# AI-ASSISTED CIDING – 8.3

**NAME: T. SAI VARUN**

**H.T NO: 2403A510C0**

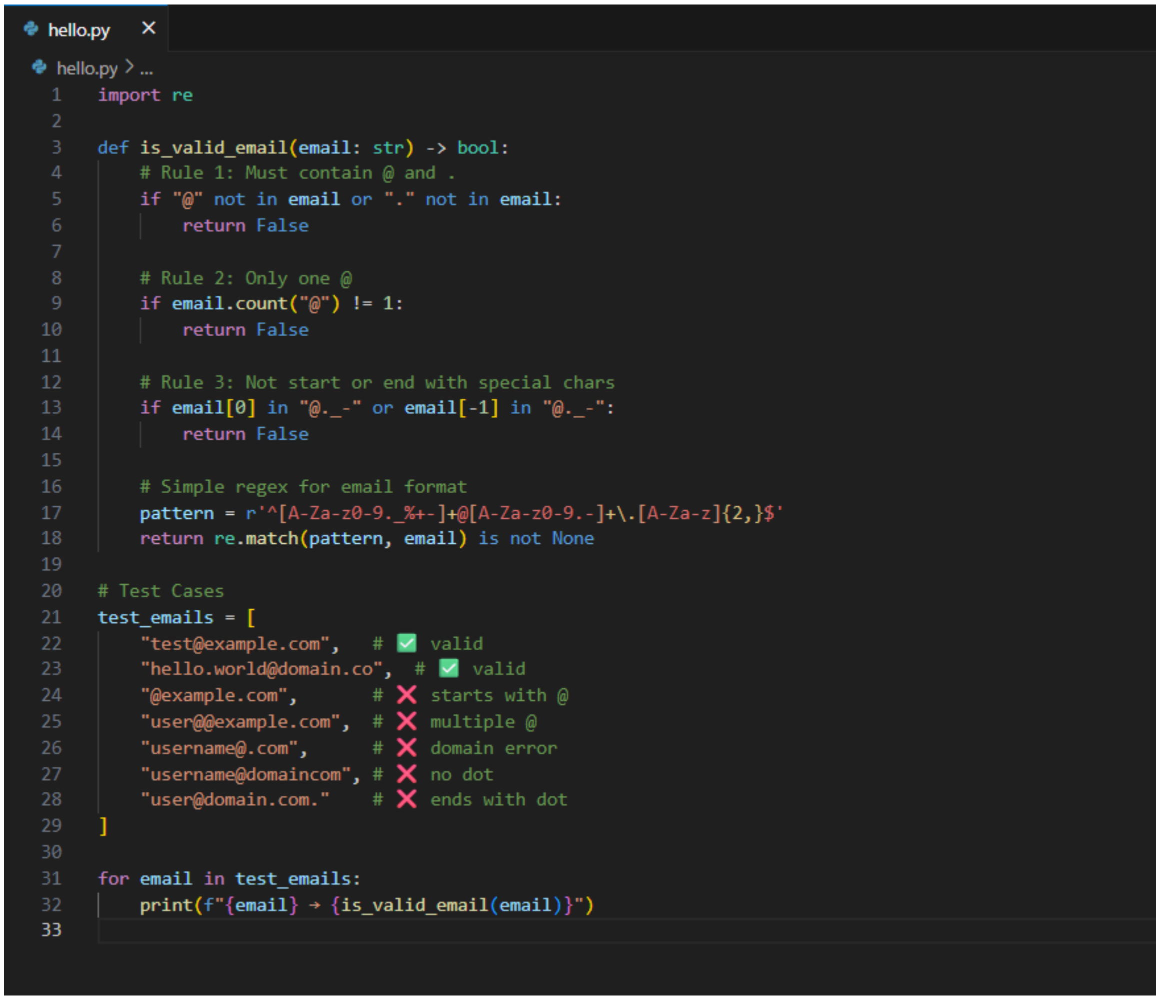
**BATCH NO: 05**

## Task #1: Email Validator

### Prompt

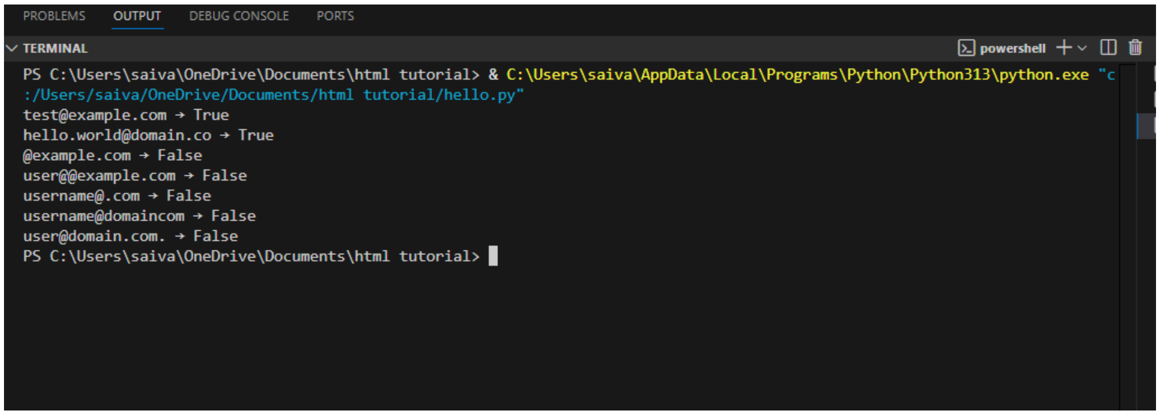
Generate Python test cases for a function is_valid_email(email) that validates email addresses with these rules: must contain @ and ., must not start or end with special characters, and should not allow multiple @. Then implement the function to pass all tests.

### Python Code



```python
import re

def is_valid_email(email: str) -> bool:
    # Rule 1: Must contain @ and .
    if "@" not in email or "." not in email:
        return False

    # Rule 2: Only one @
    if email.count("@") != 1:
        return False

    # Rule 3: Not start or end with special chars
    if email[0] in "@._-" or email[-1] in "@._-":
        return False

    # Simple regex for email format
    pattern = r'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
    return re.match(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)}")
```

