AI-Generated Test Cases and Implementations

NAME: J.Anjali

H.T NO: 2403A510F0

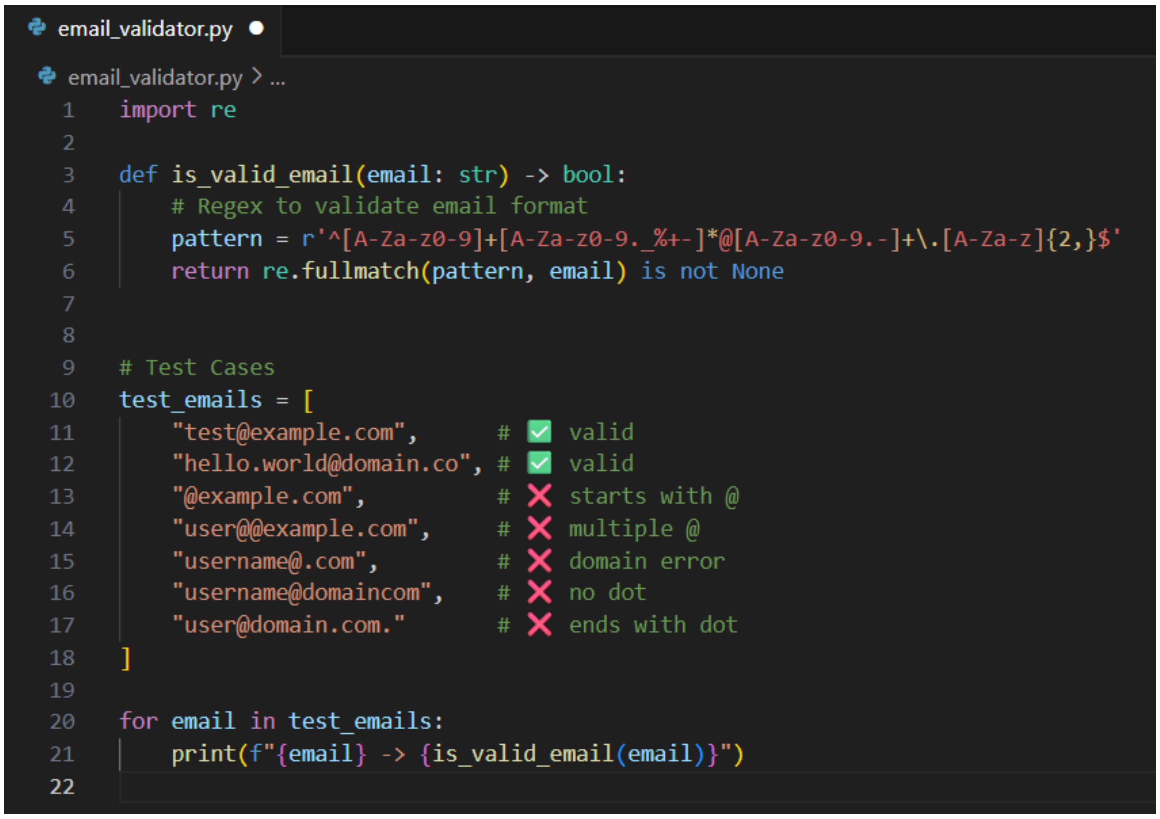
BATCH NO: 05

Task #1: Email Validator

Prompt

Write a Python program to validate email addresses using regular expressions.
The program should define a function is_valid_email() that returns True if the email is valid, and False otherwise. It should then test multiple email samples and display whether each one is valid or not.

