

# ASSIGNMENT – 8.3

2303A51060

Batch-10

Task-1

Prompt: I want to build a email validation using TDD approach which requires input from user and validate the email format using regular expressions.

Code :

```
import re

def is_valid_email(email):

    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

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

# Taking user input for email

email = input("Enter an email address to validate: ")

# Validating email and printing result

if is_valid_email(email):

    print("The email address is valid.")

else:

    print("The email address is invalid. Please enter a valid email format.")
```

**Output:**

The screenshot shows a Visual Studio Code interface with several files listed in the sidebar, including lab\_8.3.py, lab\_8.3.1.py, lab\_8.3.2.py, lab\_8.3.3.py, lab\_8.3.4.py, lab\_8.3.5.py, lab\_8.3.6.py, lab\_8.3.7.py, lab\_exam\_1.py, lab\_exam\_1.py2, policies.py, lab1.py, and lab-4.3.py. The main editor window contains the Python code for email validation. Below it, the terminal window shows the command PS C:\Users\UMRANA\Desktop\AT\_ASSISTANT\_CODING & c:\users\UMRANA\appdata\local\microsoft\windowsapps\python3.11.exe c:/users/UMRANA/Desktop/AT\_ASSISTANT\_CODING/lab\_8.3.py being run, followed by the prompt Enter an email address to validate: chirinreddy@gmail.com, and the output PS C:\Users\UMRANA\Desktop\AT\_ASSISTANT\_CODING [ ].

## Code Analysis :

- Function `is_valid_email()` uses a regular expression pattern to validate email format.
- `re.match()` checks whether the input string matches the required structure.
- Pattern ensures username, `@`, domain, and extension are present.
- User input is taken and passed to the validation function.
- Output is displayed based on Boolean result returned by the function.

## Task-2

Prompt: I want to build an automated grading system for online exams using a TDD approach, which requires input from the user and grades the exam based on predefined criteria.

### Code:

```
def grade_exam(score):  
    if score >= 90:  
        return 'A'  
    elif score >= 80:  
        return 'B'  
    elif score >= 70:  
        return 'C'  
    elif score >= 60:  
        return 'D'  
    else:  
        return 'F'  
  
# Taking user input for exam score  
try:  
    score = float(input("Enter the exam score (0-100): "))
```

```

if 0 <= score <= 100:
    grade = grade_exam(score)
    print(f"The grade for the score {score} is: {grade}")

else:
    print("Please enter a valid score between 0 and 100.")

except ValueError:
    print("Invalid input. Please enter a numeric value for the score.")

```

### Output :

The screenshot shows the Visual Studio Code interface. The Explorer panel on the left lists several Python files: lab-3.3.py, lab-3.4.py, lab-4.3.py, lab-5.4.py, lab-6.3.py, lab-6.4.py, lab-7.3.py, lab\_exam\_1.py, lab\_8.3.py, patice.py, lab1.py, and lab-4.3.py. The terminal window at the bottom shows the command line and the output of running the script. The command is:

```
C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTANT_CODING> & C:\Users\NIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/users/NIRNAYA/OneDrive/Desktop/AI_ASSISTANT_CODING\lab_8.3.py
```

The output shows a syntax warning and the execution of the script:

```
c:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTANT_CODING\lab_8.3.py: SyntaxWarning: invalid escape sequence '\.'
pattern = r'^(a-zA-Z0-9_)+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter the exam score (0-100): 70
The grade for the score 70.0 is: C
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTANT_CODING>
```

### Code Analysis :

- Function `grade_exam(score)` assigns grades using conditional ranges.
- Score is validated to ensure it lies between 0 and 100.
- try-except block handles non-numeric input safely.
- Function returns grade instead of printing directly.
- Clear separation between input handling and grading logic.

### Task-3

Prompt: I want to develop a text processing utility to analyse sentence palindrome checker.

