AI ASSISTED CODING

ASSIGNMENT 8.2

Name: G Hasini

Roll no:2503A51L41

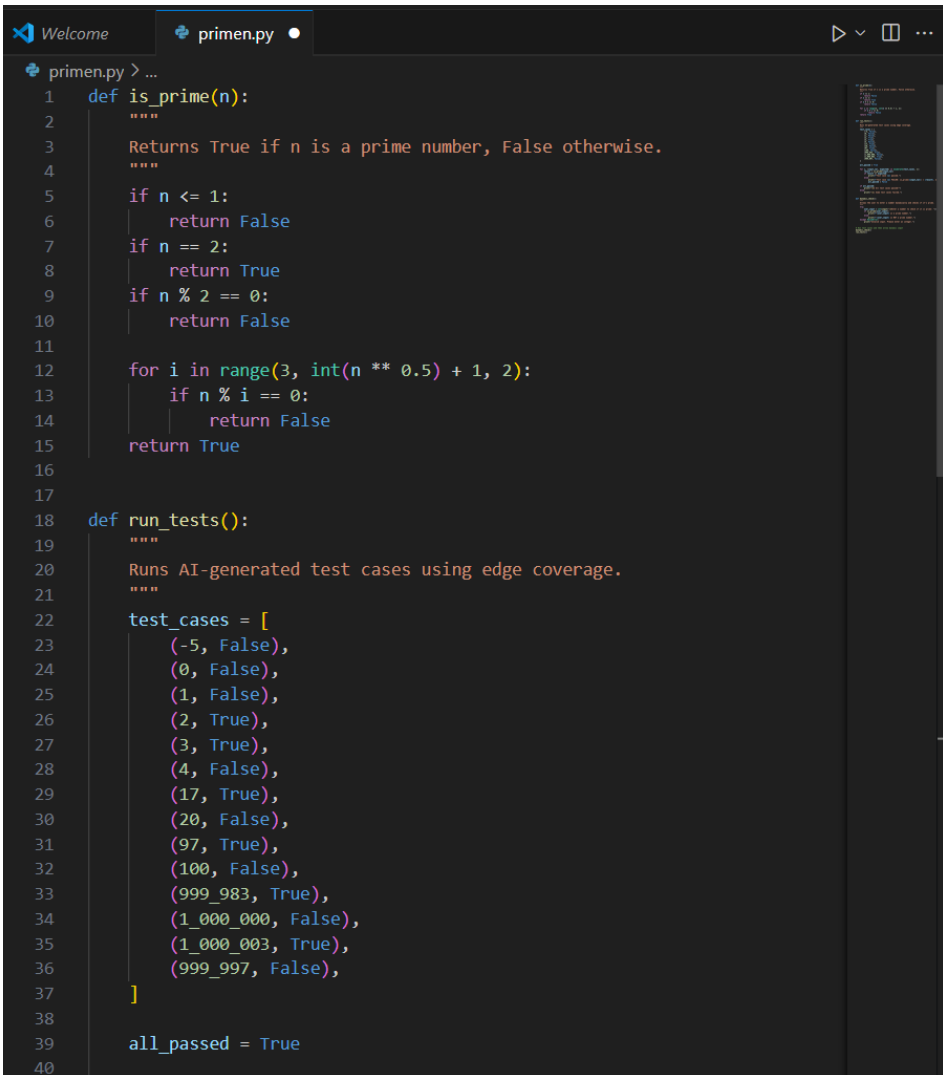
Batch:24BTCAICSB20

Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases

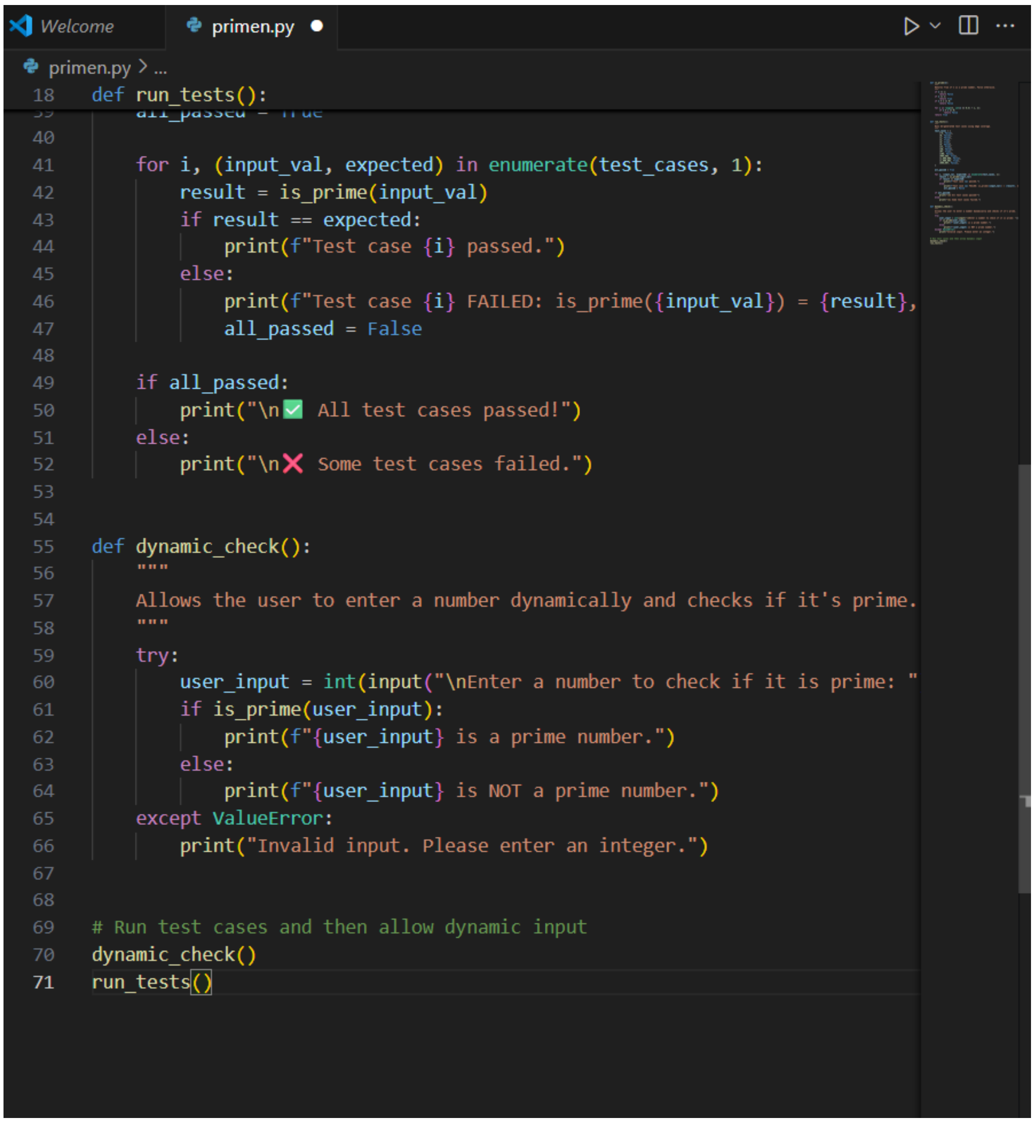
Task #1

Prompt: Use AI to generate test cases for a function is_prime(n) and then implement the function.

Code generated:

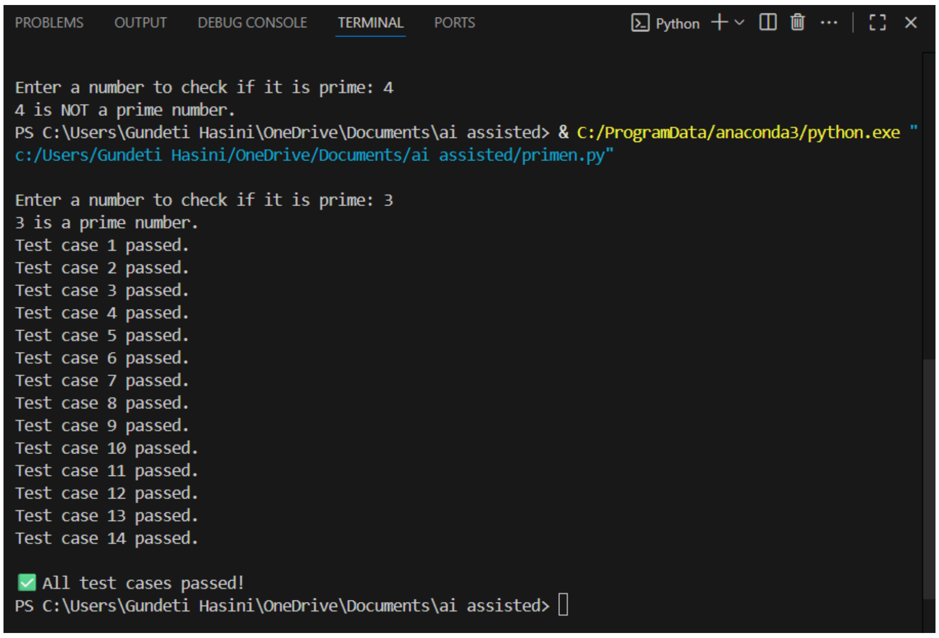


```python
def is_prime(n):
    """
    Returns True if n is a prime number, False otherwise.
    """
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return False
    return True


def run_tests():
    """
    Runs AI-generated test cases using edge coverage.
    """
    test_cases = [
        (-5, False),
        (0, False),
        (1, False),
        (2, True),
        (3, True),
        (4, False),
        (17, True),
        (20, False),
        (97, True),
        (100, False),
        (999_983, True),
        (1_000_000, False),
        (1_000_003, True),
        (999_997, False),
    ]

    all_passed = True
```



```python
def run_tests():
    all_passed = True

    for i, (input_val, expected) in enumerate(test_cases, 1):
        result = is_prime(input_val)
        if result == expected:
            print(f"Test case {i} passed.")
        else:
            print(f"Test case {i} FAILED: is_prime({input_val}) = {result},