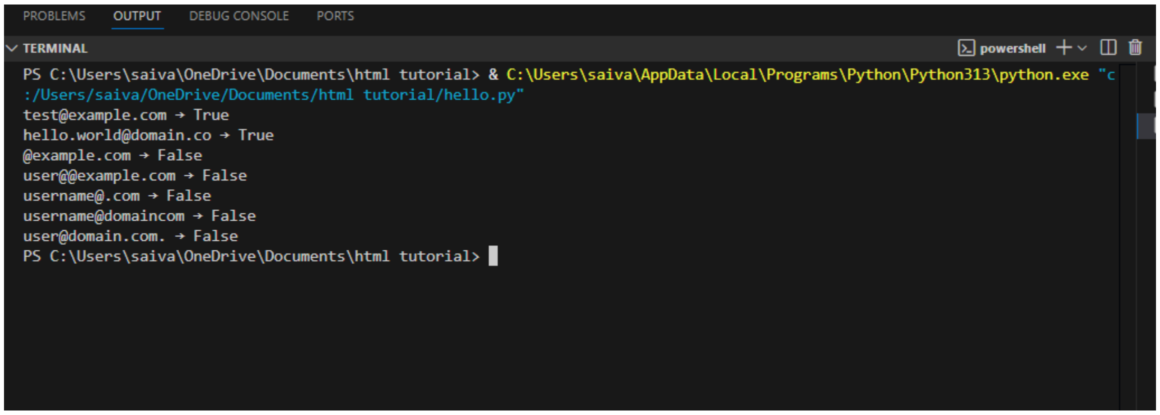
Python Code:



```python
import re

def is_valid_email(email: str) -> bool:
    # Regex to validate email format
    pattern = r'^[A-Za-z0-9]+[A-Za-z0-9._%+-]*@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
    return re.fullmatch(pattern, email) is not None


# Test Cases
test_emails = [
    "test@example.com",       # ✅ valid
    "hello.world@domain.co",  # ✅ valid
    "@example.com",           # ❌ starts with @
    "user@@example.com",      # ❌ multiple @
    "username@.com",          # ❌ domain error
    "username@domaincom",     # ❌ no dot
    "user@domain.com."        # ❌ ends with dot
]

for email in test_emails:
    print(f"{email} -> {is_valid_email(email)}")

```

AI-Generated Test Cases and Implementations

Output



```
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Python313\python.exe "c
:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
test@example.com → True
hello.world@domain.co → True
@example.com → False
user@@example.com → False
username@.com → False
username@domaincom → False
user@domain.com. → False
PS C:\Users\saiva\OneDrive\Documents\html tutorial>
```

AI-Generated Test Cases and Implementations

Observation

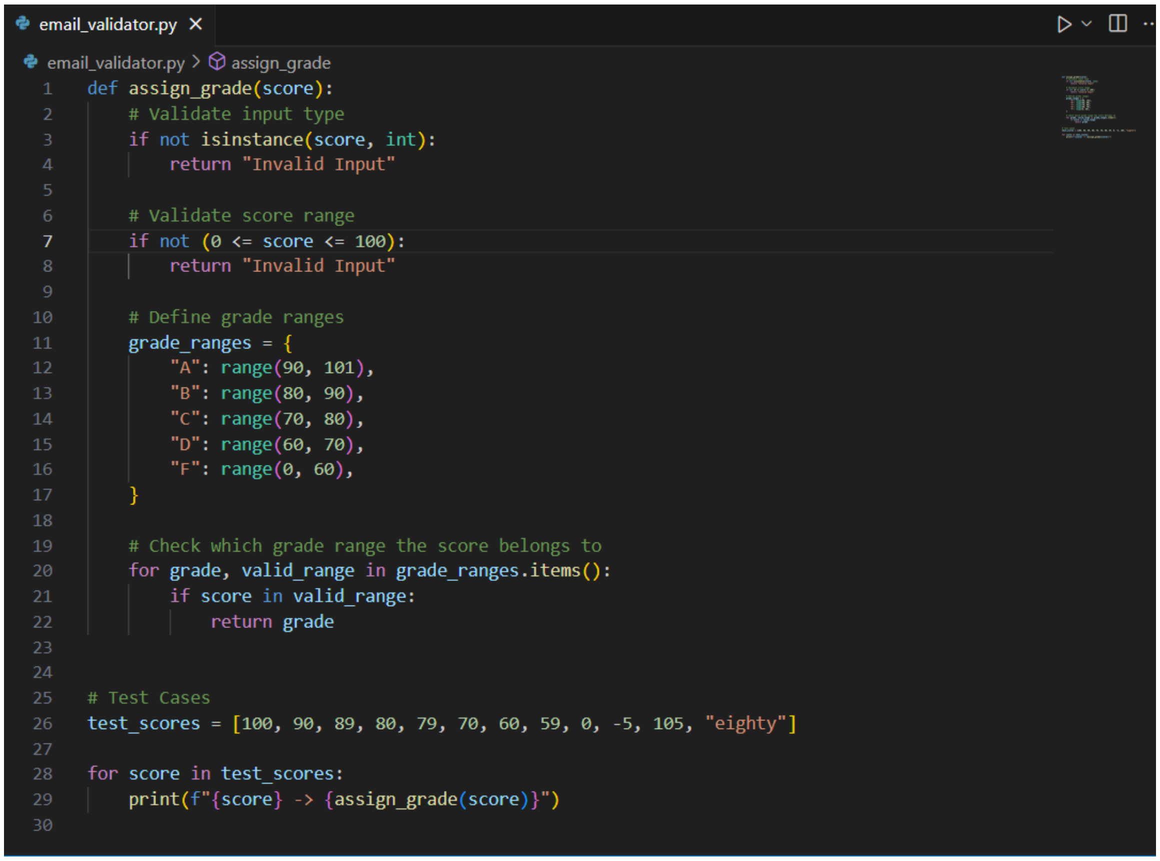
when the program is executed, it checks each test email against the regex pattern. Valid emails return True and invalid emails return False.