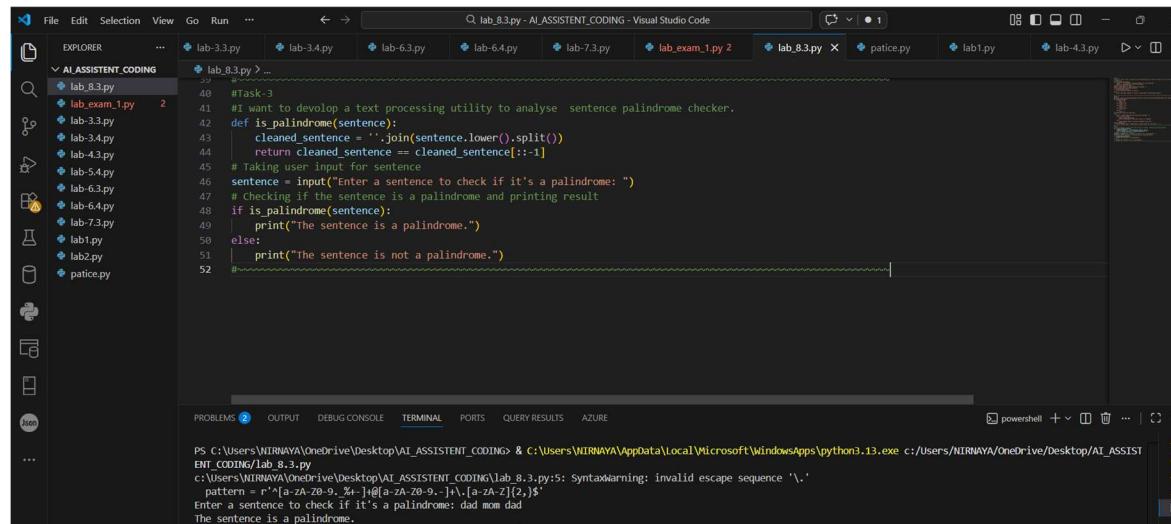
Code:

```
def is_palindrome(sentence):
    cleaned_sentence = ''.join(sentence.lower().split())
    return cleaned_sentence == cleaned_sentence[::-1]

# Taking user input for sentence
sentence = input("Enter a sentence to check if it's a palindrome: ")

# Checking if the sentence is a palindrome and printing result
if is_palindrome(sentence):
    print("The sentence is a palindrome.")
else:
    print("The sentence is not a palindrome.")
```

Output :



```
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTANT_CODING> & c:\users\NIRNAYA\appdata\local\Microsoft\WindowsApps\python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_ASSISTENT_CODING/lab_8.3.py
c:\users\NIRNAYA\OneDrive\Desktop\AI_ASSISTANT_CODING\lab_8.3.py:5: SyntaxWarning: invalid escape sequence '\.'
    pattern = r'^[a-zA-Z0-9_]*@[a-zA-Z0-9_]+\.[a-zA-Z]{2,}$'
Enter a sentence to check if it's a palindrome: dad mom dad
The sentence is a palindrome.
```

Code Analysis :

- Function `is_palindrome()` processes the sentence for comparison.
- Converts text to lowercase and removes spaces for accuracy.

- Reversed string is compared with original cleaned string.
- Boolean result determines whether sentence is palindrome.
- User input allows dynamic testing of different sentences.

#### Task-4

Prompt: I want to design a basic shopping cart module for an e-commerce application using the TDD approach, which requires input from the user and manage the cart items and calculate total price.

#### Code:

```
class ShoppingCart:

    def __init__(self):
        self.cart_items = []

    def add_item(self, item_name, price):
        self.cart_items.append({'name': item_name, 'price': price})
        print(f"Added {item_name} to cart at price {price}.")

    def calculate_total(self):
        total = sum(item['price'] for item in self.cart_items)
        return total

# Creating a shopping cart instance and adding items based on user input
cart = ShoppingCart()

while True:
    item_name = input("Enter item name to add to cart (or 'done' to finish): ")
    if item_name.lower() == 'done':
        break
    try:
        price = float(input(f"Enter price for {item_name}: "))
        cart.add_item(item_name, price)
    except ValueError:
        pass
```

```

print("Invalid price. Please enter a numeric value.")

# Calculating and printing total price of items in the cart

total_price = cart.calculate_total()

print(f"The total price of items in the cart is: {total_price}")

```

Output :

