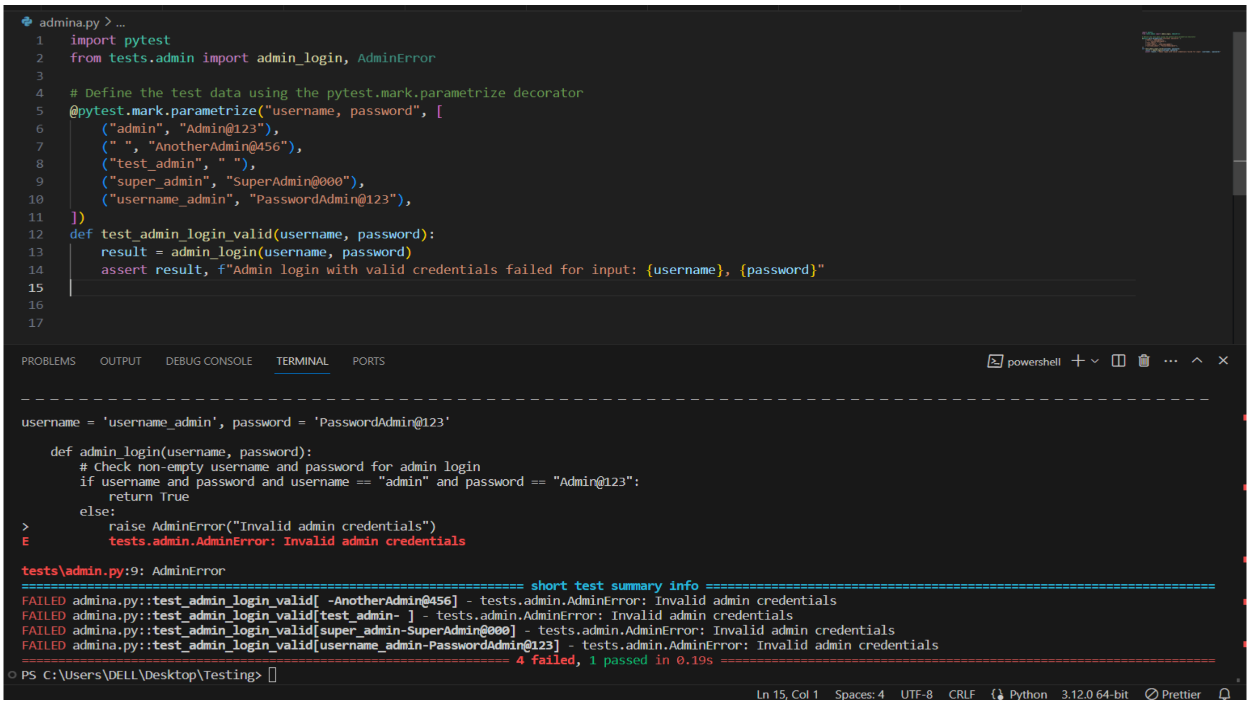
Unit Testing

Admin Side:
Admin Login:

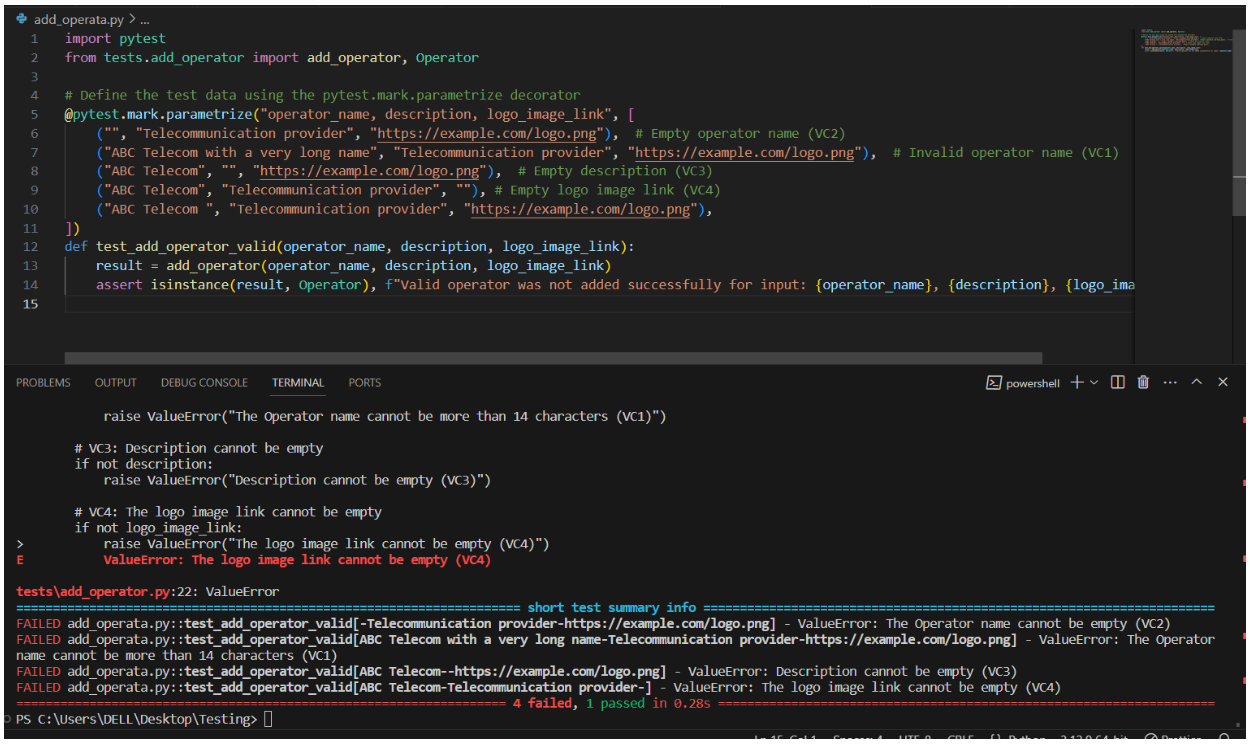


admina.py > ...

```python
import pytest
from tests.admin import admin_login, AdminError

# Define the test data using the pytest.mark.parametrize decorator
@pytest.mark.parametrize("username, password", [
    ("admin", "Admin@123"),
    (" ", "AnotherAdmin@456"),
    ("test_admin", " "),
    ("super_admin", "SuperAdmin@000"),
    ("username_admin", "PasswordAdmin@123"),
])
def test_admin_login_valid(username, password):
    result = admin_login(username, password)
    assert result, f"Admin login with valid credentials failed for input: {username}, {password}"
```

