Assingment -8.1

Hallticket no:2303A510A6

Batch no:02

#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.

• Requirements:

o Password must have at least 8 characters.

o Must include uppercase, lowercase, digit, and special

character.

o Must not contain spaces.

Example Assert Test Cases:

assert is_strong_password("Abcd@123") == True

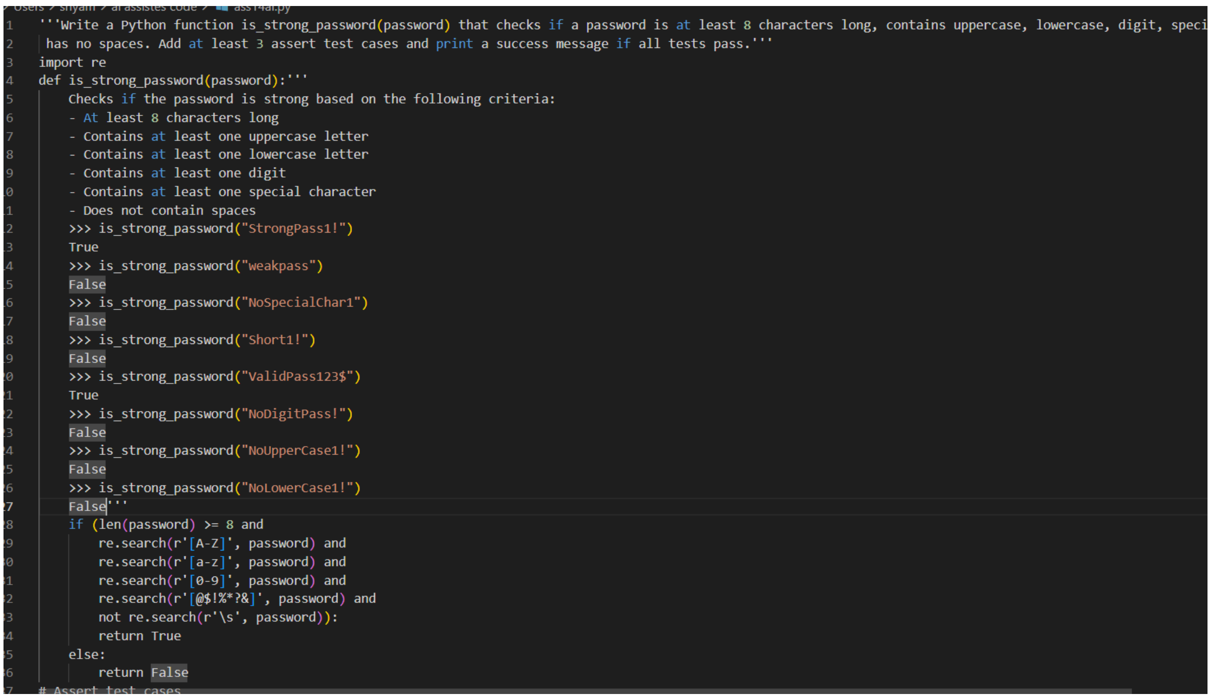
assert is_strong_password("abcd123") == False