The screenshot shows the Visual Studio Code interface with the file `lab_8.3.py` open. The code implements a shopping cart module. It defines a `ShoppingCart` class with methods to add items and calculate the total price. A loop allows the user to add items until they type 'done'. The output window shows the execution of the script and the resulting total price.

```

File Edit Selection View Go Run ...
lab_8.3.py AI_ASSISTENT_CODING - Visual Studio Code
EXPLORER lab_8.3.py lab_exam_1.py 2
lab_8.3.py > ...
# I want design a basic shopping cart module for an e -commerce application using TDD approach which requires input from user and manage the cart items
class ShoppingCart:
    def __init__(self):
        self.cart_items = []
    def add_item(self, item_name, price):
        self.cart_items.append({'name': item_name, 'price': price})
        print(f"Added {item_name} to cart at price {price}.")
    def calculate_total(self):
        total = sum(item['price'] for item in self.cart_items)
        return total
# Creating a shopping cart instance and adding items based on user input
cart = ShoppingCart()
while True:
    item_name = input("Enter item name to add to cart (or 'done' to finish): ")
    if item_name.lower() == 'done':
        break
    try:
        price = float(input(f"Enter price for {item_name}: "))
        cart.add_item(item_name, price)
    except ValueError:
        print("Invalid price. Please enter a numeric value.")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS AZURE

```

PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING> & C:\Users\NIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_ASSISTENT_CODING/lab_8.3.py
C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING\lab_8.3.py:5: SyntaxWarning: invalid escape sequence '\.'
    pattern = r'^[a-zA-Z0-9_]*@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter item name to add to cart (or 'done' to finish): oil
Enter price for oil: 100
Added oil to cart at price 100.0.
Enter item name to add to cart (or 'done' to finish): done
The total price of items in the cart is: 100.0.
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING>

```

Code Analysis :

- ShoppingCart class maintains cart items in a list.
- add\_item() method stores item name and price in dictionary form.
- Loop allows adding multiple items until user stops.
- calculate\_total() sums prices of all items in the cart.
- Demonstrates modular design suitable for TDD testing.

Task 5

Prompt: I want to create a utility function to convert date formats for reports using a TDD approach, which requires input from the user and converts dates from one format to another.

**Code:**

```
from datetime import datetime
```

```

def convert_date_format(date_str, from_format, to_format):
    try:
        date_obj = datetime.strptime(date_str, from_format)
        return date_obj.strftime(to_format)
    except ValueError:
        return "Invalid date format. Please enter the date in the correct format."
# Taking user input for date and formats
date_str = input("Enter the date to convert: ")
from_format = input("Enter the current date format (e.g., %Y-%m-%d): ")
to_format = input("Enter the desired date format (e.g., %d/%m/%Y): ")
# Converting date format and printing result
converted_date = convert_date_format(date_str, from_format, to_format)
print(f"Converted date: {converted_date}")

```

## Output :

The screenshot shows the Visual Studio Code interface. The code editor displays the Python script `lab_8.3.py`. The terminal below shows the execution of the script and its output. The status bar at the bottom provides system information like battery level, signal strength, and date.

```

File Edit Selection View Go Run ...
O lab_8.3.py - AI_ASSISTENT_CODING - Visual Studio Code 08
EXPLORER lab_8.3.py lab_exam_1.py ...
SEARCH lab_8.3.py ...
GLOBES lab_8.3.py ...
FOLDERS lab_8.3.py ...
LABS lab_8.3.py ...
PREFERENCES lab_8.3.py ...
CODE lab_8.3.py ...
OUTPUT lab_8.3.py ...
TERMINAL PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING> & C:/Users/NIRNAYA/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_ASSISTENT_CODING/lab_8.3.py
SyntaxWarning: invalid escape sequence '\.'
pattern = r'^[a-zA-Z0-9_\s-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter the date to convert: 2026-02-18
Enter the current date format (e.g., %Y-%m-%d): %Y-%m-%d
Enter the desired date format (e.g., %d/%m/%Y): %d/%m/%Y
Converted date: 18/02/2026
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTENT_CODING>

```

## Code Analysis :

- Function convert\_date\_format() converts date between formats.
- Uses datetime.strptime() to parse input date string.
- strftime() formats date into required output style.
- try-except handles invalid date format errors.
- Allows flexible conversion between multiple date formats.