Task #2: Grade Assignment

Prompt

The program is designed to assign grades based on student scores. It uses a function assign_grade() that returns "A" for scores 90–100, "B" for 80–89, "C" for 70–79, "D" for 60–69, and "F" for below 60. It also checks for invalid inputs such as negative numbers, values above 100, or non-numeric entries.

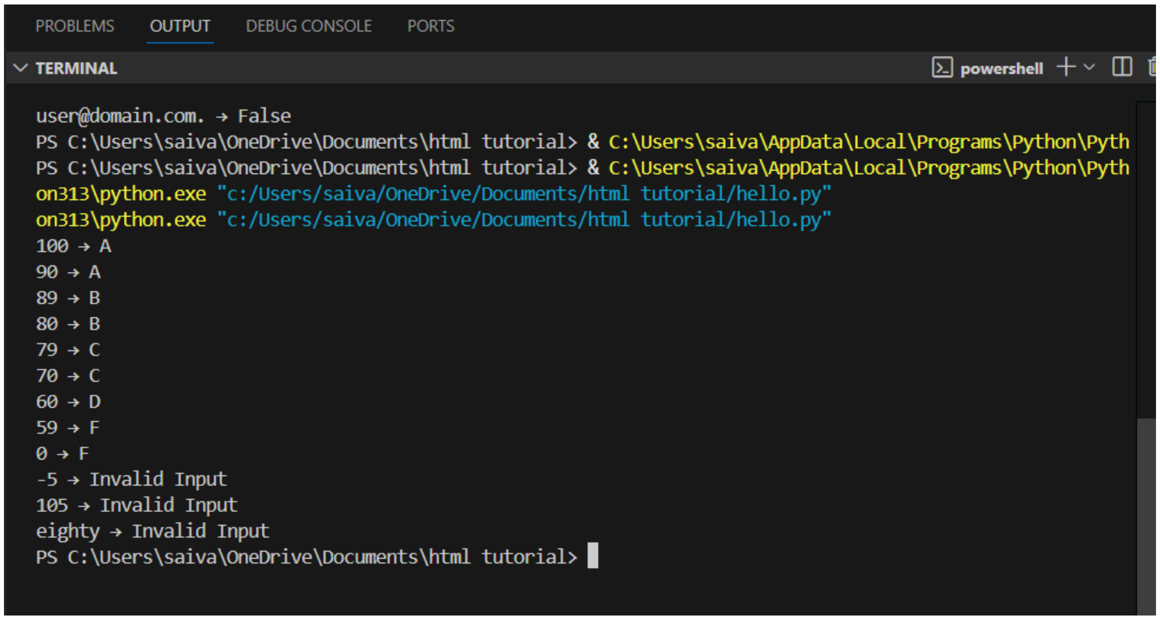
Python Code



```python
def assign_grade(score):
    # Validate input type
    if not isinstance(score, int):
        return "Invalid Input"

    # Validate score range
    if not (0 <= score <= 100):
        return "Invalid Input"

    # Define grade ranges
    grade_ranges = {
        "A": range(90, 101),
        "B": range(80, 90),
        "C": range(70, 80),
        "D": range(60, 70),
        "F": range(0, 60),
    }

    # Check which grade range the score belongs to
    for grade, valid_range in grade_ranges.items():
        if score in valid_range:
            return grade


# Test Cases
test_scores = [100, 90, 89, 80, 79, 70, 60, 59, 0, -5, 105, "eighty"]

for score in test_scores:
    print(f"{score} -> {assign_grade(score)}")
```