assert is_strong_password("ABCD@1234") == True

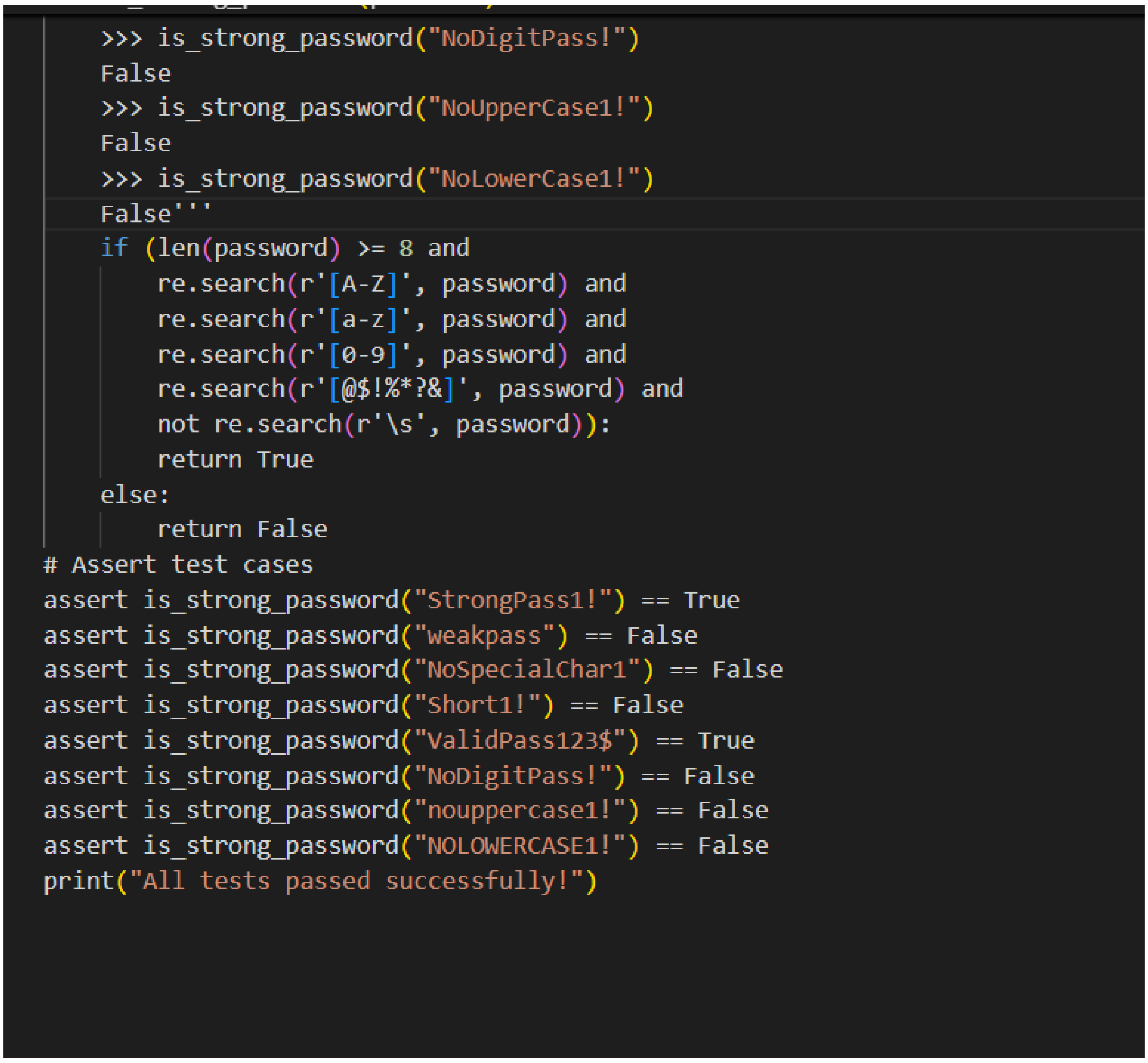
Expected Output #1:

• Password validation logic passing all AI-generated test cases.

Code:

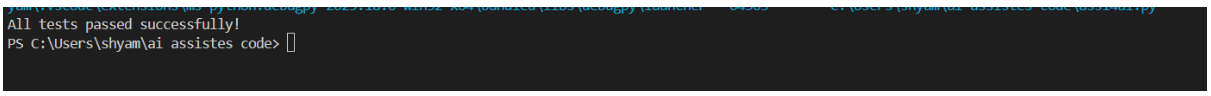


```
'''Write a Python function is_strong_password(password) that checks if a password is at least 8 characters long, contains uppercase, lowercase, digit, speci
 has no spaces. Add at least 3 assert test cases and print a success message if all tests pass.'''
import re
def is_strong_password(password):'''
    Checks if the password is strong based on the following criteria:
    - At least 8 characters long
    - Contains at least one uppercase letter
    - Contains at least one lowercase letter
    - Contains at least one digit
    - Contains at least one special character
    - Does not contain spaces
    >>> is_strong_password("StrongPass1!")
    True
    >>> is_strong_password("weakpass")
    False
    >>> is_strong_password("NoSpecialChar1")
    False
    >>> is_strong_password("Short1!")
    False
    >>> is_strong_password("ValidPass123$")
    True
    >>> is_strong_password("NoDigitPass!")
    False
    >>> is_strong_password("NoUpperCase1!")
    False
    >>> is_strong_password("NoLowerCase1!")
    False'''
    if (len(password) >= 8 and
        re.search(r'[A-Z]', password) and
        re.search(r'[a-z]', password) and
        re.search(r'[0-9]', password) and
        re.search(r'[@$!%*?&]', password) and
        not re.search(r'\s', password)):
        return True
    else:
        return False
# Assert test cases
```



```
    >>> is_strong_password("NoDigitPass!")
    False
    >>> is_strong_password("NoUpperCase1!")
    False
    >>> is_strong_password("NoLowerCase1!")
    False'''
    if (len(password) >= 8 and
        re.search(r'[A-Z]', password) and
        re.search(r'[a-z]', password) and
        re.search(r'[0-9]', password) and
        re.search(r'[@$!%*?&]', password) and
        not re.search(r'\s', password)):
        return True
    else:
        return False
# Assert test cases
assert is_strong_password("StrongPass1!") == True
assert is_strong_password("weakpass") == False
assert is_strong_password("NoSpecialChar1") == False
assert is_strong_password("Short1!") == False
assert is_strong_password("ValidPass123$") == True
assert is_strong_password("NoDigitPass!") == False
assert is_strong_password("nouppercase1!") == False
assert is_strong_password("NOLOWERCASE1!") == False
print("All tests passed successfully!")
```

Output:



```
All tests passed successfully!
PS C:\Users\shyam\ai assistes code> 
```

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.

• Requirements:

o Classify numbers as Positive, Negative, or Zero.

o Handle invalid inputs like strings and None.

o Include boundary conditions (-1, 0, 1).

Example Assert Test Cases:

assert classify_number(10) == "Positive"

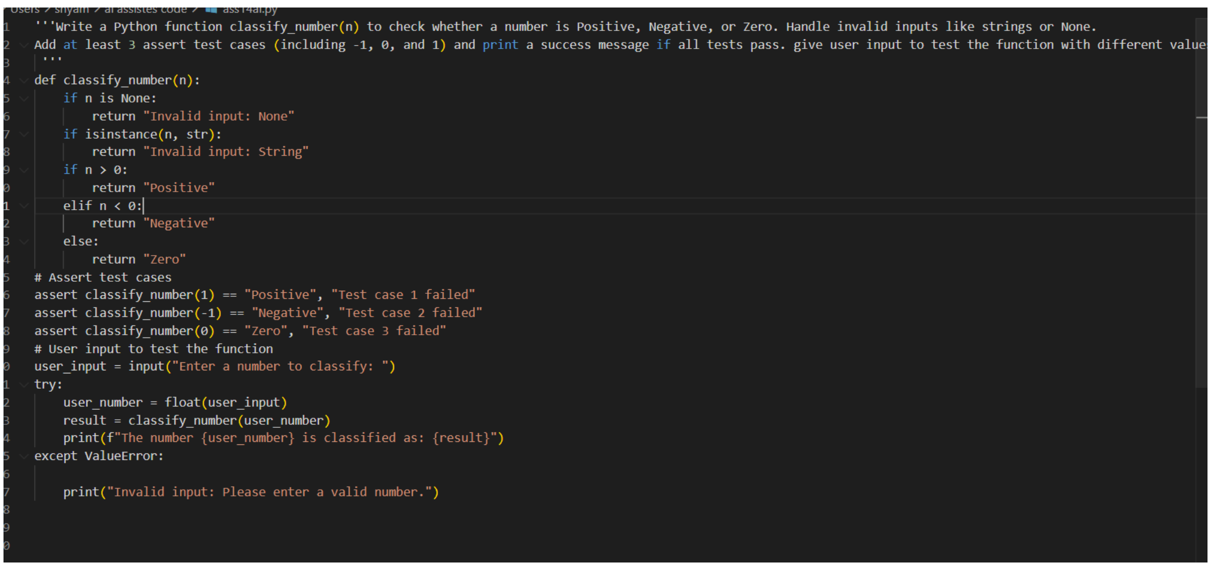
assert classify_number(-5) == "Negative"

