

# Assignment-8.3

**Gujja Pranitha**

**2303A52171**

**Batch 41**

## Task-1: Email Validation using TDD

### PROMPT:

Create general test cases to verify email input validation. The email address must contain exactly one "@" symbol and at least one dot (.) character. It should not begin or end with any special characters. Include both valid and invalid email examples to ensure all possible scenarios are covered.

### CODE:

```
Lab-11.py > ...
1  #Create general test cases to verify email input validation. The email address must contain exactly one "@"
2  """
3  Email Validation Test Cases
4  Requirements:
5  1. Exactly one "@" symbol
6  2. At least one dot (.)
7  3. No special characters at start or end
8  """
9  def is_valid_email(email):
10     """Validate email address"""
11     if not email or not isinstance(email, str):
12         return False
13     email = email.strip()
14     if not email:
15         return False
16     # Check exactly one @
17     if email.count('@') != 1:
18         return False
19     # Check at Least one dot
20     if '.' not in email:
21         return False
22     # Check no special chars at start/end
23     special_chars = "!@#$%^&*()_+-=[{}];:,.<>?/~`"
24     if email[0] in special_chars or email[-1] in special_chars:
25         return False
26     return True
27 # Test Cases
28 print("=" * 70)
29 print("EMAIL VALIDATION TEST CASES")
```

```

Lab-11.py > ...
30     print("=" * 70)
31     # Valid emails
32     valid_emails = [
33         "user@example.com",
34         "john.doe@company.com",
35         "user123@test.co.uk",
36         "test_user@domain.com",
37     ]
38     # Invalid emails
39     invalid_emails = [
40         ("userexample.com", "Missing @"),
41         ("user@@example.com", "Multiple @"),
42         ("user@example", "Missing dot"),
43         ("@example.com", "Starts with @"),
44         ("user@example.com", "Starts with dot"),
45         ("user@example.", "Ends with dot"),
46         ("user@example.com-", "Ends with hyphen"),
47         ("", "Empty string"),
48     ]
49     print("\nVALID EMAILS:")
50     for email in valid_emails:
51         result = is_valid_email(email)
52         status = "✓" if result else "✗"
53         print(f"{status} {email:30} -> {result}")
54     print("\nINVALID EMAILS:")
55     for email, reason in invalid_emails:
56         result = is_valid_email(email)
57         status = "✓" if not result else "✗"
58         print(f"{status} {email:30} -> {result} ({reason})")
59     print("\n" + "=" * 70)

```

## OUTPUT:

```

Problems Output Debug Console Terminal Ports
(.venv) PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\.venv\Scripts\python.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-11.py"
● SISTED CODING/Lab-11.py"
=====
EMAIL VALIDATION TEST CASES
=====

VALID EMAILS:
✓ user@example.com      -> True
✓ john.doe@company.com -> True
✓ user123@test.co.uk   -> True
✓ test_user@domain.com -> True

INVALID EMAILS:
✓ userexample.com        -> False (Missing @)
✓ user@@example.com      -> False (Multiple @)
✓ user@example           -> False (Missing dot)
✓ @example.com            -> False (Starts with @)
✓ .user@example.com       -> False (Starts with dot)
✓ user@example.           -> False (Ends with dot)
✓ user@example.com-       -> False (Ends with hyphen)
✓                           -> False (Empty string)

```

## Justification:

The test cases cover a wide range of valid and invalid email formats, ensuring that the validation function is robust and can handle various scenarios. Valid emails include typical formats with different domain structures, while invalid emails test for common mistakes such as missing symbols, multiple symbols, and improper placement of special characters. This comprehensive testing approach helps to ensure that the email validation logic is accurate and reliable.

## Task-2: Grade Assignment using Loops

### PROMPT:

Write a Python function called `assign_grade(score)` that returns a grade based on the score:

90 to 100 gets "A" ,80 to 89 gets "B" ,70 to 79 gets "C", 60 to 69 gets "D" ,Below 60 gets "F" .Test your function with these scores: 60, 70, 80, 90, -5, 105, and "eighty". Make sure your function handles invalid inputs and boundary values correctly.