            all_passed = False

    if all_passed:
        print("\n✅ All test cases passed!")
    else:
        print("\n❌ Some test cases failed.")


def dynamic_check():
    """
    Allows the user to enter a number dynamically and checks if it's prime.
    """
    try:
        user_input = int(input("\nEnter a number to check if it is prime: "
        if is_prime(user_input):
            print(f"{user_input} is a prime number.")
        else:
            print(f"{user_input} is NOT a prime number.")
    except ValueError:
        print("Invalid input. Please enter an integer.")


# Run test cases and then allow dynamic input
dynamic_check()
run_tests()
```

Output :



```
Enter a number to check if it is prime: 4
4 is NOT a prime number.
PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted> & C:/ProgramData/anaconda3/python.exe "
c:/Users/Gundeti Hasini/OneDrive/Documents/ai assisted/primen.py"

Enter a number to check if it is prime: 3
3 is a prime number.
Test case 1 passed.
Test case 2 passed.
Test case 3 passed.
Test case 4 passed.
Test case 5 passed.
Test case 6 passed.
Test case 7 passed.
Test case 8 passed.
Test case 9 passed.
Test case 10 passed.
Test case 11 passed.
Test case 12 passed.
Test case 13 passed.
Test case 14 passed.

✅ All test cases passed!
PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted> 
```