## 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> 
```

## Observation

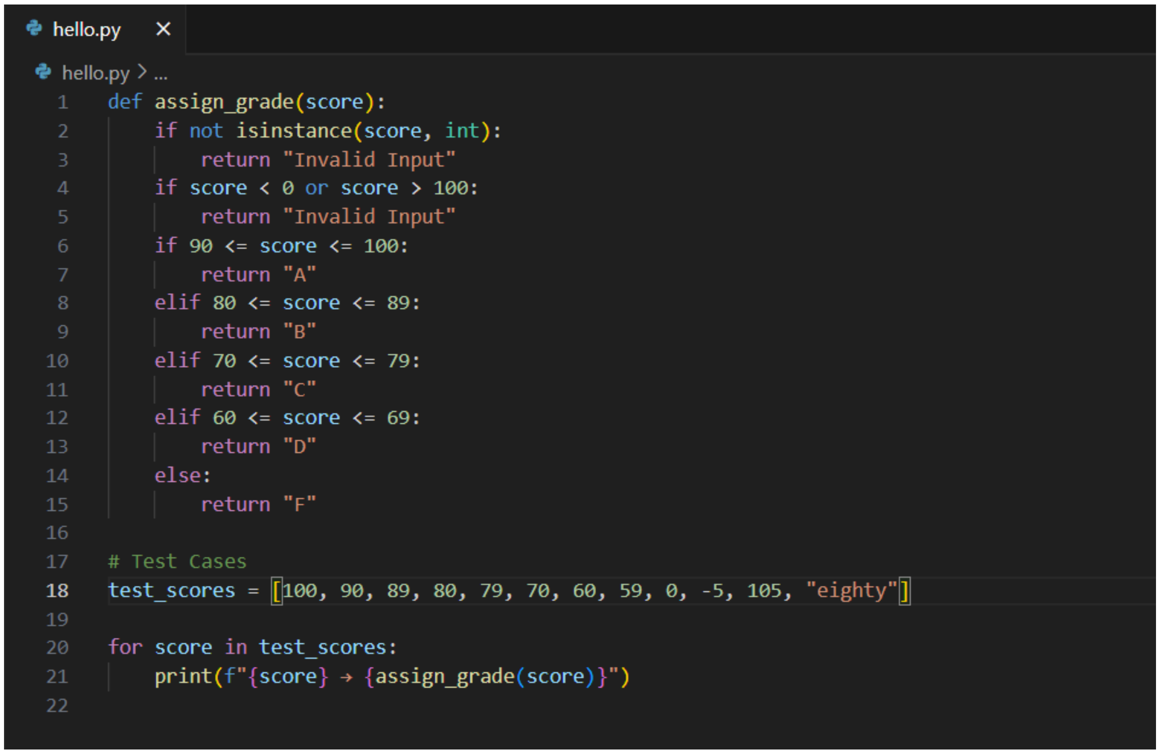
The function correctly validates emails according to the rules. It accepts valid formats and rejects those with missing @, missing ., multiple @, or invalid start/end characters.

## Task #2: Grade Assignment

### Prompt

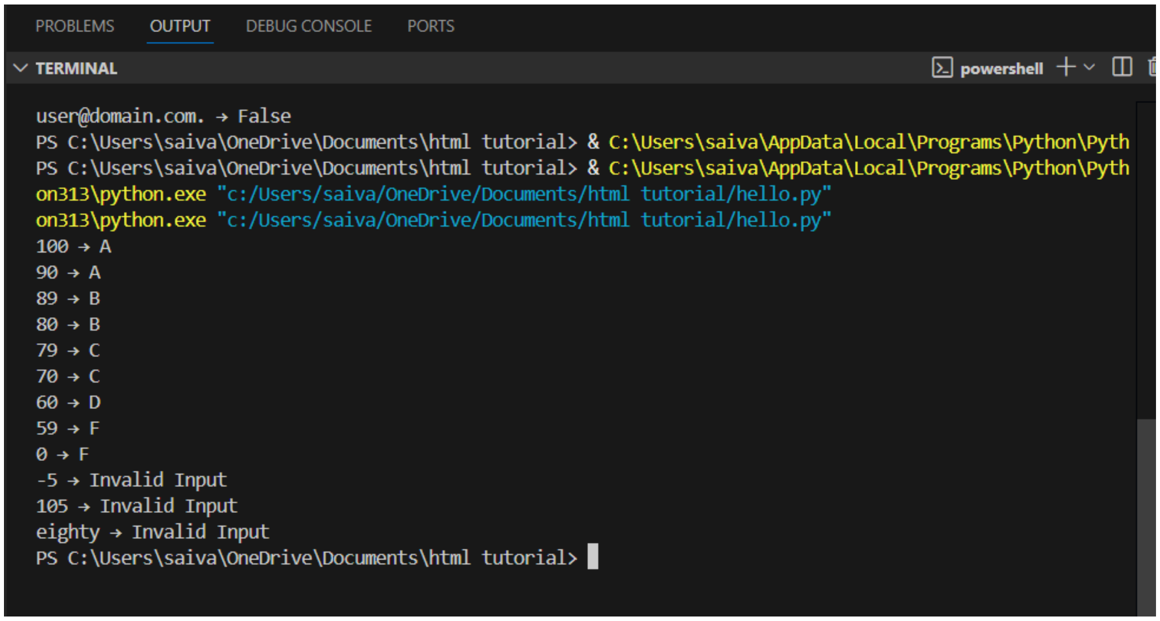
Generate boundary and invalid input test cases for assign_grade(score) where: 90-100 = A, 80-89 = B, 70-79 = C, 60-69 = D, <60 = F. Handle invalid values like negative, >100, and strings.

## Python Code



