ASSIGNMENT – 8.3

2303A51060

Batch-10

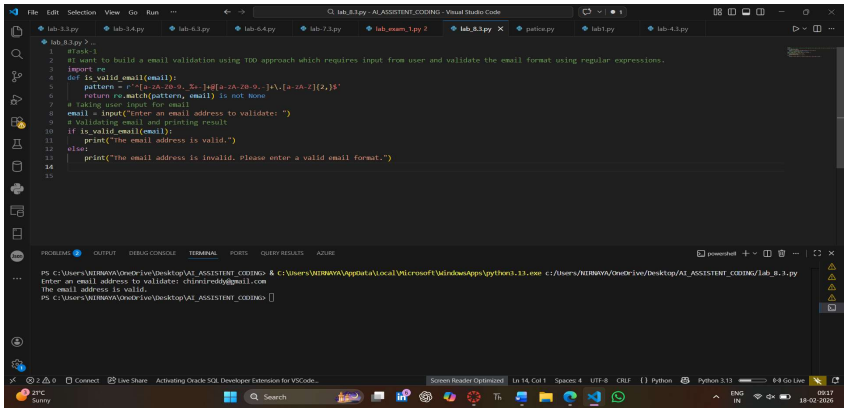
Task-1

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:



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

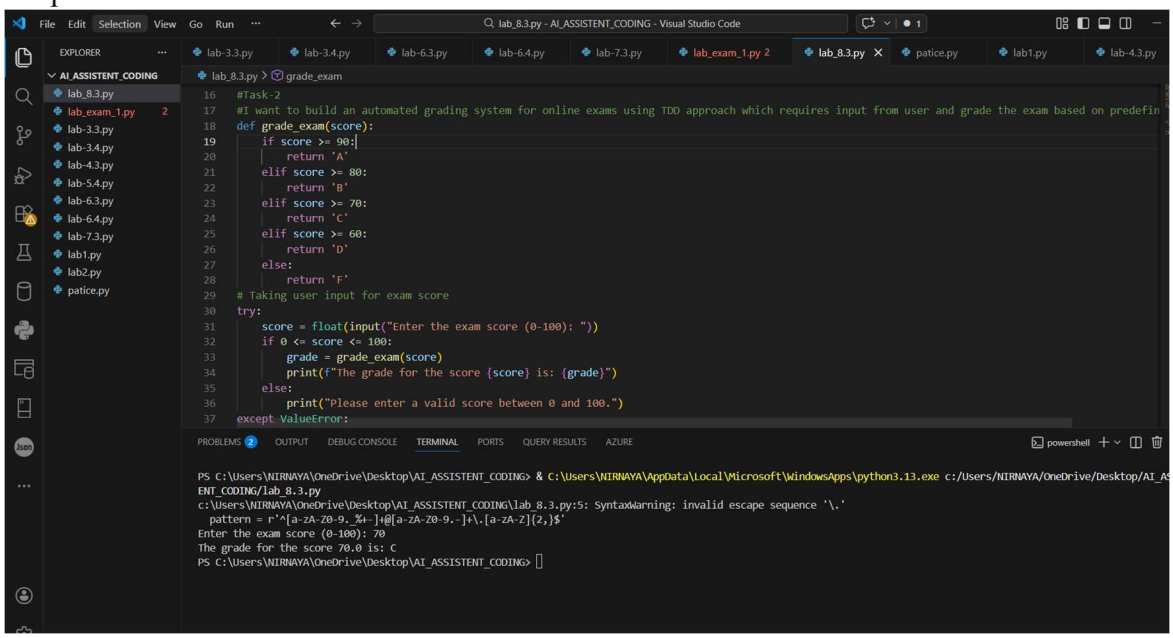
Prompt: I want to build an automated grading system for online exams using a TDD approach, which requires input from the user and grades the exam based on predefined criteria.

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 :



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.

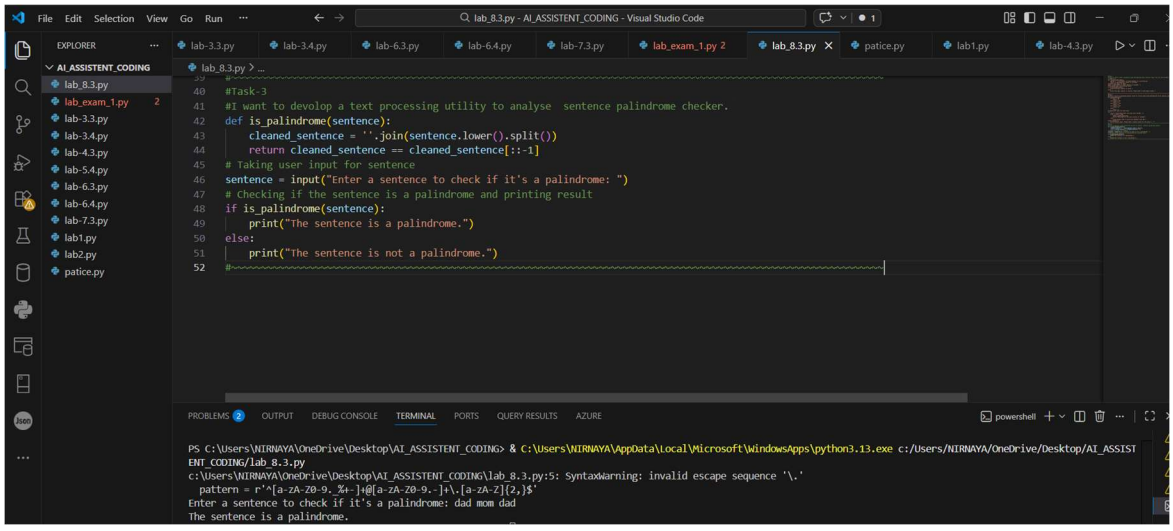
Task-3

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 :