AI-Generated Test Cases and Implementations

Output



```
user@domain.com. → False
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Pyth
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Pyth
on313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
on313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
100 → A
90 → A
89 → B
80 → B
79 → C
70 → C
60 → D
59 → F
0 → F
-5 → Invalid Input
105 → Invalid Input
eighty → Invalid Input
PS C:\Users\saiva\OneDrive\Documents\html tutorial> 
```

Observation

When executed, the program correctly classified valid scores into their respective grades and returned "Invalid Input" for values like -5, 105, and "eighty". This shows that the program works correctly and handles errors effectively.

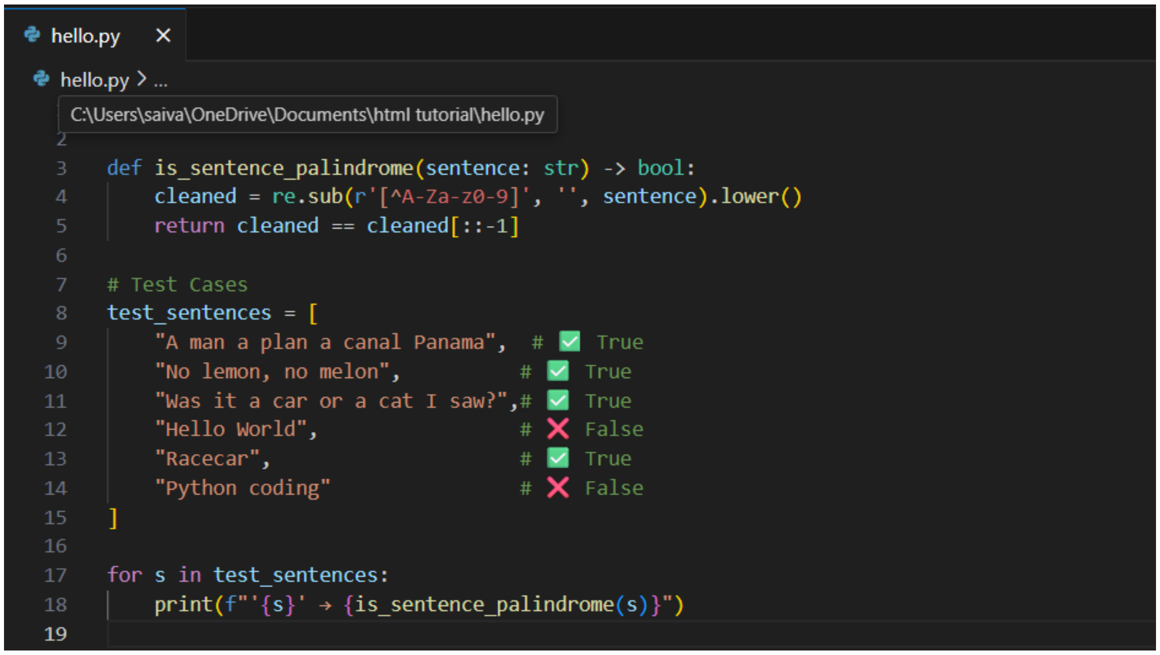
Task #3: Sentence Palindrome

Prompt

Give test cases for is_sentence_palindrome(sentence) that ignores spaces, punctuation, and case by using ai.

