ASSIGNMENT-8.1

HT NO:2303A51706

Batch No:28

Course:AI Assisted Coding

Task Description 1: (Password Strength Validator – Apply AI in

Security Context)

• **Task: Apply AI to generate at least 3 assert test cases for**

is_strong_password(password) and implement the validator

function.

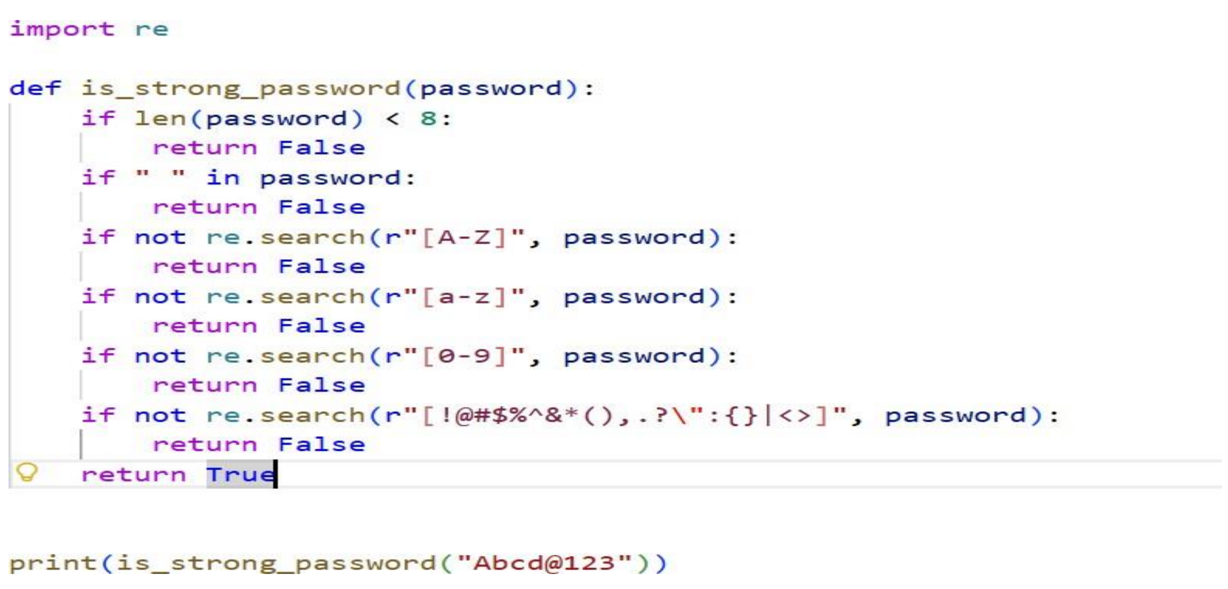
Prompt:

Generate at least 3 assert test cases for a Python function is_strong_password(password). Requirements:

- Minimum 8 characters
- Must contain uppercase, lowercase, digit, and special character
- Must not contain spaces

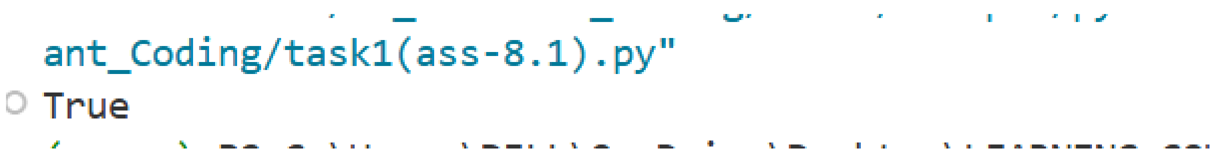
  Then implement the function so all test cases pass.

1.General Testing



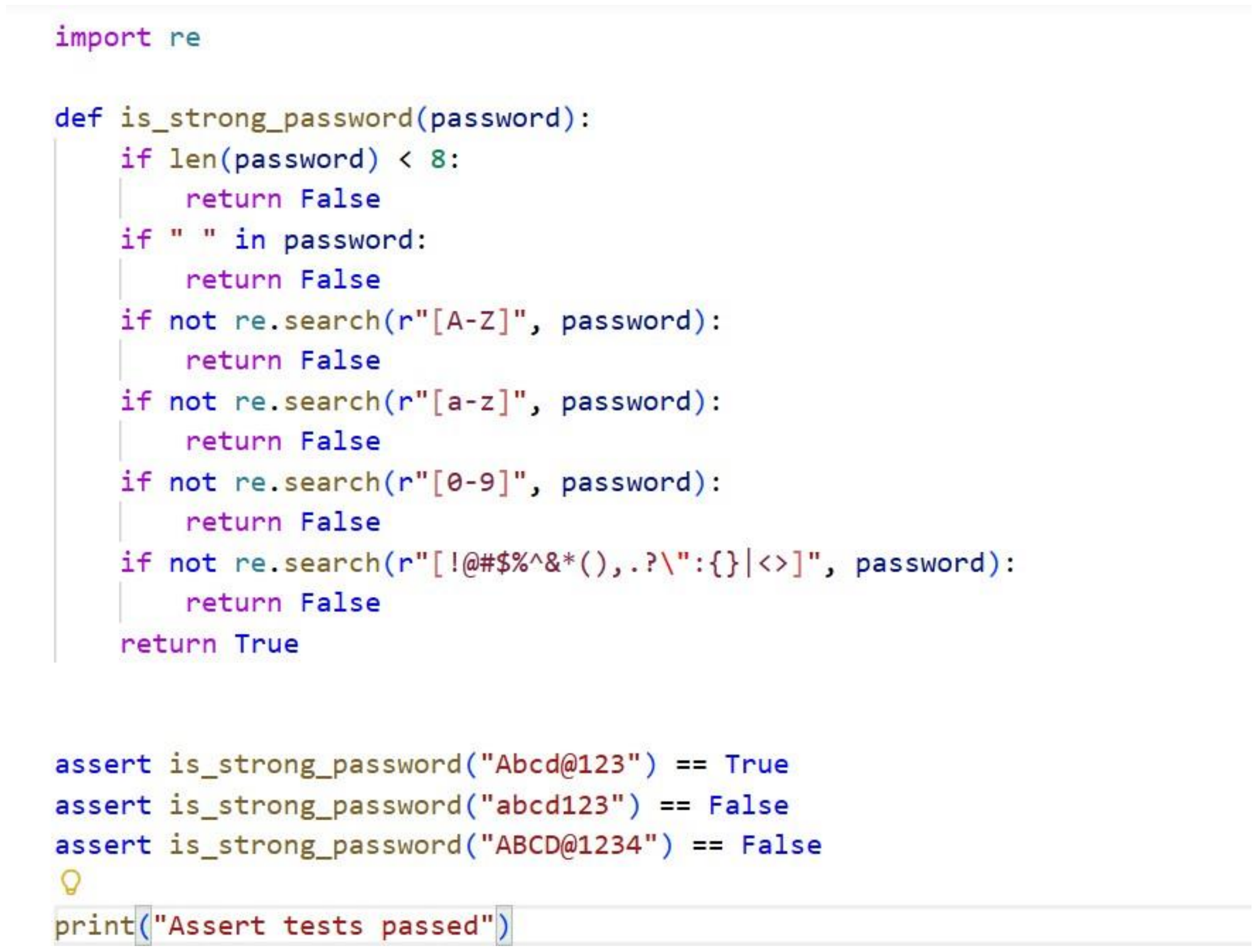
```python
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        return False
    return True


print(is_strong_password("Abcd@123"))
```

Output:



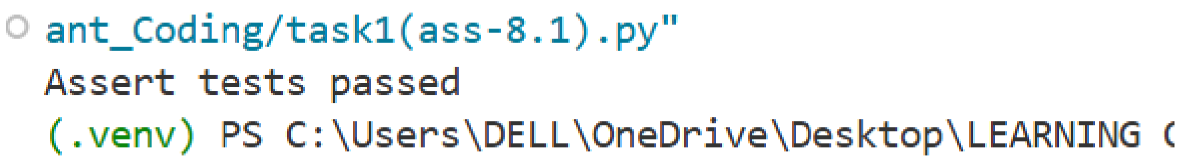
```
ant_Coding/task1(ass-8.1).py"
True
```

2. Assert testing



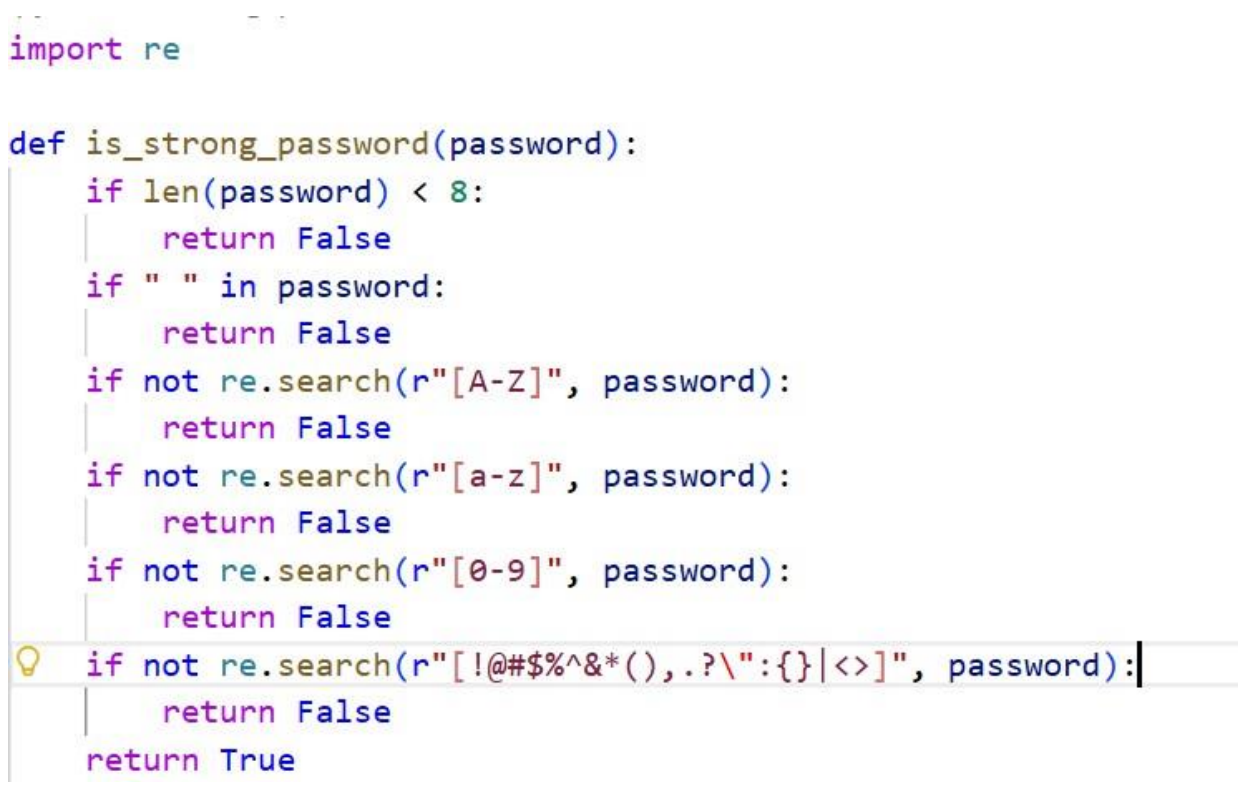
```python
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        return False
    return True


assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123") == False
assert is_strong_password("ABCD@1234") == False

print("Assert tests passed")
```

Output:



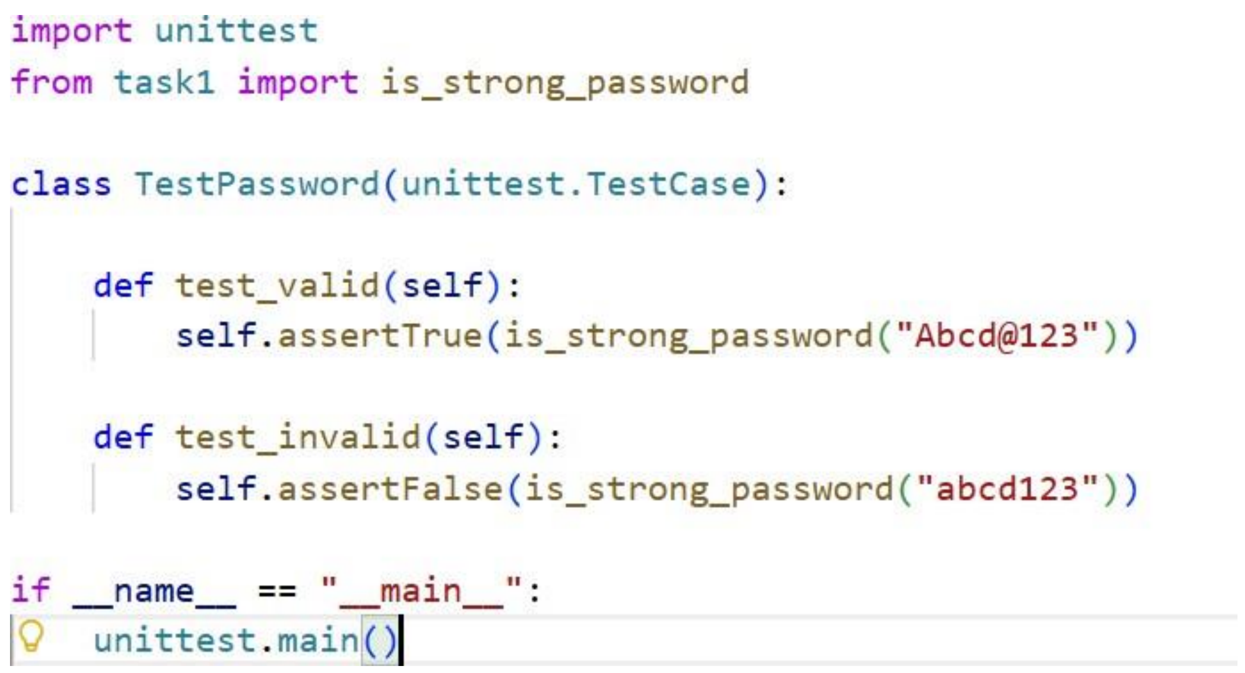
```
ant_Coding/task1(ass-8.1).py"
Assert tests passed
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING (
```