assert classify_number(0) == "Zero"

Expected Output #2:

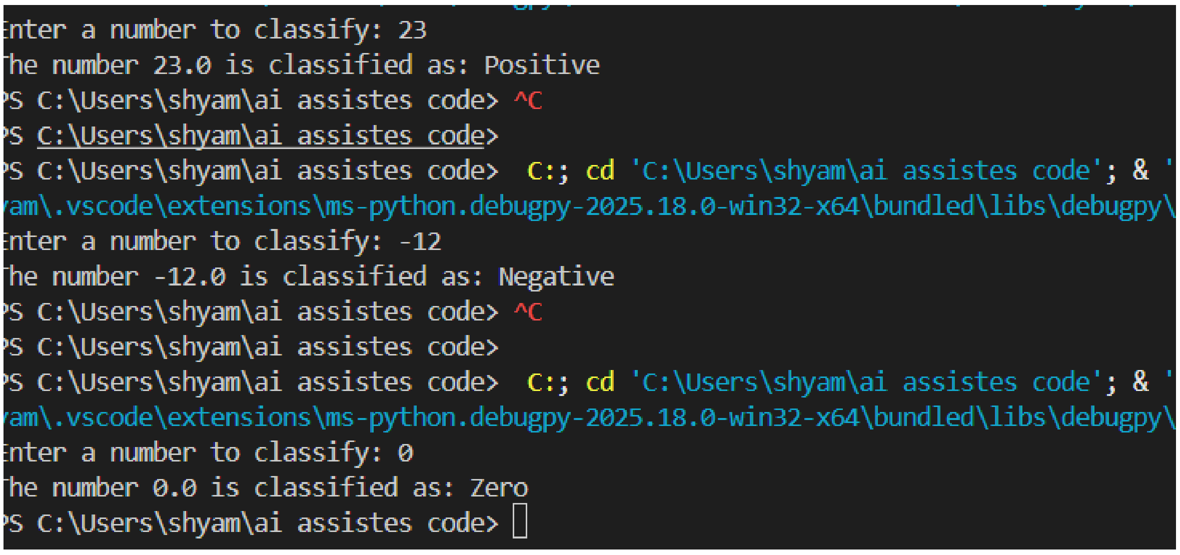
• Classification logic passing all assert tests.

Code:



```python
'''Write a Python function classify_number(n) to check whether a number is Positive, Negative, or Zero. Handle invalid inputs like strings or None.
Add at least 3 assert test cases (including -1, 0, and 1) and print a success message if all tests pass. give user input to test the function with different value
 '''
def classify_number(n):
    if n is None:
        return "Invalid input: None"
    if isinstance(n, str):
        return "Invalid input: String"
    if n > 0:
        return "Positive"
    elif n < 0:
        return "Negative"
    else:
        return "Zero"
# Assert test cases
assert classify_number(1) == "Positive", "Test case 1 failed"
assert classify_number(-1) == "Negative", "Test case 2 failed"
assert classify_number(0) == "Zero", "Test case 3 failed"
# User input to test the function
user_input = input("Enter a number to classify: ")
try:
    user_number = float(user_input)
    result = classify_number(user_number)
    print(f"The number {user_number} is classified as: {result}")
except ValueError:

    print("Invalid input: Please enter a valid number.")
```

Output:



```
nter a number to classify: 23
he number 23.0 is classified as: Positive
S C:\Users\shyam\ai assistes code> ^C
S C:\Users\shyam\ai assistes code>
S C:\Users\shyam\ai assistes code>  C:; cd 'C:\Users\shyam\ai assistes code'; & '
am\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\
nter a number to classify: -12
he number -12.0 is classified as: Negative
S C:\Users\shyam\ai assistes code> ^C
S C:\Users\shyam\ai assistes code>
S C:\Users\shyam\ai assistes code>  C:; cd 'C:\Users\shyam\ai assistes code'; & '
am\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\
nter a number to classify: 0
he number 0.0 is classified as: Zero
S C:\Users\shyam\ai assistes code>
```

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.

