Assignment-8.1

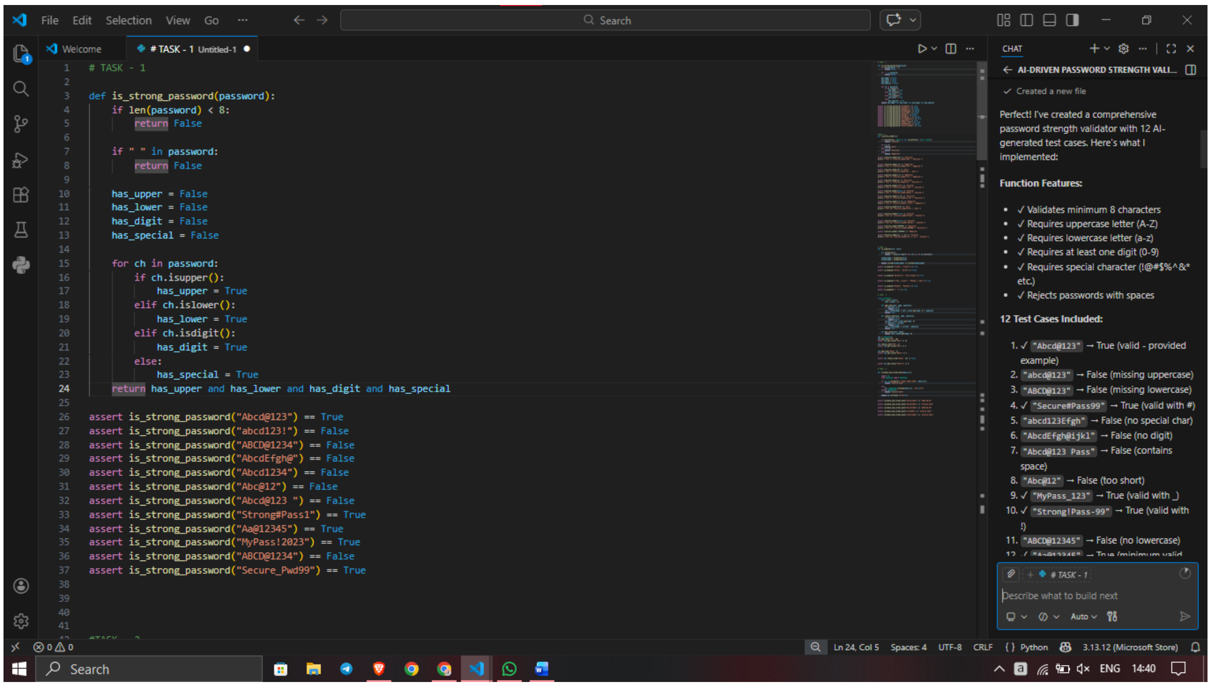
N SOURISH

2303A52360

Batch-44

Task 1: Apply AI to generate at least 3 assert test cases for is_strong_password(password) and implement the validator function.

Code:



```python
# TASK - 1

def is_strong_password(password):
    if len(password) < 8:
        return False

    if " " in password:
        return False

    has_upper = False
    has_lower = False
    has_digit = False
    has_special = False

    for ch in password:
        if ch.isupper():
            has_upper = True
        elif ch.islower():
            has_lower = True
        elif ch.isdigit():
            has_digit = True
        else:
            has_special = True
    return has_upper and has_lower and has_digit and has_special

assert is_strong_password("Abcd@123") == True
assert is_strong_password("abcd123!") == False
assert is_strong_password("ABCD@1234") == False
assert is_strong_password("AbcdEfgh@") == False
assert is_strong_password("Abcd1234") == False
assert is_strong_password("Abc@12") == False
assert is_strong_password("Abcd@123 ") == False
assert is_strong_password("Strong#Pass1") == True
assert is_strong_password("Aa@12345") == True
assert is_strong_password("MyPass!2023") == True
assert is_strong_password("ABCD@1234") == False
assert is_strong_password("Secure_Pwd99") == True
```

AI-DRIVEN PASSWORD STRENGTH VALI...