3. Unit testing



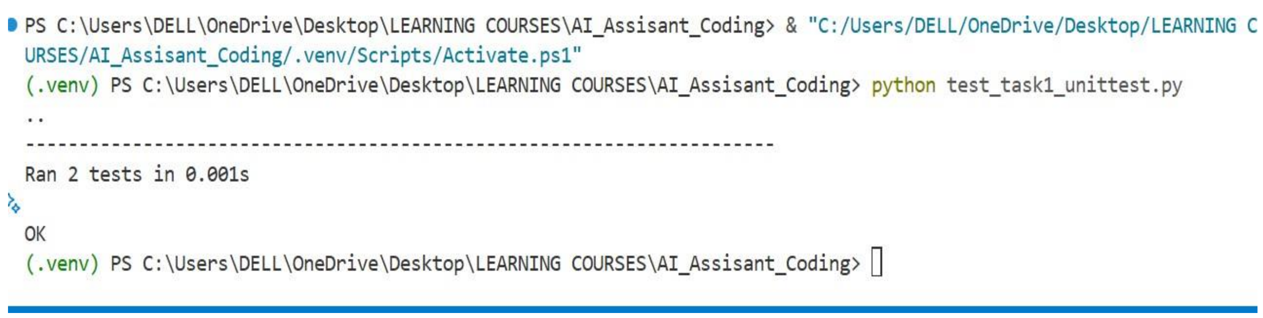
```python
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        return False
    return True
```

Create Next python file to run Test Cases:



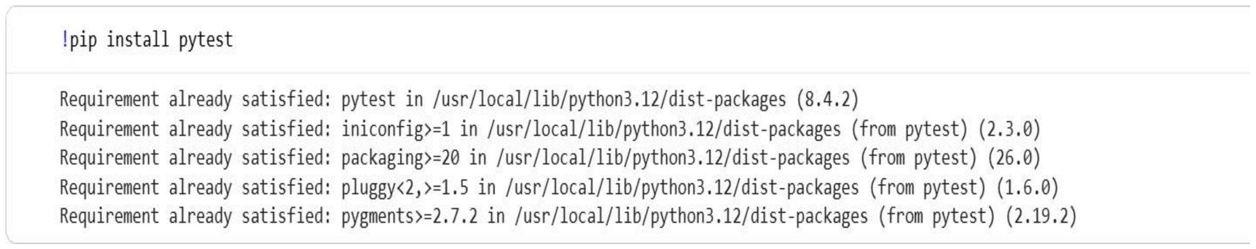
```python
import unittest
from task1 import is_strong_password

class TestPassword(unittest.TestCase):

    def test_valid(self):
        self.assertTrue(is_strong_password("Abcd@123"))

    def test_invalid(self):
        self.assertFalse(is_strong_password("abcd123"))

if __name__ == "__main__":
    unittest.main()
```

Output:



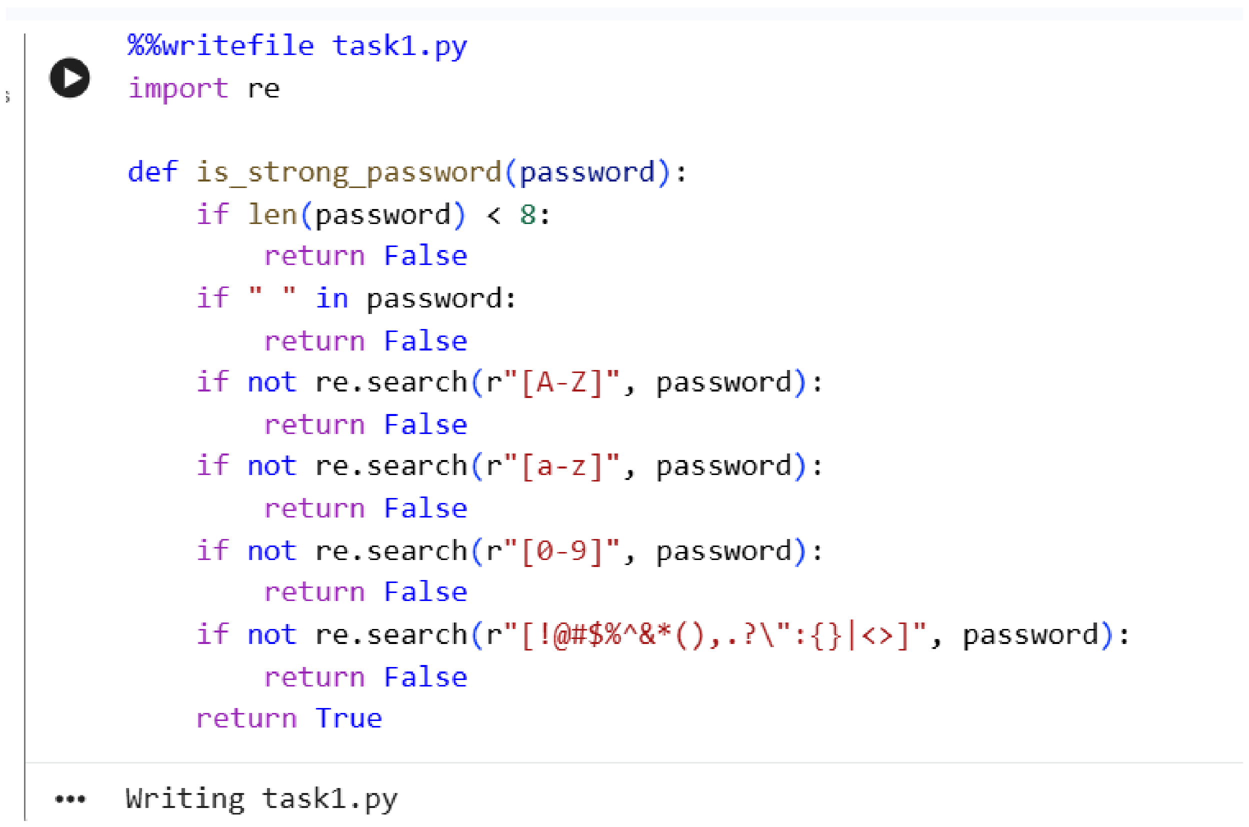
```
PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEARNING C
URSES/AI_Assisant_Coding/.venv/Scripts/Activate.ps1"
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python test_task1_unittest.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> 
```

4. Pytest and Output:



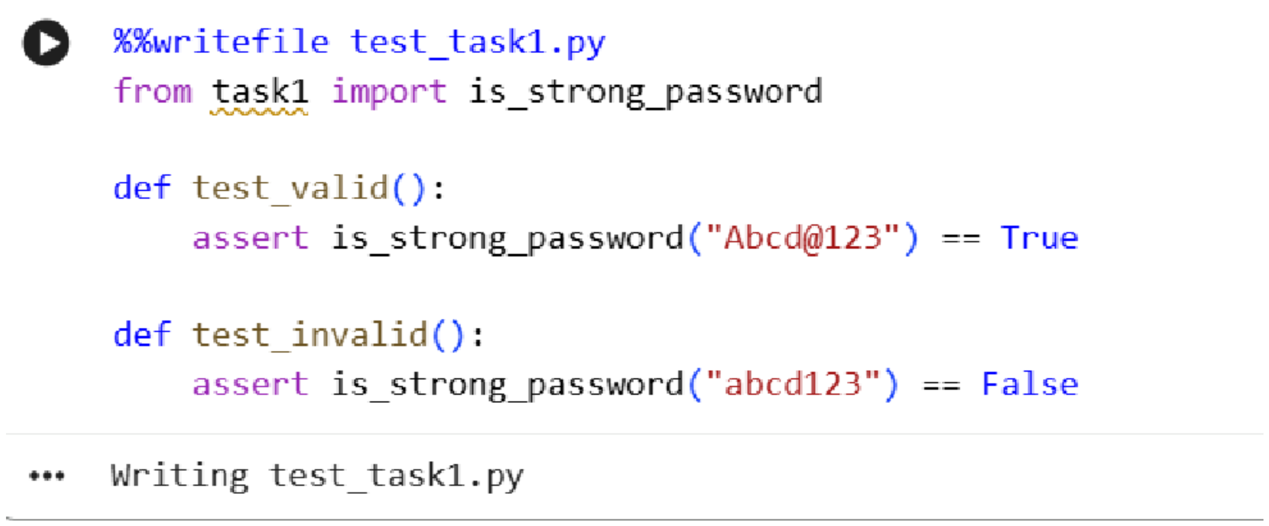
```
!pip install pytest
```

```
Requirement already satisfied: pytest in /usr/local/lib/python3.12/dist-packages (8.4.2)
Requirement already satisfied: iniconfig>=1 in /usr/local/lib/python3.12/dist-packages (from pytest) (2.3.0)
Requirement already satisfied: packaging>=20 in /usr/local/lib/python3.12/dist-packages (from pytest) (26.0)
Requirement already satisfied: pluggy<2,>=1.5 in /usr/local/lib/python3.12/dist-packages (from pytest) (1.6.0)
Requirement already satisfied: pygments>=2.7.2 in /usr/local/lib/python3.12/dist-packages (from pytest) (2.19.2)
```



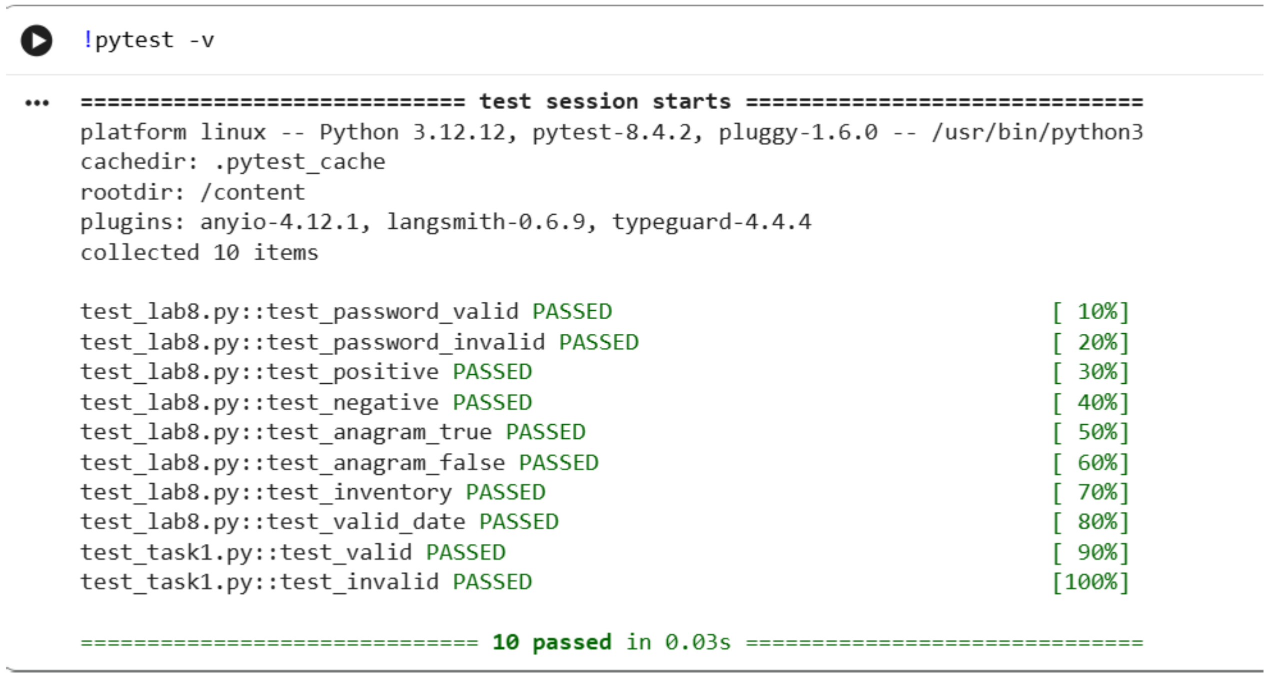
```python
%%writefile task1.py
import re

def is_strong_password(password):
    if len(password) < 8:
        return False
    if " " in password:
        return False
    if not re.search(r"[A-Z]", password):
        return False
    if not re.search(r"[a-z]", password):
        return False
    if not re.search(r"[0-9]", password):
        return False
    if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        return False
    return True
```

```
Writing task1.py
```



```
%%writefile test_task1.py
from task1 import is_strong_password

def test_valid():
    assert is_strong_password("Abcd@123") == True

def test_invalid():
    assert is_strong_password("abcd123") == False
```

```
Writing test_task1.py
```



```
!pytest -v
```

```
============================= test session starts ==============================
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED                                 [ 10%]
test_lab8.py::test_password_invalid PASSED                               [ 20%]
test_lab8.py::test_positive PASSED                                       [ 30%]
test_lab8.py::test_negative PASSED                                       [ 40%]
test_lab8.py::test_anagram_true PASSED                                   [ 50%]
test_lab8.py::test_anagram_false PASSED                                  [ 60%]
test_lab8.py::test_inventory PASSED                                      [ 70%]
test_lab8.py::test_valid_date PASSED                                     [ 80%]
test_task1.py::test_valid PASSED                                         [ 90%]
test_task1.py::test_invalid PASSED                                       [100%]

============================== 10 passed in 0.03s ==============================
```

Justification:

This task applies AI in a **security context**.

