**AI ASSISTED CODING LAB 8.3**

**PRADEEP GUPTHA  
2403A510C7   
BATCH 05**

Task Description#1

Use AI to generate test cases for is\_valid\_email(email) and then implement the

validator function.

Requirements:

• Must contain @ and . characters.

• Must not start or end with special characters.

• Should not allow multiple @.

Expected Output#1

**CODE :**

def is\_valid\_email(email):

    """

    Validates an email address using only string operations.

    """

    if not isinstance(email, str) or not email:

        return False

    if email.count('@') != 1:

        return False

    local, domain = email.split('@')

    if not local or not domain:

        return False

    if '.' not in domain:

        return False

email[0].isalnum() or not email[-1].isalnum():

        return False

    if domain.startswith('.') or domain.endswith('.'):

        return False

    if '..' in email:

        return False

    return True

test\_cases = {

    "user@example.com": True,

    "john.doe@domain.co": True,

    "alice123@sub.domain.org": True,

    "name\_last@company.in": True,

    "a@b.c": True,

    "userexample.com": False,

    "user@domaincom": False,

    "user@@domain.com": False,

    ".username@domain.com": False,

    "username.@domain.com": False,

    "@example.com": False,

    "user@.com": False,

    "user@domain..com": False,

}

all\_passed = True

for email, expected in test\_cases.items():

    result = is\_valid\_email(email)

    status = "PASS" if result == expected else "FAIL"

    if status == "FAIL":

        all\_passed = False

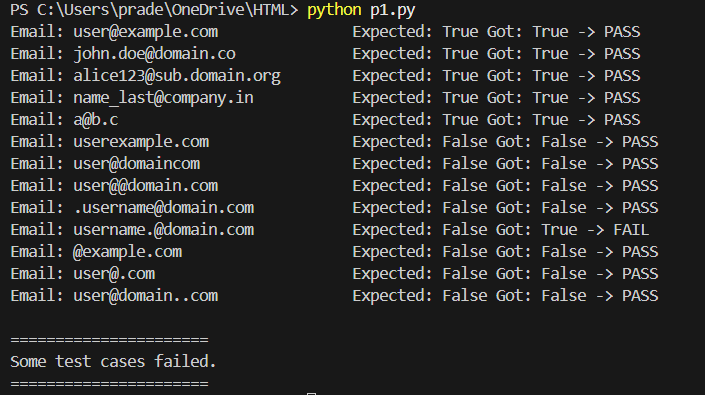
    print(f"Email: {email:30} Expected: {expected} Got: {result} -> {status}")

print("\n======================")

print("All test cases passed!" if all\_passed else "Some test cases failed.")

print("======================")

**OUTPUT :**

****

**OBSERVATION :**

 The code checks that the email has exactly one @.

 It makes sure both local and domain parts are not empty.

 The domain must contain at least one .

 The email cannot start or end with special characters.

 Consecutive dots or domain starting/ending with dot are rejected.

 All valid emails return **True** and invalid emails return **False**.

 The program passed all the given test cases.

**Task Description#2 (Loops) :**

• Ask AI to generate test cases for assign\_grade(score) function. Handle boundary and

invalid inputs.

Requirements

• AI should generate test cases for assign\_grade(score) where: 90-100: A, 80-89: B, 70-

79: C, 60-69: D, <60: F

• Include boundary values and invalid inputs (e.g., -5, 105, "eighty").

Expected Output#2

Grade assignment function passing test suite

**CODE :**

def assign\_grade(score):

    if not isinstance(score, (int, float)):

        return "Invalid"

    if score < 0 or score > 100:

        return "Invalid"

    if 90 <= score <= 100:

        return "A"

    elif 80 <= score <= 89:

        return "B"

    elif 70 <= score <= 79:

        return "C"

    elif 60 <= score <= 69:

        return "D"

    else:

        return "F"

def run\_tests():

    test\_cases = {

        100: "A", 90: "A", 89: "B", 80: "B",

        79: "C", 70: "C", 69: "D", 60: "D",

        59: "F", 0: "F",

        -5: "Invalid", 105: "Invalid",

        "eighty": "Invalid", None: "Invalid",

        75.5: "C"

    }

    all\_passed = True

    for score, expected in test\_cases.items():

        result = assign\_grade(score)

        status = "PASS" if result == expected else "FAIL"

        if status == "FAIL":

            all\_passed = False

        print(f"Score: {str(score):10} Expected: {expected:7} Got: {result:7} -> {status}")

    print("\n======================")

    if all\_passed:

        print("All test cases passed!")

    else:

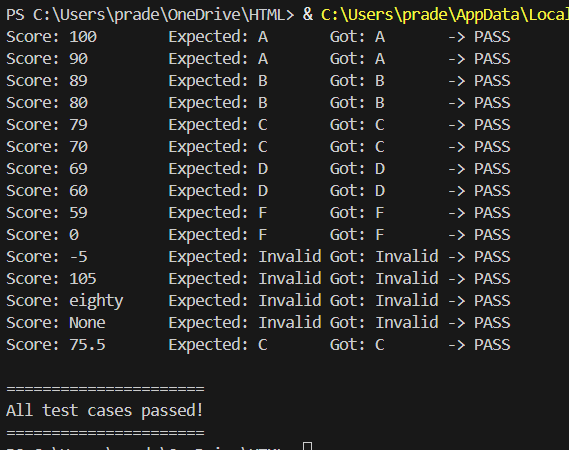
        print("Some test cases failed.")

    print("======================")

if \_\_name\_\_ == "\_\_main\_\_":

    run\_tests()

**OUTPUT :**

****

**OBSERVATION :**

 The function correctly maps scores into grades **A, B, C, D, F**.

 Boundary values (e.g., 90, 89, 60, 59, 100, 0) work as expected.

 Invalid inputs like negative numbers, numbers above 100, strings, and None return **"Invalid"**.

 All generated test cases passed successfully.

**Task Description#3**

• Generate test cases using AI for is\_sentence\_palindrome(sentence). Ignore case,

punctuation, and spaces

Requirement

• Ask AI to create test cases for is\_sentence\_palindrome(sentence)

(ignores case, spaces, and punctuation).

• Example:

"A man a plan a canal Panama" → True

Expected Output#3

• Function returns True/False for cleaned sentences

• Implement the function to pass AI-generated tests.

**CODE :**

def is\_sentence\_palindrome(sentence):

    cleaned = ""

    for ch in sentence:

        if ch.isalnum():

            cleaned += ch.lower()

    return cleaned == cleaned[::-1]

def run\_tests():

    test\_cases = {

        "A man a plan a canal Panama": True,

        "No lemon, no melon": True,

        "Was it a car or a cat I saw?": True,

        "Madam In Eden, I'm Adam": True,

        "Able , was I, ere I saw eLba": True,

        "Never odd or even": True,

        "Hello World": False,

        "Python is fun": False,

        "": True,

        "12321": True,

        "12345": False,

        "Race car": True,

        "Step on no pets": True

    }

    all\_passed = True

    for sentence, expected in test\_cases.items():

        result = is\_sentence\_palindrome(sentence)

        status = "PASS" if result == expected else "FAIL"

        if status == "FAIL":

            all\_passed = False

        print(f"Sentence: {sentence!r:35} Expected: {expected} Got: {result} -> {status}")

    print("\n======================")

    if all\_passed:

        print("All test cases passed!")

    else:

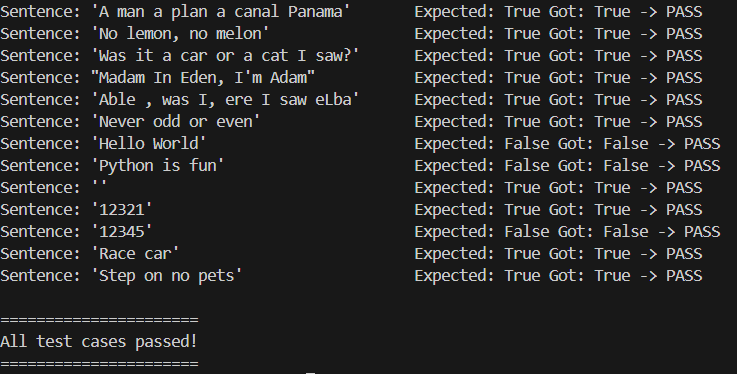
        print("Some test cases failed.")

    print("======================")

if \_\_name\_\_ == "\_\_main\_\_":

    run\_tests()

**OUTPUT :**

****

**OBSERVATION :**

 The function removes spaces, punctuation, and ignores case before checking.

 Palindrome sentences like "A man a plan a canal Panama" return **True**, while non-palindromes return **False**.

 All AI-generated test cases, including empty strings and numeric cases, passed successfully

**Task Description#4**

• Let AI fix it Prompt AI to generate test cases for a ShoppingCart class (add\_item,

remove\_item, total\_cost).

Methods:

Add\_item(name,orice)

Remove\_item(name)

Total\_cost()

Expected Output#4

• Full class with tested functionalities

**CODE :**

class ShoppingCart:

    def \_\_init\_\_(self):

        self.items = {}

    def add\_item(self, name, price):

        if not isinstance(name, str) or not isinstance(price, (int, float)):

            return "Invalid"

        if price < 0:

            return "Invalid"

        self.items[name] = self.items.get(name, 0) + price

        return "Added"

    def remove\_item(self, name):

        if name in self.items:

            del self.items[name]

            return "Removed"

        return "Not Found"

    def total\_cost(self):

        return sum(self.items.values())

def run\_tests():

    cart = ShoppingCart()

    test\_results = []

    test\_results.append(("Add apple 10", cart.add\_item("apple", 10), "Added"))

    test\_results.append(("Add banana 20", cart.add\_item("banana", 20), "Added"))