```python
def assign_grade(score):
    if not isinstance(score, int):
        return "Invalid Input"
    if score < 0 or score > 100:
        return "Invalid Input"
    if 90 <= score <= 100:
        return "A"
    elif 80 <= score <= 89:
        return "B"
    elif 70 <= score <= 79:
        return "C"
    elif 60 <= score <= 69:
        return "D"
    else:
        return "F"

# 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)}")
```

## 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

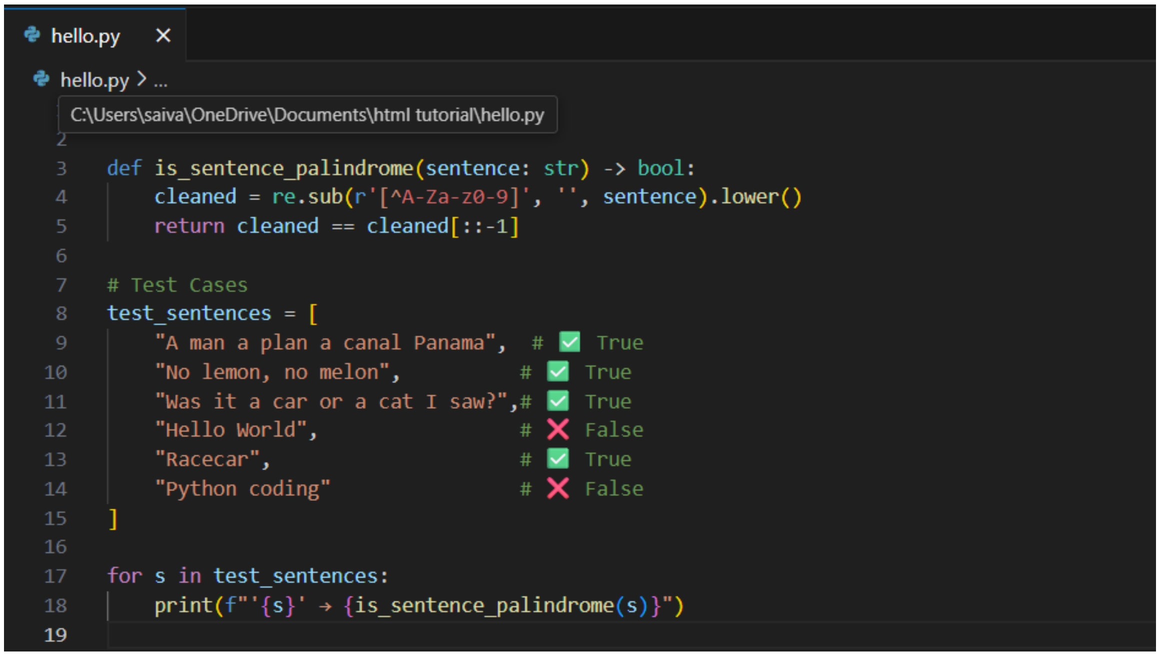
The function handles all grade boundaries properly and rejects invalid inputs like negative scores, values above 100, and non-numeric strings.

## Task #3: Sentence Palindrome

### Prompt

Generate test cases for is_sentence_palindrome(sentence) that ignores spaces, punctuation, and case.

### Python Code



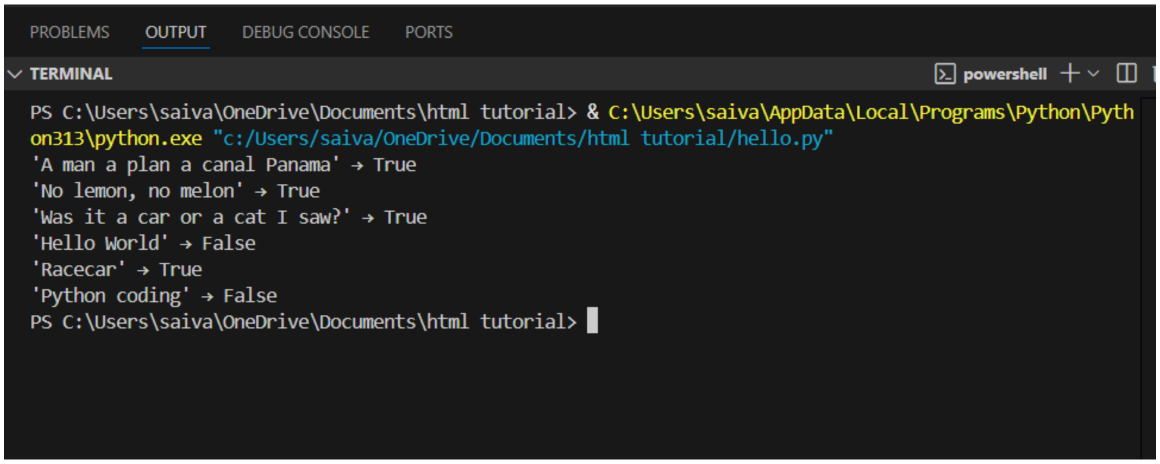
hello.py

hello.py > ...

C:\Users\saiva\OneDrive\Documents\html tutorial\hello.py