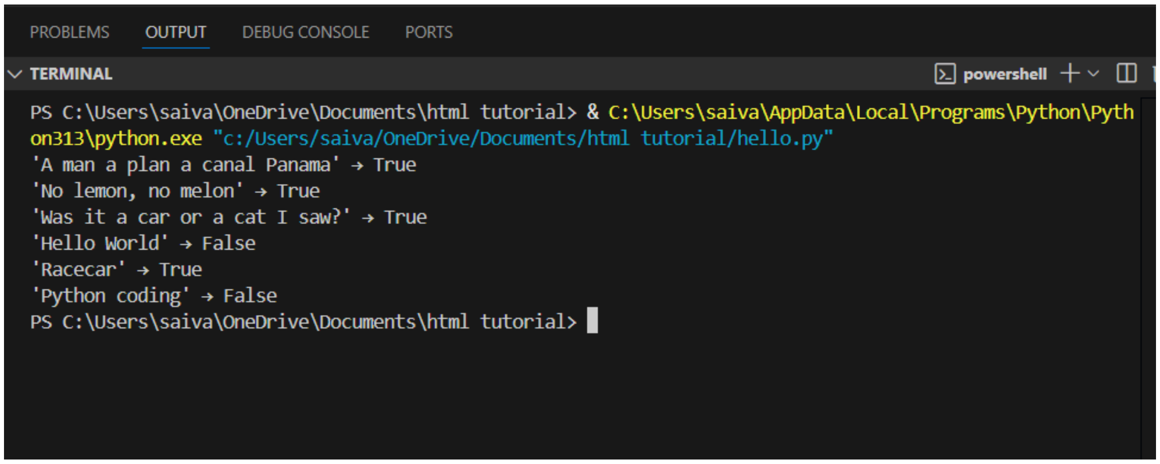
AI-Generated Test Cases and Implementations

Python Code



```python
def is_sentence_palindrome(sentence: str) -> bool:
    cleaned = re.sub(r'[^A-Za-z0-9]', '', sentence).lower()
    return cleaned == cleaned[::-1]

# Test Cases
test_sentences = [
    "A man a plan a canal Panama",  # ✅ True
    "No lemon, no melon",          # ✅ True
    "Was it a car or a cat I saw?",# ✅ True
    "Hello World",                 # ❌ False
    "Racecar",                     # ✅ True
    "Python coding"                # ❌ False
]

for s in test_sentences:
    print(f"'{s}' → {is_sentence_palindrome(s)}")
```

Output



```
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Pyth
on313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
'A man a plan a canal Panama' → True
'No lemon, no melon' → True
'Was it a car or a cat I saw?' → True
'Hello World' → False
'Racecar' → True
'Python coding' → False
PS C:\Users\saiva\OneDrive\Documents\html tutorial>
```

Observation

The code correctly checks palindromes by ignoring case, spaces, and punctuation, returning **True** for valid palindromes and **False** otherwise.