- AI generates edge-case test scenarios.
- TDD approach ensures validation rules are clearly defined before coding.
- Improves reliability of password validation.
- Prevents weak or insecure passwords.
- Demonstrates input validation and regex handling.

→Shows importance of secure coding

→ Encourages validation before implementation

Task Description 2: (Number Classification with Loops – Apply AI for Edge Case Handling)

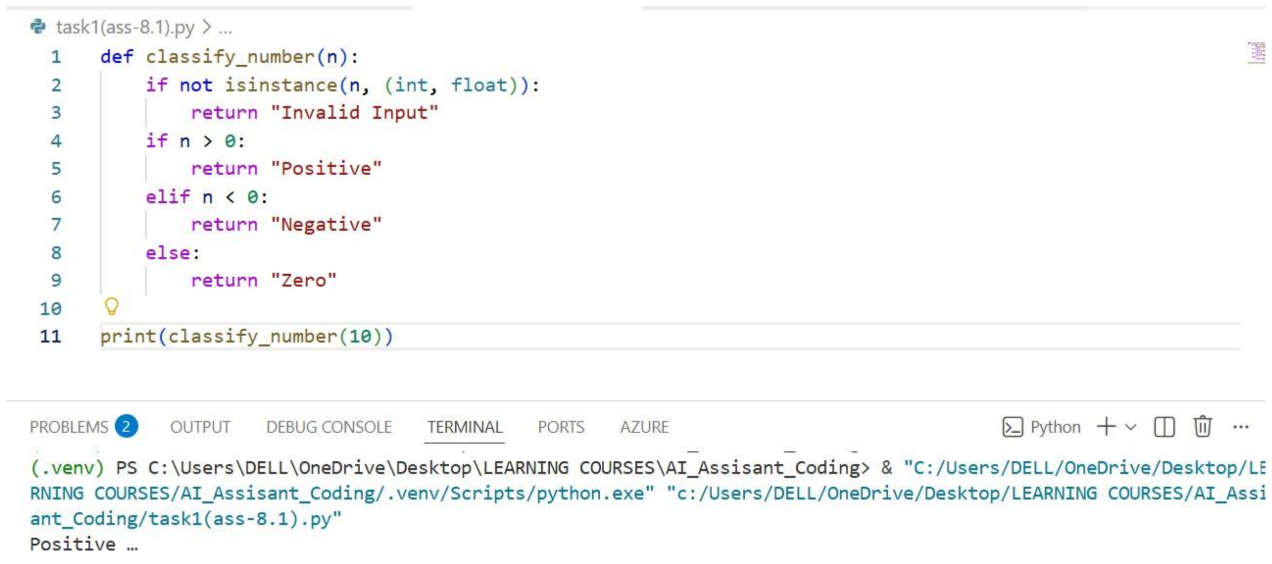
• **Task: Use AI to generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.**

Prompt:

Generate at least 3 assert test cases for a function classify_number(n) that:

- Returns "Positive", "Negative", or "Zero"
- Handles invalid inputs like strings and None
- Includes boundary values (-1, 0, 1)
  Then implement the function using loops so all tests pass.

1. General Testing and Output

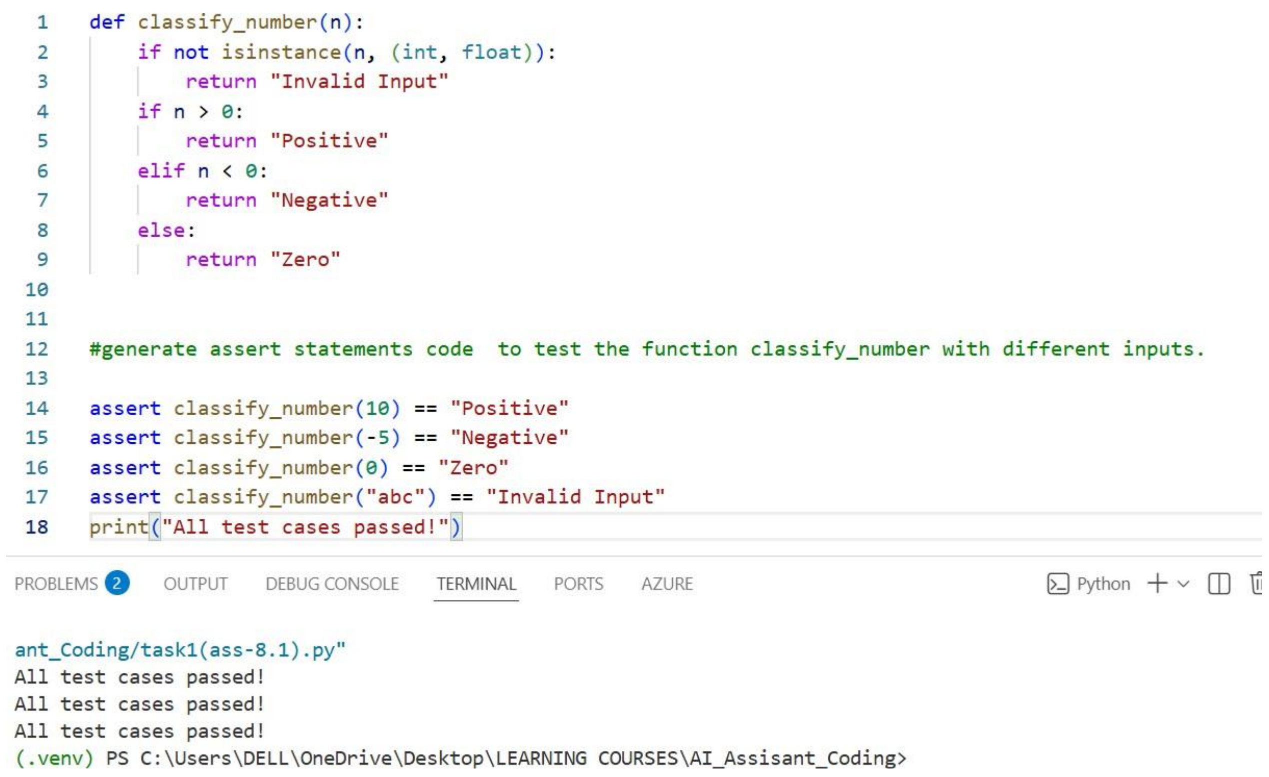


task1(ass-8.1).py > ...

```python
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"

print(classify_number(10))
```

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LE
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assi
ant_Coding/task1(ass-8.1).py"
Positive …
```

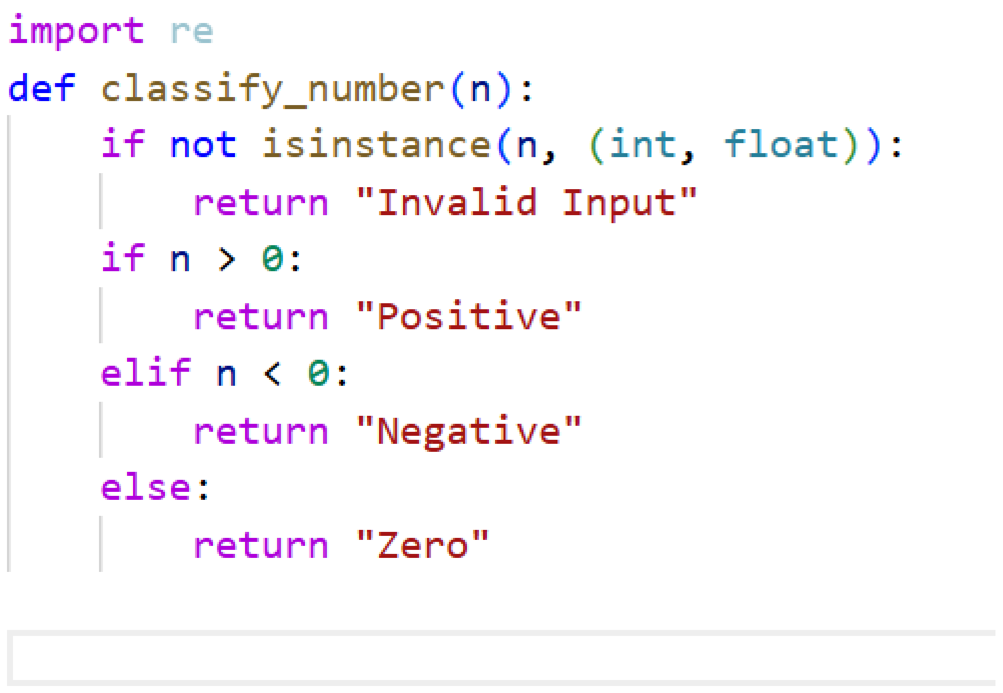
2. Assert testing and Output



```python
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"


#generate assert statements code  to test the function classify_number with different inputs.

assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number("abc") == "Invalid Input"
print("All test cases passed!")
```

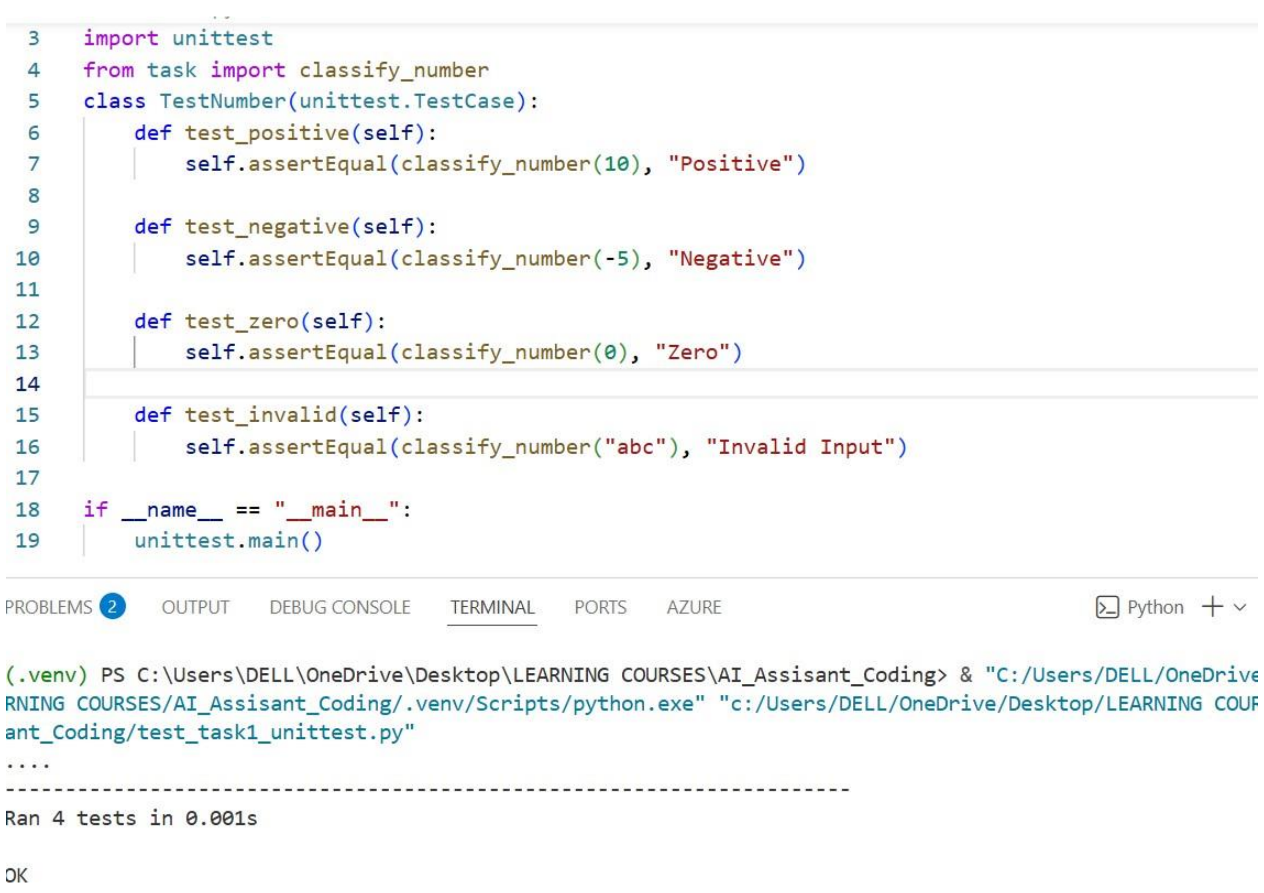
```
ant_Coding/task1(ass-8.1).py"
All test cases passed!
All test cases passed!
All test cases passed!
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

