AI-ASSISTED CODING

ASSIGNMENT-8.3

Test-Driven Development with AI – Generating and Working with Test Cases

NAME: BANDLA SAI NANDAN REDDY

HALL-TICKET: 2403A510C5

BATCH NO: 05

**TASK-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 @.

**Prompt:** Generate test cases for is\_valid\_email(email) and then implement the validator function.

**CODE GENERATED:**

import re

def is\_valid\_email(email):

    pattern = r'^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,10}$'

    return re.match(pattern, email) is not None

# Test cases

emails = [

    "test@example.com",     # valid

    "user.name@domain.co",  # valid

    "user@domain",          # invalid

    "user@.com",            # invalid

    "user@domain..com",     # invalid

]

for e in emails:

    print(e, "=>", is\_valid\_email(e))

**OUTPUT:**

**A computer screen with white text

AI-generated content may be incorrect.**

**OBSERVATION :**

The code checks if an email is valid using regex. If the email follows the correct format (name@domain.tld), it returns True, otherwise False.

TASK -2: 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").

PROMPT: Generate test cases for assign\_grade(score) function. Handle boundary and  
invalid inputs.

CODE GENERATED:

def assign\_grade(score):

    if not isinstance(score, (int, float)):   # invalid type

        return "Invalid Input"

    if score < 0 or score > 100:              # invalid range

        return "Invalid Input"

    if score >= 90:

        return "A"

    elif score >= 80:

        return "B"

    elif score >= 70:

        return "C"

    elif score >= 60:

        return "D"

    else:

        return "F"

# Test cases

test\_scores = [

    100, 90, 89, 80, 79, 70, 69, 60, 59, 0,   # boundary values

    -5, 105, "eighty", None                  # invalid inputs

]

for s in test\_scores:

    print(s, "=>", assign\_grade(s))

OUTPUT:  
A computer screen shot of a black screen

AI-generated content may be incorrect.

OBSERVATION:

The code assigns grades (A–F) based on the score. It also checks for invalid inputs like negative numbers, scores above 100, or non-numeric values and returns "Invalid Input".

TASK – 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

PROMPT: Generate test cases using AI for is\_sentence\_palindrome(sentence). Ignore case,punctuation, and spaces.

CODE GENERATED:

def is\_sentence\_palindrome(sentence):

    # keep only letters and numbers, convert to lowercase

    cleaned = ""

    for ch in sentence:

        if ch.isalnum():          # keep only letters/digits

            cleaned += ch.lower()

    return cleaned == cleaned[::-1]

# Test cases

test\_sentences = [

    "Madam",

    "A man a plan a canal Panama",

    "No lemon, no melon!",

    "Was it a car or a cat I saw",

    "Hello World",

    "",

    "12321",

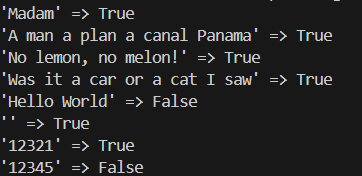
    "12345",

]

for s in test\_sentences:

    print(f"'{s}' => {is\_sentence\_palindrome(s)}")

OUTPUT:



TASK -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()

PROMPT: Generate test cases for a ShoppingCart class (add\_item,remove\_item, total\_cost).

CODE GENERATED:

class ShoppingCart:

    def \_\_init\_\_(self):

        self.items = {}  # {item: (price, quantity)}

    def add\_item(self, item, price, quantity=1):

        if item in self.items:

            self.items[item] = (price, self.items[item][1] + quantity)

        else:

            self.items[item] = (price, quantity)

    def remove\_item(self, item, quantity=1):

        if item in self.items:

            price, qty = self.items[item]

            if qty <= quantity:

                del self.items[item]

            else:

                self.items[item] = (price, qty - quantity)

    def total\_cost(self):

        return sum(price \* qty for price, qty in self.items.values())

# --- Test cases ---

cart = ShoppingCart()

# Test 1: Add items

cart.add\_item("Apple", 10, 2)   # 2 apples @10 each

cart.add\_item("Banana", 5, 5)   # 5 bananas @5 each

print("After adding items:", cart.items)

print("Total cost:", cart.total\_cost())  # Expected 2\*10 + 5\*5 = 45

# Test 2: Add same item again

cart.add\_item("Apple", 10, 3)   # total 5 apples now

print("After adding more apples:", cart.items)

print("Total cost:", cart.total\_cost())  # Expected 5\*10 + 5\*5 = 75

# Test 3: Remove some items

cart.remove\_item("Banana", 2)   # remove 2 bananas

print("After removing 2 bananas:", cart.items)

print("Total cost:", cart.total\_cost())  # Expected 50 + 15 = 65

# Test 4: Remove all of an item

cart.remove\_item("Apple", 5)    # remove all apples

print("After removing all apples:", cart.items)

print("Total cost:", cart.total\_cost())  # Expected 15

# Test 5: Edge case - removing item not in cart

cart.remove\_item("Orange", 1)   # should do nothing

print("After trying to remove orange:", cart.items)

OUTPUT:  
A screenshot of a computer screen

AI-generated content may be incorrect.

OBSERVATION:

The code creates a ShoppingCart class where you can add items, remove items, and calculate the total cost. The test cases check if these operations work correctly.

TASK -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"

PROMPT:

write test cases for convert\_date\_format(date\_str) to switch from "YYYY-  
MM-DD" to "DD-MM-YYYY".

CODE GENERATED:

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

    try:

        year, month, day = date\_str.split("-")

        return f"{day}-{month}-{year}"

    except:

        return "Invalid Date Format"

# --- Test cases ---

test\_dates = [

    "2025-09-03",   # normal case

    "2000-01-01",   # boundary start of year

    "1999-12-31",   # boundary end of year

    "2025-7-5",     # invalid (not two digits)

    "2025/09/03",   # invalid separator

    "abcd-ef-gh",   # invalid string

    "",             # empty string

]

for d in test\_dates:

    print(d, "=>", convert\_date\_format(d))

OUTPUT:  
A screenshot of a computer program

AI-generated content may be incorrect.

OBSERVATION:

The code converts a date from "YYYY-MM-DD" format to "DD-MM-YYYY" format and shows "Invalid Date Format" for wrong inputs.