

Lab Assignment-8.3

2303A51546

Batch-10

TASK-1:

Prompt:

write a python program to develop a user registration system that requires reliable email input validation

CODE:

```
import re

class UserRegistration:

    def __init__(self, name, email):

        self.name = name

        self.email = email

    def validate_email(self):

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

        if re.match(pattern, self.email):

            return True

        else:

            return False

    def display(self):

        print(f"Name: {self.name}")

        print(f"Email: {self.email}")

        if self.validate_email():

            print("Email is valid.")

        else:

            print("Email is invalid.")

# Test Cases

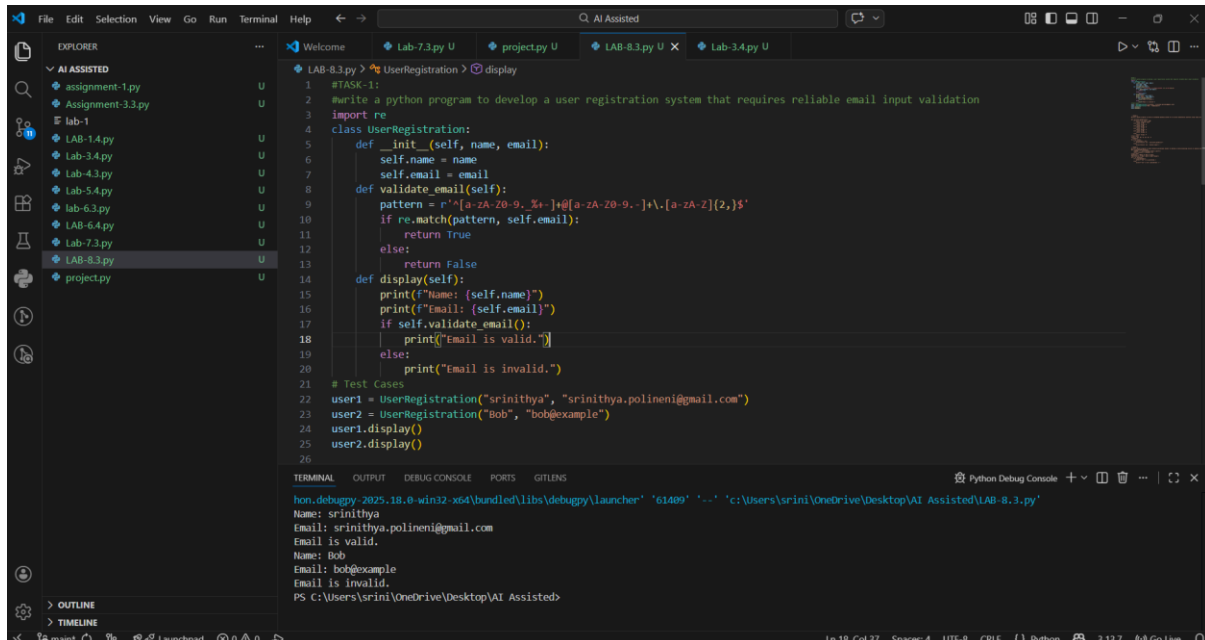
user1 = UserRegistration("srinithya", "srinithya.polineni@gmail.com")
```

```
user2 = UserRegistration("Bob", "bob@example")
```

```
user1.display()
```

```
user2.display()
```

Output:



```
LAB-8.3.py > UserRegistration > display
1 #TASK-1:
2 #write a python program to develop a user registration system that requires reliable email input validation
3 import re
4 class UserRegistration:
5     def __init__(self, name, email):
6         self.name = name
7         self.email = email
8     def validate_email(self):
9         pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
10        if re.match(pattern, self.email):
11            return True
12        else:
13            return False
14    def display(self):
15        print(f"Name: {self.name}")
16        print(f"Email: {self.email}")
17        if self.validate_email():
18            print("Email is valid.")
19        else:
20            print("Email is invalid.")
21
22 # Test Cases
23 user1 = UserRegistration("srinithya", "srinithya.polineni@gmail.com")
24 user2 = UserRegistration("Bob", "bob@example")
25 user1.display()
26 user2.display()
```

```
hon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher '61409' '-' 'c:\Users\sriini\OneDrive\Desktop\AI Assisted\LAB-8.3.py'
Name: srinithya
Email: srinithya.polineni@gmail.com
Email is valid.
Name: Bob
Email: bob@example
Email is invalid.
PS C:\Users\sriini\OneDrive\Desktop\AI Assisted>
```

Analysis:

In this task, a user registration system is implemented to validate email addresses. A class is used to store user details such as name and email. The email validation is done using a regular expression, which checks whether the email follows a proper format containing @ and .. The program correctly identifies valid and invalid email addresses and displays the result. This approach ensures reliable email input validation.