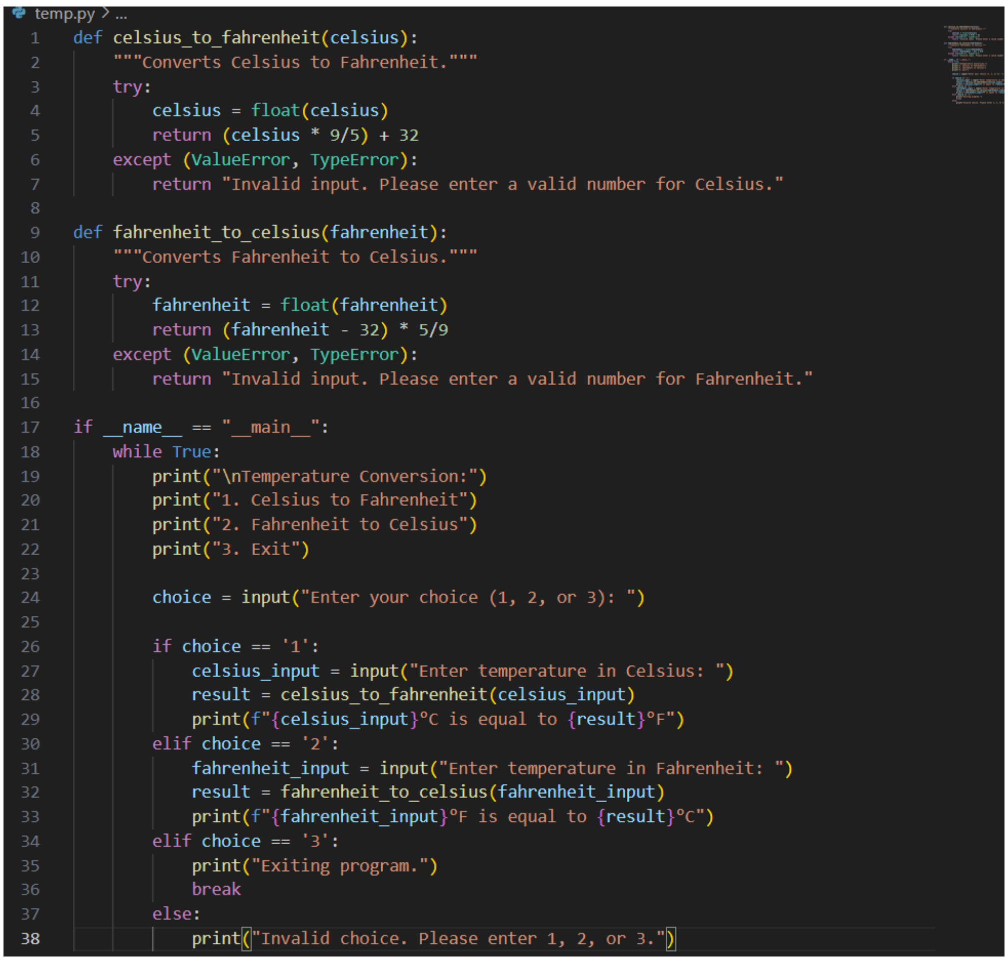
Observation:

- The function correctly checks if numbers greater than 1 are prime.
- Handles edge cases like negative numbers, 0, and 1 properly.
- Returns true for 2, the smallest prime, and excludes other even numbers quickly, Uses an efficient check up to the square root of the number, skipping even divisors.
- Tests cover a wide range of cases including large primes and non-primes.
- The test runner prints clear pass/fail results and confirms when all tests pass. Includes a dynamic user input section with error handling for invalid entries.

<u>Task #2</u>

<u>Prompt</u>: Ask AI to generate test cases for celsius_to_fahrenheit(c) and fahrenheit_to_celsius(f)

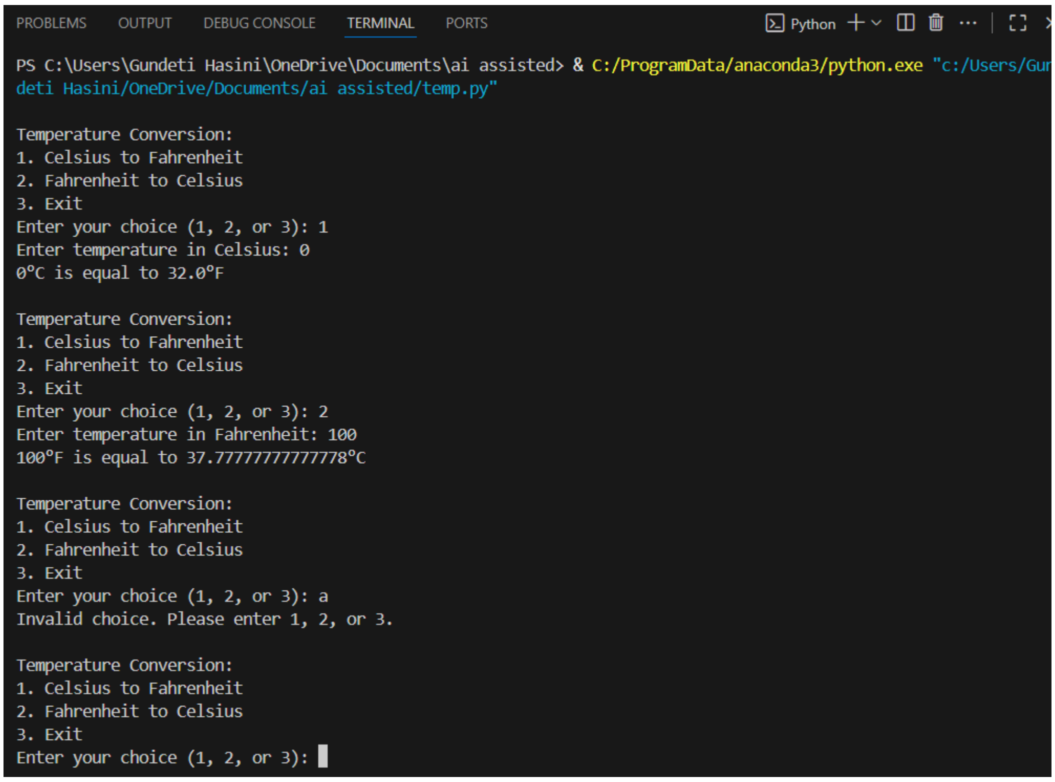
Code generated:



temp.py > ...

```python
def celsius_to_fahrenheit(celsius):
    """Converts Celsius to Fahrenheit."""
    try:
        celsius = float(celsius)
        return (celsius * 9/5) + 32
    except (ValueError, TypeError):
        return "Invalid input. Please enter a valid number for Celsius."

def fahrenheit_to_celsius(fahrenheit):
    """Converts Fahrenheit to Celsius."""
    try:
        fahrenheit = float(fahrenheit)
        return (fahrenheit - 32) * 5/9
    except (ValueError, TypeError):
        return "Invalid input. Please enter a valid number for Fahrenheit."

if __name__ == "__main__":
    while True:
        print("\nTemperature Conversion:")
        print("1. Celsius to Fahrenheit")
        print("2. Fahrenheit to Celsius")
        print("3. Exit")

        choice = input("Enter your choice (1, 2, or 3): ")

        if choice == '1':
            celsius_input = input("Enter temperature in Celsius: ")
            result = celsius_to_fahrenheit(celsius_input)
            print(f"{celsius_input}°C is equal to {result}°F")
        elif choice == '2':
            fahrenheit_input = input("Enter temperature in Fahrenheit: ")
            result = fahrenheit_to_celsius(fahrenheit_input)
            print(f"{fahrenheit_input}°F is equal to {result}°C")
        elif choice == '3':
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please enter 1, 2, or 3.")
```