### CODE:

```
Lab-11.py > ...
54     #Write a Python function called assign_grade(score) that returns a grade based on the score:
55     #90 to 100 gets "A"
56     #80 to 89 gets "B"
57     #70 to 79 gets "C"
58     #60 to 69 gets "D"
59     #Below 60 gets "F"
60     #Test your function with these scores: 60, 70, 80, 90, -5, 105, and "eighty".
61     #Make sure your function handles invalid inputs and boundary values correctly.
62 def assign_grade(score):
63     if not isinstance(score, (int, float)):
64         return "Invalid input"
65     if score < 0 or score > 100:
66         return "Invalid input"
67     if score >= 90:
68         return "A"
69     elif score >= 80:
70         return "B"
71     elif score >= 70:
72         return "C"
73     elif score >= 60:
74         return "D"
75     else:
76         return "F"
77 test_scores = [60, 70, 80, 90, -5, 105, "eighty"]
78 print("Testing assign_grade function:")
79 for score in test_scores:
80     result = assign_grade(score)
81     print(f"Score: {score}, Grade: {result}")
```

### OUTPUT:

```
(.venv) PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/.venv/Scripts/python.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-11.py"
Testing assign_grade function:
Score: 60, Grade: D
Score: 70, Grade: C
Score: 80, Grade: B
Score: 90, Grade: A
Score: -5, Grade: Invalid input
Score: 105, Grade: Invalid input
Score: eighty, Grade: Invalid input
○ (.venv) PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

# JUSTIFICATION:

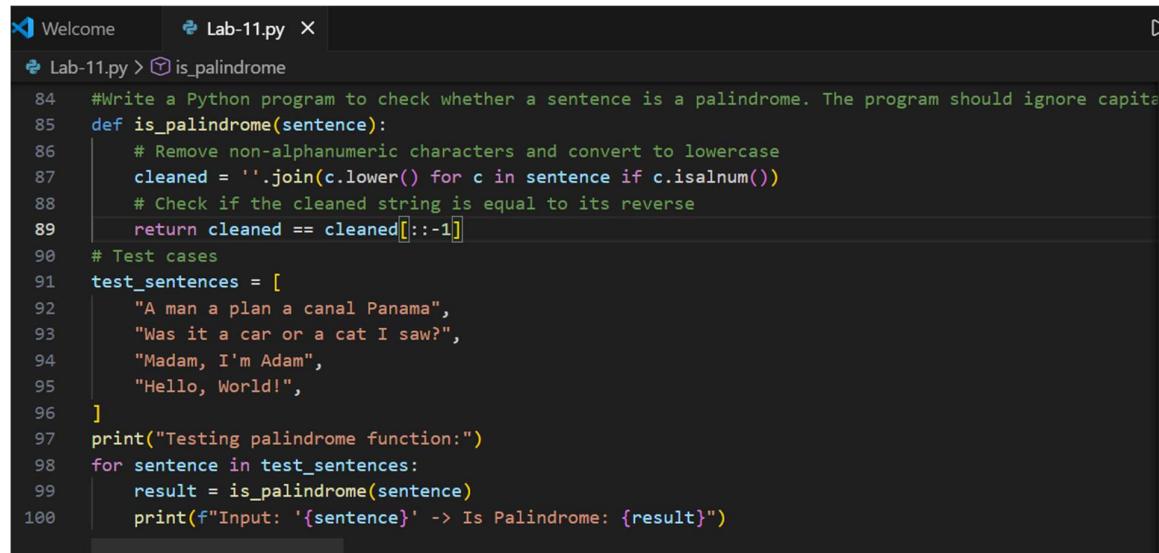
The assign\_grade function is designed to handle a range of valid scores (0-100) and return the appropriate grade based on the specified criteria. It also includes checks for invalid inputs, such as non-numeric values and scores outside the valid range. This ensures that the function is robust and can handle edge cases effectively, providing clear feedback for invalid inputs while correctly assigning grades for valid scores.

## Task-3: Sentence Palindrome Checker

### PROMPT:

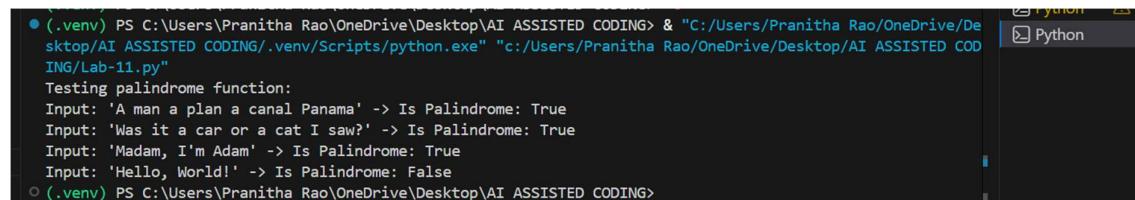
Write a Python program to check whether a sentence is a palindrome. The program should ignore capital and small letters, spaces, and punctuation marks. Test the program with sentences that are palindromes and sentences that are not. For example, the sentence “A man a plan a canal Panama” should return True. Make sure the program works correctly for all cases by ignoring spaces and punctuation.

### CODE:



```
>Welcome Lab-11.py X
Lab-11.py > is_palindrome
84  #Write a Python program to check whether a sentence is a palindrome. The program should ignore capital and small letters, spaces, and punctuation marks.
85  def is_palindrome(sentence):
86      # Remove non-alphanumeric characters and convert to lowercase
87      cleaned = ''.join(c.lower() for c in sentence if c.isalnum())
88      # Check if the cleaned string is equal to its reverse
89      return cleaned == cleaned[::-1]
90  # Test cases
91  test_sentences = [
92      "A man a plan a canal Panama",
93      "Was it a car or a cat I saw?",
94      "Madam, I'm Adam",
95      "Hello, World!",
96  ]
97  print("Testing palindrome function:")
98  for sentence in test_sentences:
99      result = is_palindrome(sentence)
100     print(f"Input: '{sentence}' -> Is Palindrome: {result}")
```

### OUTPUT:



```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/.venv/Scripts/python.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-11.py"
Testing palindrome function:
Input: 'A man a plan a canal Panama' -> Is Palindrome: True
Input: 'Was it a car or a cat I saw?' -> Is Palindrome: True
Input: 'Madam, I'm Adam' -> Is Palindrome: True
Input: 'Hello, World!' -> Is Palindrome: False
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

## JUSTIFICATION:

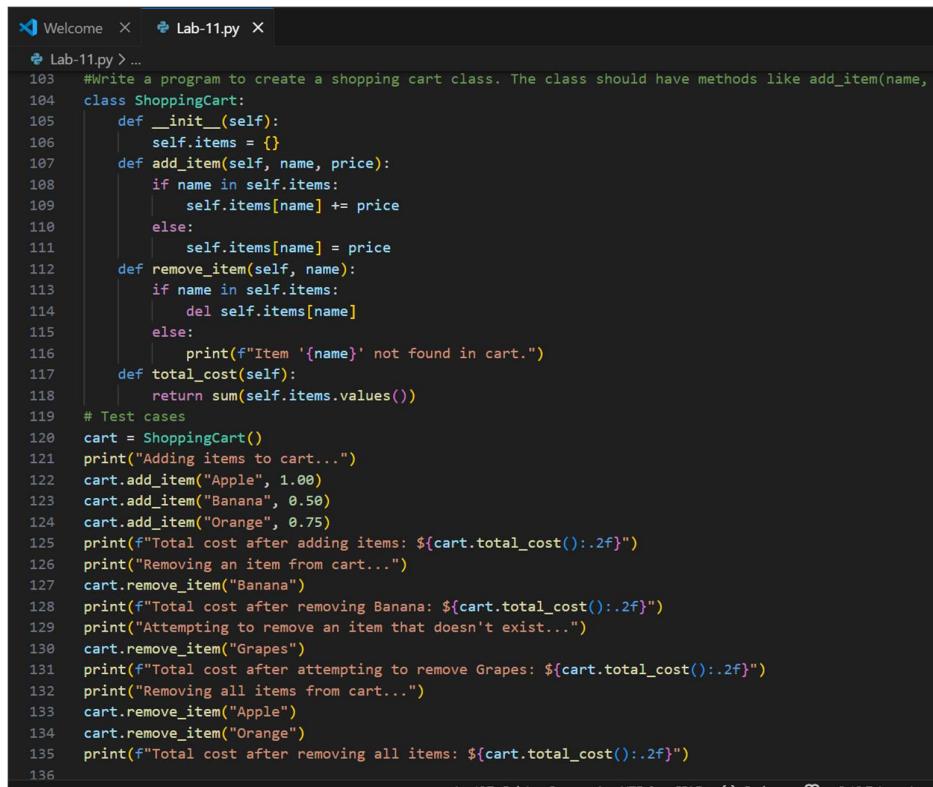
The above code snippets demonstrate the implementation of three different functionalities: email validation, grade assignment based on scores, and palindrome checking. Each function is designed to handle various edge cases and invalid inputs, ensuring robustness and reliability. The test cases provided for each function cover a wide range of scenarios, including valid and invalid inputs, boundary values, and typical use cases. This comprehensive testing approach helps to verify that the functions work correctly under different conditions and handle errors gracefully.