TASK-2:

Prompt:

write a python program to bulid an automated grading system for an online examination platform using loops and aslo include invalid input like -5,eighty

CODE:

```
def calculate_grade(score):
```

```
    if score < 0 or score > 100:
```

```
        return "Invalid input"
```

```
    elif score >= 90:
```

```
    return "Grade: A"
```

```
elif score >= 80:
```

```
    return "Grade: B"
```

```
elif score >= 70:
```

```
    return "Grade: C"
```

```
elif score >= 60:
```

```
    return "Grade: D"
```

```
else:
```

```
    return "Grade: F"
```

Test Cases

```
scores = [95, 85, 75, 65, 55, -5
```

```
, "eighty"]
```

```
for s in scores:
```

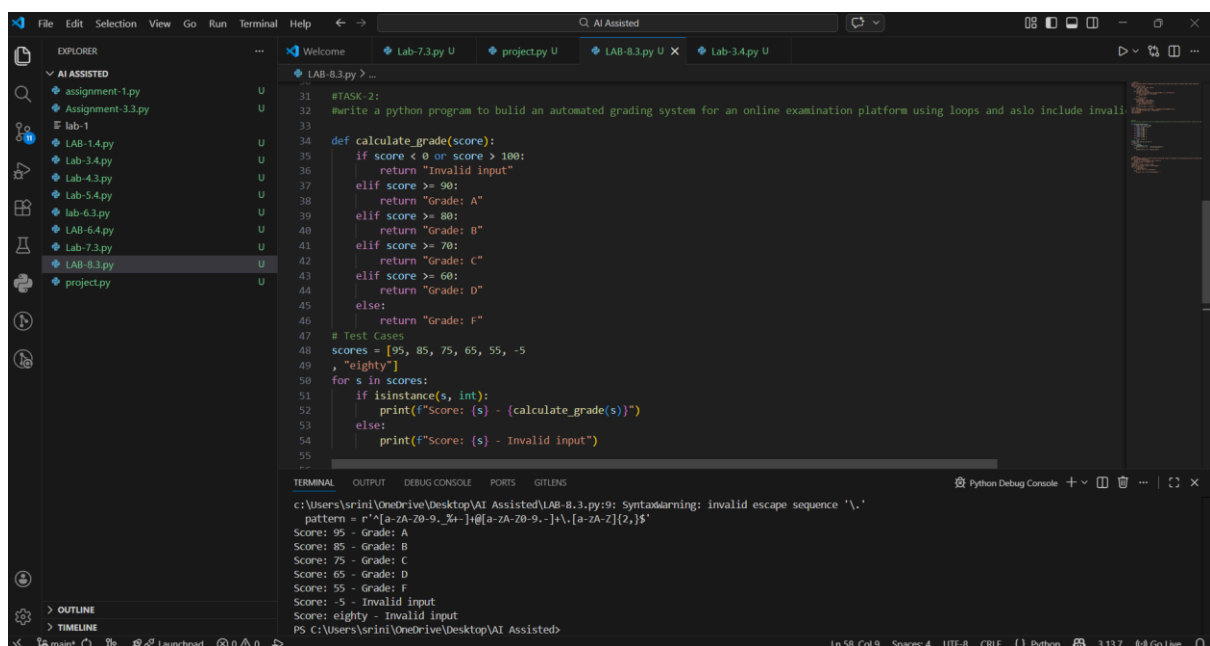
```
    if isinstance(s, int):
```

```
        print(f"Score: {s} - {calculate_grade(s)}")
```

```
    else:
```

```
        print(f"Score: {s} - Invalid input")
```