3.Unit testing and Output



```python
import re
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
```

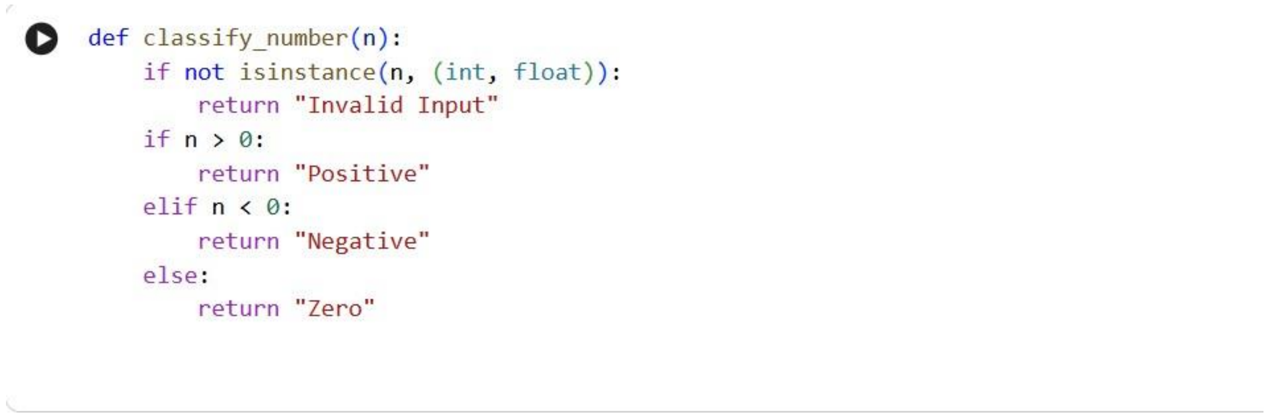
Create Next python file to run Test Cases:



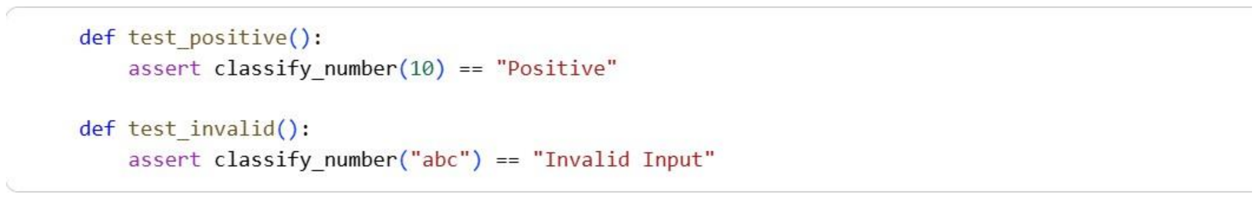
```
import unittest
from task import classify_number
class TestNumber(unittest.TestCase):
    def test_positive(self):
        self.assertEqual(classify_number(10), "Positive")

    def test_negative(self):
        self.assertEqual(classify_number(-5), "Negative")

    def test_zero(self):
        self.assertEqual(classify_number(0), "Zero")

    def test_invalid(self):
        self.assertEqual(classify_number("abc"), "Invalid Input")

if __name__ == "__main__":
    unittest.main()
```

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COUF
ant_Coding/test_task1_unittest.py"
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```

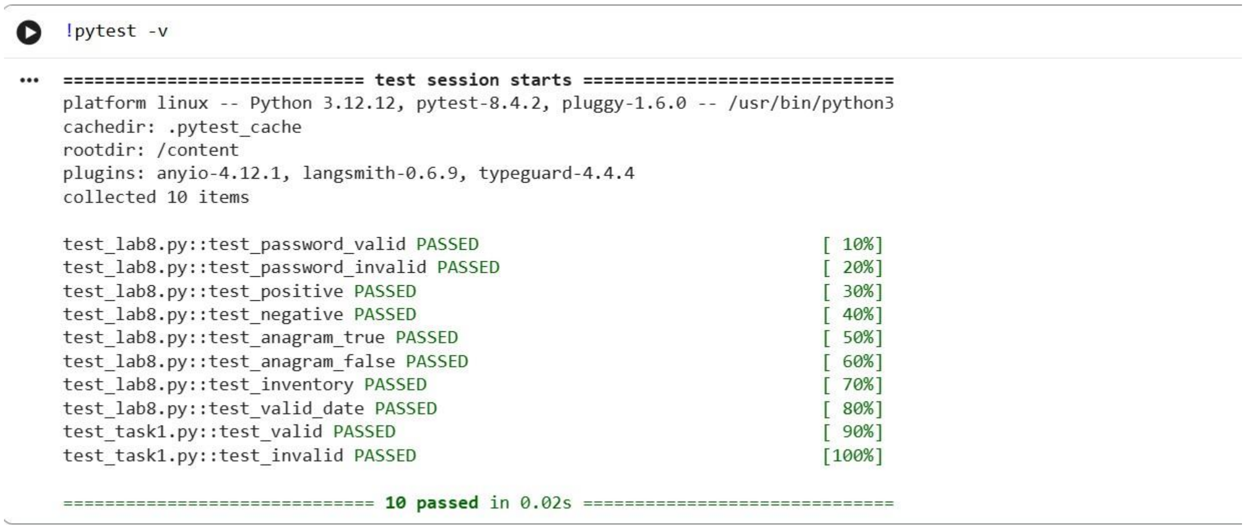
4. Pytest and Output:



```
def classify_number(n):
    if not isinstance(n, (int, float)):
        return "Invalid Input"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
```



```
def test_positive():
    assert classify_number(10) == "Positive"

def test_invalid():
    assert classify_number("abc") == "Invalid Input"
```



```
!pytest -v

============================ test session starts ==============================
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED                                  [ 10%]
test_lab8.py::test_password_invalid PASSED                                [ 20%]
test_lab8.py::test_positive PASSED                                        [ 30%]
test_lab8.py::test_negative PASSED                                        [ 40%]
test_lab8.py::test_anagram_true PASSED                                    [ 50%]
test_lab8.py::test_anagram_false PASSED                                   [ 60%]
test_lab8.py::test_inventory PASSED                                       [ 70%]
test_lab8.py::test_valid_date PASSED                                      [ 80%]
test_task1.py::test_valid PASSED                                          [ 90%]
test_task1.py::test_invalid PASSED                                        [100%]

============================== 10 passed in 0.02s ==============================
```

Justification:

This task focuses on **edge case handling**.

- AI helps identify boundary conditions.
- Ensures handling of invalid inputs.
- Demonstrates defensive programming.
- Reinforces loop usage and type checking.

→ Encourages robustness

→ Improves error handling

→ Demonstrates TDD validation logic

Task Description 3: (Anagram Checker – Apply AI for String Analysis)

• Task: Use AI to generate at least 3 assert test cases for is_anagram(str1, str2) and implement the function.

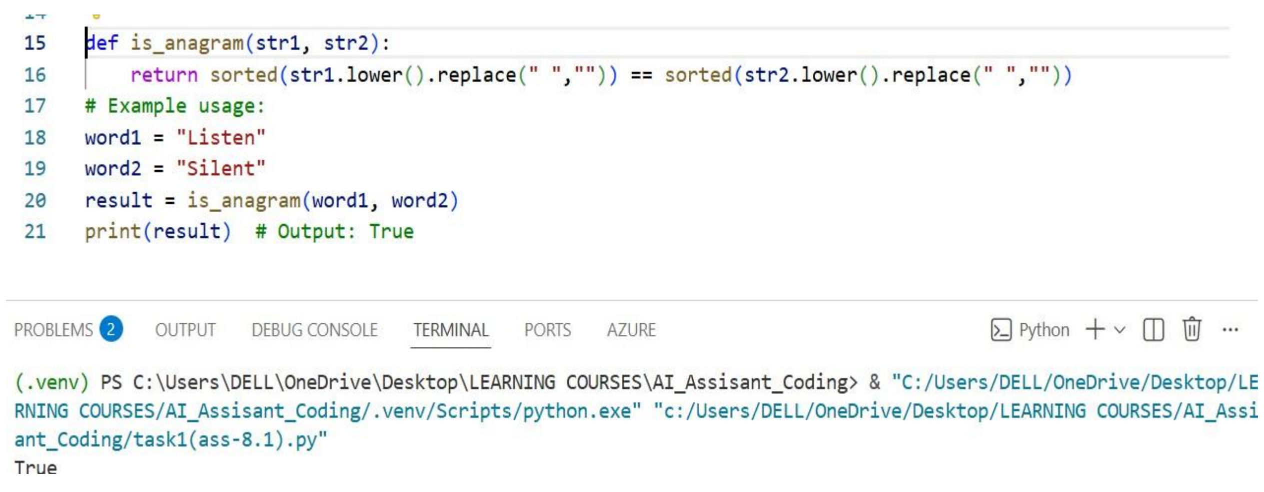
Prompt:

Generate at least 3 assert test cases for a function is_anagram(str1, str2) that:

- Ignores case
- Ignores spaces and punctuation

- Handles empty strings
  Then implement the function so all tests pass.

1. General Testing and Output:

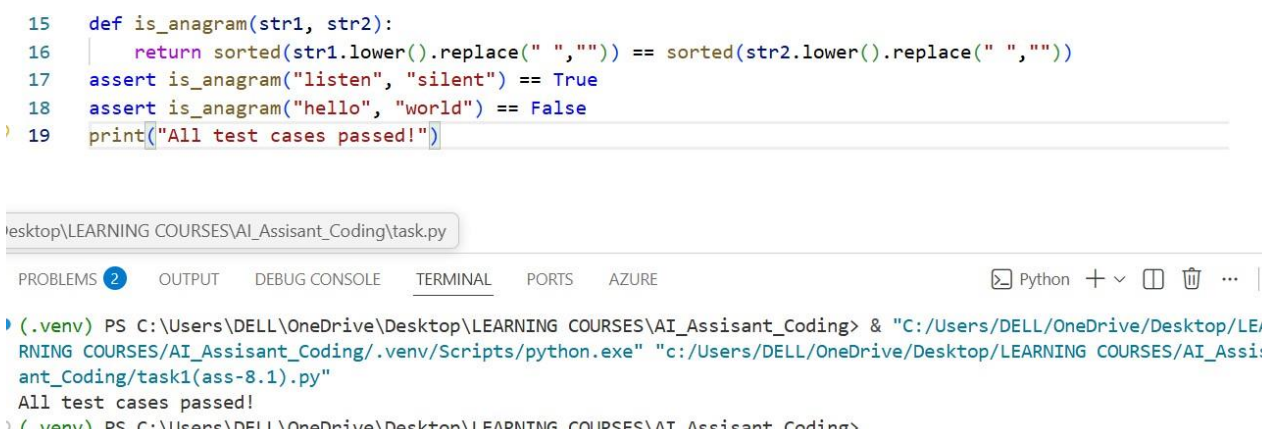


```python
def is_anagram(str1, str2):
    return sorted(str1.lower().replace(" ","")) == sorted(str2.lower().replace(" ",""))
# Example usage:
word1 = "Listen"
word2 = "Silent"
result = is_anagram(word1, word2)
print(result)  # Output: True
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LE
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assi
ant_Coding/task1(ass-8.1).py"
True
```