Output:



```
PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted> & C:/ProgramData/anaconda3/python.exe "c:/Users/Gun
deti Hasini/OneDrive/Documents/ai assisted/temp.py"

Temperature Conversion:
1. Celsius to Fahrenheit
2. Fahrenheit to Celsius
3. Exit
Enter your choice (1, 2, or 3): 1
Enter temperature in Celsius: 0
0°C is equal to 32.0°F

Temperature Conversion:
1. Celsius to Fahrenheit
2. Fahrenheit to Celsius
3. Exit
Enter your choice (1, 2, or 3): 2
Enter temperature in Fahrenheit: 100
100°F is equal to 37.77777777777778°C

Temperature Conversion:
1. Celsius to Fahrenheit
2. Fahrenheit to Celsius
3. Exit
Enter your choice (1, 2, or 3): a
Invalid choice. Please enter 1, 2, or 3.

Temperature Conversion:
1. Celsius to Fahrenheit
2. Fahrenheit to Celsius
3. Exit
Enter your choice (1, 2, or 3): 
```

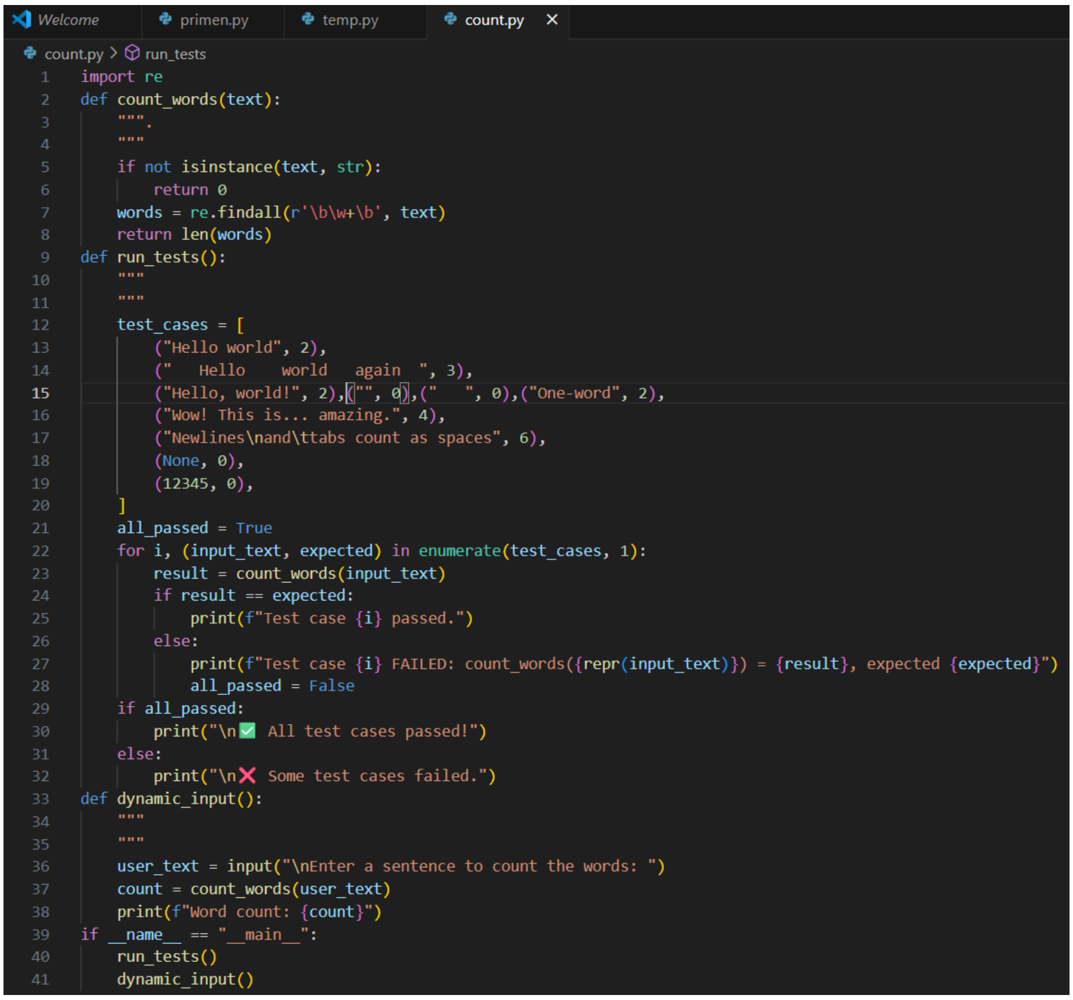
Observation:

- Uses correct formulas for temperature conversion,
- Handles edge cases like 0°C = 32°F and 100°C = 212°F
- Supports decimal inputs
- Safely handles invalid inputs like strings and None using try-except
- Includes automated test cases for all input types.

Task#3

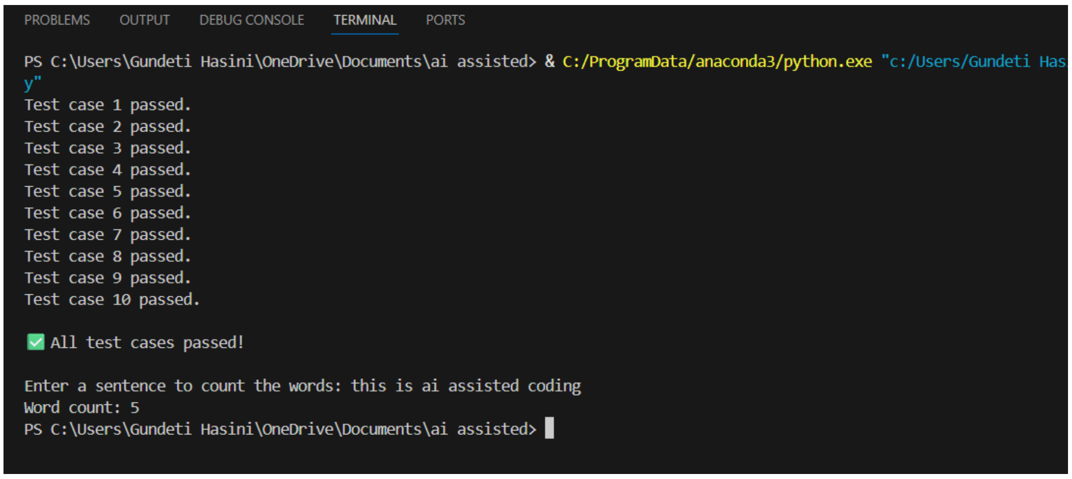
Prompt: Use AI to write test cases for a function count_words(text) that returns the number of words in a sentence.