Output:



The screenshot shows a Visual Studio Code editor window with a Python file named 'LAB-8.3.py'. The code defines a 'calculate_grade' function with a series of if-elif-else statements to assign grades based on scores. It also includes a list of test scores and a loop that prints the grade for each score. The terminal at the bottom shows the output of the script, which correctly assigns grades to the numerical scores and identifies the string 'eighty' as invalid input. A syntax warning is also visible in the terminal output.

```
def calculate_grade(score):
    if score < 0 or score > 100:
        return "Invalid input"
    elif score >= 90:
        return "Grade: A"
    elif score >= 80:
        return "Grade: B"
    elif score >= 70:
        return "Grade: C"
    elif score >= 60:
        return "Grade: D"
    else:
        return "Grade: F"

# Test Cases
scores = [95, 85, 75, 65, 55, -5, "eighty"]
for s in scores:
    if isinstance(s, int):
        print(f"Score: {s} - {calculate_grade(s)}")
    else:
        print(f"Score: {s} - Invalid input")
```

Terminal Output:

```
c:\Users\srin\OneDrive\Desktop\AI Assisted\LAB-8.3.py:9: SyntaxWarning: invalid escape sequence '\.'
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Score: 95 - Grade: A
Score: 85 - Grade: B
Score: 75 - Grade: C
Score: 65 - Grade: D
Score: 55 - Grade: F
Score: -5 - Invalid input
Score: eighty - Invalid input
PS C:\Users\srin\OneDrive\Desktop\AI Assisted
```

Analysis:

This task focuses on building an automated grading system using conditional statements and loops. The program assigns grades based on the score ranges provided. Boundary values are handled correctly, and invalid inputs such as negative numbers and non-numeric values are detected and handled safely. This prevents incorrect grading and improves the robustness of the program.

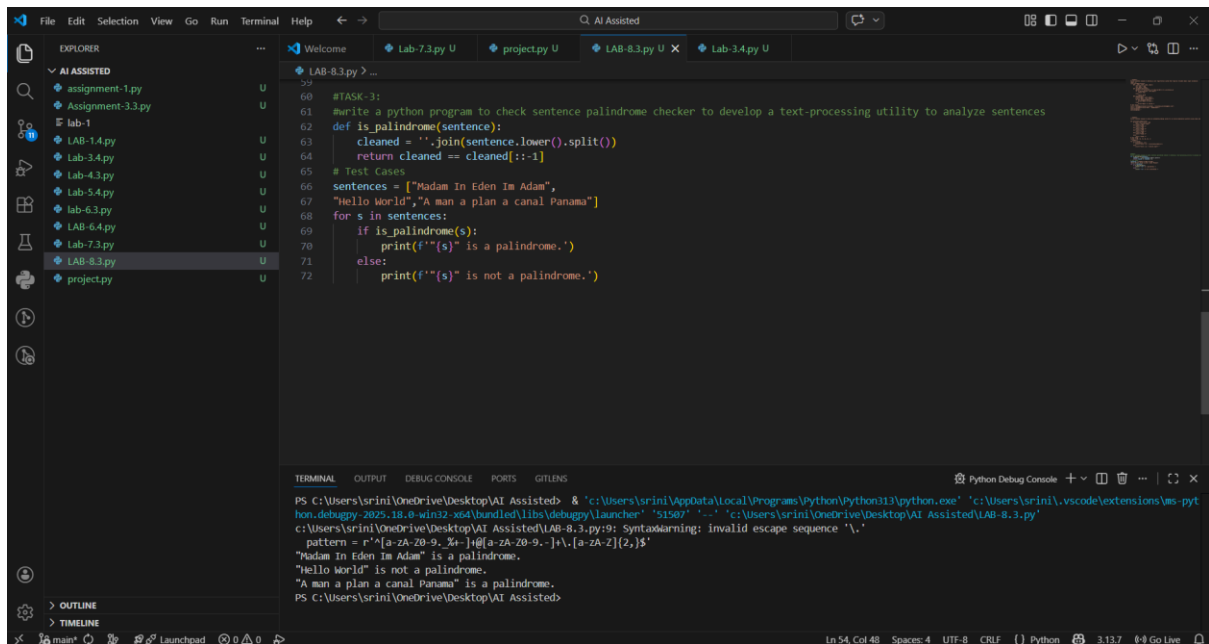
TASK-3:**Prompt:**

write a python program to check sentence palindrome checker to develop a text-processing utility to analyze sentences