2. Assert testing and Output:



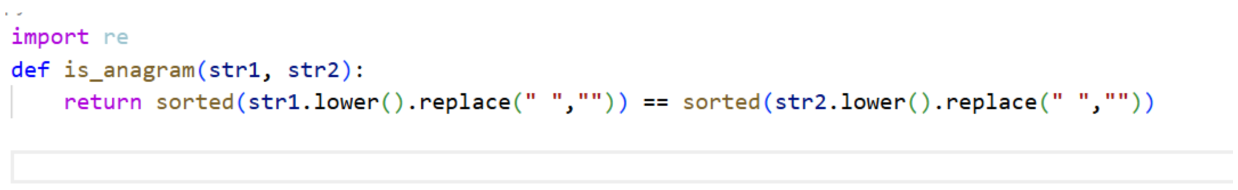
```python
def is_anagram(str1, str2):
    return sorted(str1.lower().replace(" ","")) == sorted(str2.lower().replace(" ",""))
assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
print("All test cases passed!")
```

esktop\LEARNING COURSES\AI_Assisant_Coding\task.py

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE Python

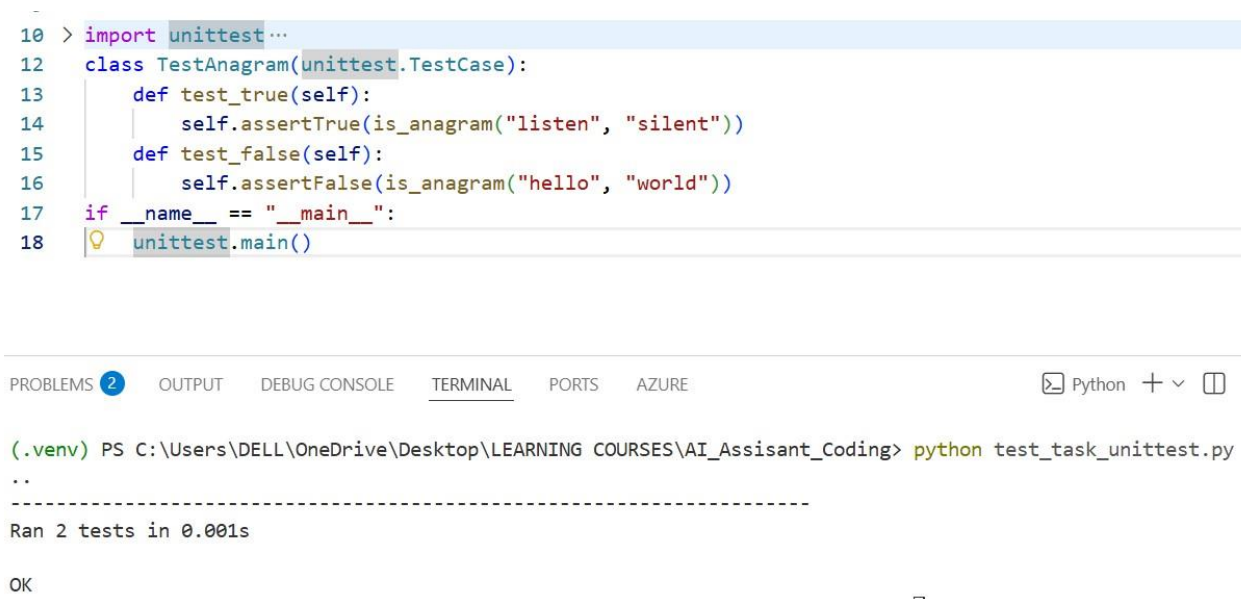
```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LE
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assi
ant_Coding/task1(ass-8.1).py"
All test cases passed!
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

3. Unit testing and Output:



```python
import re
def is_anagram(str1, str2):
    return sorted(str1.lower().replace(" ","")) == sorted(str2.lower().replace(" ",""))
```

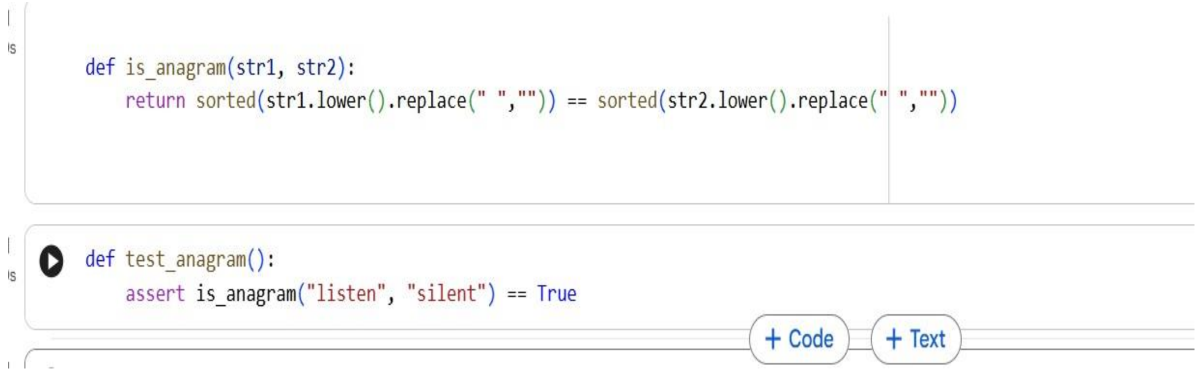
Create Next python file to run Test Cases:



```python
import unittest···
class TestAnagram(unittest.TestCase):
    def test_true(self):
        self.assertTrue(is_anagram("listen", "silent"))
    def test_false(self):
        self.assertFalse(is_anagram("hello", "world"))
if __name__ == "__main__":
    unittest.main()
```

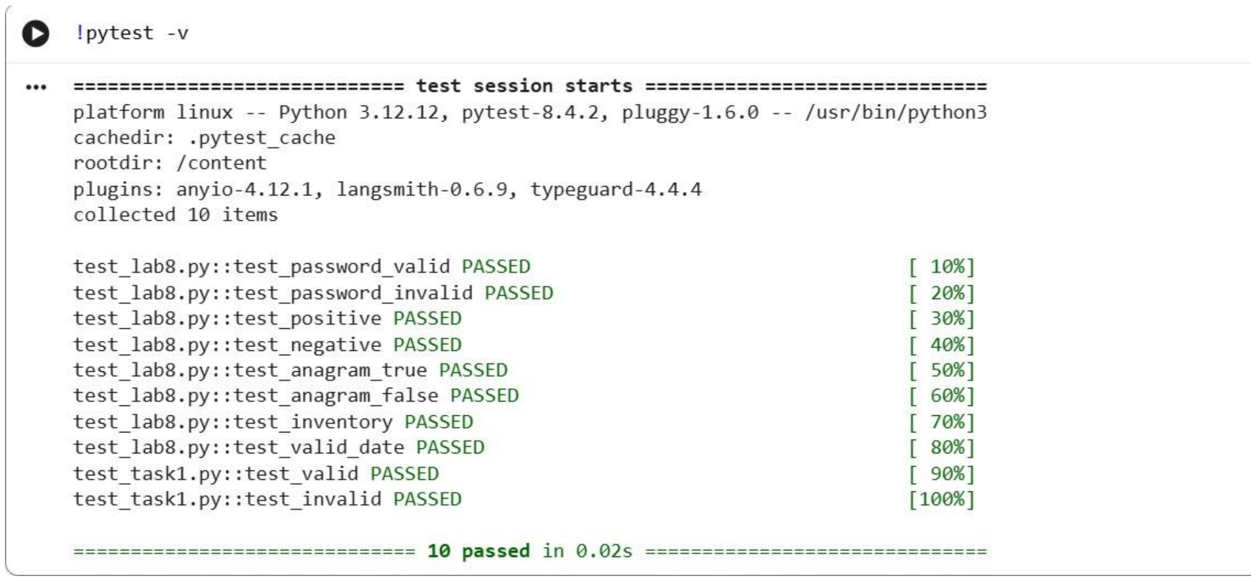
```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python test_task_unittest.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

4. Pytest and Output:



```python
def is_anagram(str1, str2):
    return sorted(str1.lower().replace(" ","")) == sorted(str2.lower().replace(" ",""))
```

```python
def test_anagram():
    assert is_anagram("listen", "silent") == True
```



```
!pytest -v

============================= test session starts ==============================
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED                                 [ 10%]
test_lab8.py::test_password_invalid PASSED                               [ 20%]
test_lab8.py::test_positive PASSED                                       [ 30%]
test_lab8.py::test_negative PASSED                                       [ 40%]
test_lab8.py::test_anagram_true PASSED                                   [ 50%]
test_lab8.py::test_anagram_false PASSED                                  [ 60%]
test_lab8.py::test_inventory PASSED                                      [ 70%]
test_lab8.py::test_valid_date PASSED                                     [ 80%]
test_task1.py::test_valid PASSED                                         [ 90%]
test_task1.py::test_invalid PASSED                                       [100%]

============================== 10 passed in 0.02s ==============================
```

Justification:

This task applies AI for **string analysis**.

- AI helps detect tricky test cases.
- Validates normalization (case folding, filtering).
- Ensures string preprocessing before comparison.
- Covers empty and identical cases.

→ Improves text processing logic
→ Demonstrates data cleaning
→ Shows importance of normalization

Task Description 4: (Inventory Class – Apply AI to Simulate Real-World Inventory System)

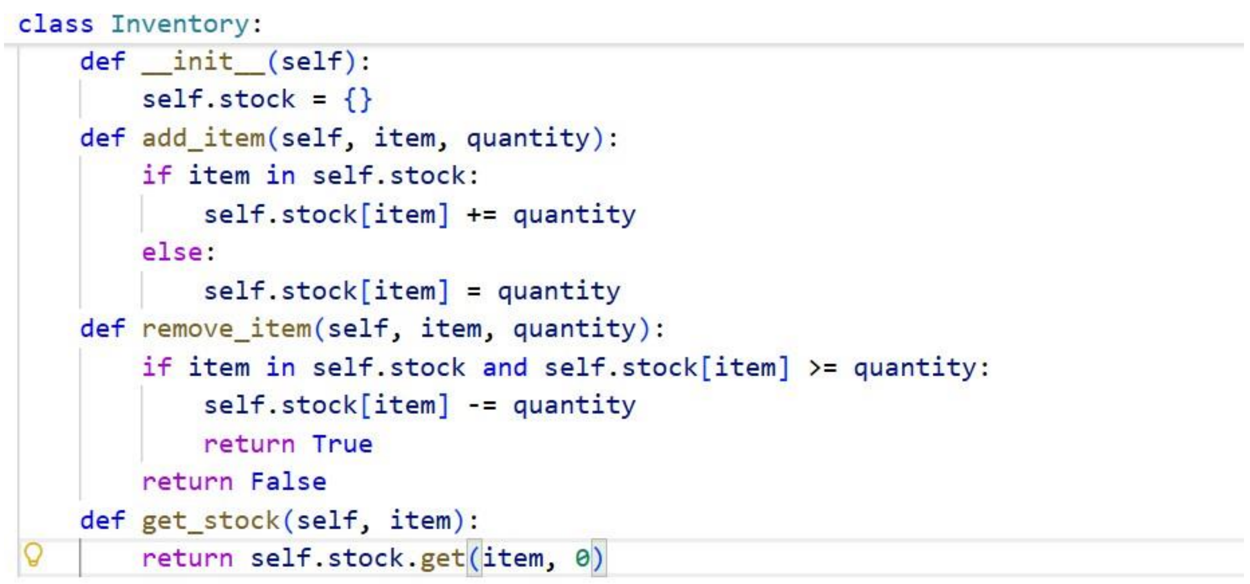
• Task: Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.

Prompt:

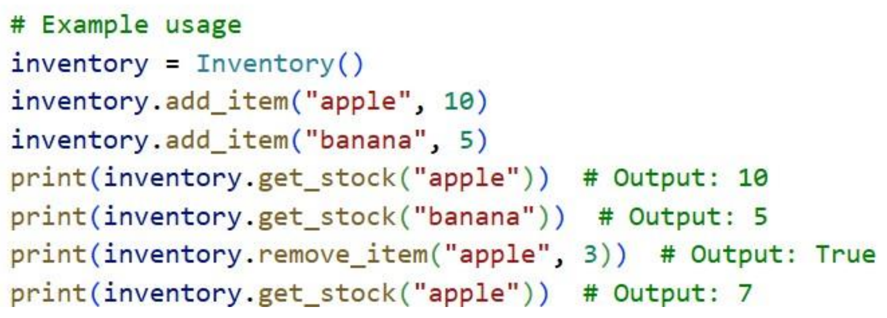
Generate at least 3 assert-based test cases for an Inventory class that supports:

- add_item(name, quantity)
- remove_item(name, quantity)
- get_stock(name)
  Then implement the class so all assertions pass.