• Requirements:

o Ignore case, spaces, and punctuation.

o Handle edge cases (empty strings, identical words).

Example Assert Test Cases:

assert is_anagram("listen", "silent") == True

assert is_anagram("hello", "world") == False

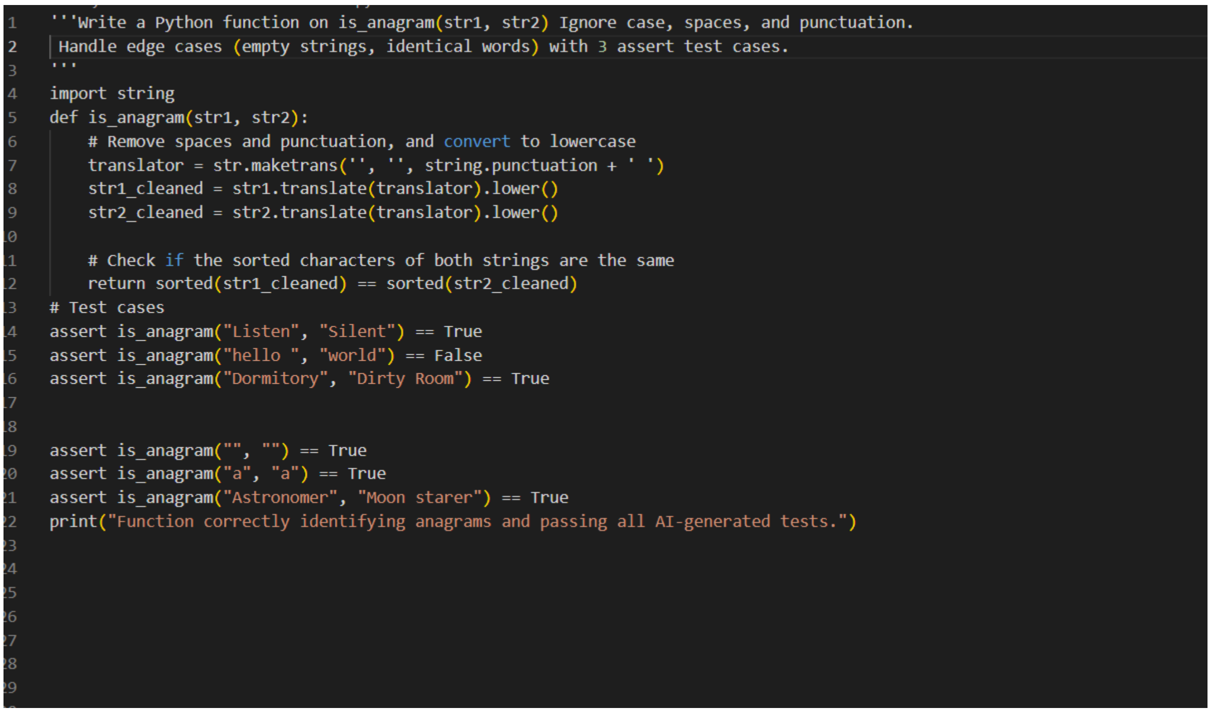
assert is_anagram("Dormitory", "Dirty Room") == True

Expected Output #3:

• Function correctly identifying anagrams and passing all AI-

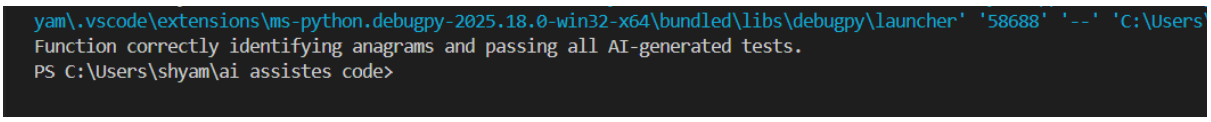
generated tests.

