

## 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
```

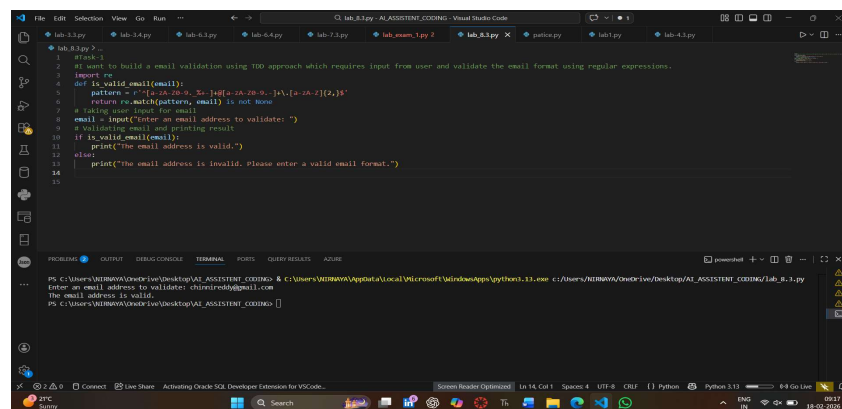
```
if is_valid_email(email):
```

```
    print("The email address is valid.")
```

```
else:
```

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

Output:



```
1 # Task 1
2 # I want to build a email validation using TDD approach which requires input from user and validate the email format using regular expressions.
3 import re
4 def is_valid_email(email):
5     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
6     return re.match(pattern, email) is not None
7 # Taking user input for email
8 email = input("Enter an email address to validate: ")
9 # Validating email and printing result
10 if is_valid_email(email):
11     print("The email address is valid.")
12 else:
13     print("The email address is invalid. Please enter a valid email format.")
14
15
```

Terminal Output:

```
PS C:\Users\NIRAV\OneDrive\Desktop\AI_ASSISTENT_CODING> python lab_8_1.py
Enter an email address to validate: chirereddy@gmail.com
The email address is valid.
PS C:\Users\NIRAV\OneDrive\Desktop\AI_ASSISTENT_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}")
```

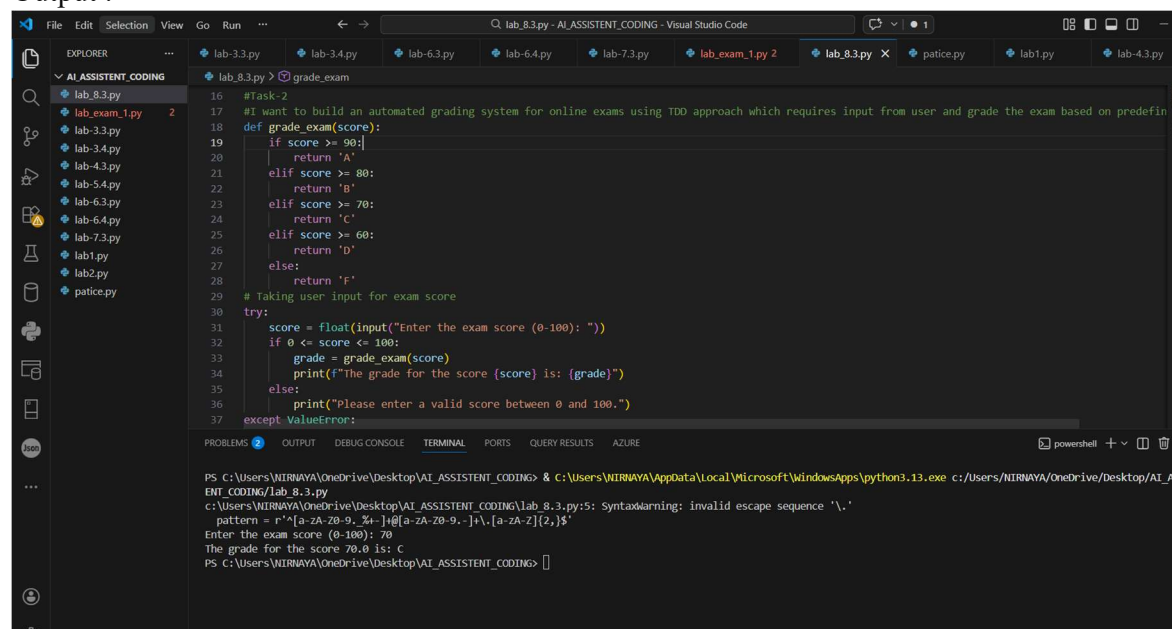
```
else:
```

```
    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 pane on the left shows a project named 'AI\_ASSISTENT\_CODING' with several files, including 'lab\_exam\_1.py' which is selected. The main editor shows the code for 'lab\_exam\_1.py', which is a Python script for an automated grading system. The code includes a function 'grade\_exam(score)' that returns grades 'A' through 'F' based on score ranges, and a main block that takes user input, validates it, and prints the grade. The output pane at the bottom shows the execution of the script, where the user enters '70' and the program outputs 'The grade for the score 70.0 is: C'. There is a syntax warning in the output pane about an invalid escape sequence in a regex pattern.

```
16 #Task-2
17 #I want to build an automated grading system for online exams using TDD approach which requires input from user and grade the exam based on predefined
18 def grade_exam(score):
19     if score >= 90:
20         return 'A'
21     elif score >= 80:
22         return 'B'
23     elif score >= 70:
24         return 'C'
25     elif score >= 60:
26         return 'D'
27     else:
28         return 'F'
29 # Taking user input for exam score
30 try:
31     score = float(input("Enter the exam score (0-100): "))
32     if 0 <= score <= 100:
33         grade = grade_exam(score)
34         print(f"The grade for the score {score} is: {grade}")
35     else:
36         print("Please enter a valid score between 0 and 100.")
37 except ValueError:
```

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 the exam score (0-100): 70  
The grade for the score 70.0 is: C  
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI\_ASSISTENT\_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

- 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:
```

```

        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 a Visual Studio Code editor with a file explorer on the left containing a folder named 'AI\_ASSISTENT\_CODING'. The main editor displays a file named 'lab\_8.3.py' with the following Python code:

```

53 #Task-4
54 #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 i
55 class ShoppingCart:
56     def __init__(self):
57         self.cart_items = []
58     def add_item(self, item_name, price):
59         self.cart_items.append({'name': item_name, 'price': price})
60         print(f"Added {item_name} to cart at price {price}.")
61     def calculate_total(self):
62         total = sum(item['price'] for item in self.cart_items)
63         return total
64 # Creating a shopping cart instance and adding items based on user input
65 cart = ShoppingCart()
66 while True:
67     item_name = input("Enter item name to add to cart (or 'done' to finish): ")
68     if item_name.lower() == 'done':
69         break
70     try:
71         price = float(input(f"Enter price for {item_name}: "))
72         cart.add_item(item_name, price)
73     except ValueError:
74         print("Invalid price. Please enter a numeric value.")

```

The terminal at the bottom shows the execution of the script:

```

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 a Visual Studio Code editor with a Python file named `lab_8.3.py` open. The code defines a function `convert_date_format` that takes a date string, a source format, and a target format as input. It uses `datetime.strptime` to parse the input date and `datetime.strftime` to format it according to the target format. A `try-except` block handles `ValueError` exceptions, returning a message about an invalid date format. Below the function, the script prompts the user for a date to convert, the current date format, and the desired date format. It then calls the `convert_date_format` function and prints the result.

The terminal output shows the execution of the script. It displays the prompts for the date, current format, and target format, followed by the converted date. The output is as follows:

```

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_%.+ ]{0,20}$'
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.