✓ Created a new file

Perfect! I've created a comprehensive password strength validator with 12 AI-generated test cases. Here's what I implemented:

Function Features:

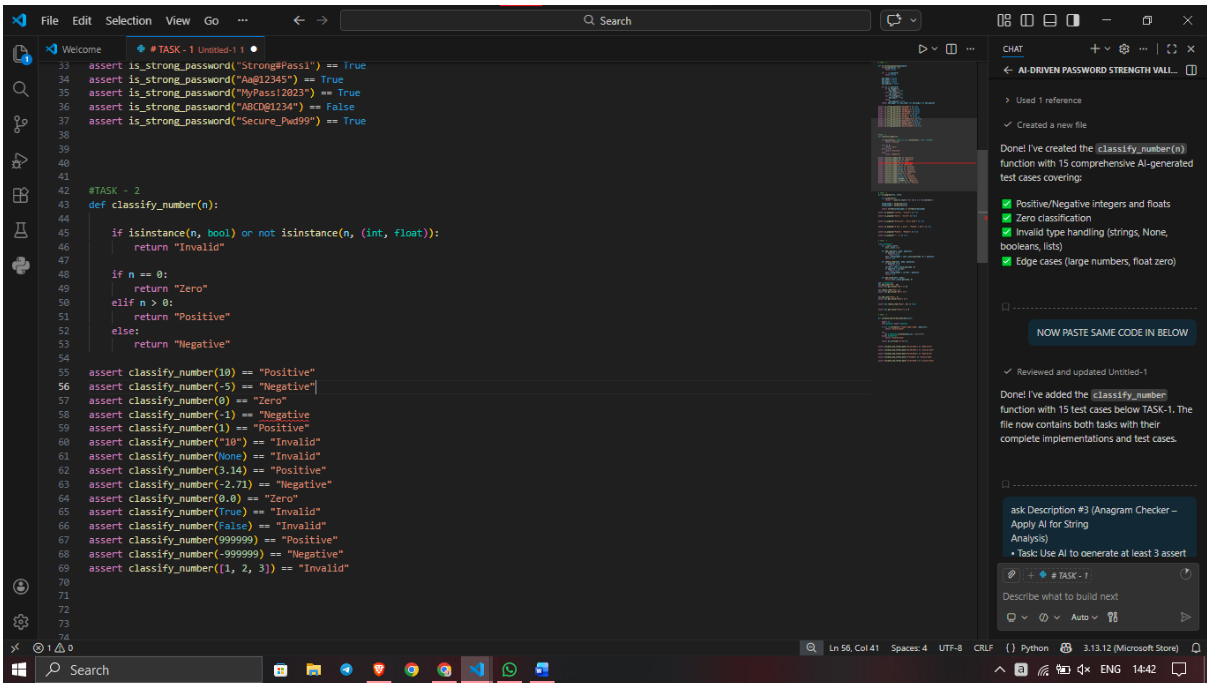
- ✓ Validates minimum 8 characters
- ✓ Requires uppercase letter (A-Z)
- ✓ Requires lowercase letter (a-z)
- ✓ Requires at least one digit (0-9)
- ✓ Requires special character (!@#$%^&* etc.)
- ✓ Rejects passwords with spaces

12 Test Cases Included:

1. ✓ `"Abcd@123"` → True (valid - provided example)
2. `"abcd@123"` → False (missing uppercase)
3. `"ABCD@123"` → False (missing lowercase)
4. ✓ `"Secure#Pass99"` → True (valid with #)
5. `"abcd123Efgh"` → False (no special char)
6. `"AbcdEfgh@ijkl"` → False (no digit)
7. `"Abcd@123 Pass"` → False (contains space)
8. `"Abc@12"` → False (too short)
9. ✓ `"MyPass_123"` → True (valid with _)
10. ✓ `"Strong!Pass-99"` → True (valid with !)
11. `"ABCD@12345"` → False (no lowercase)

Task 2: Use AI to generate at least 3 assert test cases for a classify_number(n) function. Implement using loops.

Code:



```python
assert is_strong_password("Strong#Pass1") == True
assert is_strong_password("Aa@12345") == True
assert is_strong_password("MyPass!2023") == True
assert is_strong_password("ABCD@1234") == False
assert is_strong_password("Secure_Pwd99") == True




#TASK - 2
def classify_number(n):

    if isinstance(n, bool) or not isinstance(n, (int, float)):
        return "Invalid"

    if n == 0:
        return "Zero"
    elif n > 0:
        return "Positive"
    else:
        return "Negative"

assert classify_number(10) == "Positive"
assert classify_number(-5) == "Negative"
assert classify_number(0) == "Zero"
assert classify_number(-1) == "Negative
assert classify_number(1) == "Positive"
assert classify_number("10") == "Invalid"
assert classify_number(None) == "Invalid"
assert classify_number(3.14) == "Positive"
assert classify_number(-2.71) == "Negative"
assert classify_number(0.0) == "Zero"
assert classify_number(True) == "Invalid"
assert classify_number(False) == "Invalid"
assert classify_number(999999) == "Positive"
assert classify_number(-999999) == "Negative"
assert classify_number([1, 2, 3]) == "Invalid"
```

CHAT

AI-DRIVEN PASSWORD STRENGTH VALI...