Code:



```python
'''Write a Python function on is_anagram(str1, str2) Ignore case, spaces, and punctuation.
 Handle edge cases (empty strings, identical words) with 3 assert test cases.
'''
import string
def is_anagram(str1, str2):
    # Remove spaces and punctuation, and convert to lowercase
    translator = str.maketrans('', '', string.punctuation + ' ')
    str1_cleaned = str1.translate(translator).lower()
    str2_cleaned = str2.translate(translator).lower()

    # Check if the sorted characters of both strings are the same
    return sorted(str1_cleaned) == sorted(str2_cleaned)
# Test cases
assert is_anagram("Listen", "Silent") == True
assert is_anagram("hello ", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True


assert is_anagram("", "") == True
assert is_anagram("a", "a") == True
assert is_anagram("Astronomer", "Moon starer") == True
print("Function correctly identifying anagrams and passing all AI-generated tests.")
```

Output:



```
yam\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58688' '--' 'C:\Users\
Function correctly identifying anagrams and passing all AI-generated tests.
PS C:\Users\shyam\ai assistes code>
```

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.

• Methods:

o add_item(name, quantity)

o remove_item(name, quantity)

o get_stock(name)

Example Assert Test Cases:

inv = Inventory()

inv.add_item("Pen", 10)

assert inv.get_stock("Pen") == 10

inv.remove_item("Pen", 5)

assert inv.get_stock("Pen") == 5

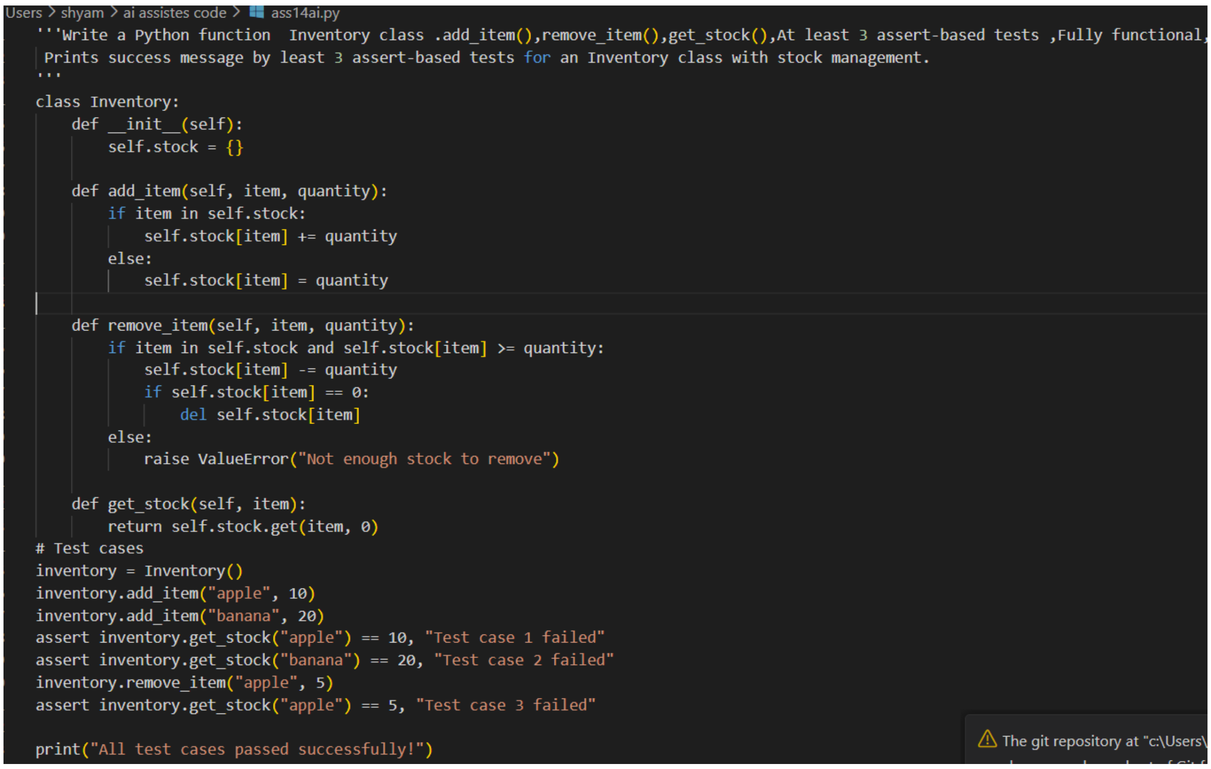
inv.add_item("Book", 3)