CODE:

```
def is_palindrome(sentence):  
    cleaned = ".join(sentence.lower().split())  
    return cleaned == cleaned[::-1]  
  
# Test Cases  
  
sentences = ["Madam In Eden Im Adam",  
             "Hello World","A man a plan a canal Panama"]  
  
for s in sentences:  
    if is_palindrome(s):  
        print(f'"{s}" is a palindrome.')  
    else:  
        print(f'"{s}" is not a palindrome.')
```

Output:



Analysis:

In this task, a function is used to check whether a given sentence is a palindrome. The sentence is converted to lowercase and spaces are removed before comparison. The cleaned string is then compared with its reverse to determine whether it is a palindrome. The program successfully identifies both palindromic and non-palindromic sentences.

TASK-4:

Prompt:

Write a python program to design a basic shopping cart module for an e-commerce application.add item and remove item and total cost

CODE:

```
class ShoppingCart:
```

```
    def __init__(self):
```

```
        self.cart = {}
```

```
    def add_item(self, item, price):
```

```
        self.cart[item] = price
```

```
        print(f"Added {item} to cart at ₹{price}")
```

```
    def remove_item(self, item):
```

```
        if item in self.cart:
```

```

        del self.cart[item]

        print(f"Removed {item} from cart")

    else:

        print(f"{item} not found in cart")

def total_cost(self):

    return sum(self.cart.values())

# Test Cases

cart = ShoppingCart()

cart.add_item("Laptop", 50000)

cart.add_item("Headphones", 2000)

cart.add_item("Mouse", 500)

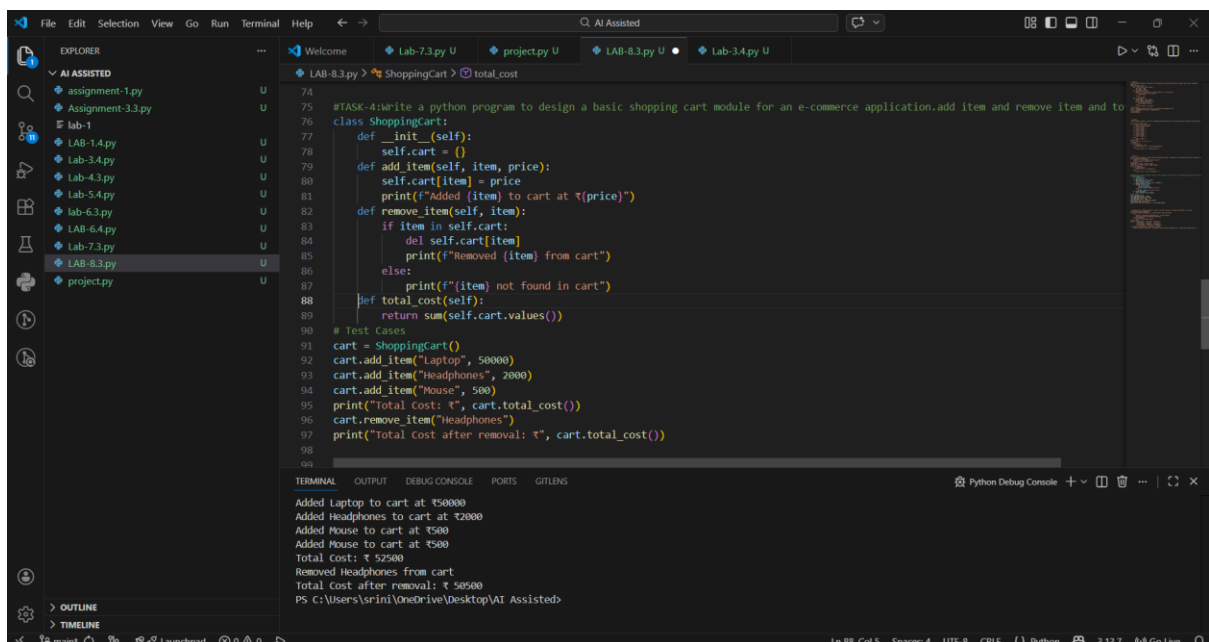
print("Total Cost: ₹", cart.total_cost())

cart.remove_item("Headphones")

print("Total Cost after removal: ₹", cart.total_cost())