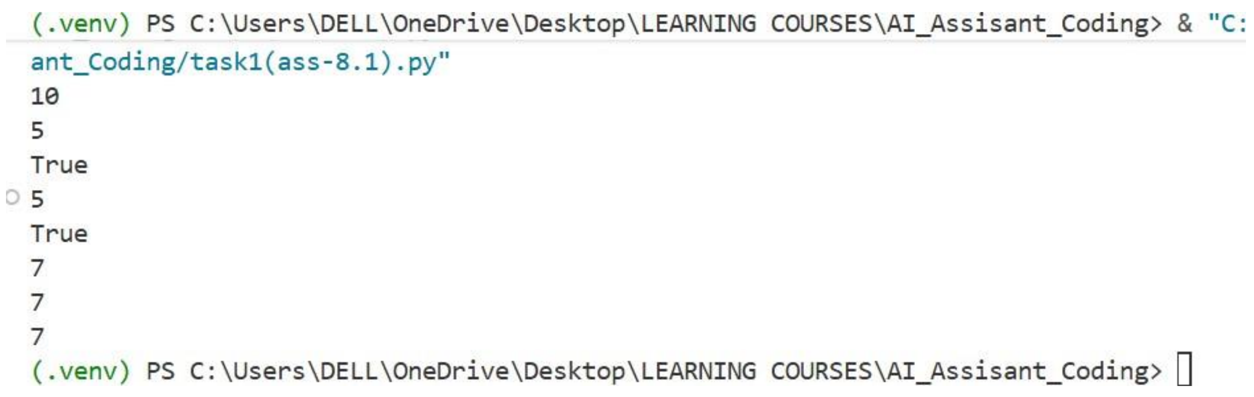
1.General Testing and Output:



```python
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False
    def get_stock(self, item):
        return self.stock.get(item, 0)
```

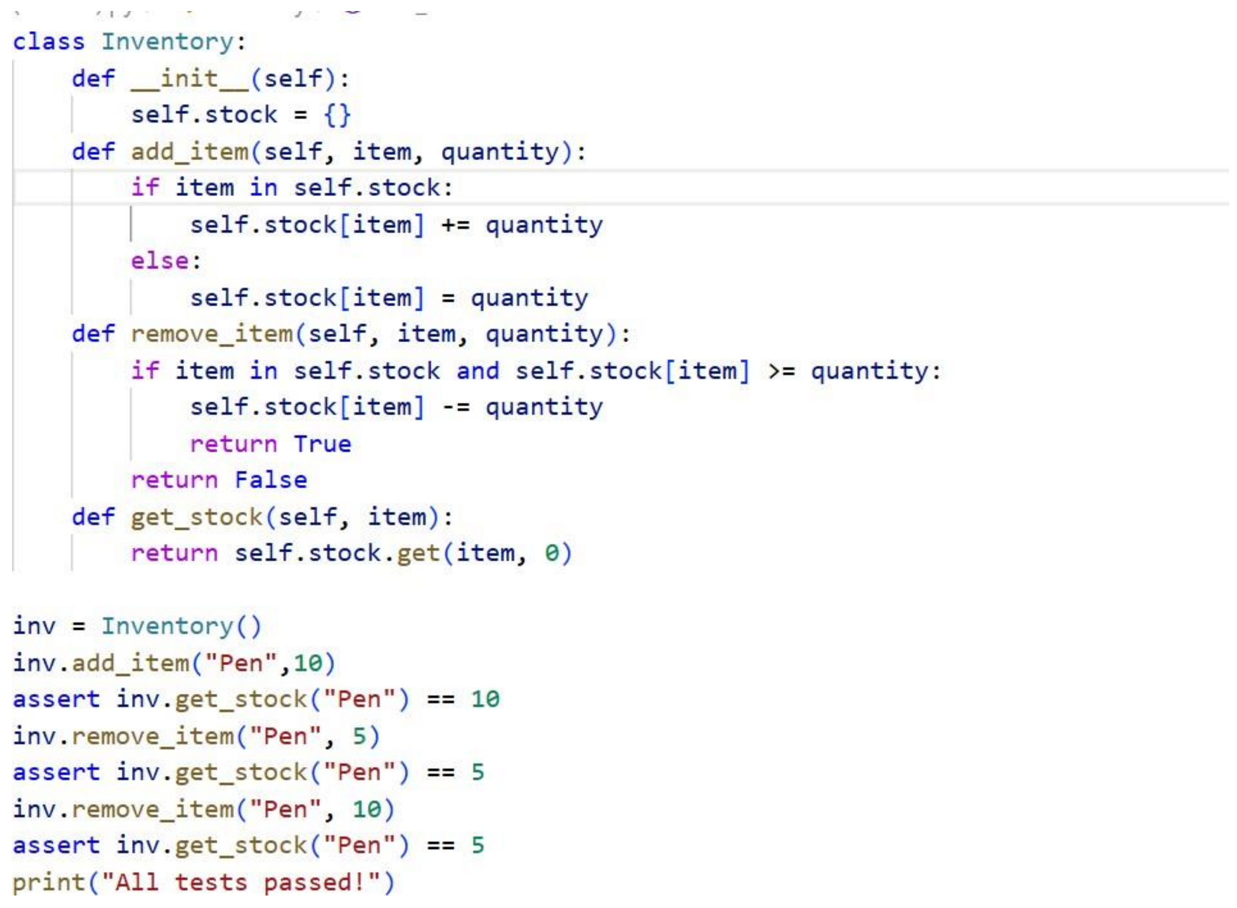


```python
# Example usage
inventory = Inventory()
inventory.add_item("apple", 10)
inventory.add_item("banana", 5)
print(inventory.get_stock("apple"))  # Output: 10
print(inventory.get_stock("banana"))  # Output: 5
print(inventory.remove_item("apple", 3))  # Output: True
print(inventory.get_stock("apple"))  # Output: 7
```

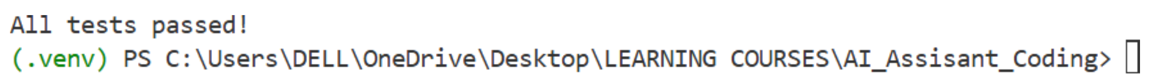


```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:
ant_Coding/task1(ass-8.1).py"
10
5
True
5
True
7
7
7
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

2. Assert testing and Output:

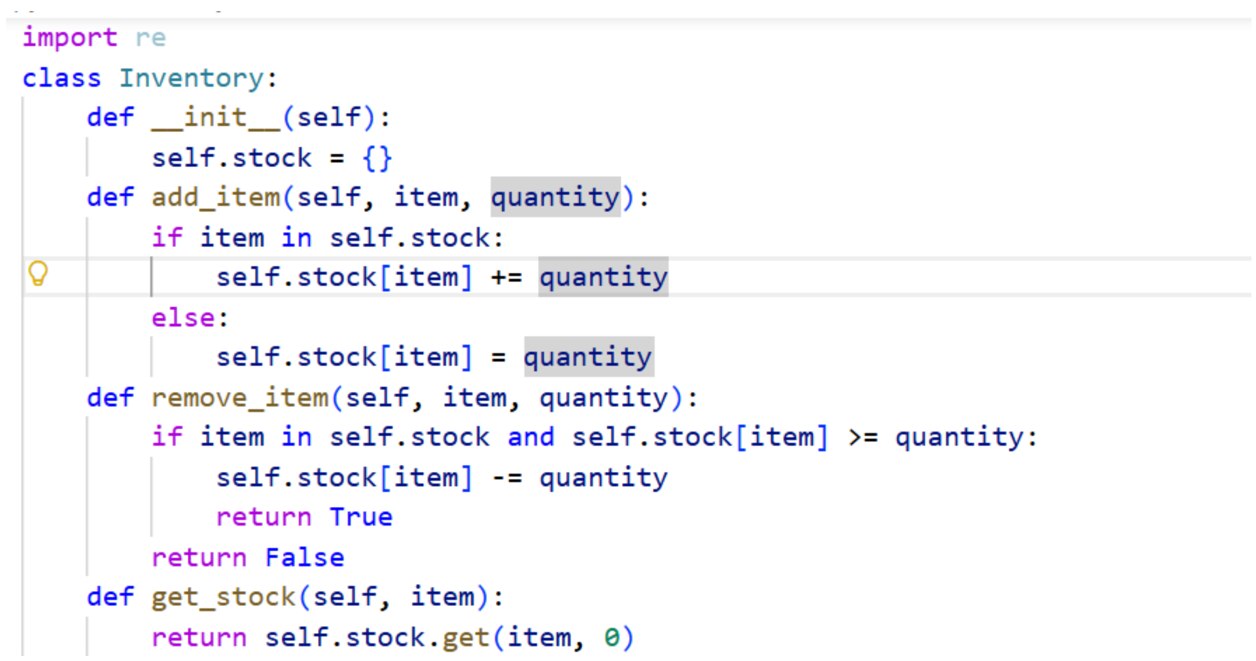


```python
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False
    def get_stock(self, item):
        return self.stock.get(item, 0)

inv = Inventory()
inv.add_item("Pen",10)
assert inv.get_stock("Pen") == 10
inv.remove_item("Pen", 5)
assert inv.get_stock("Pen") == 5
inv.remove_item("Pen", 10)
assert inv.get_stock("Pen") == 5
print("All tests passed!")
```



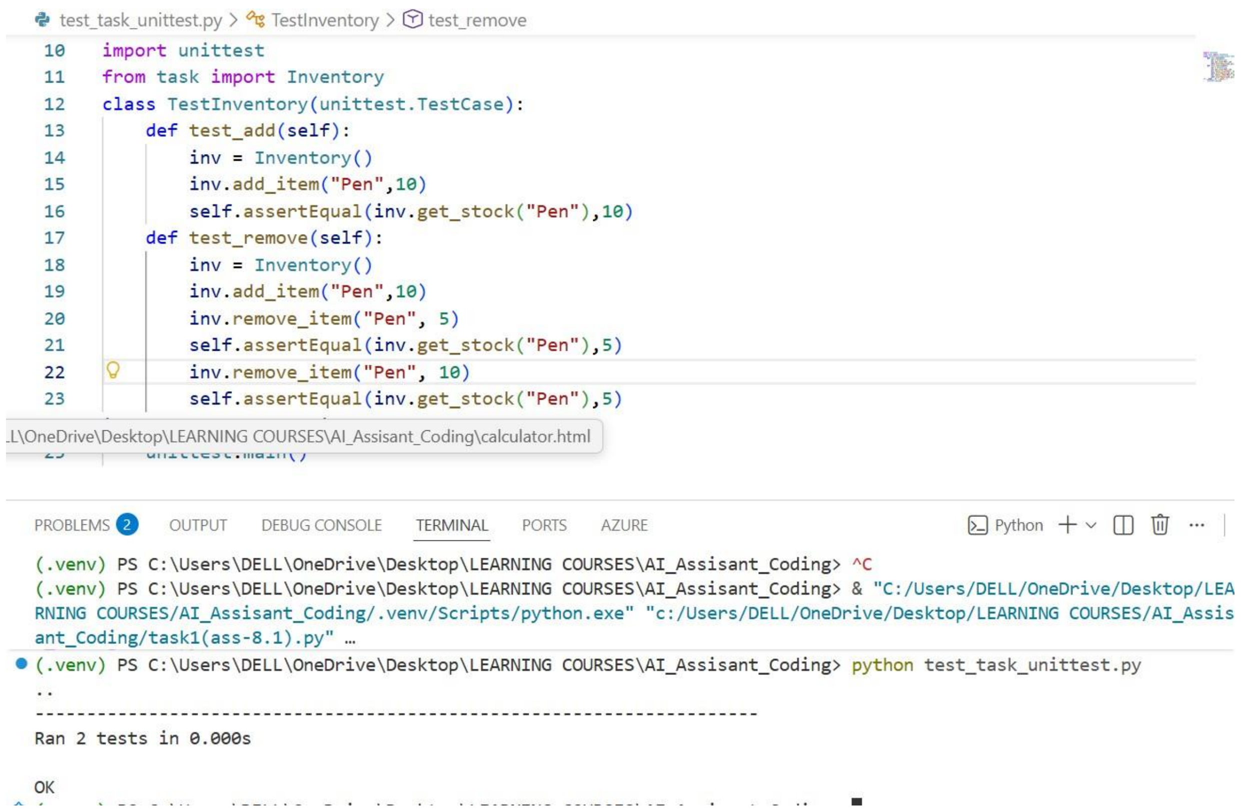
```
All tests passed!
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

3. Unit testing and Output:



```python
import re
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False
    def get_stock(self, item):
        return self.stock.get(item, 0)
```