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

username = 'username_admin', password = 'PasswordAdmin@123'

    def admin_login(username, password):
        # Check non-empty username and password for admin login
        if username and password and username == "admin" and password == "Admin@123":
            return True
        else:
>           raise AdminError("Invalid admin credentials")
E           tests.admin.AdminError: Invalid admin credentials

tests\admin.py:9: AdminError
==================================================================== short test summary info ====================================================================
FAILED admina.py::test_admin_login_valid[ -AnotherAdmin@456] - tests.admin.AdminError: Invalid admin credentials
FAILED admina.py::test_admin_login_valid[test_admin- ] - tests.admin.AdminError: Invalid admin credentials
FAILED admina.py::test_admin_login_valid[super_admin-SuperAdmin@000] - tests.admin.AdminError: Invalid admin credentials
FAILED admina.py::test_admin_login_valid[username_admin-PasswordAdmin@123] - tests.admin.AdminError: Invalid admin credentials
=================================================================== 4 failed, 1 passed in 0.19s ===================================================================
PS C:\Users\DELL\Desktop\Testing>
```

Add Operator:

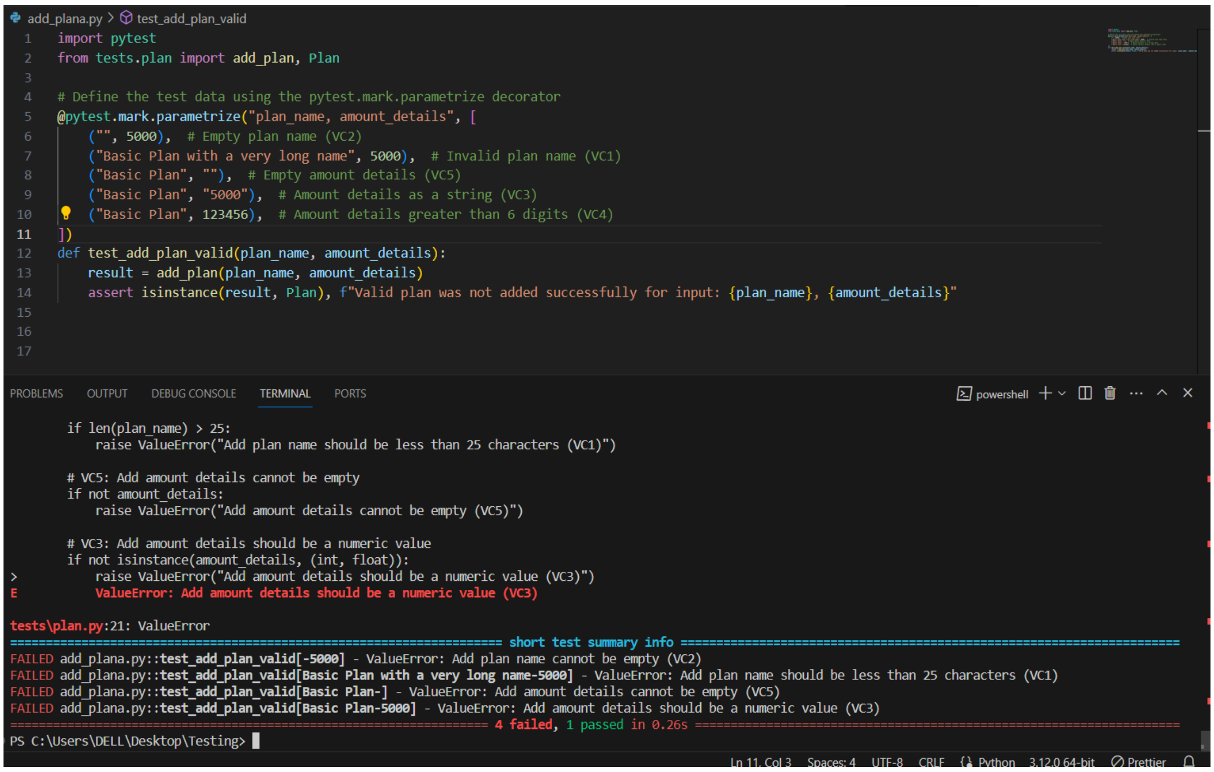


add_operata.py > ...

```python
import pytest
from tests.add_operator import add_operator, Operator

# Define the test data using the pytest.mark.parametrize decorator
@pytest.mark.parametrize("operator_name, description, logo_image_link", [
    ("", "Telecommunication provider", "https://example.com/logo.png"),  # Empty operator name (VC2)
    ("ABC Telecom with a very long name", "Telecommunication provider", "https://example.com/logo.png"),  # Invalid operator name (VC1)
    ("ABC Telecom", "", "https://example.com/logo.png"),  # Empty description (VC3)
    ("ABC Telecom", "Telecommunication provider", ""), # Empty logo image link (VC4)
    ("ABC Telecom ", "Telecommunication provider", "https://example.com/logo.png"),
])
def test_add_operator_valid(operator_name, description, logo_image_link):
    result = add_operator(operator_name, description, logo_image_link)
    assert isinstance(result, Operator), f"Valid operator was not added successfully for input: {operator_name}, {description}, {logo_ima
```

```
            raise ValueError("The Operator name cannot be more than 14 characters (VC1)")

        # VC3: Description cannot be empty
        if not description:
            raise ValueError("Description cannot be empty (VC3)")

        # VC4: The logo image link cannot be empty
        if not logo_image_link:
>           raise ValueError("The logo image link cannot be empty (VC4)")
E           ValueError: The logo image link cannot be empty (VC4)

tests\add_operator.py:22: ValueError
==================================================================== short test summary info ====================================================================
FAILED add_operata.py::test_add_operator_valid[-Telecommunication provider-https://example.com/logo.png] - ValueError: The Operator name cannot be empty (VC2)
FAILED add_operata.py::test_add_operator_valid[ABC Telecom with a very long name-Telecommunication provider-https://example.com/logo.png] - ValueError: The Operator
name cannot be more than 14 characters (VC1)
FAILED add_operata.py::test_add_operator_valid[ABC Telecom--https://example.com/logo.png] - ValueError: Description cannot be empty (VC3)
FAILED add_operata.py::test_add_operator_valid[ABC Telecom-Telecommunication provider-] - ValueError: The logo image link cannot be empty (VC4)
=================================================================== 4 failed, 1 passed in 0.28s ===================================================================
PS C:\Users\DELL\Desktop\Testing>
```

Add Plans:



add_plana.py > test_add_plan_valid

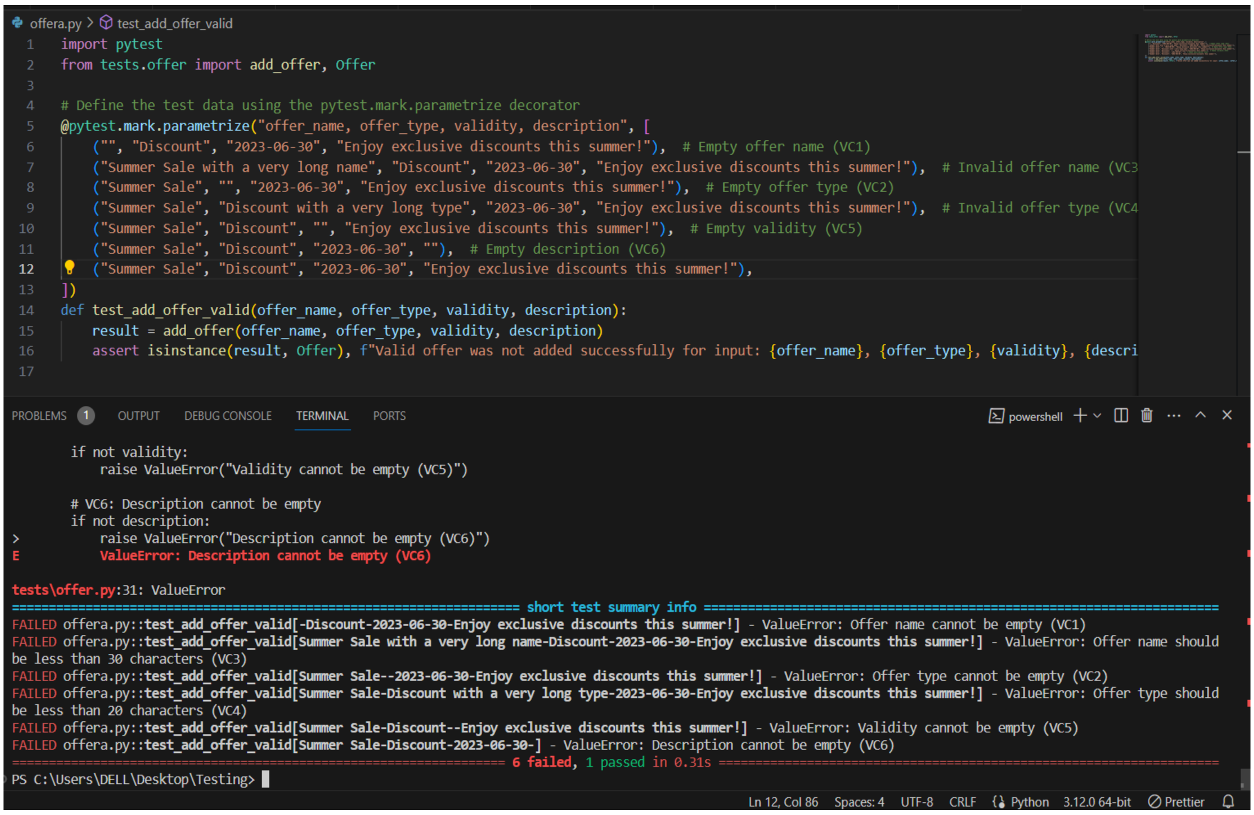
```python
import pytest
from tests.plan import add_plan, Plan