```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

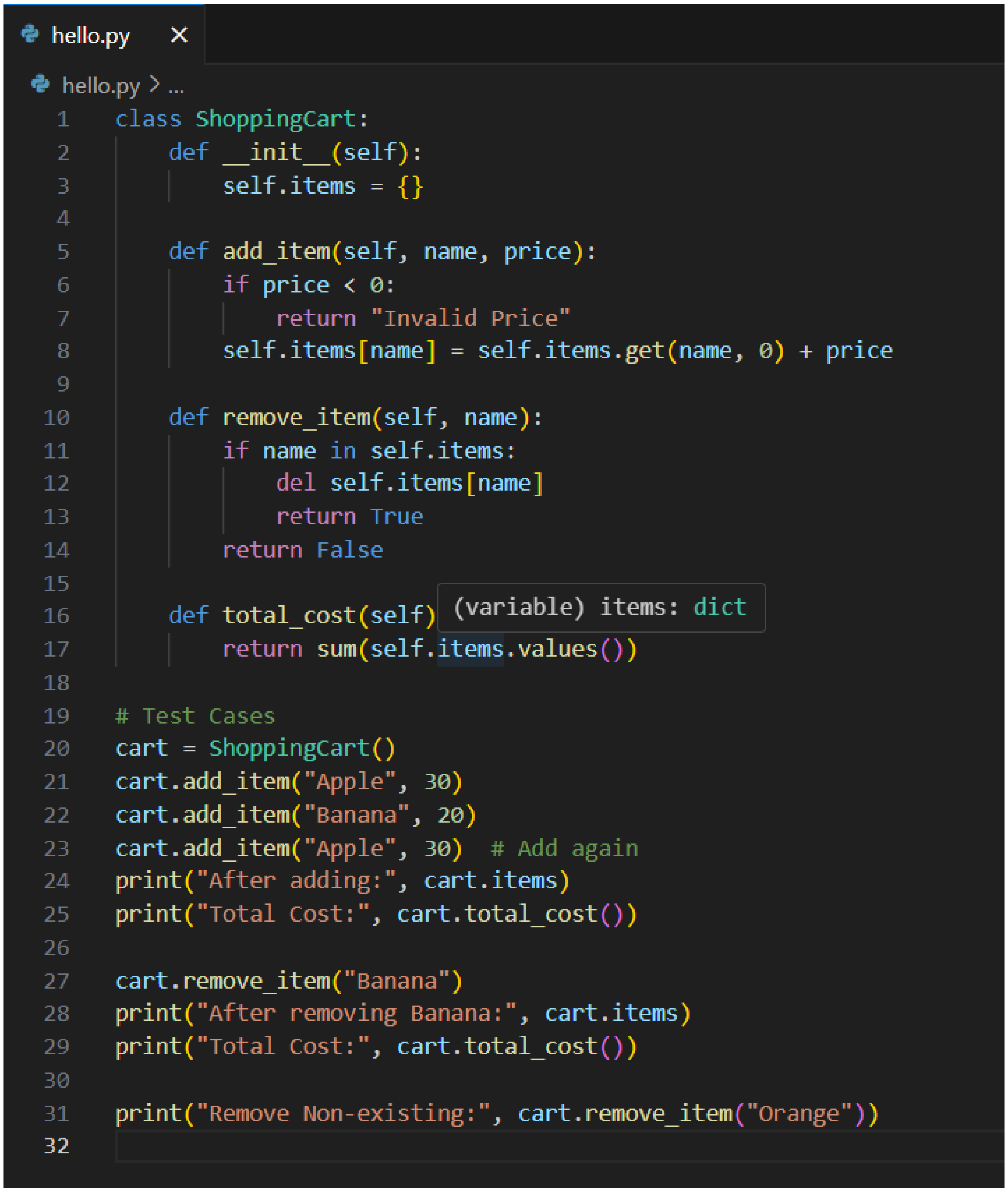
The function successfully ignores spaces, punctuation, and case, correctly identifying palindrome sentences.