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:
```

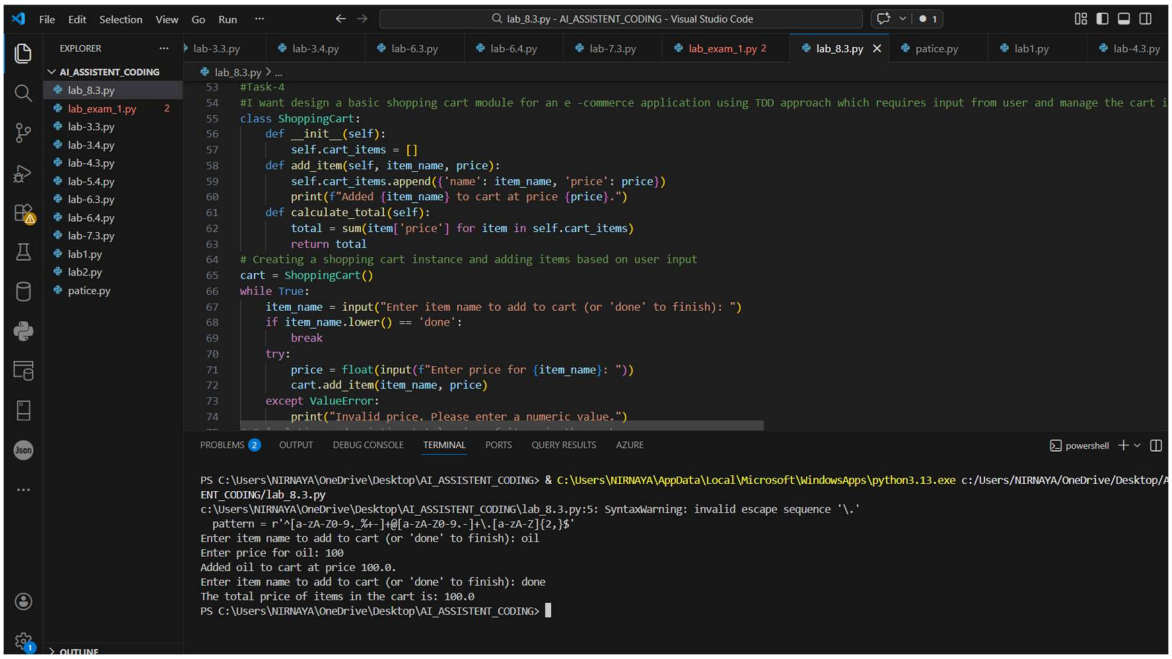
print("Invalid price. Please enter a numeric value.")

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 :



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

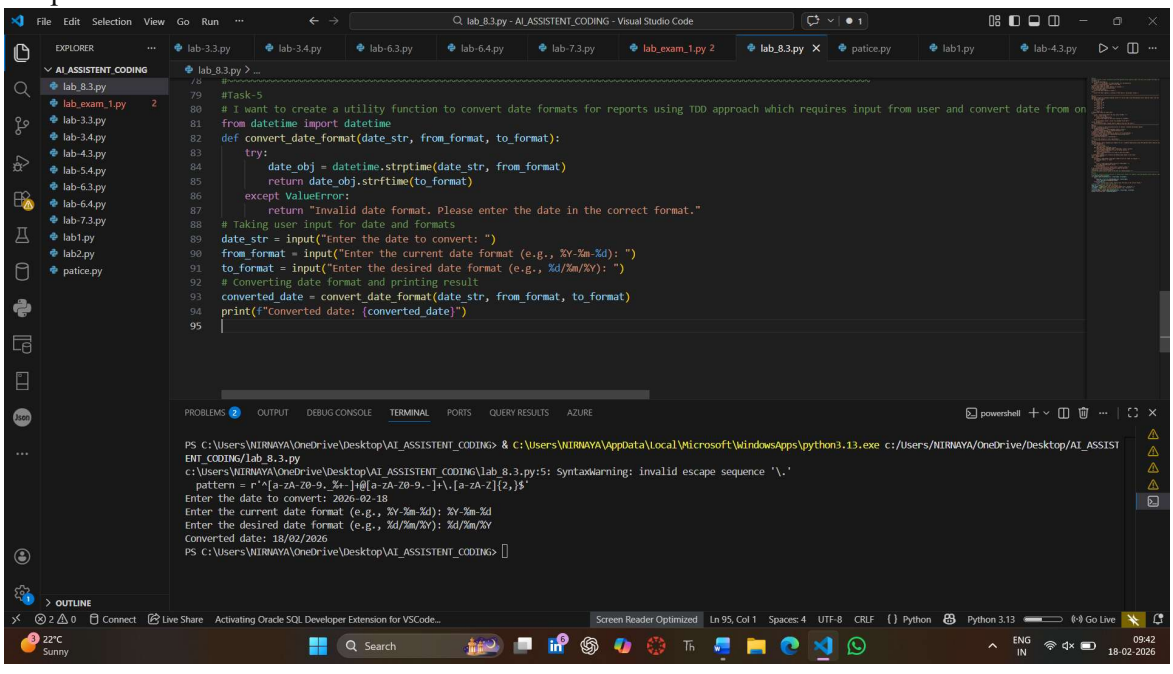
Prompt: I want to create a utility function to convert date formats for reports using a TDD approach, which requires input from the user and converts dates from one format to another.

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 :



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.