# Define the test data using the pytest.mark.parametrize decorator
@pytest.mark.parametrize("plan_name, amount_details", [
    ("", 5000),  # Empty plan name (VC2)
    ("Basic Plan with a very long name", 5000),  # Invalid plan name (VC1)
    ("Basic Plan", ""),  # Empty amount details (VC5)
    ("Basic Plan", "5000"),  # Amount details as a string (VC3)
    ("Basic Plan", 123456),  # Amount details greater than 6 digits (VC4)
])
def test_add_plan_valid(plan_name, amount_details):
    result = add_plan(plan_name, amount_details)
    assert isinstance(result, Plan), f"Valid plan was not added successfully for input: {plan_name}, {amount_details}"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
    if len(plan_name) > 25:
        raise ValueError("Add plan name should be less than 25 characters (VC1)")

    # VC5: Add amount details cannot be empty
    if not amount_details:
        raise ValueError("Add amount details cannot be empty (VC5)")

    # VC3: Add amount details should be a numeric value
    if not isinstance(amount_details, (int, float)):
>       raise ValueError("Add amount details should be a numeric value (VC3)")
E       ValueError: Add amount details should be a numeric value (VC3)

tests\plan.py:21: ValueError
================================================================ short test summary info ================================================================
FAILED add_plana.py::test_add_plan_valid[-5000] - ValueError: Add plan name cannot be empty (VC2)
FAILED add_plana.py::test_add_plan_valid[Basic Plan with a very long name-5000] - ValueError: Add plan name should be less than 25 characters (VC1)
FAILED add_plana.py::test_add_plan_valid[Basic Plan-] - ValueError: Add amount details cannot be empty (VC5)
FAILED add_plana.py::test_add_plan_valid[Basic Plan-5000] - ValueError: Add amount details should be a numeric value (VC3)
=============================================================== 4 failed, 1 passed in 0.26s ===============================================================
PS C:\Users\DELL\Desktop\Testing>
```

Ln 11, Col 3 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Prettier

Add Offers:



offera.py > test_add_offer_valid

```python
import pytest
from tests.offer import add_offer, Offer

# Define the test data using the pytest.mark.parametrize decorator
@pytest.mark.parametrize("offer_name, offer_type, validity, description", [
    ("", "Discount", "2023-06-30", "Enjoy exclusive discounts this summer!"),  # Empty offer name (VC1)
    ("Summer Sale with a very long name", "Discount", "2023-06-30", "Enjoy exclusive discounts this summer!"),  # Invalid offer name (VC3
    ("Summer Sale", "", "2023-06-30", "Enjoy exclusive discounts this summer!"),  # Empty offer type (VC2)
    ("Summer Sale", "Discount with a very long type", "2023-06-30", "Enjoy exclusive discounts this summer!"),  # Invalid offer type (VC4
    ("Summer Sale", "Discount", "", "Enjoy exclusive discounts this summer!"),  # Empty validity (VC5)
    ("Summer Sale", "Discount", "2023-06-30", ""),  # Empty description (VC6)
    ("Summer Sale", "Discount", "2023-06-30", "Enjoy exclusive discounts this summer!"),
])
def test_add_offer_valid(offer_name, offer_type, validity, description):
    result = add_offer(offer_name, offer_type, validity, description)
    assert isinstance(result, Offer), f"Valid offer was not added successfully for input: {offer_name}, {offer_type}, {validity}, {descri
```

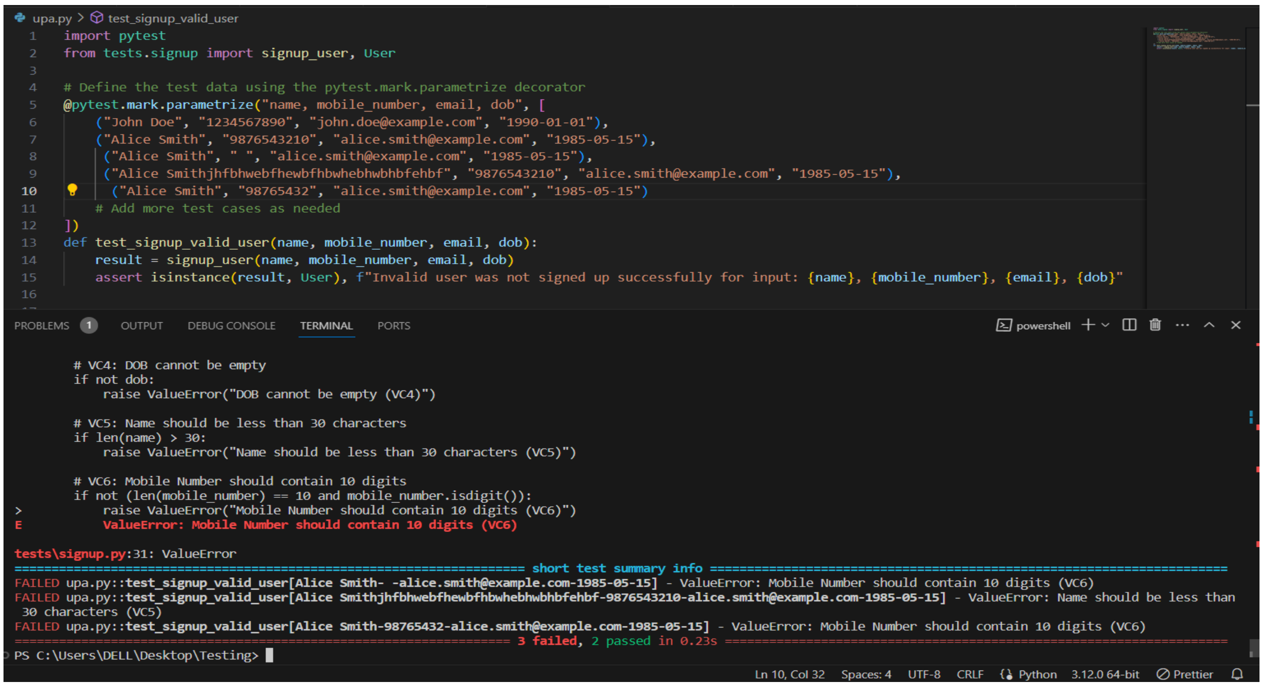
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
    if not validity:
        raise ValueError("Validity cannot be empty (VC5)")

    # VC6: Description cannot be empty
    if not description:
>       raise ValueError("Description cannot be empty (VC6)")
E       ValueError: Description cannot be empty (VC6)

tests\offer.py:31: ValueError
================================================================ short test summary info ================================================================
FAILED offera.py::test_add_offer_valid[-Discount-2023-06-30-Enjoy exclusive discounts this summer!] - ValueError: Offer name cannot be empty (VC1)
FAILED offera.py::test_add_offer_valid[Summer Sale with a very long name-Discount-2023-06-30-Enjoy exclusive discounts this summer!] - ValueError: Offer name should
be less than 30 characters (VC3)
FAILED offera.py::test_add_offer_valid[Summer Sale--2023-06-30-Enjoy exclusive discounts this summer!] - ValueError: Offer type cannot be empty (VC2)
FAILED offera.py::test_add_offer_valid[Summer Sale-Discount with a very long type-2023-06-30-Enjoy exclusive discounts this summer!] - ValueError: Offer type should
be less than 20 characters (VC4)
FAILED offera.py::test_add_offer_valid[Summer Sale-Discount--Enjoy exclusive discounts this summer!] - ValueError: Validity cannot be empty (VC5)
FAILED offera.py::test_add_offer_valid[Summer Sale-Discount-2023-06-30-] - ValueError: Description cannot be empty (VC6)
=============================================================== 6 failed, 1 passed in 0.31s ===============================================================
PS C:\Users\DELL\Desktop\Testing>
```