## 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

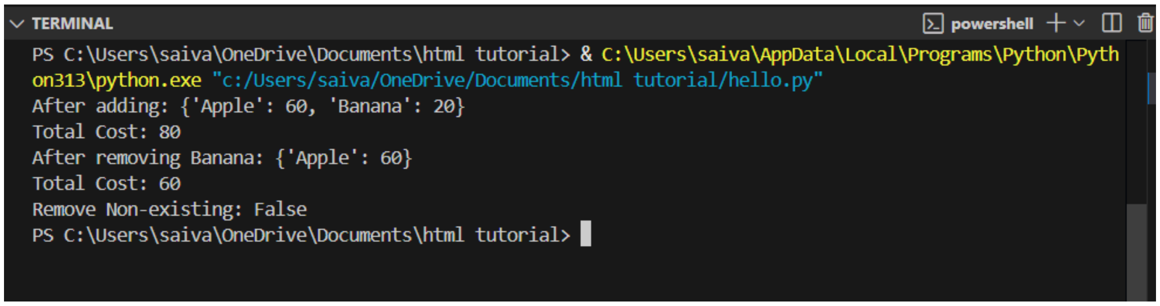

hello.py

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"))
```

(variable) items: dict

## 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

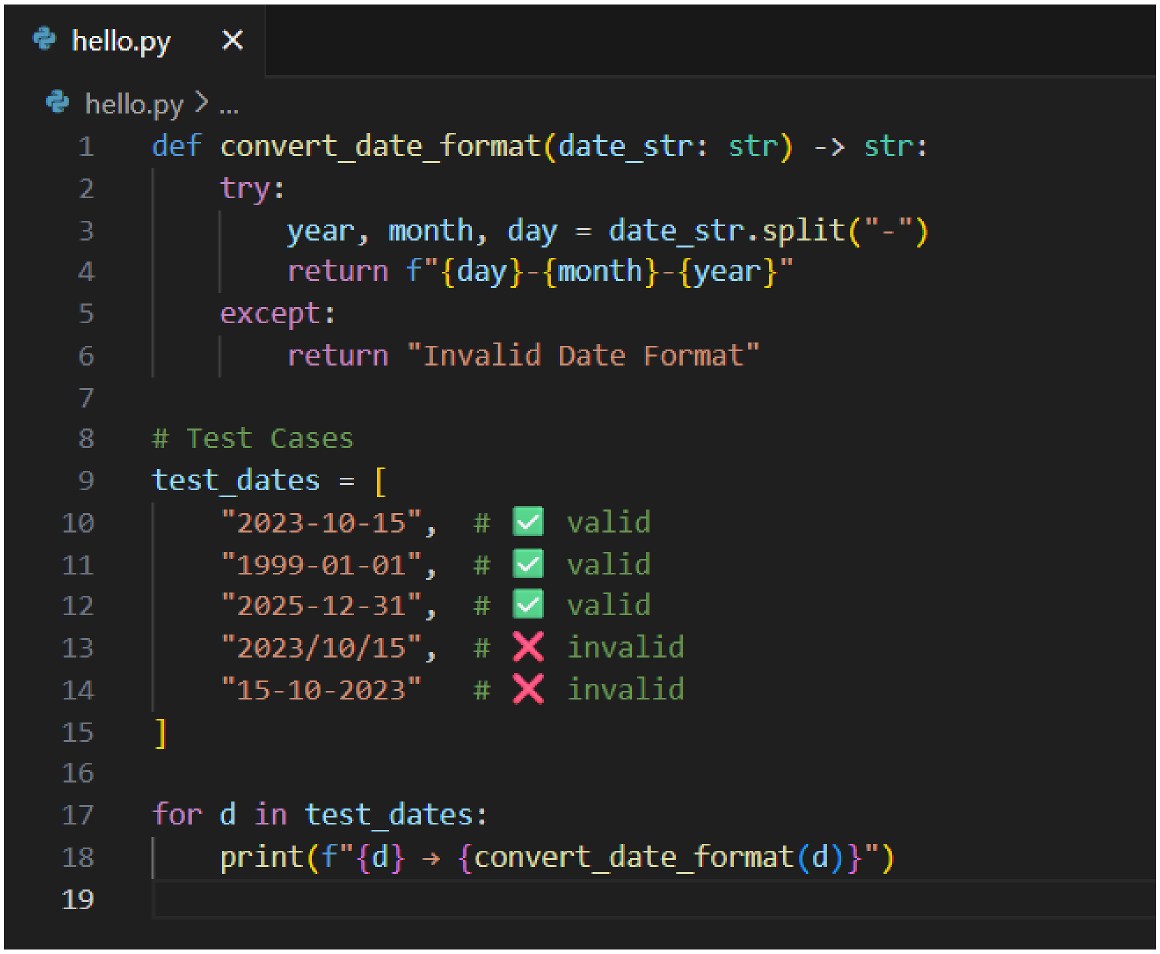
The class supports adding items (with cumulative prices), removing items, and calculating total cost correctly. Invalid removals return False.