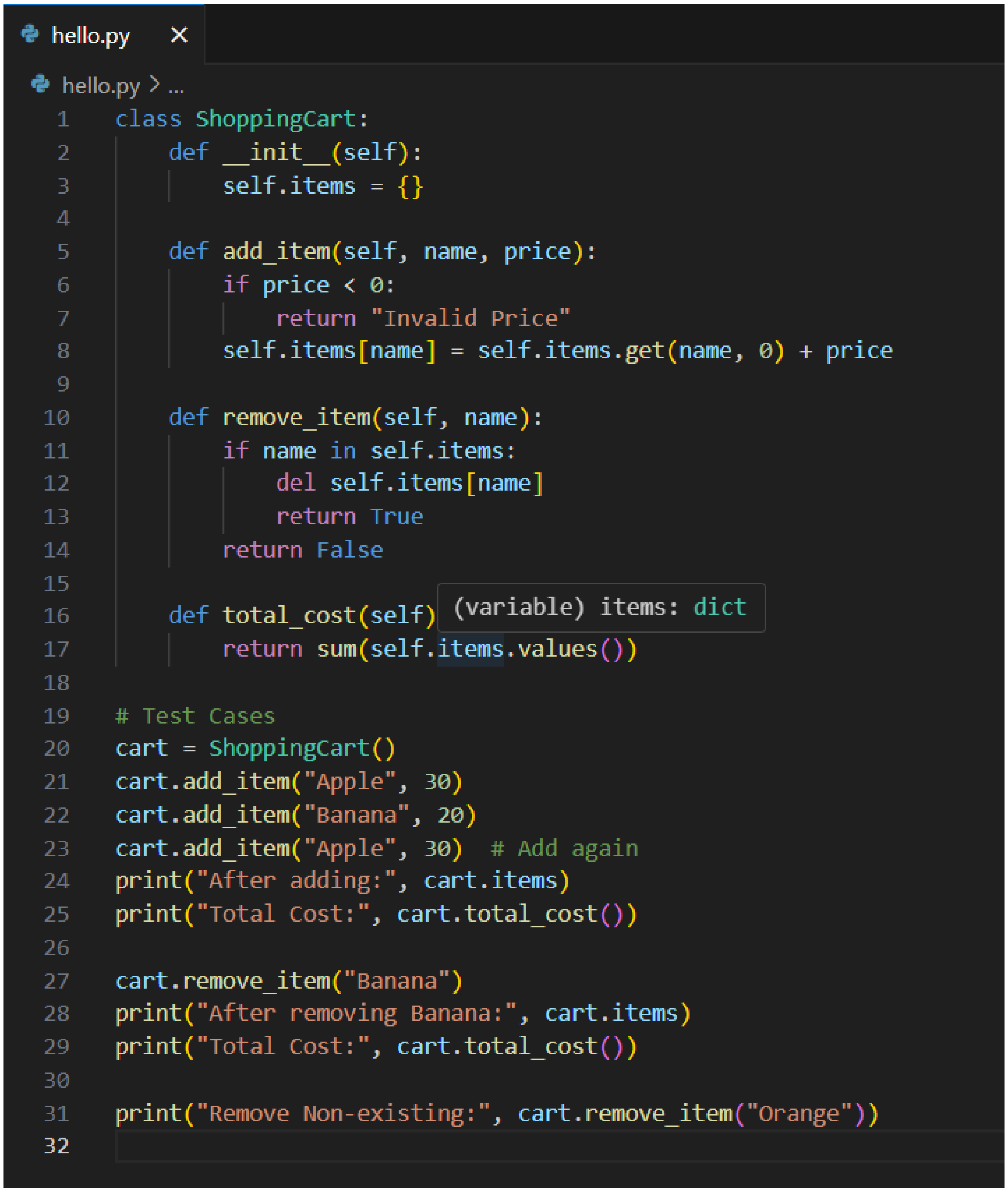
AI-Generated Test Cases and Implementations

Task #4: Shopping Cart

Prompt

Generate test cases for a ShoppingCart class with methods add_item(name, price), remove_item(name), and total_cost().

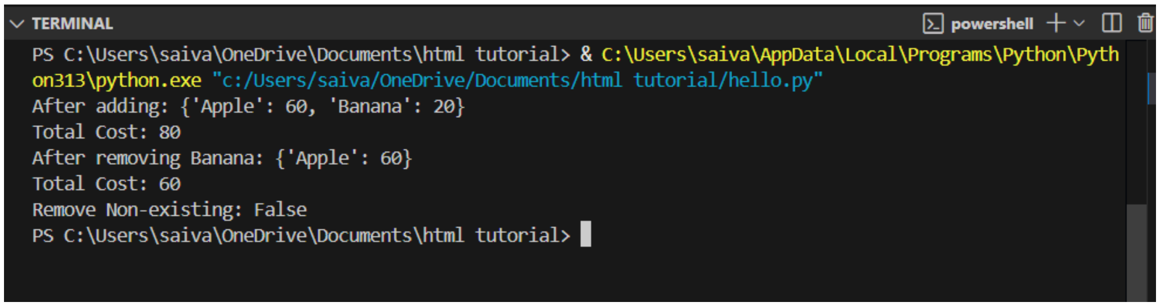
Python Code


hello.py

hello.py > ...