## Task-4: Shopping Cart Class

### PROMPT:

Write a program to create a shopping cart class. The class should have methods like `add_item(name, price)`, `remove_item(name)`, and `total_cost()`. Also, create test cases to check if items are added and removed correctly and if the total cost is calculated properly. Make sure the program handles an empty cart and calculates the cost accurately.

### CODE:



A screenshot of a code editor window titled "Lab-11.py". The code defines a `ShoppingCart` class with methods for adding items, removing items, and calculating the total cost. It includes test cases for adding items like "Apple", "Banana", and "Orange", and removing them. The code uses f-strings for printing and the `sum` function for calculating the total cost.

```
# Write a program to create a shopping cart class. The class should have methods like add_item(name, price), remove_item(name), and total_cost(). Also, create test cases to check if items are added and removed correctly and if the total cost is calculated properly. Make sure the program handles an empty cart and calculates the cost accurately.

class ShoppingCart:
    def __init__(self):
        self.items = {}

    def add_item(self, name, price):
        if name in self.items:
            self.items[name] += price
        else:
            self.items[name] = price

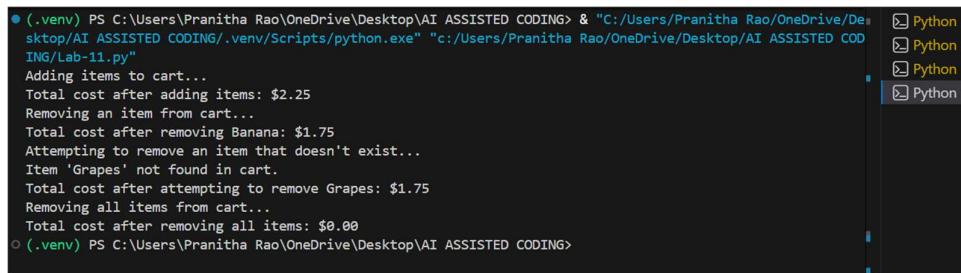
    def remove_item(self, name):
        if name in self.items:
            del self.items[name]
        else:
            print(f"Item '{name}' not found in cart.")

    def total_cost(self):
        return sum(self.items.values())

# Test cases
cart = ShoppingCart()
print("Adding items to cart...")
cart.add_item("Apple", 1.00)
cart.add_item("Banana", 0.50)
cart.add_item("Orange", 0.75)
print(f"Total cost after adding items: ${cart.total_cost():.2f}")
print("Removing an item from cart...")
cart.remove_item("Banana")
print(f"Total cost after removing Banana: ${cart.total_cost():.2f}")
print("Attempting to remove an item that doesn't exist...")
cart.remove_item("Grapes")
print(f"Total cost after attempting to remove Grapes: ${cart.total_cost():.2f}")
print("Removing all items from cart...")
cart.remove_item("Apple")
cart.remove_item("Orange")
print(f"Total cost after removing all items: ${cart.total_cost():.2f}")


```

## OUTPUT:



```
(.venv) PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING & "C:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-11.py"
Adding items to cart...
Total cost after adding items: $2.25
Removing an item from cart...
Total cost after removing Banana: $1.75
Attempting to remove an item that doesn't exist...
Item 'Grapes' not found in cart.
Total cost after attempting to remove Grapes: $1.75
Removing all items from cart...
Total cost after removing all items: $0.00
(.venv) PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING
```