Code generated:



```python
import re
def count_words(text):
    """.
    """
    if not isinstance(text, str):
        return 0
    words = re.findall(r'\b\w+\b', text)
    return len(words)
def run_tests():
    """
    """
    test_cases = [
        ("Hello world", 2),
        ("   Hello    world   again  ", 3),
        ("Hello, world!", 2),("", 0),("   ", 0),("One-word", 2),
        ("Wow! This is... amazing.", 4),
        ("Newlines\nand\ttabs count as spaces", 6),
        (None, 0),
        (12345, 0),
    ]
    all_passed = True
    for i, (input_text, expected) in enumerate(test_cases, 1):
        result = count_words(input_text)
        if result == expected:
            print(f"Test case {i} passed.")
        else:
            print(f"Test case {i} FAILED: count_words({repr(input_text)}) = {result}, expected {expected}")
            all_passed = False
    if all_passed:
        print("\n✅ All test cases passed!")
    else:
        print("\n❌ Some test cases failed.")
def dynamic_input():
    """
    """
    user_text = input("\nEnter a sentence to count the words: ")
    count = count_words(user_text)
    print(f"Word count: {count}")
if __name__ == "__main__":
    run_tests()
    dynamic_input()
```

Output:



```
PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted> & C:/ProgramData/anaconda3/python.exe "c:/Users/Gundeti Has
y"
Test case 1 passed.
Test case 2 passed.
Test case 3 passed.
Test case 4 passed.
Test case 5 passed.
Test case 6 passed.
Test case 7 passed.
Test case 8 passed.
Test case 9 passed.
Test case 10 passed.

✅ All test cases passed!

Enter a sentence to count the words: this is ai assisted coding
Word count: 5
PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted>
```

Observation:

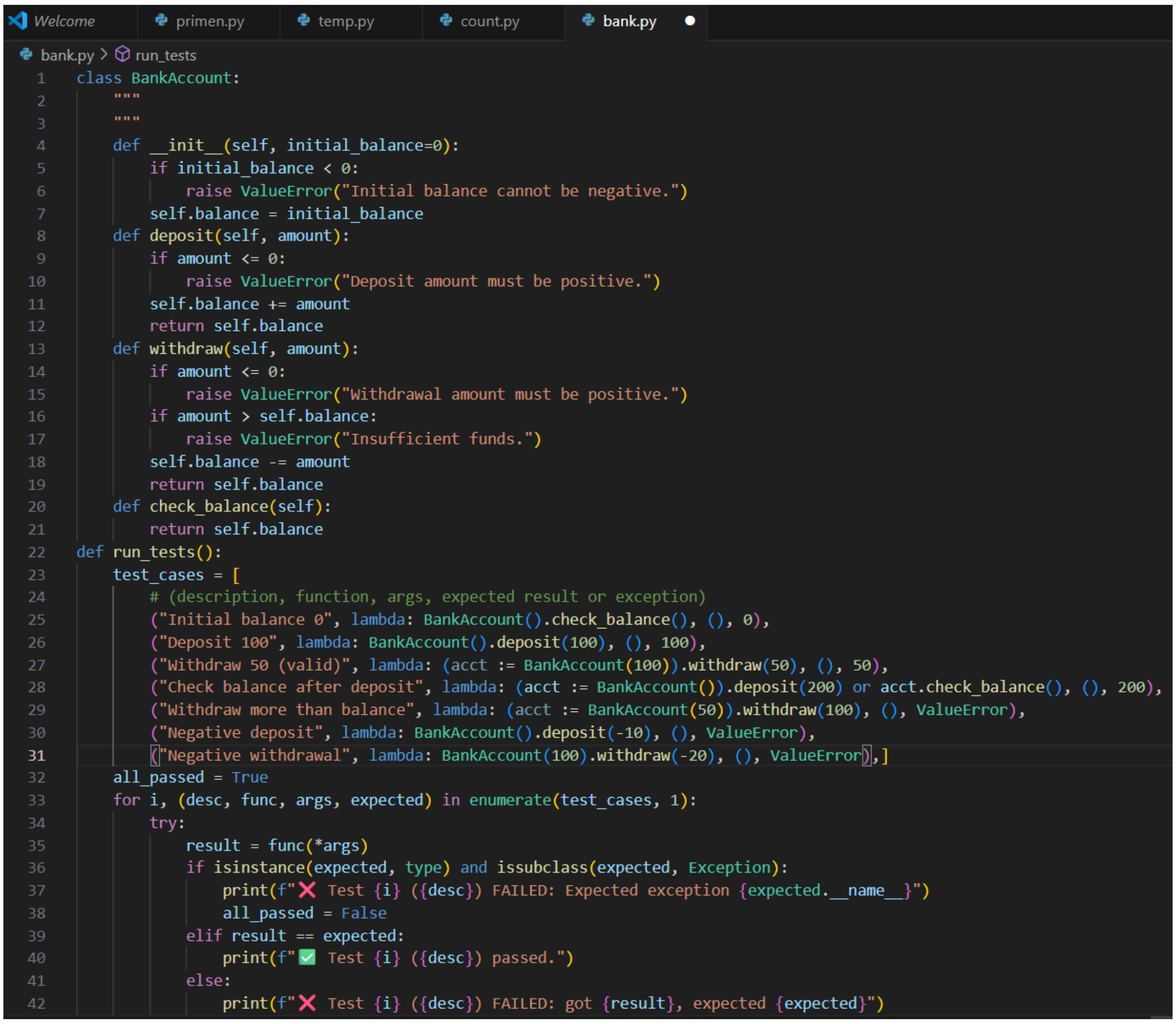
- Good use of regex (\b\w+\b)-This correctly matches words while ignoring punctuation.

- Robust error handling-You check if not isinstance(text, str): return 0, which prevents crashes for invalid inputs.
- Comprehensive test cases-Covers normal text, multiple spaces, punctuation, tabs/newlines, None, and numbers.
- Clear testing framework-Uses run_tests() to validate correctness with informative messages (passed / failed).