assert inv.get_stock("Book") == 3

Expected Output #4:

• Fully functional class passing all assertions.

Code:

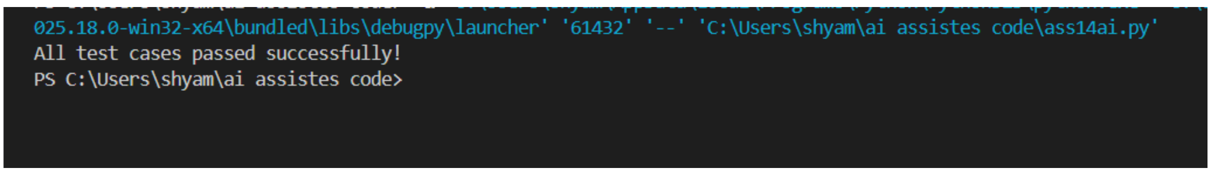


Users > shyam > ai assistes code > ass14ai.py

```python
'''Write a Python function  Inventory class .add_item(),remove_item(),get_stock(),At least 3 assert-based tests ,Fully functional,
 Prints success message by least 3 assert-based tests for an Inventory class with stock management.
'''
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
            if self.stock[item] == 0:
                del self.stock[item]
        else:
            raise ValueError("Not enough stock to remove")

    def get_stock(self, item):
        return self.stock.get(item, 0)
# Test cases
inventory = Inventory()
inventory.add_item("apple", 10)
inventory.add_item("banana", 20)
assert inventory.get_stock("apple") == 10, "Test case 1 failed"
assert inventory.get_stock("banana") == 20, "Test case 2 failed"
inventory.remove_item("apple", 5)
assert inventory.get_stock("apple") == 5, "Test case 3 failed"

print("All test cases passed successfully!")
```

The git repository at "c:\Users\

Output:



```
025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61432' '--' 'C:\Users\shyam\ai assistes code\ass14ai.py'
All test cases passed successfully!
PS C:\Users\shyam\ai assistes code>
```

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.

• Requirements:

o Validate "MM/DD/YYYY" format.

o Handle invalid dates.

o Convert valid dates to "YYYY-MM-DD".

Example Assert Test Cases:

assert validate_and_format_date("10/15/2023") == "2023-10-15"