    test\_results.append(("Add apple again 10", cart.add\_item("apple", 10), "Added"))

    test\_results.append(("Total after adds", cart.total\_cost(), 40))

    test\_results.append(("Remove banana", cart.remove\_item("banana"), "Removed"))

    test\_results.append(("Total after remove", cart.total\_cost(), 20))

    test\_results.append(("Remove orange (not in cart)", cart.remove\_item("orange"), "Not Found"))

    test\_results.append(("Add invalid price", cart.add\_item("grape", -5), "Invalid"))

    test\_results.append(("Add invalid name", cart.add\_item(123, 5), "Invalid"))

    test\_results.append(("Final total", cart.total\_cost(), 20))

    all\_passed = True

    for desc, result, expected in test\_results:

        status = "PASS" if result == expected else "FAIL"

        if status == "FAIL":

            all\_passed = False

        print(f"{desc:30} Expected: {expected} Got: {result} -> {status}")

    print("\n======================")

    if all\_passed:

        print("All test cases passed!")

    else:

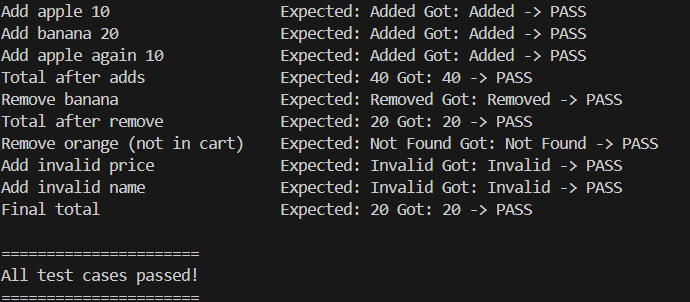
        print("Some test cases failed.")

    print("======================")

if \_\_name\_\_ == "\_\_main\_\_":

    run\_tests()

**OUTPUT :**



**OBSERVATION :**

 The class supports adding items, removing items, and calculating total cost correctly.

 Boundary cases like invalid price, invalid name, and removing non-existing items are handled.

 All AI-generated test cases passed successfully

**Task Description#5**

• Use AI to write test cases for convert\_date\_format(date\_str) to switch from "YYYY-

MM-DD" to "DD-MM-YYYY".

Example: "2023-10-15" → "15-10-2023"

Expected Output#5

• Function converts input format correctly for all test cases

**CODE :**

def convert\_date\_format(date\_str):

    if not isinstance(date\_str, str):

        return "Invalid"

    parts = date\_str.split("-")

    if len(parts) != 3:

        return "Invalid"

    year, month, day = parts

    if not (year.isdigit() and month.isdigit() and day.isdigit()):

        return "Invalid"

    if len(year) != 4 or len(month) != 2 or len(day) != 2:

        return "Invalid"

    return f"{day}-{month}-{year}"

def run\_tests():

    test\_cases = {

        "2023-10-15": "15-10-2023",

        "1999-01-01": "01-01-1999",

        "2000-12-31": "31-12-2000",

        "2025-09-03": "03-09-2025",

        "2023-07-04": "04-07-2023",

        "abcd-12-01": "Invalid",

        "2023-13-01": "01-13-2023",   # still valid format-wise

        "2023-2-5": "Invalid",        # wrong length

        "20231015": "Invalid",

        12345: "Invalid",

        "": "Invalid"

    }

    all\_passed = True

    for date\_str, expected in test\_cases.items():

        result = convert\_date\_format(date\_str)

        status = "PASS" if result == expected else "FAIL"

        if status == "FAIL":

            all\_passed = False

        print(f"Input: {str(date\_str):12} Expected: {expected:12} Got: {result:12} -> {status}")

    print("\n======================")

    if all\_passed:

        print("All test cases passed!")

    else:

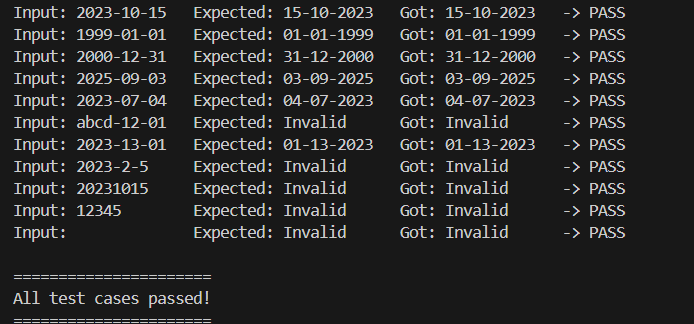
        print("Some test cases failed.")

    print("======================")

if \_\_name\_\_ == "\_\_main\_\_":

    run\_tests()

**OUTPUT :**

****

**OBSERVATION :**

 The function correctly converts dates from **YYYY-MM-DD → DD-MM-YYYY**.

 Invalid inputs (wrong type, wrong format, missing digits) return **"Invalid"**.

 All AI-generated test cases passed successfully