## Task #5: Date Format Converter

### Prompt

Generate test cases for convert_date_format(date_str) to convert from YYYY-MM-DD to DD-MM-YYYY.

## Python Code

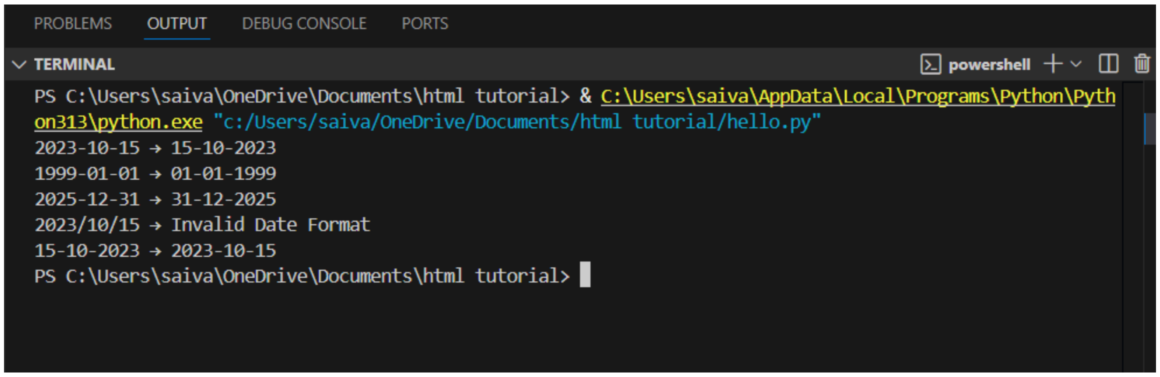


hello.py ×

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> 
```

## Observation

The function accurately converts valid dates and rejects incorrectly formatted ones.