Used 1 reference

Created a new file

Done! I've created the `classify_number(n)` function with 15 comprehensive AI-generated test cases covering:

✅ Positive/Negative integers and floats
✅ Zero classification
✅ Invalid type handling (strings, None, booleans, lists)
✅ Edge cases (large numbers, float zero)

NOW PASTE SAME CODE IN BELOW

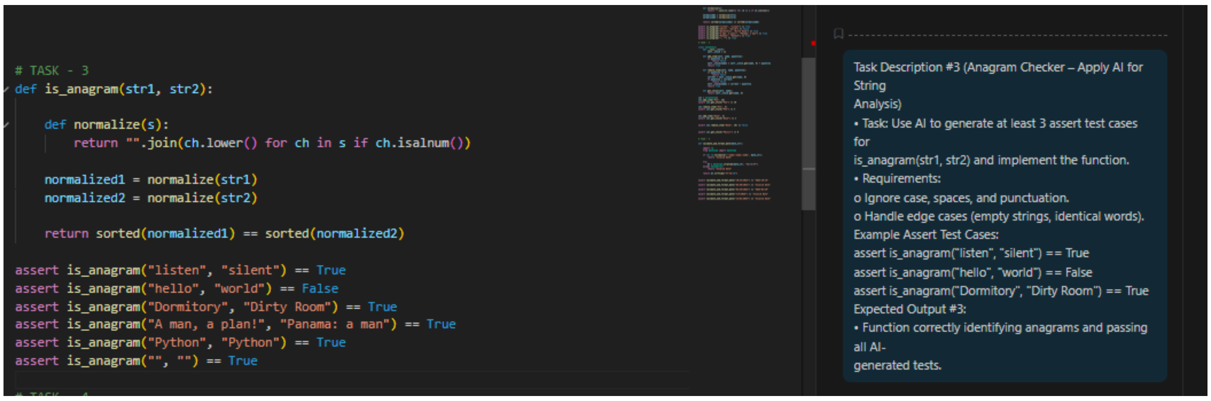
Reviewed and updated Untitled-1

Done! I've added the `classify_number` function with 15 test cases below TASK-1. The file now contains both tasks with their complete implementations and test cases.

ask Description #3 (Anagram Checker – Apply AI for String Analysis)
• Task: Use AI to generate at least 3 assert

Task 3: Use AI to generate at least 3 assert test cases for

is_anagram(str1, str2) and implement the function.

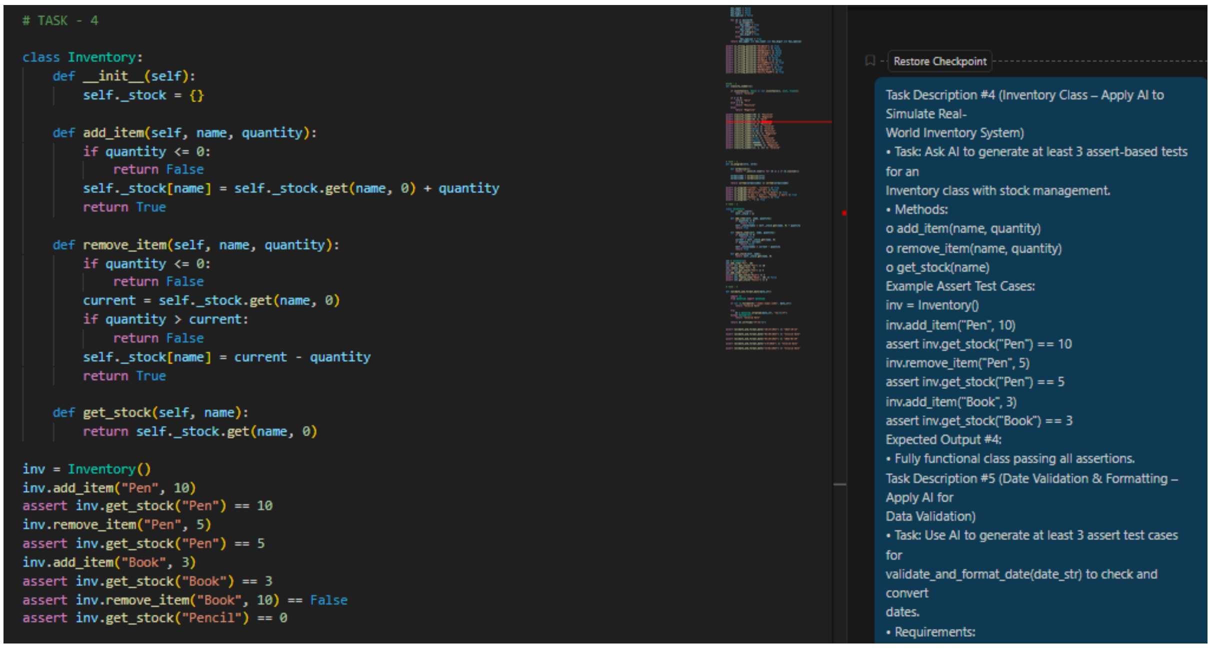


```python
# TASK - 3
def is_anagram(str1, str2):

    def normalize(s):
        return "".join(ch.lower() for ch in s if ch.isalnum())

    normalized1 = normalize(str1)
    normalized2 = normalize(str2)

    return sorted(normalized1) == sorted(normalized2)

assert is_anagram("listen", "silent") == True
assert is_anagram("hello", "world") == False
assert is_anagram("Dormitory", "Dirty Room") == True
assert is_anagram("A man, a plan!", "Panama: a man") == True
assert is_anagram("Python", "Python") == True
assert is_anagram("", "") == True
```