Ln 12, Col 86 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Prettier

User Side:

Sign Up:

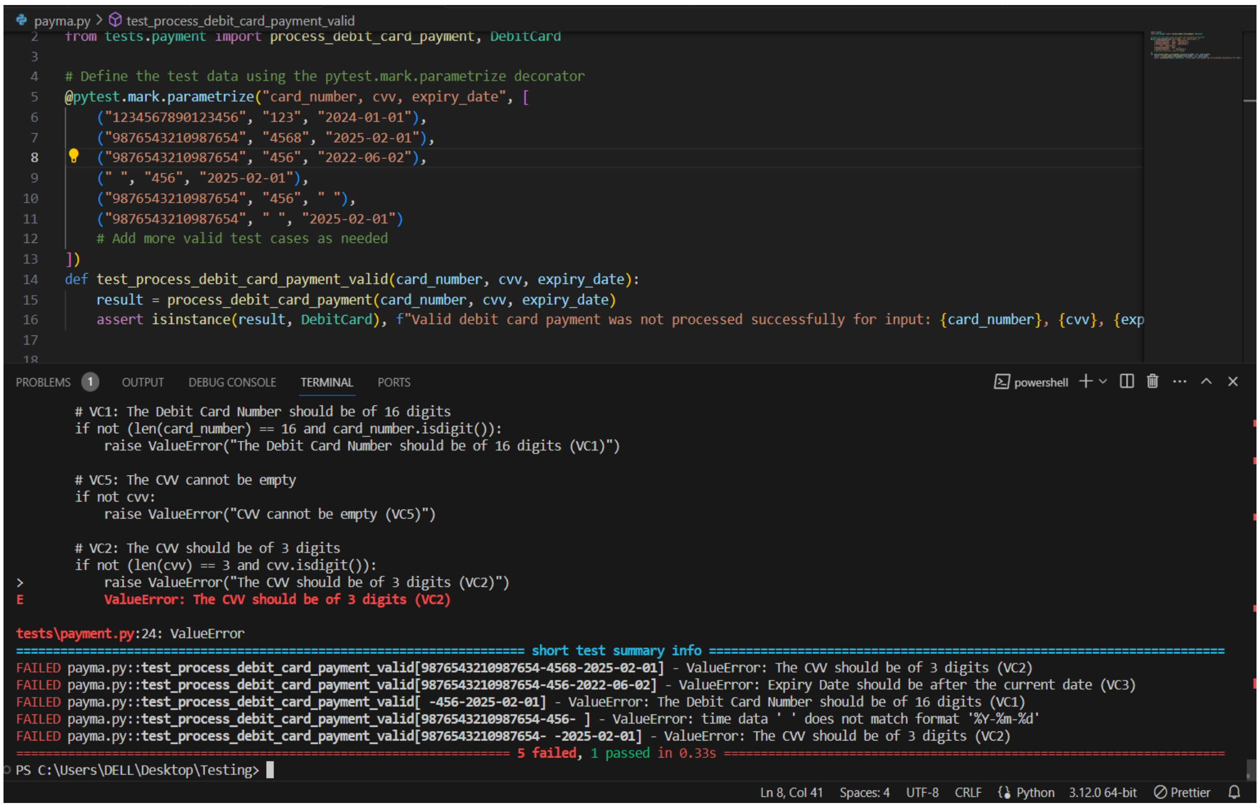


```python
import pytest
from tests.signup import signup_user, User

# Define the test data using the pytest.mark.parametrize decorator
@pytest.mark.parametrize("name, mobile_number, email, dob", [
    ("John Doe", "1234567890", "john.doe@example.com", "1990-01-01"),
    ("Alice Smith", "9876543210", "alice.smith@example.com", "1985-05-15"),
     ("Alice Smith", " ", "alice.smith@example.com", "1985-05-15"),
     ("Alice Smithjhfbhwebfhewbfhbwhebhwbhbfehbf", "9876543210", "alice.smith@example.com", "1985-05-15"),
      ("Alice Smith", "98765432", "alice.smith@example.com", "1985-05-15")
    # Add more test cases as needed
])
def test_signup_valid_user(name, mobile_number, email, dob):
    result = signup_user(name, mobile_number, email, dob)
    assert isinstance(result, User), f"Invalid user was not signed up successfully for input: {name}, {mobile_number}, {email}, {dob}"
```

```
        # VC4: DOB cannot be empty
        if not dob:
            raise ValueError("DOB cannot be empty (VC4)")

        # VC5: Name should be less than 30 characters
        if len(name) > 30:
            raise ValueError("Name should be less than 30 characters (VC5)")

        # VC6: Mobile Number should contain 10 digits
        if not (len(mobile_number) == 10 and mobile_number.isdigit()):
>           raise ValueError("Mobile Number should contain 10 digits (VC6)")
E           ValueError: Mobile Number should contain 10 digits (VC6)

tests\signup.py:31: ValueError
======================================================================= short test summary info ========================================================================
FAILED upa.py::test_signup_valid_user[Alice Smith- -alice.smith@example.com-1985-05-15] - ValueError: Mobile Number should contain 10 digits (VC6)
FAILED upa.py::test_signup_valid_user[Alice Smithjhfbhwebfhewbfhbwhebhwbhbfehbf-9876543210-alice.smith@example.com-1985-05-15] - ValueError: Name should be less than
 30 characters (VC5)
FAILED upa.py::test_signup_valid_user[Alice Smith-98765432-alice.smith@example.com-1985-05-15] - ValueError: Mobile Number should contain 10 digits (VC6)
===================================================================== 3 failed, 2 passed in 0.23s ======================================================================
PS C:\Users\DELL\Desktop\Testing>
```

Payment Mode:

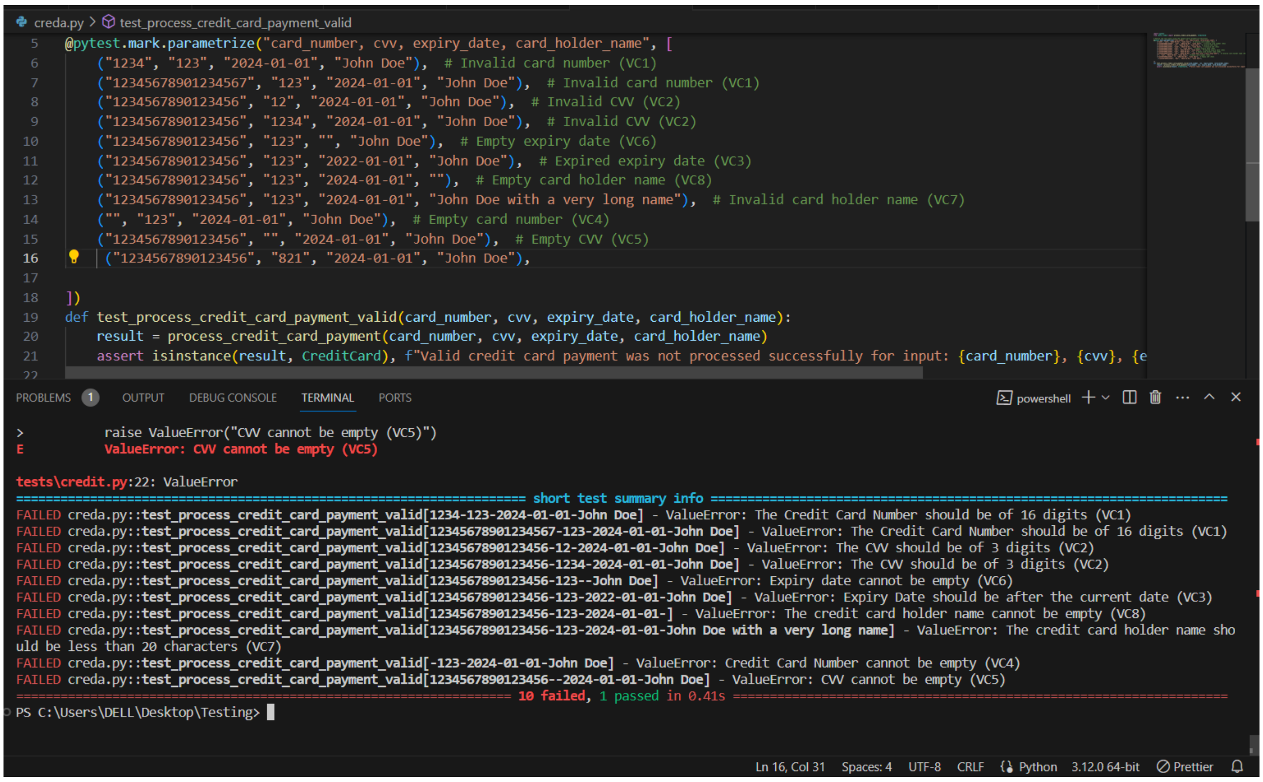
Debit Card:



```python
from tests.payment import process_debit_card_payment, DebitCard

# Define the test data using the pytest.mark.parametrize decorator
@pytest.mark.parametrize("card_number, cvv, expiry_date", [
    ("1234567890123456", "123", "2024-01-01"),
    ("9876543210987654", "4568", "2025-02-01"),
    ("9876543210987654", "456", "2022-06-02"),
    (" ", "456", "2025-02-01"),
    ("9876543210987654", "456", " "),
    ("9876543210987654", " ", "2025-02-01")
    # Add more valid test cases as needed
])
def test_process_debit_card_payment_valid(card_number, cvv, expiry_date):
    result = process_debit_card_payment(card_number, cvv, expiry_date)
    assert isinstance(result, DebitCard), f"Valid debit card payment was not processed successfully for input: {card_number}, {cvv}, {exp
```

```
        # VC1: The Debit Card Number should be of 16 digits
        if not (len(card_number) == 16 and card_number.isdigit()):
            raise ValueError("The Debit Card Number should be of 16 digits (VC1)")

        # VC5: The CVV cannot be empty
        if not cvv:
            raise ValueError("CVV cannot be empty (VC5)")

        # VC2: The CVV should be of 3 digits
        if not (len(cvv) == 3 and cvv.isdigit()):
>           raise ValueError("The CVV should be of 3 digits (VC2)")
E           ValueError: The CVV should be of 3 digits (VC2)

tests\payment.py:24: ValueError
======================================================================= short test summary info ========================================================================
FAILED payma.py::test_process_debit_card_payment_valid[9876543210987654-4568-2025-02-01] - ValueError: The CVV should be of 3 digits (VC2)
FAILED payma.py::test_process_debit_card_payment_valid[9876543210987654-456-2022-06-02] - ValueError: Expiry Date should be after the current date (VC3)
FAILED payma.py::test_process_debit_card_payment_valid[ -456-2025-02-01] - ValueError: The Debit Card Number should be of 16 digits (VC1)
FAILED payma.py::test_process_debit_card_payment_valid[9876543210987654-456- ] - ValueError: time data ' ' does not match format '%Y-%m-%d'
FAILED payma.py::test_process_debit_card_payment_valid[9876543210987654- -2025-02-01] - ValueError: The CVV should be of 3 digits (VC2)
===================================================================== 5 failed, 1 passed in 0.33s ======================================================================
PS C:\Users\DELL\Desktop\Testing>
```

Credit Card:



creda.py > test_process_credit_card_payment_valid

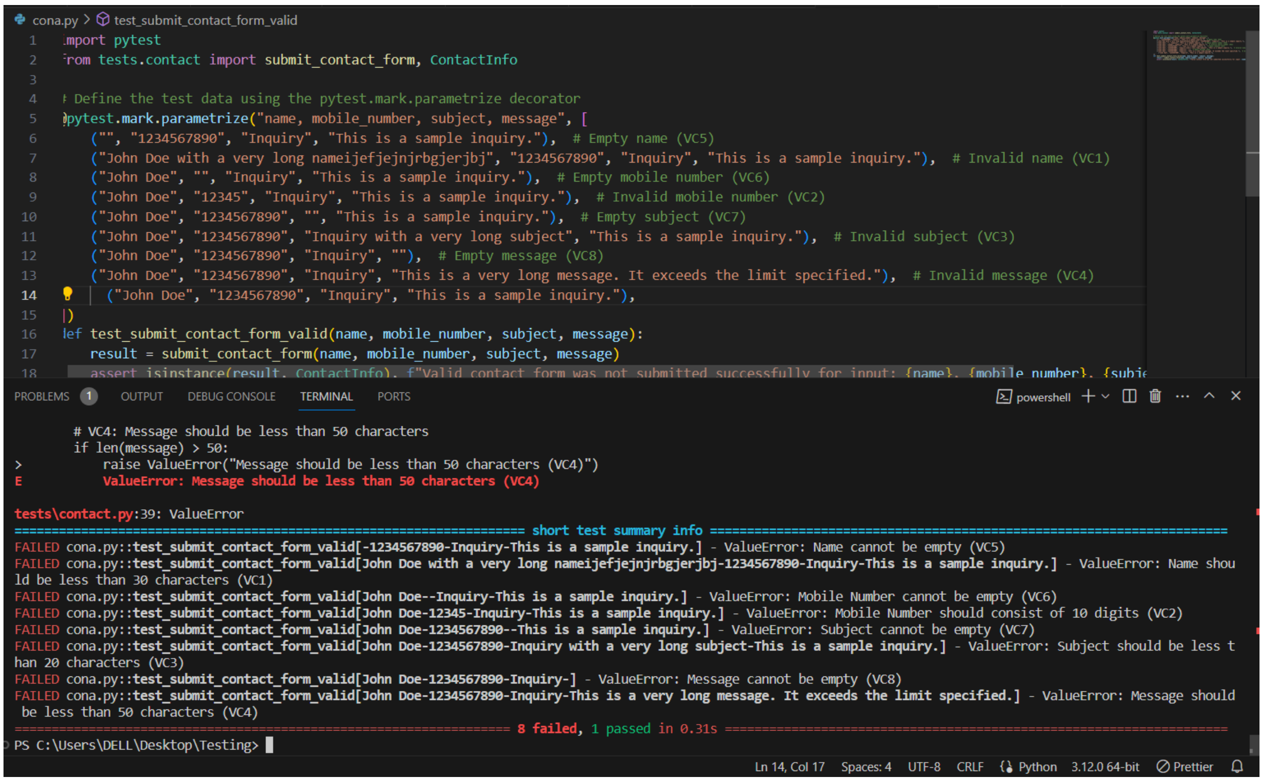
```python
@pytest.mark.parametrize("card_number, cvv, expiry_date, card_holder_name", [
    ("1234", "123", "2024-01-01", "John Doe"),  # Invalid card number (VC1)
    ("12345678901234567", "123", "2024-01-01", "John Doe"),  # Invalid card number (VC1)
    ("1234567890123456", "12", "2024-01-01", "John Doe"),  # Invalid CVV (VC2)
    ("1234567890123456", "1234", "2024-01-01", "John Doe"),  # Invalid CVV (VC2)
    ("1234567890123456", "123", "", "John Doe"),  # Empty expiry date (VC6)
    ("1234567890123456", "123", "2022-01-01", "John Doe"),  # Expired expiry date (VC3)
    ("1234567890123456", "123", "2024-01-01", ""),  # Empty card holder name (VC8)
    ("1234567890123456", "123", "2024-01-01", "John Doe with a very long name"),  # Invalid card holder name (VC7)
    ("", "123", "2024-01-01", "John Doe"),  # Empty card number (VC4)
    ("1234567890123456", "", "2024-01-01", "John Doe"),  # Empty CVV (VC5)
     ("1234567890123456", "821", "2024-01-01", "John Doe"),

])
def test_process_credit_card_payment_valid(card_number, cvv, expiry_date, card_holder_name):
    result = process_credit_card_payment(card_number, cvv, expiry_date, card_holder_name)
    assert isinstance(result, CreditCard), f"Valid credit card payment was not processed successfully for input: {card_number}, {cvv}, {e
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
>           raise ValueError("CVV cannot be empty (VC5)")
E           ValueError: CVV cannot be empty (VC5)

tests\credit.py:22: ValueError
=============================================================== short test summary info ================================================================
FAILED creda.py::test_process_credit_card_payment_valid[1234-123-2024-01-01-John Doe] - ValueError: The Credit Card Number should be of 16 digits (VC1)
FAILED creda.py::test_process_credit_card_payment_valid[12345678901234567-123-2024-01-01-John Doe] - ValueError: The Credit Card Number should be of 16 digits (VC1)
FAILED creda.py::test_process_credit_card_payment_valid[1234567890123456-12-2024-01-01-John Doe] - ValueError: The CVV should be of 3 digits (VC2)
FAILED creda.py::test_process_credit_card_payment_valid[1234567890123456-1234-2024-01-01-John Doe] - ValueError: The CVV should be of 3 digits (VC2)
FAILED creda.py::test_process_credit_card_payment_valid[1234567890123456-123--John Doe] - ValueError: Expiry date cannot be empty (VC6)
FAILED creda.py::test_process_credit_card_payment_valid[1234567890123456-123-2022-01-01-John Doe] - ValueError: Expiry Date should be after the current date (VC3)
FAILED creda.py::test_process_credit_card_payment_valid[1234567890123456-123-2024-01-01-] - ValueError: The credit card holder name cannot be empty (VC8)
FAILED creda.py::test_process_credit_card_payment_valid[1234567890123456-123-2024-01-01-John Doe with a very long name] - ValueError: The credit card holder name sho
uld be less than 20 characters (VC7)
FAILED creda.py::test_process_credit_card_payment_valid[-123-2024-01-01-John Doe] - ValueError: Credit Card Number cannot be empty (VC4)
FAILED creda.py::test_process_credit_card_payment_valid[1234567890123456--2024-01-01-John Doe] - ValueError: CVV cannot be empty (VC5)
============================================================= 10 failed, 1 passed in 0.41s =============================================================
PS C:\Users\DELL\Desktop\Testing> 
```

Ln 16, Col 31 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Prettier

Contact Us:



cona.py > test_submit_contact_form_valid

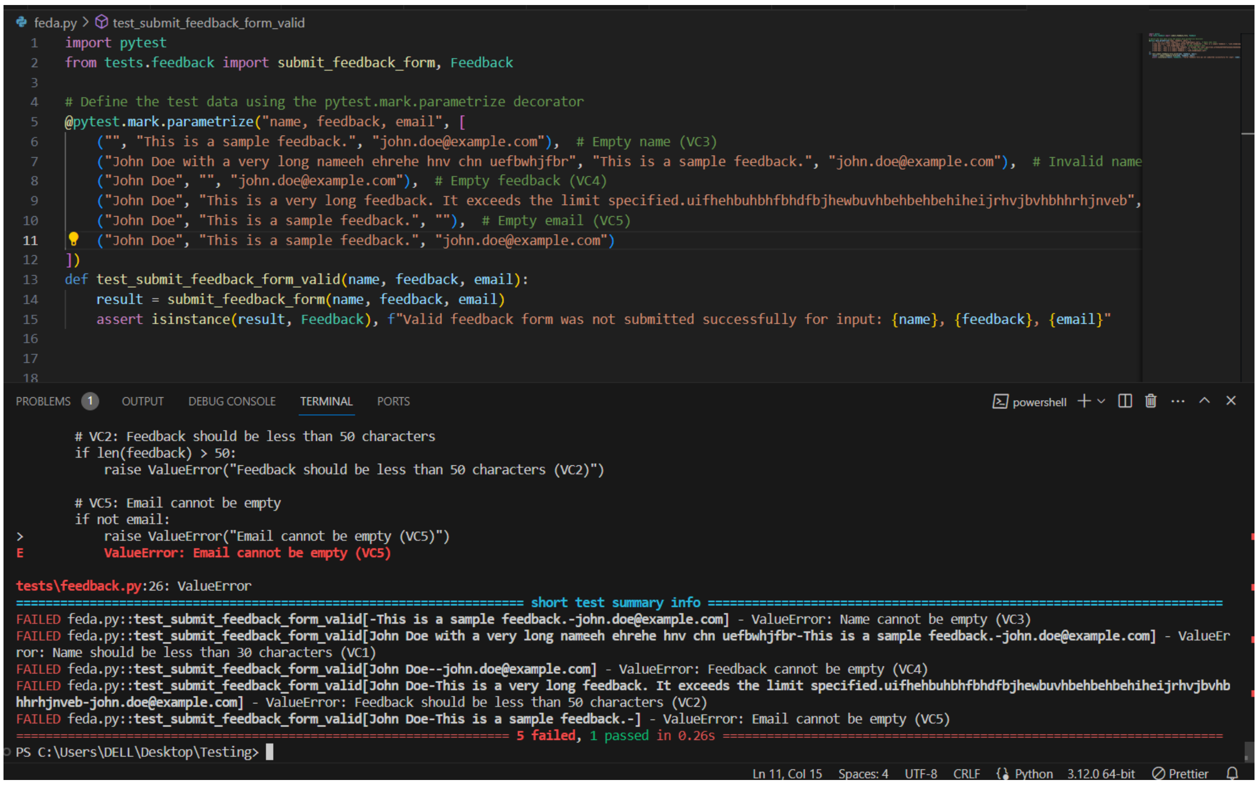
```python
mport pytest
rom tests.contact import submit_contact_form, ContactInfo

 Define the test data using the pytest.mark.parametrize decorator
pytest.mark.parametrize("name, mobile_number, subject, message", [
   ("", "1234567890", "Inquiry", "This is a sample inquiry."),  # Empty name (VC5)
   ("John Doe with a very long nameijefjejnjrbgjerjbj", "1234567890", "Inquiry", "This is a sample inquiry."),  # Invalid name (VC1)
   ("John Doe", "", "Inquiry", "This is a sample inquiry."),  # Empty mobile number (VC6)
   ("John Doe", "12345", "Inquiry", "This is a sample inquiry."),  # Invalid mobile number (VC2)
   ("John Doe", "1234567890", "", "This is a sample inquiry."),  # Empty subject (VC7)
   ("John Doe", "1234567890", "Inquiry with a very long subject", "This is a sample inquiry."),  # Invalid subject (VC3)
   ("John Doe", "1234567890", "Inquiry", ""),  # Empty message (VC8)
   ("John Doe", "1234567890", "Inquiry", "This is a very long message. It exceeds the limit specified."),  # Invalid message (VC4)
     ("John Doe", "1234567890", "Inquiry", "This is a sample inquiry."),
)
lef test_submit_contact_form_valid(name, mobile_number, subject, message):
   result = submit_contact_form(name, mobile_number, subject, message)
   assert isinstance(result, ContactInfo), f"Valid contact form was not submitted successfully for input: {name}, {mobile_number}, {subje
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell

```
        # VC4: Message should be less than 50 characters
        if len(message) > 50:
>           raise ValueError("Message should be less than 50 characters (VC4)")
E           ValueError: Message should be less than 50 characters (VC4)

tests\contact.py:39: ValueError
=============================================================== short test summary info ================================================================
FAILED cona.py::test_submit_contact_form_valid[-1234567890-Inquiry-This is a sample inquiry.] - ValueError: Name cannot be empty (VC5)
FAILED cona.py::test_submit_contact_form_valid[John Doe with a very long nameijefjejnjrbgjerjbj-1234567890-Inquiry-This is a sample inquiry.] - ValueError: Name shou
ld be less than 30 characters (VC1)
FAILED cona.py::test_submit_contact_form_valid[John Doe--Inquiry-This is a sample inquiry.] - ValueError: Mobile Number cannot be empty (VC6)
FAILED cona.py::test_submit_contact_form_valid[John Doe-12345-Inquiry-This is a sample inquiry.] - ValueError: Mobile Number should consist of 10 digits (VC2)
FAILED cona.py::test_submit_contact_form_valid[John Doe-1234567890--This is a sample inquiry.] - ValueError: Subject cannot be empty (VC7)
FAILED cona.py::test_submit_contact_form_valid[John Doe-1234567890-Inquiry with a very long subject-This is a sample inquiry.] - ValueError: Subject should be less t
han 20 characters (VC3)
FAILED cona.py::test_submit_contact_form_valid[John Doe-1234567890-Inquiry-] - ValueError: Message cannot be empty (VC8)
FAILED cona.py::test_submit_contact_form_valid[John Doe-1234567890-Inquiry-This is a very long message. It exceeds the limit specified.] - ValueError: Message should
 be less than 50 characters (VC4)
============================================================== 8 failed, 1 passed in 0.31s =============================================================
PS C:\Users\DELL\Desktop\Testing> 
```

Ln 14, Col 17 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Prettier

Feedback Form:



feda.py > test_submit_feedback_form_valid

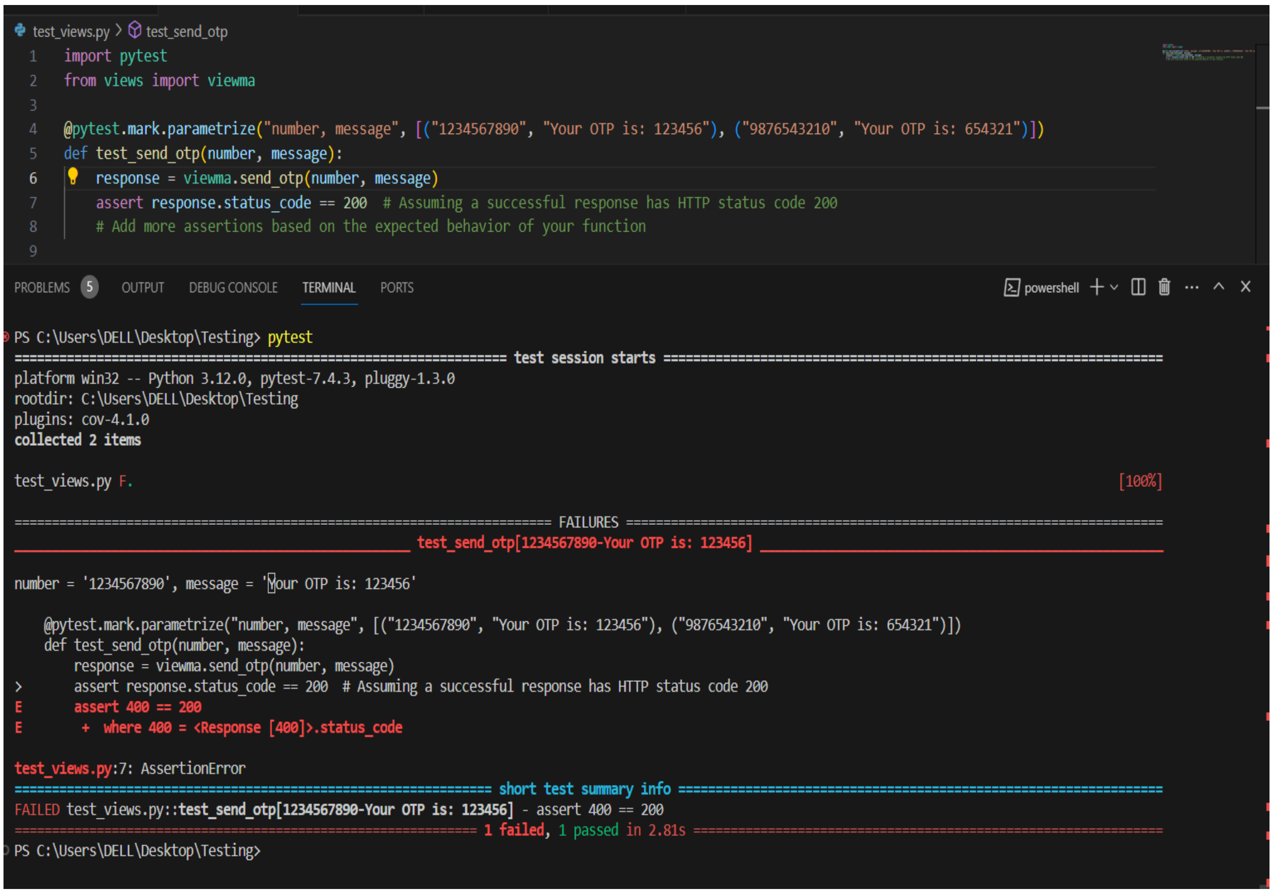
```python
import pytest
from tests.feedback import submit_feedback_form, Feedback

# Define the test data using the pytest.mark.parametrize decorator
@pytest.mark.parametrize("name, feedback, email", [
    ("", "This is a sample feedback.", "john.doe@example.com"),  # Empty name (VC3)
    ("John Doe with a very long nameeh ehrehe hnv chn uefbwhjfbr", "This is a sample feedback.", "john.doe@example.com"),  # Invalid name
    ("John Doe", "", "john.doe@example.com"),  # Empty feedback (VC4)
    ("John Doe", "This is a very long feedback. It exceeds the limit specified.uifhehbuhbhfbhdfbjhewbuvhbehbehbehiheijrhvjbvhbhhrhjnveb",
    ("John Doe", "This is a sample feedback.", ""),  # Empty email (VC5)
    ("John Doe", "This is a sample feedback.", "john.doe@example.com")
])
def test_submit_feedback_form_valid(name, feedback, email):
    result = submit_feedback_form(name, feedback, email)
    assert isinstance(result, Feedback), f"Valid feedback form was not submitted successfully for input: {name}, {feedback}, {email}"
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
        # VC2: Feedback should be less than 50 characters
        if len(feedback) > 50:
            raise ValueError("Feedback should be less than 50 characters (VC2)")