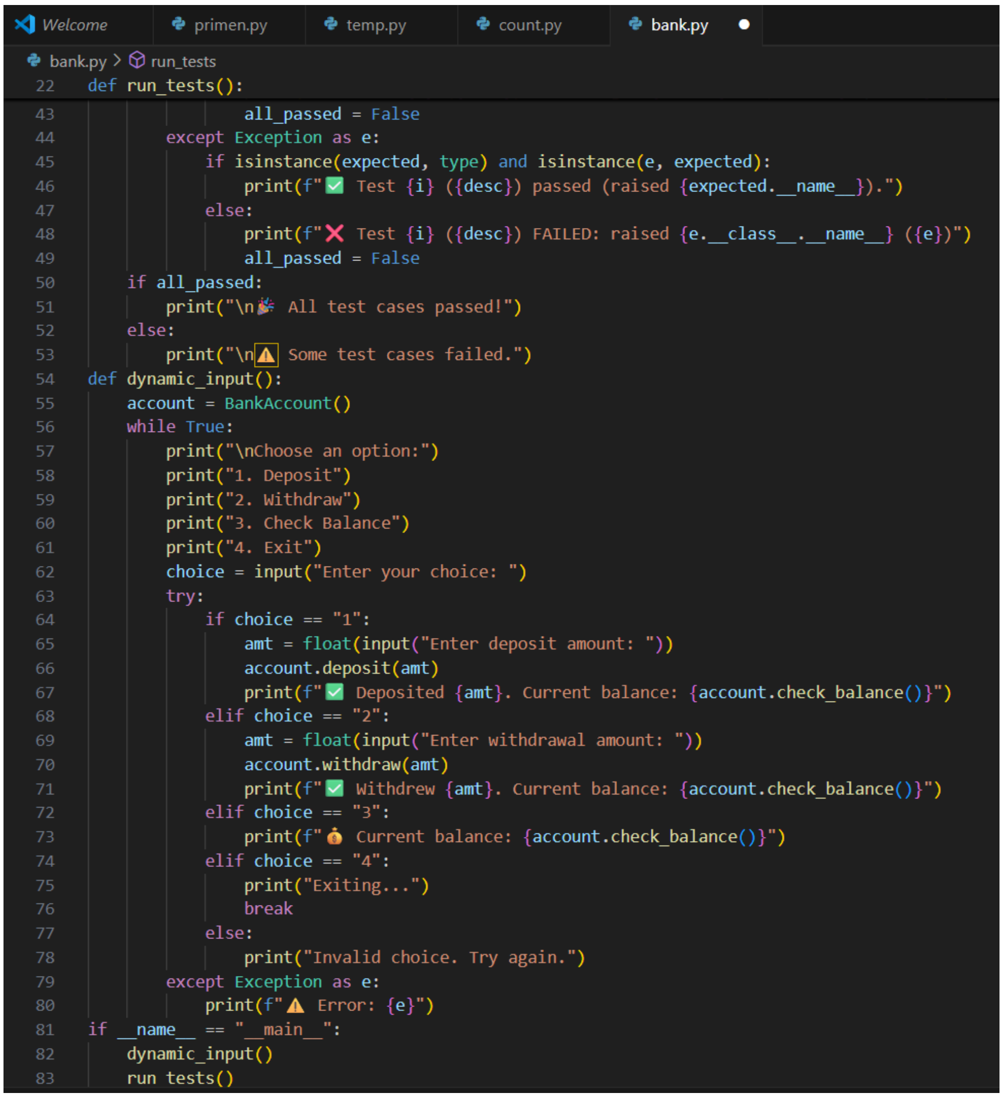
Task#4

Prompt: Generate test cases for a BankAccount class with: deposit(amount), withdraw(amount), check_balance()

Code generated:

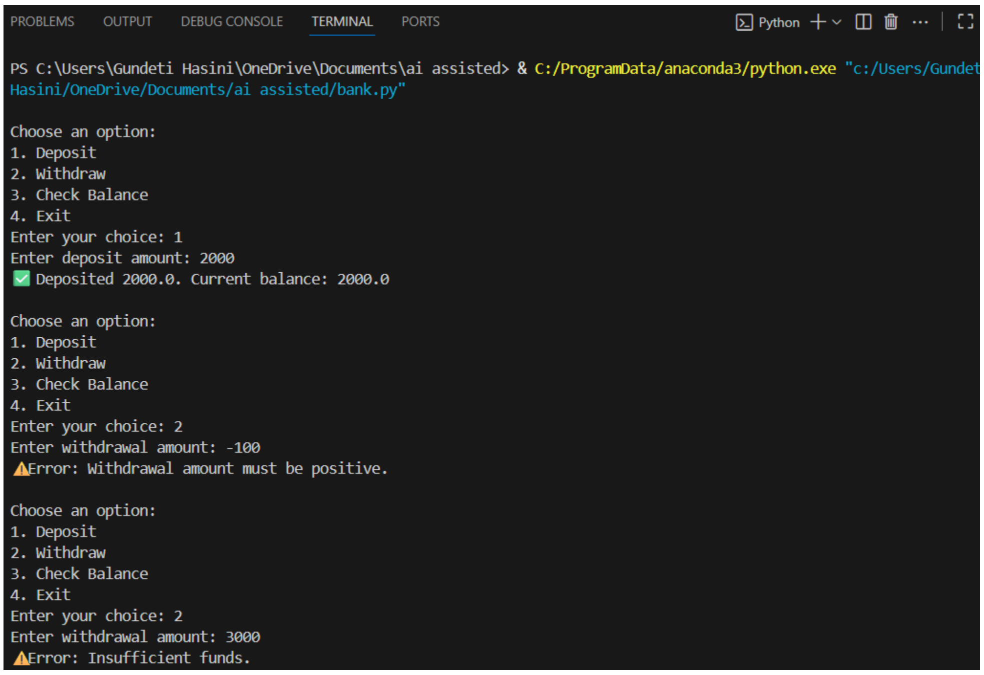


```python
class BankAccount:
    """
    """
    def __init__(self, initial_balance=0):
        if initial_balance < 0:
            raise ValueError("Initial balance cannot be negative.")
        self.balance = initial_balance
    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Deposit amount must be positive.")
        self.balance += amount
        return self.balance
    def withdraw(self, amount):
        if amount <= 0:
            raise ValueError("Withdrawal amount must be positive.")
        if amount > self.balance:
            raise ValueError("Insufficient funds.")
        self.balance -= amount
        return self.balance
    def check_balance(self):
        return self.balance
def run_tests():
    test_cases = [
        # (description, function, args, expected result or exception)
        ("Initial balance 0", lambda: BankAccount().check_balance(), (), 0),
        ("Deposit 100", lambda: BankAccount().deposit(100), (), 100),
        ("Withdraw 50 (valid)", lambda: (acct := BankAccount(100)).withdraw(50), (), 50),
        ("Check balance after deposit", lambda: (acct := BankAccount()).deposit(200) or acct.check_balance(), (), 200),
        ("Withdraw more than balance", lambda: (acct := BankAccount(50)).withdraw(100), (), ValueError),
        ("Negative deposit", lambda: BankAccount().deposit(-10), (), ValueError),
        ("Negative withdrawal", lambda: BankAccount(100).withdraw(-20), (), ValueError),]
    all_passed = True
    for i, (desc, func, args, expected) in enumerate(test_cases, 1):
        try:
            result = func(*args)
            if isinstance(expected, type) and issubclass(expected, Exception):
                print(f"❌ Test {i} ({desc}) FAILED: Expected exception {expected.__name__}")
                all_passed = False
            elif result == expected:
                print(f"✅ Test {i} ({desc}) passed.")
            else:
                print(f"❌ Test {i} ({desc}) FAILED: got {result}, expected {expected}")
```



```python
def run_tests():
                all_passed = False
        except Exception as e:
            if isinstance(expected, type) and isinstance(e, expected):
                print(f"✅ Test {i} ({desc}) passed (raised {expected.__name__}).")
            else:
                print(f"❌ Test {i} ({desc}) FAILED: raised {e.__class__.__name__} ({e})")
                all_passed = False
    if all_passed:
        print("\n🎉 All test cases passed!")
    else:
        print("\n⚠️ Some test cases failed.")
def dynamic_input():
    account = BankAccount()
    while True:
        print("\nChoose an option:")
        print("1. Deposit")
        print("2. Withdraw")
        print("3. Check Balance")
        print("4. Exit")
        choice = input("Enter your choice: ")
        try:
            if choice == "1":
                amt = float(input("Enter deposit amount: "))
                account.deposit(amt)
                print(f"✅ Deposited {amt}. Current balance: {account.check_balance()}")
            elif choice == "2":
                amt = float(input("Enter withdrawal amount: "))
                account.withdraw(amt)
                print(f"✅ Withdrew {amt}. Current balance: {account.check_balance()}")
            elif choice == "3":
                print(f"💰 Current balance: {account.check_balance()}")
            elif choice == "4":
                print("Exiting...")
                break
            else:
                print("Invalid choice. Try again.")
        except Exception as e:
            print(f"⚠️ Error: {e}")
if __name__ == "__main__":
    dynamic_input()
    run_tests()
```

Output:



```
PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted> & C:/ProgramData/anaconda3/python.exe "c:/Users/Gundet
Hasini/OneDrive/Documents/ai assisted/bank.py"

Choose an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice: 1
Enter deposit amount: 2000
✅ Deposited 2000.0. Current balance: 2000.0

Choose an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice: 2
Enter withdrawal amount: -100
⚠️Error: Withdrawal amount must be positive.

Choose an option:
1. Deposit
2. Withdraw
3. Check Balance
4. Exit
Enter your choice: 2
Enter withdrawal amount: 3000
⚠️Error: Insufficient funds.
```

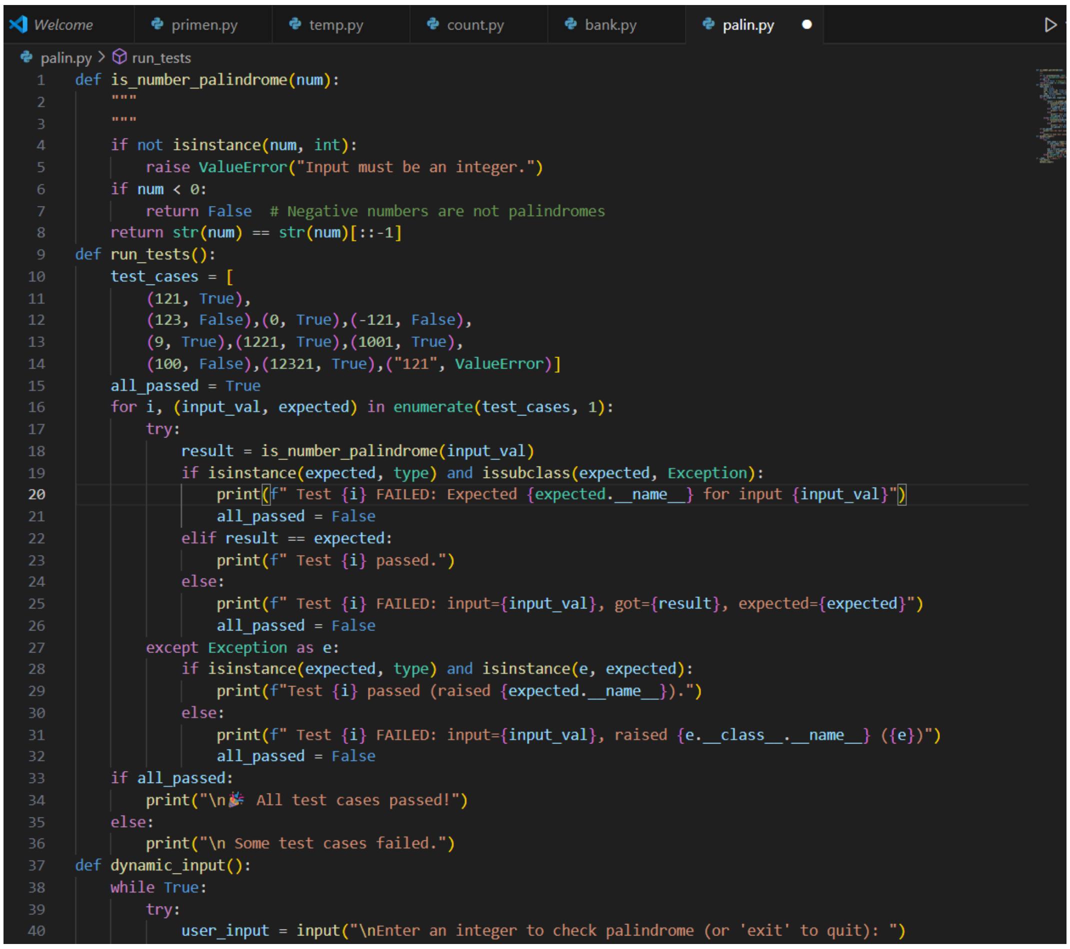
Observation:

- Clear class design – deposit, withdraw, and check_balance methods follow good OOP principles.
- Input validation – negative deposits/withdrawals and over-withdrawals correctly raise errors.
- Comprehensive test cases – covers valid transactions, invalid inputs, and edge cases.
- Readable error handling in tests – shows whether expected results or exceptions occurred.
- Dynamic input menu – allows interactive deposits, withdrawals, and balance checks.