Create Next python file to run Test Cases:



test_task_unittest.py > TestInventory > test_remove

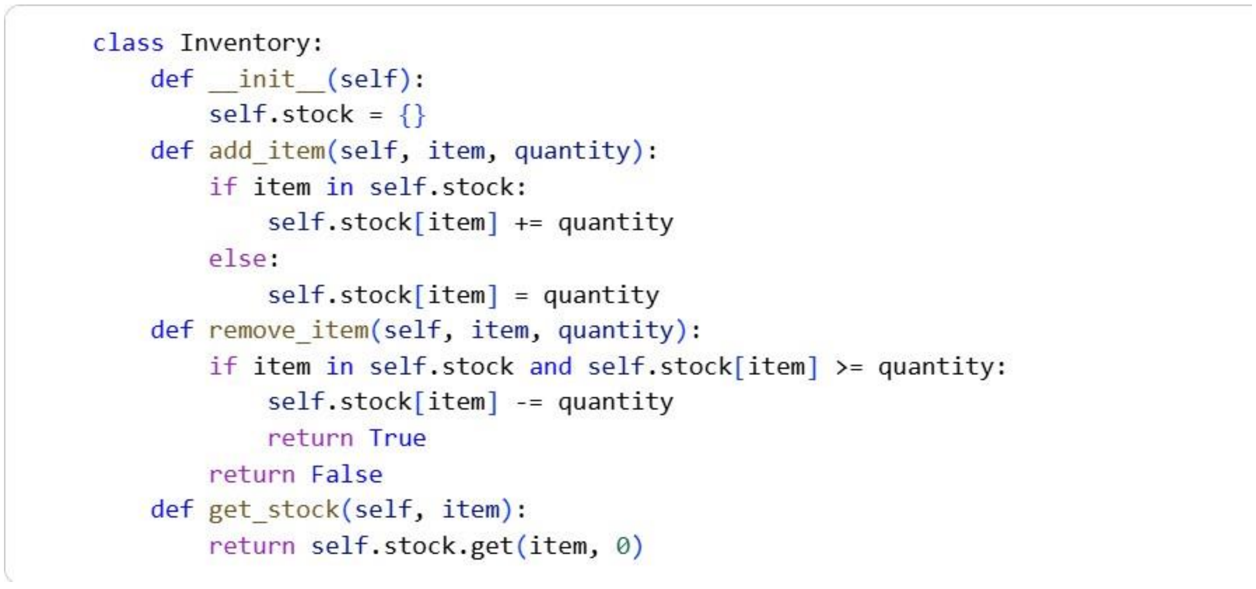
```python
import unittest
from task import Inventory
class TestInventory(unittest.TestCase):
    def test_add(self):
        inv = Inventory()
        inv.add_item("Pen",10)
        self.assertEqual(inv.get_stock("Pen"),10)
    def test_remove(self):
        inv = Inventory()
        inv.add_item("Pen",10)
        inv.remove_item("Pen", 5)
        self.assertEqual(inv.get_stock("Pen"),5)
        inv.remove_item("Pen", 10)
        self.assertEqual(inv.get_stock("Pen"),5)
```

.L\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding\calculator.html

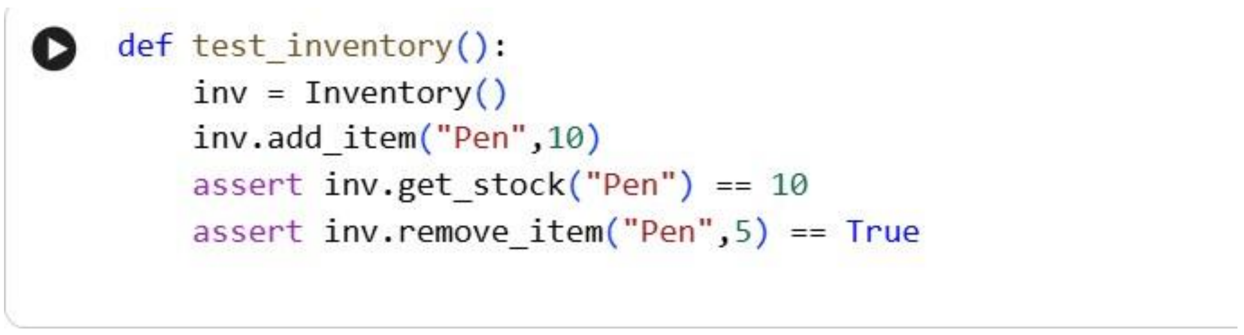
```
    unittest.main()
```

```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> ^C
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Desktop/LEA
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI_Assis
ant_Coding/task1(ass-8.1).py" …
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python test_task_unittest.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

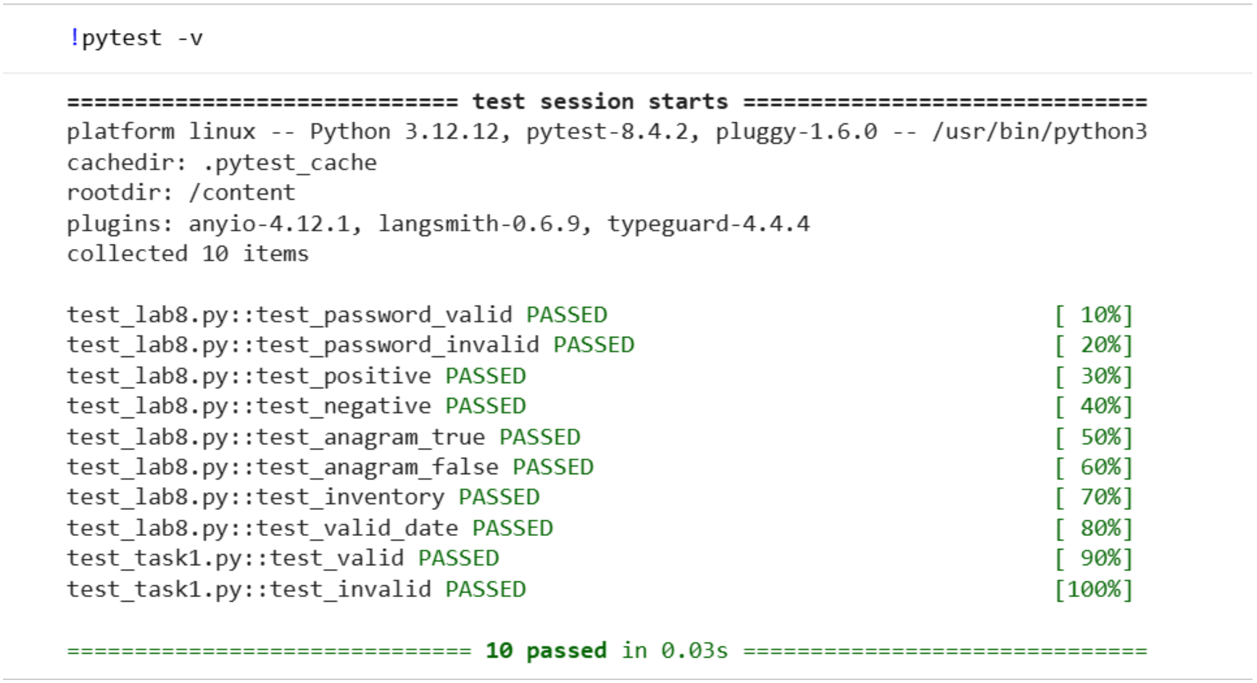
4. Pytest and Output:



```
class Inventory:
    def __init__(self):
        self.stock = {}
    def add_item(self, item, quantity):
        if item in self.stock:
            self.stock[item] += quantity
        else:
            self.stock[item] = quantity
    def remove_item(self, item, quantity):
        if item in self.stock and self.stock[item] >= quantity:
            self.stock[item] -= quantity
            return True
        return False
    def get_stock(self, item):
        return self.stock.get(item, 0)
```



```
def test_inventory():
    inv = Inventory()
    inv.add_item("Pen",10)
    assert inv.get_stock("Pen") == 10
    assert inv.remove_item("Pen",5) == True
```



```
!pytest -v
```

```
============================== test session starts ==============================
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED                                 [ 10%]
test_lab8.py::test_password_invalid PASSED                               [ 20%]
test_lab8.py::test_positive PASSED                                       [ 30%]
test_lab8.py::test_negative PASSED                                       [ 40%]
test_lab8.py::test_anagram_true PASSED                                   [ 50%]
test_lab8.py::test_anagram_false PASSED                                  [ 60%]
test_lab8.py::test_inventory PASSED                                      [ 70%]
test_lab8.py::test_valid_date PASSED                                     [ 80%]
test_task1.py::test_valid PASSED                                         [ 90%]
test_task1.py::test_invalid PASSED                                       [100%]

============================== 10 passed in 0.03s ==============================
```

Justification:

This task simulates a **real-world system**.

- AI generates realistic stock management tests.
- Encourages object-oriented design.
- Validates state updates inside a class.
- Demonstrates data storage using dictionaries.

→ Applies OOP concepts
→ Simulates real inventory system
→ Encourages state validation

Task Description 5 (Date Validation & Formatting – Apply AI for

Data Validation)

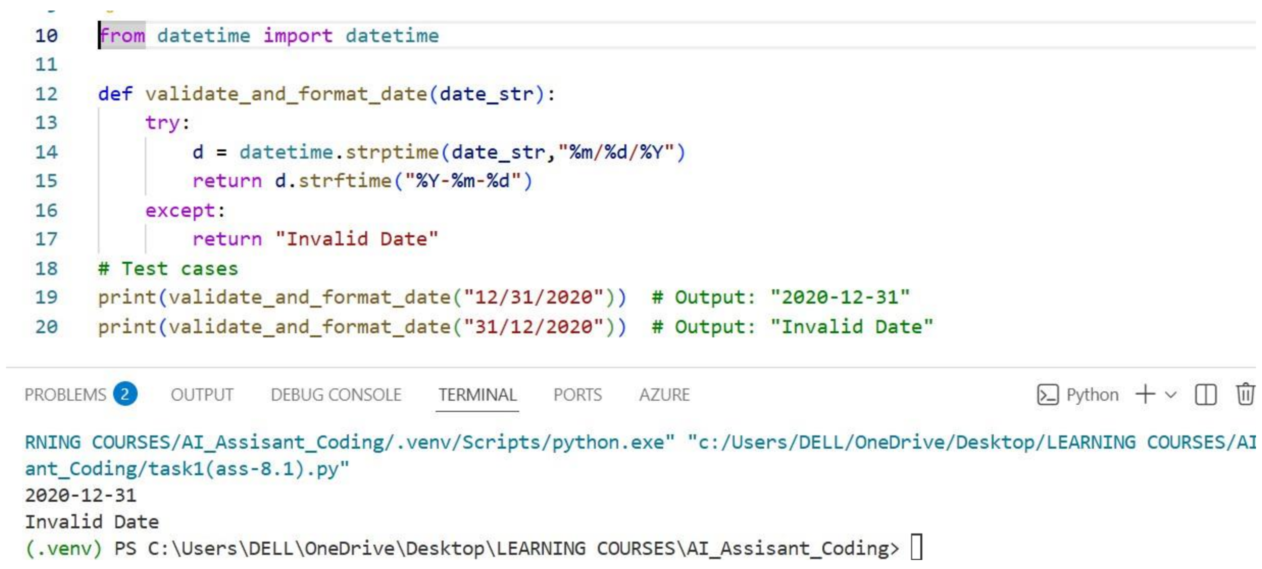
• Task: Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.

Prompt:

Generate at least 3 assert test cases for a function validate_and_format_date(date_str) that:

- Validates "MM/DD/YYYY" format
- Handles invalid dates
- Converts valid dates to "YYYY-MM-DD"
  Then implement the function so all test cases pass.

1. General Testing and Output:

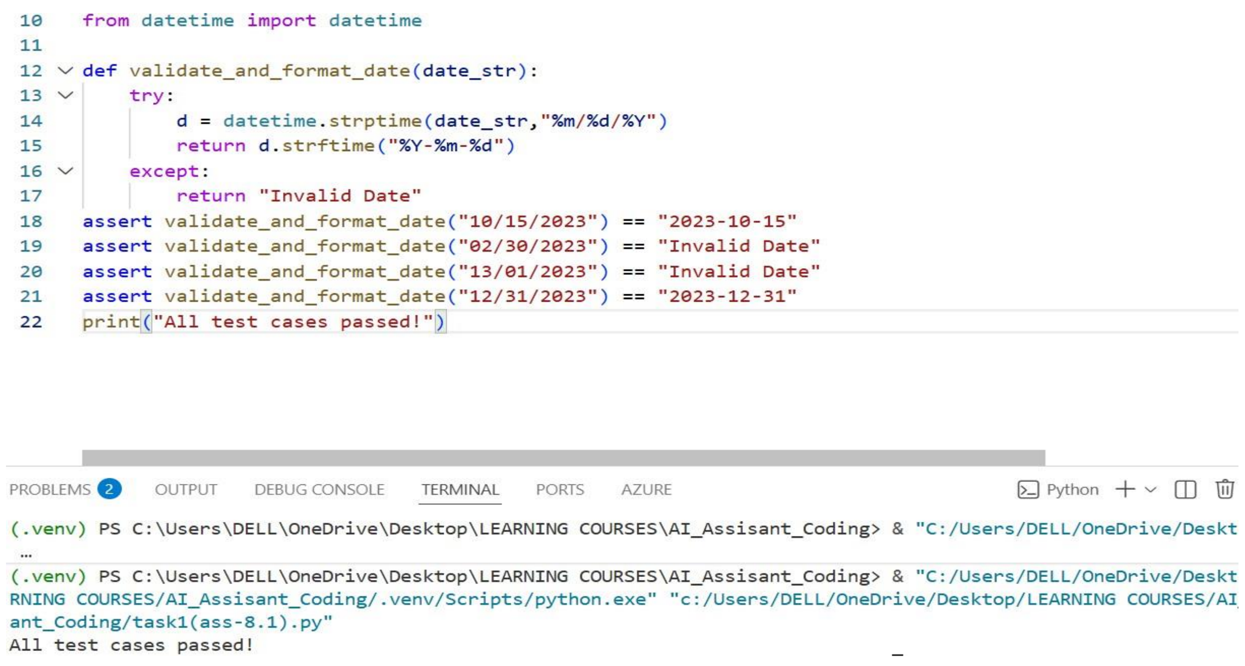