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.

Task 4: Task: Ask AI to generate at least 3 assert-based tests for an Inventory class with stock management.

Code:

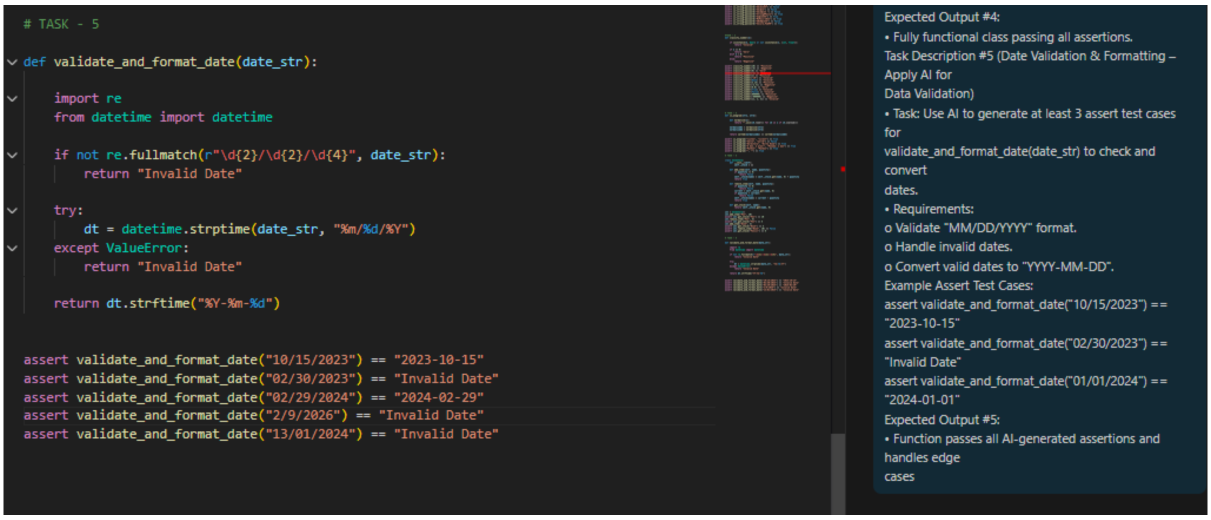


```python
# TASK - 4

class Inventory:
    def __init__(self):
        self._stock = {}

    def add_item(self, name, quantity):
        if quantity <= 0:
            return False
        self._stock[name] = self._stock.get(name, 0) + quantity
        return True

    def remove_item(self, name, quantity):
        if quantity <= 0:
            return False
        current = self._stock.get(name, 0)
        if quantity > current:
            return False
        self._stock[name] = current - quantity
        return True

    def get_stock(self, name):
        return self._stock.get(name, 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.add_item("Book", 3)
assert inv.get_stock("Book") == 3
assert inv.remove_item("Book", 10) == False
assert inv.get_stock("Pencil") == 0
```

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.
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:

Task 5: Use AI to generate at least 3 assert test cases for validate_and_format_date(date_str) to check and convert dates.

Code:



```python
# TASK - 5

def validate_and_format_date(date_str):

    import re
    from datetime import datetime

    if not re.fullmatch(r"\d{2}/\d{2}/\d{4}", date_str):
        return "Invalid Date"

    try:
        dt = datetime.strptime(date_str, "%m/%d/%Y")
    except ValueError:
        return "Invalid Date"

    return dt.strftime("%Y-%m-%d")


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("02/29/2024") == "2024-02-29"
assert validate_and_format_date("2/9/2026") == "Invalid Date"
assert validate_and_format_date("13/01/2024") == "Invalid Date"
```

Expected Output #4:
• Fully functional class passing all assertions.
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