        # VC5: Email cannot be empty
        if not email:
>           raise ValueError("Email cannot be empty (VC5)")
E           ValueError: Email cannot be empty (VC5)

tests\feedback.py:26: ValueError
============================================================ short test summary info ============================================================
FAILED feda.py::test_submit_feedback_form_valid[-This is a sample feedback.-john.doe@example.com] - ValueError: Name cannot be empty (VC3)
FAILED feda.py::test_submit_feedback_form_valid[John Doe with a very long nameeh ehrehe hnv chn uefbwhjfbr-This is a sample feedback.-john.doe@example.com] - ValueEr
ror: Name should be less than 30 characters (VC1)
FAILED feda.py::test_submit_feedback_form_valid[John Doe--john.doe@example.com] - ValueError: Feedback cannot be empty (VC4)
FAILED feda.py::test_submit_feedback_form_valid[John Doe-This is a very long feedback. It exceeds the limit specified.uifhehbuhbhfbhdfbjhewbuvhbehbehbehiheijrhvjbvhb
hhrhjnveb-john.doe@example.com] - ValueError: Feedback should be less than 50 characters (VC2)
FAILED feda.py::test_submit_feedback_form_valid[John Doe-This is a sample feedback.-] - ValueError: Email cannot be empty (VC5)
============================================================ 5 failed, 1 passed in 0.26s ============================================================
PS C:\Users\DELL\Desktop\Testing>
```

Ln 11, Col 15 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Prettier

Validation of Otp and Phone No:



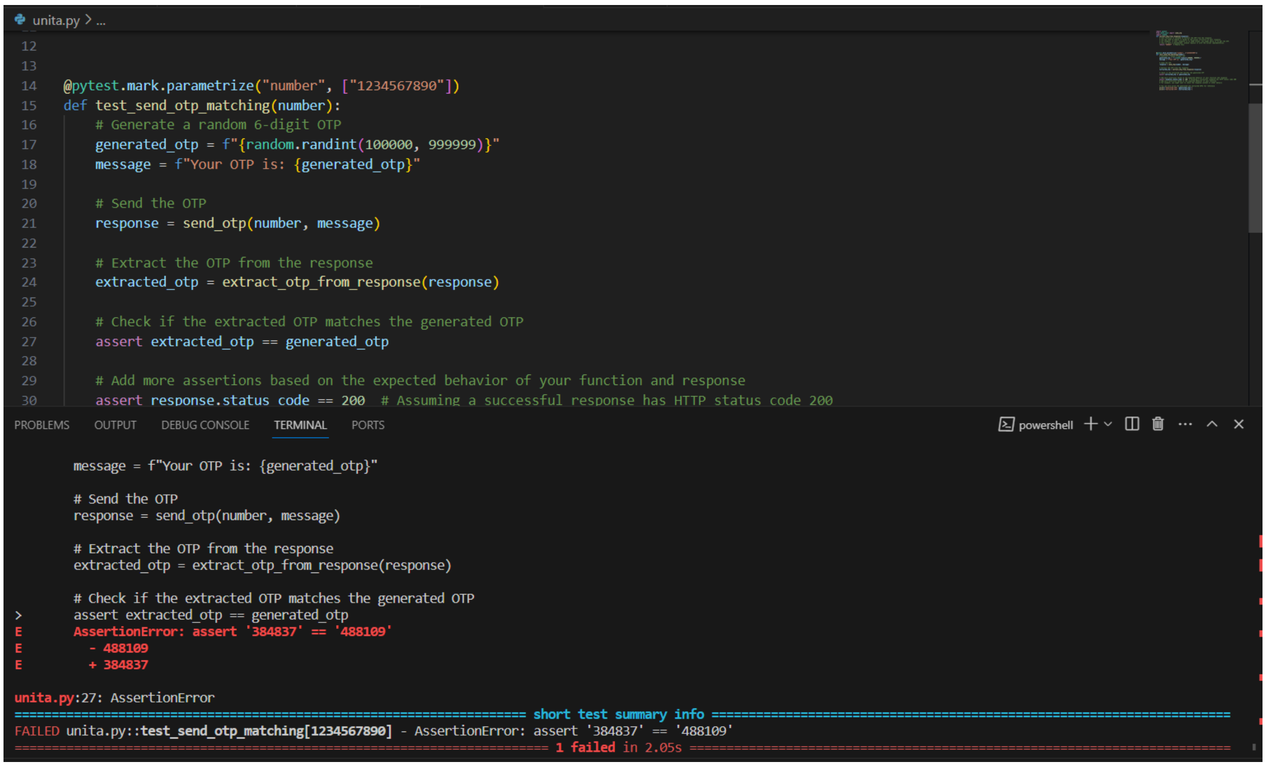
test_views.py > test_send_otp

```python
import pytest
from views import viewma

@pytest.mark.parametrize("number, message", [("1234567890", "Your OTP is: 123456"), ("9876543210", "Your OTP is: 654321")])
def test_send_otp(number, message):
    response = viewma.send_otp(number, message)
    assert response.status_code == 200  # Assuming a successful response has HTTP status code 200
    # Add more assertions based on the expected behavior of your function
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\Testing> pytest
============================================================ test session starts ============================================================
platform win32 -- Python 3.12.0, pytest-7.4.3, pluggy-1.3.0
rootdir: C:\Users\DELL\Desktop\Testing
plugins: cov-4.1.0
collected 2 items

test_views.py F.                                                                                                                      [100%]

================================================================== FAILURES ==================================================================
__________________________________________________ test_send_otp[1234567890-Your OTP is: 123456] __________________________________________________

number = '1234567890', message = 'Your OTP is: 123456'

    @pytest.mark.parametrize("number, message", [("1234567890", "Your OTP is: 123456"), ("9876543210", "Your OTP is: 654321")])
    def test_send_otp(number, message):
        response = viewma.send_otp(number, message)
>       assert response.status_code == 200  # Assuming a successful response has HTTP status code 200
E       assert 400 == 200
E        +  where 400 = <Response [400]>.status_code

test_views.py:7: AssertionError
============================================================ short test summary info ============================================================
FAILED test_views.py::test_send_otp[1234567890-Your OTP is: 123456] - assert 400 == 200
============================================================ 1 failed, 1 passed in 2.81s ============================================================
PS C:\Users\DELL\Desktop\Testing>
```

Otp Matching:



unita.py > ...

```python


@pytest.mark.parametrize("number", ["1234567890"])
def test_send_otp_matching(number):
    # Generate a random 6-digit OTP
    generated_otp = f"{random.randint(100000, 999999)}"
    message = f"Your OTP is: {generated_otp}"

    # Send the OTP
    response = send_otp(number, message)

    # Extract the OTP from the response
    extracted_otp = extract_otp_from_response(response)

    # Check if the extracted OTP matches the generated OTP
    assert extracted_otp == generated_otp

    # Add more assertions based on the expected behavior of your function and response
    assert response.status_code == 200  # Assuming a successful response has HTTP status code 200
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
        message = f"Your OTP is: {generated_otp}"

        # Send the OTP
        response = send_otp(number, message)

        # Extract the OTP from the response
        extracted_otp = extract_otp_from_response(response)

        # Check if the extracted OTP matches the generated OTP
>       assert extracted_otp == generated_otp
E       AssertionError: assert '384837' == '488109'
E         - 488109
E         + 384837

unita.py:27: AssertionError
==================================================================== short test summary info =====================================================================
FAILED unita.py::test_send_otp_matching[1234567890] - AssertionError: assert '384837' == '488109'
======================================================================= 1 failed in 2.05s ========================================================================
```