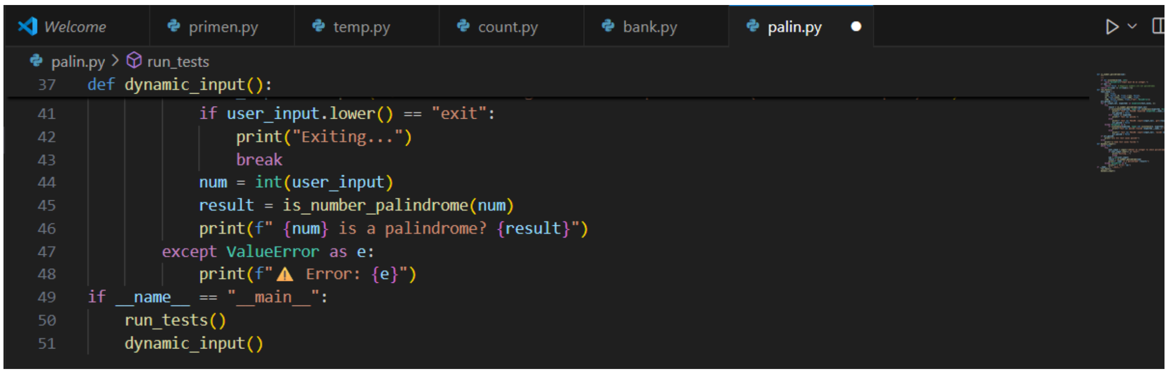
Task#5

Prompt: Generate test cases for is_number_palindrome(num), which checks if an integer reads the same backward.

Code generated:

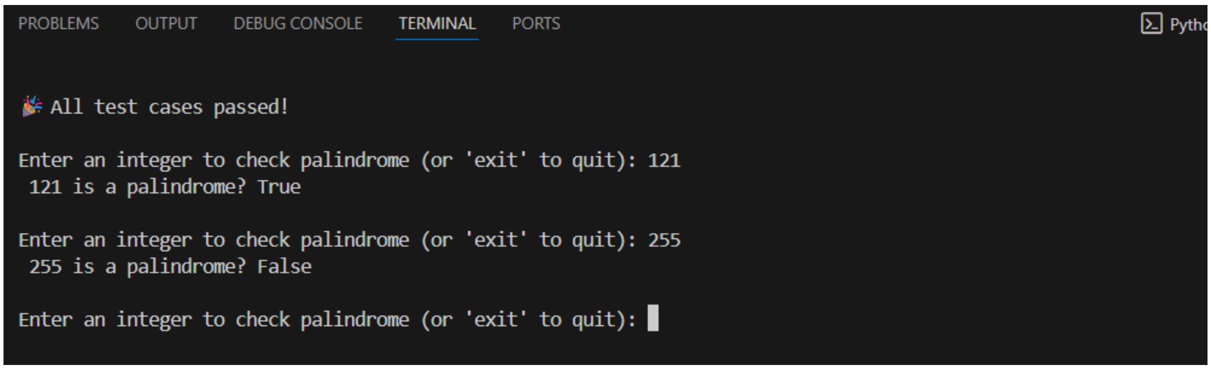


```python
def is_number_palindrome(num):
    """
    """
    if not isinstance(num, int):
        raise ValueError("Input must be an integer.")
    if num < 0:
        return False  # Negative numbers are not palindromes
    return str(num) == str(num)[::-1]
def run_tests():
    test_cases = [
        (121, True),
        (123, False),(0, True),(-121, False),
        (9, True),(1221, True),(1001, True),
        (100, False),(12321, True),("121", ValueError)]
    all_passed = True
    for i, (input_val, expected) in enumerate(test_cases, 1):
        try:
            result = is_number_palindrome(input_val)
            if isinstance(expected, type) and issubclass(expected, Exception):
                print(f" Test {i} FAILED: Expected {expected.__name__} for input {input_val}")
                all_passed = False
            elif result == expected:
                print(f" Test {i} passed.")
            else:
                print(f" Test {i} FAILED: input={input_val}, got={result}, expected={expected}")
                all_passed = False
        except Exception as e:
            if isinstance(expected, type) and isinstance(e, expected):
                print(f"Test {i} passed (raised {expected.__name__}).")
            else:
                print(f" Test {i} FAILED: input={input_val}, raised {e.__class__.__name__} ({e})")
                all_passed = False
    if all_passed:
        print("\n🎉 All test cases passed!")
    else:
        print("\n Some test cases failed.")
def dynamic_input():
    while True:
        try:
            user_input = input("\nEnter an integer to check palindrome (or 'exit' to quit): ")
```



```python
def dynamic_input():
            if user_input.lower() == "exit":
                print("Exiting...")
                break
            num = int(user_input)
            result = is_number_palindrome(num)
            print(f" {num} is a palindrome? {result}")
        except ValueError as e:
            print(f"⚠ Error: {e}")
if __name__ == "__main__":
    run_tests()
    dynamic_input()
```

Output:



```
🎉 All test cases passed!

Enter an integer to check palindrome (or 'exit' to quit): 121
 121 is a palindrome? True

Enter an integer to check palindrome (or 'exit' to quit): 255
 255 is a palindrome? False

Enter an integer to check palindrome (or 'exit' to quit): 
```

Observation:

- Correctly handles positives, negatives, 0, and single-digit numbers.
- Good error handling for non-integer input.
- Test cases cover edge cases and expected failures.
- Dynamic input makes it interactive**.**