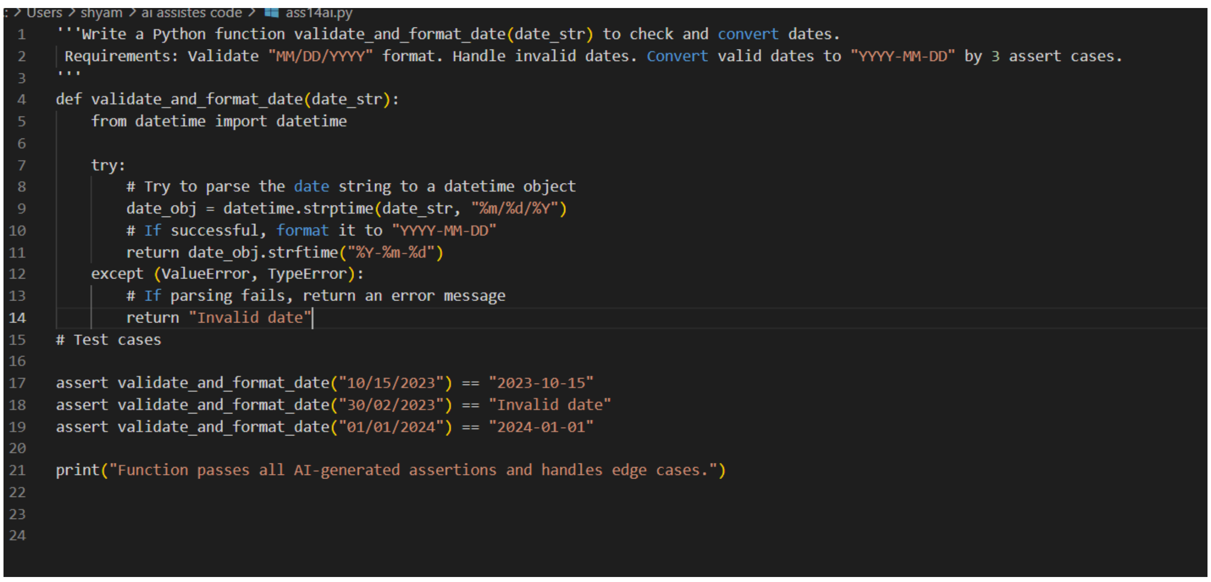
assert validate_and_format_date("02/30/2023") == "Invalid Date"

assert validate_and_format_date("01/01/2024") == "2024-01-01"

Expected Output #5:

• Function passes all AI-generated assertions and handles edge Cases

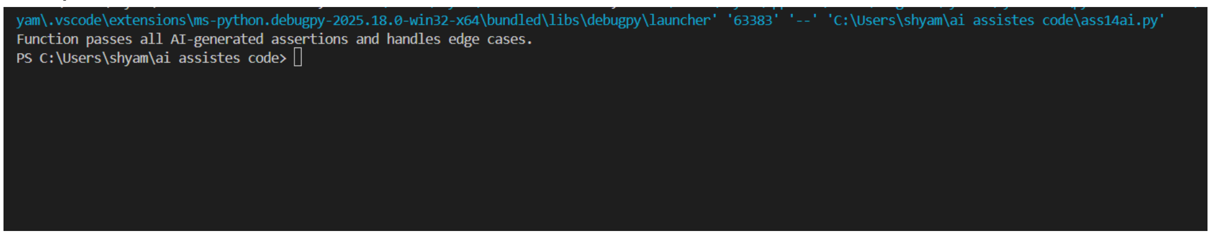
Code:



: > Users > shyam > ai assistes code > ass14ai.py

```python
'''Write a Python function validate_and_format_date(date_str) to check and convert dates.
 Requirements: Validate "MM/DD/YYYY" format. Handle invalid dates. Convert valid dates to "YYYY-MM-DD" by 3 assert cases.
'''
def validate_and_format_date(date_str):
    from datetime import datetime

    try:
        # Try to parse the date string to a datetime object
        date_obj = datetime.strptime(date_str, "%m/%d/%Y")
        # If successful, format it to "YYYY-MM-DD"
        return date_obj.strftime("%Y-%m-%d")
    except (ValueError, TypeError):
        # If parsing fails, return an error message
        return "Invalid date"
# Test cases

assert validate_and_format_date("10/15/2023") == "2023-10-15"
assert validate_and_format_date("30/02/2023") == "Invalid date"
assert validate_and_format_date("01/01/2024") == "2024-01-01"

print("Function passes all AI-generated assertions and handles edge cases.")
```

Output:



```
yam\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63383' '--' 'C:\Users\shyam\ai assistes code\ass14ai.py'
Function passes all AI-generated assertions and handles edge cases.
PS C:\Users\shyam\ai assistes code> 
```