  
14 # Task 1: Buggy grading logic (simulate failing tests)  
15 # -----  
16 def grade(marks):  
17     if marks > 90:           # Incorrect logic (90 not included in A)  
18         return "A"  
19     elif marks >= 80:  
20         return "B"
```

C:\Users\Praneeth Cheekati\Downloads\sunny > exam 3.py > ...

```
21     elif marks >= 70:
22         return "C"
23     elif marks >= 60:
24         return "D"
25     else:
26         return "F"
27
28     print("Running Failing Tests...\n")
29     for marks, expected in test_data.items():
30         output = grade(marks)
31         status = "PASS" if output == expected else "FAIL"
32         print(f"Input: {marks} | Expected: {expected} | Got: {output} | Result: {status}")
33
34     # -----
35     # Task 2: Correct code until all tests pass
36     # -----
37     def grade(marks):
38         if marks >= 90:             # Correct logic
39             return "A"
40         elif marks >= 80:
```

C:\Users\Praneeth Cheekati\Downloads\sunny > exam 3.py > ...

```
37     def grade(marks):
39         return "A"
40         elif marks >= 80:
41             return "B"
42         elif marks >= 70:
43             return "C"
44         elif marks >= 60:
45             return "D"
46         else:
47             return "F"
48
49     print("\nRunning After Fix...\n")
50     for marks, expected in test_data.items():
51         output = grade(marks)
52         status = "PASS" if output == expected else "FAIL"
53         print(f"Input: {marks} | Expected: {expected} | Got: {output} | Result: {status}")
54
```

OUTPUT:

Running After Fix...

Input: 90		Expected: A		Got: A		Result: PASS
Input: 95		Expected: A		Got: A		Result: PASS
Input: 89		Expected: B		Got: B		Result: PASS
Input: 79		Expected: C		Got: C		Result: PASS
Input: 60		Expected: D		Got: D		Result: PASS
Input: 59		Expected: F		Got: F		Result: PASS

PS C:\WINDOWS\System32\WindowsPowerShell\v1.0>

OBSERVATION:

After updating the grading logic, all test cases passed, confirming the defect was fixed.
The system now correctly assigns Grade A for marks ≥ 90 and works accurately for all grade boundaries.