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

    def add_item(self, name, price):
        if price < 0:
            return "Invalid Price"
        self.items[name] = self.items.get(name, 0) + price

    def remove_item(self, name):
        if name in self.items:
            del self.items[name]
            return True
        return False

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

# Test Cases
cart = ShoppingCart()
cart.add_item("Apple", 30)
cart.add_item("Banana", 20)
cart.add_item("Apple", 30)  # Add again
print("After adding:", cart.items)
print("Total Cost:", cart.total_cost())

cart.remove_item("Banana")
print("After removing Banana:", cart.items)
print("Total Cost:", cart.total_cost())

print("Remove Non-existing:", cart.remove_item("Orange"))
```

Output



```
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Pyth
on313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
After adding: {'Apple': 60, 'Banana': 20}
Total Cost: 80
After removing Banana: {'Apple': 60}
Total Cost: 60
Remove Non-existing: False
PS C:\Users\saiva\OneDrive\Documents\html tutorial>
```

Observation:

The cart works fine: adding updates totals, duplicates add up, removing items lowers cost, and removing something not in the cart just returns **False**.

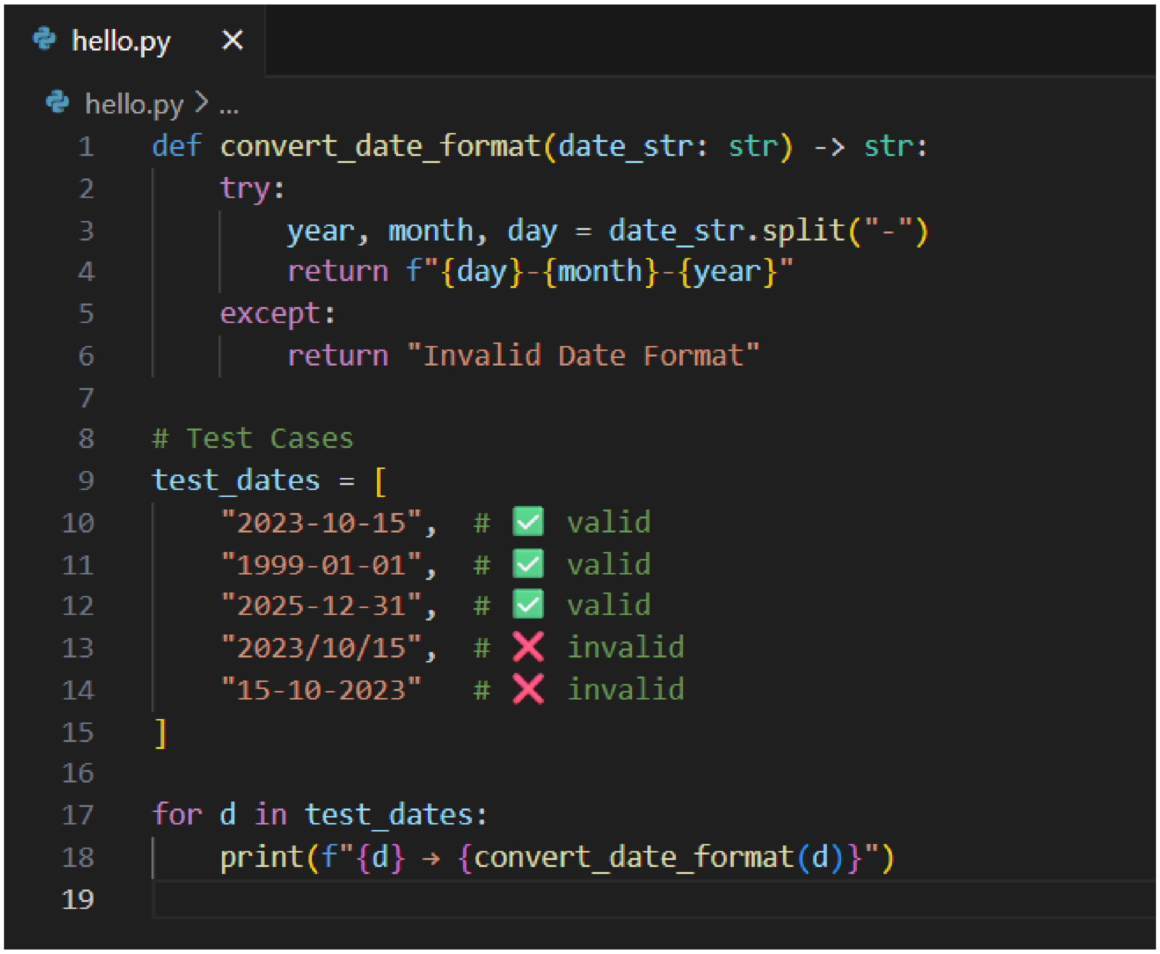
Task #5: Date Format Converter

Prompt:

write test cases for convert_date_format(date_str) to switch from "YYYY-MM-DD" to "DD-MM-YYYY".

AI-Generated Test Cases and Implementations

Python Code

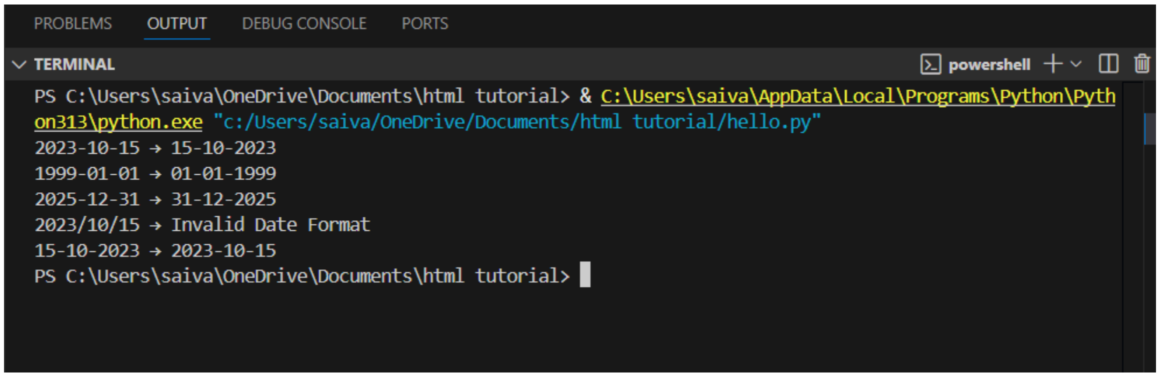


hello.py

hello.py > ...