```python
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        d = datetime.strptime(date_str,"%m/%d/%Y")
        return d.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"
# Test cases
print(validate_and_format_date("12/31/2020"))  # Output: "2020-12-31"
print(validate_and_format_date("31/12/2020"))  # Output: "Invalid Date"
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

```
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI
ant_Coding/task1(ass-8.1).py"
2020-12-31
Invalid Date
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding>
```

2. Assert testing and Output:

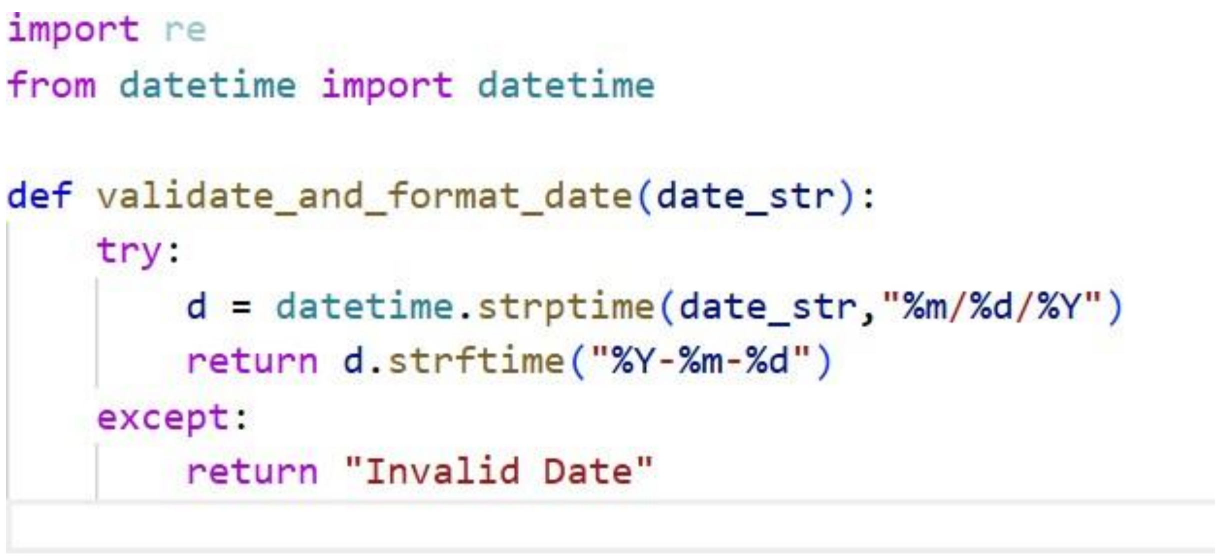


```python
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        d = datetime.strptime(date_str,"%m/%d/%Y")
        return d.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"
assert validate_and_format_date("10/15/2023") == "2023-10-15"
assert validate_and_format_date("02/30/2023") == "Invalid Date"
assert validate_and_format_date("13/01/2023") == "Invalid Date"
assert validate_and_format_date("12/31/2023") == "2023-12-31"
print("All test cases passed!")
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

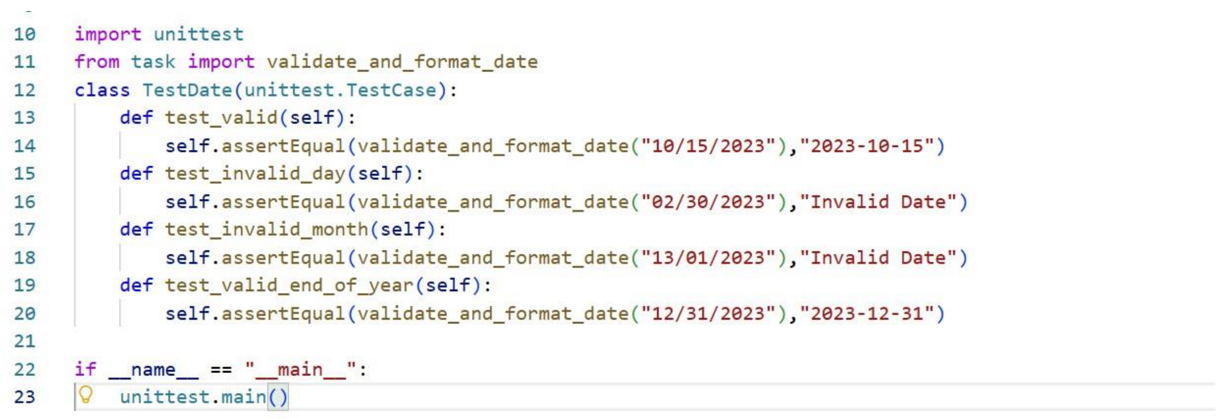
```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Deskt
…
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> & "C:/Users/DELL/OneDrive/Deskt
RNING COURSES/AI_Assisant_Coding/.venv/Scripts/python.exe" "c:/Users/DELL/OneDrive/Desktop/LEARNING COURSES/AI
ant_Coding/task1(ass-8.1).py"
All test cases passed!
```

3. Unit testing

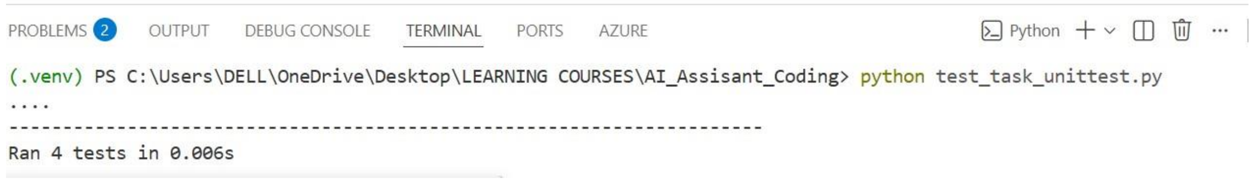


```python
import re
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        d = datetime.strptime(date_str,"%m/%d/%Y")
        return d.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"
```

Create Next python file to run Test Cases:

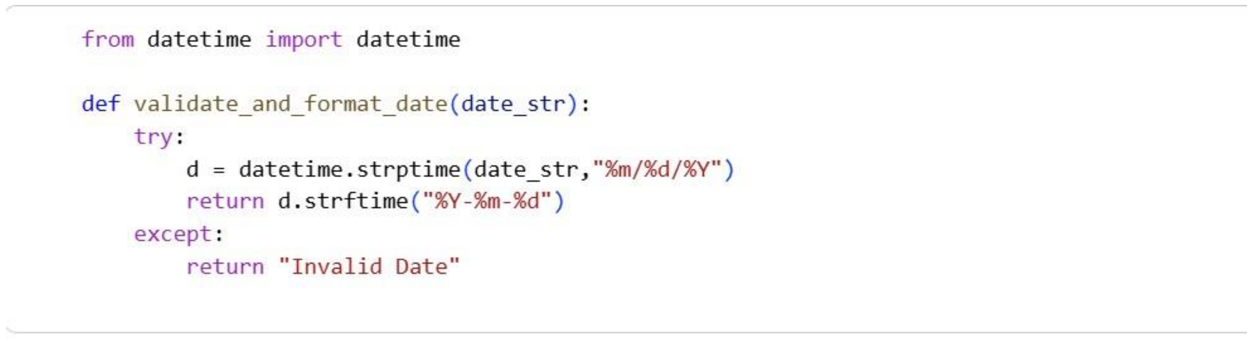


```python
import unittest
from task import validate_and_format_date
class TestDate(unittest.TestCase):
    def test_valid(self):
        self.assertEqual(validate_and_format_date("10/15/2023"),"2023-10-15")
    def test_invalid_day(self):
        self.assertEqual(validate_and_format_date("02/30/2023"),"Invalid Date")
    def test_invalid_month(self):
        self.assertEqual(validate_and_format_date("13/01/2023"),"Invalid Date")
    def test_valid_end_of_year(self):
        self.assertEqual(validate_and_format_date("12/31/2023"),"2023-12-31")

if __name__ == "__main__":
    unittest.main()
```

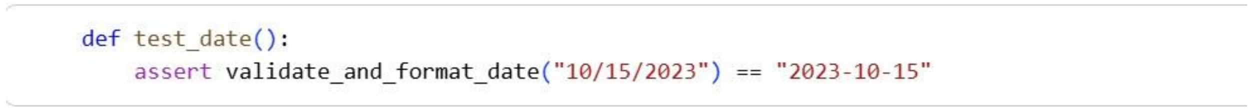


```
(.venv) PS C:\Users\DELL\OneDrive\Desktop\LEARNING COURSES\AI_Assisant_Coding> python test_task_unittest.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.006s
```

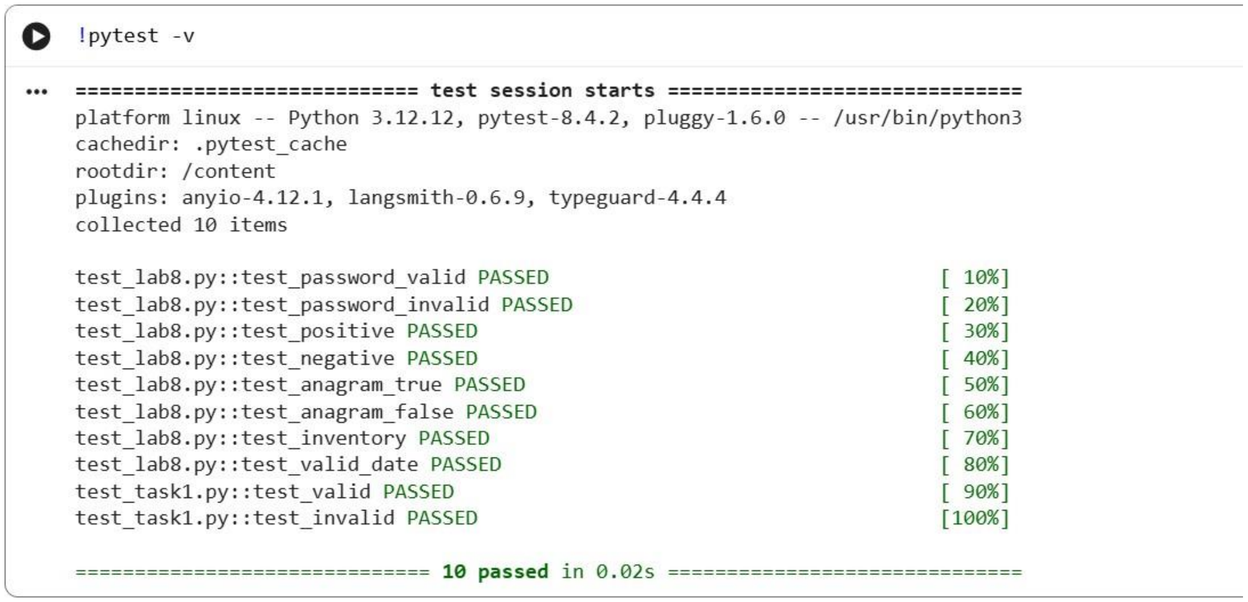
4. Pytest and Output:



```python
from datetime import datetime

def validate_and_format_date(date_str):
    try:
        d = datetime.strptime(date_str,"%m/%d/%Y")
        return d.strftime("%Y-%m-%d")
    except:
        return "Invalid Date"
```



```python
def test_date():
    assert validate_and_format_date("10/15/2023") == "2023-10-15"
```



```
!pytest -v
```

```
============================= test session starts ==============================
platform linux -- Python 3.12.12, pytest-8.4.2, pluggy-1.6.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: anyio-4.12.1, langsmith-0.6.9, typeguard-4.4.4
collected 10 items

test_lab8.py::test_password_valid PASSED                                 [ 10%]
test_lab8.py::test_password_invalid PASSED                               [ 20%]
test_lab8.py::test_positive PASSED                                       [ 30%]
test_lab8.py::test_negative PASSED                                       [ 40%]
test_lab8.py::test_anagram_true PASSED                                   [ 50%]
test_lab8.py::test_anagram_false PASSED                                  [ 60%]
test_lab8.py::test_inventory PASSED                                      [ 70%]
test_lab8.py::test_valid_date PASSED                                     [ 80%]
test_task1.py::test_valid PASSED                                         [ 90%]
test_task1.py::test_invalid PASSED                                       [100%]

============================== 10 passed in 0.02s ==============================
```

Justification:

This task focuses on **data validation**.

- AI generates invalid date cases (e.g., Feb 30).
- Ensures strict format validation.
- Demonstrates exception handling.
- Converts data into standardized format.

→ Reinforces error handling

→ Demonstrates datetime usage

→ Encourages clean data formatting