## JUSTIFICATION:

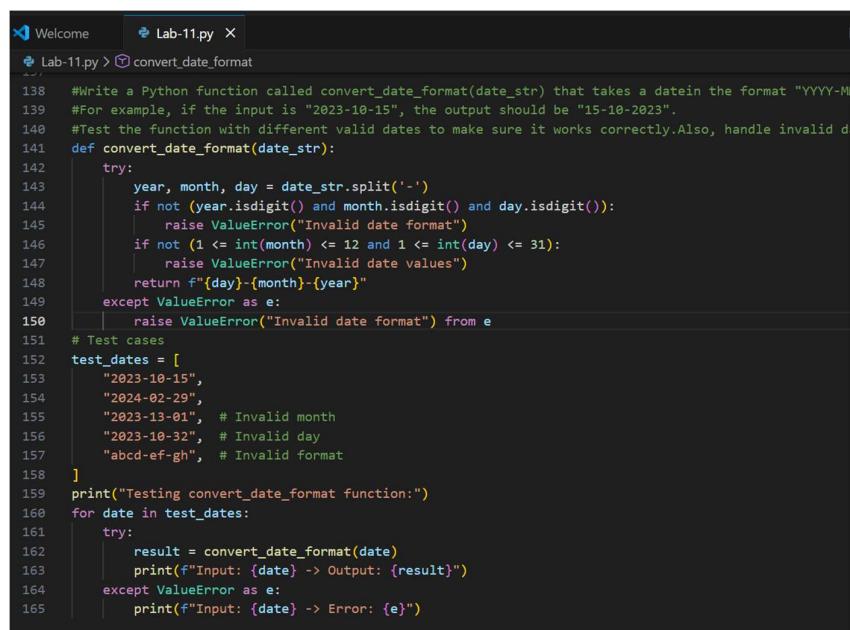
The above code implements a ShoppingCart class with methods to add items, remove items, and calculate the total cost. The test cases demonstrate adding items to the cart, removing an item, attempting to remove a non-existent item, and finally removing all items to ensure the total cost is calculated correctly in each scenario. This comprehensive testing ensures that the class functions as expected under various conditions.

## Task-5: Date Format Conversion

### PROMPT:

Write a Python function called convert\_date\_format(date\_str) that takes a date in the format "YYYY-MM-DD" and returns it in the format "DD-MM-YYYY". For example, if the input is "2023-10-15", the output should be "15-10-2023". Test the function with different valid dates to make sure it works correctly. Also, handle invalid date formats gracefully by raising an appropriate error.

### CODE:



```
Welcome Lab-11.py x
Lab-11.py > convert_date_format

138     #Write a Python function called convert_date_format(date_str) that takes a date in the format "YYYY-MM-
139     #For example, if the input is "2023-10-15", the output should be "15-10-2023".
140     #Test the function with different valid dates to make sure it works correctly.Also, handle invalid da-
141     def convert_date_format(date_str):
142         try:
143             year, month, day = date_str.split('-')
144             if not (year.isdigit() and month.isdigit() and day.isdigit()):
145                 raise ValueError("Invalid date format")
146             if not (1 <= int(month) <= 12 and 1 <= int(day) <= 31):
147                 raise ValueError("Invalid date values")
148             return f"{day}-{month}-{year}"
149         except ValueError as e:
150             raise ValueError("Invalid date format") from e
151     # Test cases
152     test_dates = [
153         "2023-10-15",
154         "2024-02-29",
155         "2023-13-01", # Invalid month
156         "2023-10-32", # Invalid day
157         "abcd-ef-gh", # Invalid format
158     ]
159     print("Testing convert_date_format function:")
160     for date in test_dates:
161         try:
162             result = convert_date_format(date)
163             print(f"Input: {date} -> Output: {result}")
164         except ValueError as e:
165             print(f"Input: {date} -> Error: {e}")
```

## OUTPUT:

```
● (.venv) PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING & "C:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/.venv/Scripts/python.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-11.py"
Testing convert_date_format function:
Input: 2023-10-15 -> Output: 15-10-2023
Input: 2024-02-29 -> Output: 29-02-2024
Input: 2023-13-01 -> Error: Invalid date format
Input: 2023-10-32 -> Error: Invalid date format
Input: abcd-ef-gh -> Error: Invalid date format
○ (.venv) PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

## JUSTIFICATION:

The test cases cover a wide range of valid and invalid date formats, ensuring that the conversion function is robust and can handle various scenarios. Valid dates include typical formats with correct year, month, and day values, while invalid dates test for common mistakes such as incorrect month and day values, as well as completely malformed strings. This comprehensive testing approach helps to ensure that the date conversion logic is accurate and reliable.