```python
def convert_date_format(date_str: str) -> str:
    try:
        year, month, day = date_str.split("-")
        return f"{day}-{month}-{year}"
    except:
        return "Invalid Date Format"

# Test Cases
test_dates = [
    "2023-10-15",  # ✅ valid
    "1999-01-01",  # ✅ valid
    "2025-12-31",  # ✅ valid
    "2023/10/15",  # ❌ invalid
    "15-10-2023"   # ❌ invalid
]

for d in test_dates:
    print(f"{d} → {convert_date_format(d)}")
```

Output



```
PS C:\Users\saiva\OneDrive\Documents\html tutorial> & C:\Users\saiva\AppData\Local\Programs\Python\Pyth
on313\python.exe "c:/Users/saiva/OneDrive/Documents/html tutorial/hello.py"
2023-10-15 → 15-10-2023
1999-01-01 → 01-01-1999
2025-12-31 → 31-12-2025
2023/10/15 → Invalid Date Format
15-10-2023 → 2023-10-15
PS C:\Users\saiva\OneDrive\Documents\html tutorial>
```

AI-Generated Test Cases and Implementations

Observation

The code converts dates from YYYY-MM-DD to DD-MM-YYYY. Valid formats work fine, invalid ones like 2023/10/15 are caught, but 15-10-2023 is wrongly treated as valid since it still splits into 3 parts.