```

Output:



The screenshot shows a VS Code editor with a Python file named 'total_cost.py'. The code defines a 'ShoppingCart' class with methods for adding items, removing items, and calculating the total cost. It also includes test cases that create a cart, add items (Laptop, Headphones, Mouse), calculate the total cost, remove an item (Headphones), and calculate the total cost again.

The terminal output shows the execution of the script, displaying the following messages:

```

Added Laptop to cart at ₹50000
Added Headphones to cart at ₹2000
Added Mouse to cart at ₹500
Added Mouse to cart at ₹500
Total Cost: ₹ 52500
Removed Headphones from cart
Total Cost after removal: ₹ 50500
PS C:\Users\sriniv\OneDrive\Desktop\AI Assisted

```

Analysis:

This task implements a basic shopping cart system using a class. A dictionary is used to store items and their prices. The program allows adding items, removing items, and calculating the total cost. It correctly updates the cart after each operation and handles

cases where an item does not exist in the cart. This demonstrates the use of object-oriented programming concepts.

TASK-5:

Prompt:

write a python program to create a utility function to convert date formats for reports

CODE:

```
from datetime import datetime

def convert_date_format(date_str, current_format, desired_format):

    try:

        date_obj = datetime.strptime(date_str, current_format)

        return date_obj.strftime(desired_format)

    except ValueError:

        return "Invalid date format"

# Test Cases

dates = [("2024-06-15", "%Y-%m-%d", "%d/%m/%Y"),

        ("15/06/2024", "%d/%m/%Y", "%Y-%m-%d"),

        ("06-15-2024", "%m-%d-%Y", "%Y/%m/%d"),

        ("invalid-date", "%Y-%m-%d", "%d/%m/%Y")]

for date_str, current_fmt, desired_fmt in dates:

    print(f"Original: {date_str} - Converted: {convert_date_format(date_str, current_fmt, desired_fmt)}")
```

Output:

The screenshot shows a Visual Studio Code editor window with a Python file named `LAB-8.3.py` open. The file contains a utility function `convert_date_format` that takes a date string, a current format, and a desired format as arguments. It uses the `datetime` module to parse the date and then reformat it. The function includes exception handling for `ValueError` to manage invalid date inputs. A list of test cases is provided, including valid dates in different formats and an invalid date. The terminal output shows the results of running the script, displaying the original date, the converted date, and the error message for the invalid date.

```
102
103 #TASK 8: write a python program to create a utility function to convert date formats for reports
104 from datetime import datetime
105 def convert_date_format(date_str, current_format, desired_format):
106     try:
107         date_obj = datetime.strptime(date_str, current_format)
108         return date_obj.strftime(desired_format)
109     except ValueError:
110         return "Invalid date format"
111 # Test Cases
112 dates = [("2024-06-15", "%Y-%m-%d", "%d/%m/%Y"),
113          ("15/06/2024", "%d/%m/%Y", "%Y-%m-%d"),
114          ("06-15-2024", "%m-%d-%Y", "%Y/%m-%d"),
115          ("invalid-date", "%Y-%m-%d", "%d/%m/%Y")]
116 for date_str, current_fmt, desired_fmt in dates:
117     print(f"Original: {date_str} - Converted: {convert_date_format(date_str, current_fmt, desired_fmt)}")
```

TERMINAL OUTPUT:

```
Original: 2024-06-15 - Converted: 15/06/2024
Original: 15/06/2024 - Converted: 2024-06-15
Original: 06-15-2024 - Converted: 2024/06/15
Original: invalid-date - Converted: Invalid date format
PS C:\Users\srini\OneDrive\Desktop\AI Assisted
```

Analysis:

In this task, a utility function is created to convert date formats using the `datetime` module. The program converts a date from one format to another and uses exception handling to manage invalid date inputs. This ensures that the program does not crash and provides meaningful output for incorrect formats.