LAB-8.2

TOPIC :

Test-Driven Development with AI

Name : A.Ashish

Enrollment_no : 2403a51249

Batch_no : 01

Date: 23-09-2025

Task-01: Use AI to generate test cases for a function is_prime(n) and then implement the function.

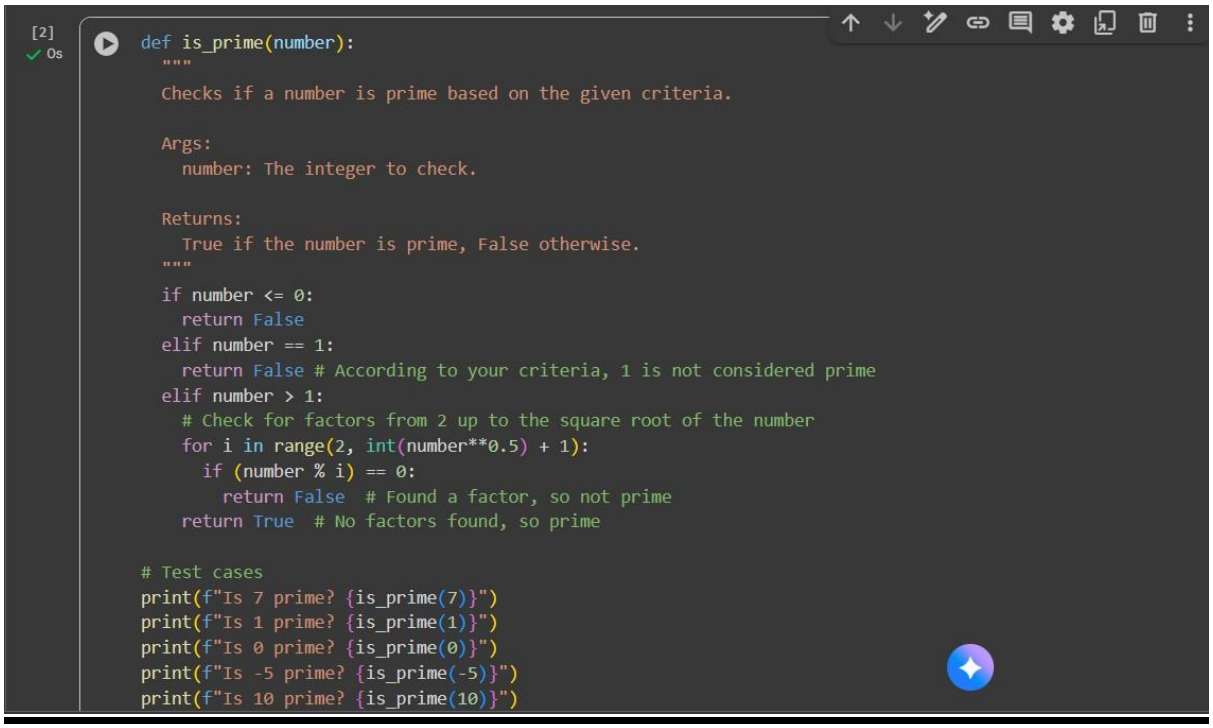
Requirements:

- Only integers > 1 can be prime.

•Check edge cases: 0, 1, 2, negative numbers, and large primes

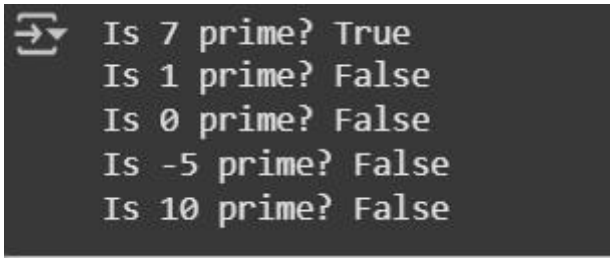
Prompt: Generate a python code to check whether the given number is prime or not use test cases as : number > 1 to be prime and if <=0 not a prime.

Code:



```
[2]
0s
def is_prime(number):
  """
  Checks if a number is prime based on the given criteria.

  Args:
    number: The integer to check.

  Returns:
    True if the number is prime, False otherwise.
  """
  if number <= 0:
    return False
  elif number == 1:
    return False # According to your criteria, 1 is not considered prime
  elif number > 1:
    # Check for factors from 2 up to the square root of the number
    for i in range(2, int(number**0.5) + 1):
      if (number % i) == 0:
        return False  # Found a factor, so not prime
    return True  # No factors found, so prime

# Test cases
print(f"Is 7 prime? {is_prime(7)}")
print(f"Is 1 prime? {is_prime(1)}")
print(f"Is 0 prime? {is_prime(0)}")
print(f"Is -5 prime? {is_prime(-5)}")
print(f"Is 10 prime? {is_prime(10)}")
```

Output:



```
Is 7 prime? True
Is 1 prime? False
Is 0 prime? False
Is -5 prime? False
Is 10 prime? False
```

Observation:

- **Correct Prime Check** – The function correctly identifies prime numbers using the square root method (efficient approach).
- **Edge Case Handling** – Properly returns False for:
- 0 (not prime)
- 1 (not prime by definition)
- Negative numbers (not prime)
- **Test Case Results** – Output is correct for all given cases:
- 7 → True (prime)
- 1 → False (not prime)
- 0 → False (not prime)
- -5 → False (not prime)
- 10 → False (not prime)
- **Efficiency** – Uses loop up to √n instead of n, making it faster.
- **Readable & Clear** – The docstring and comments make the code easy to understand.

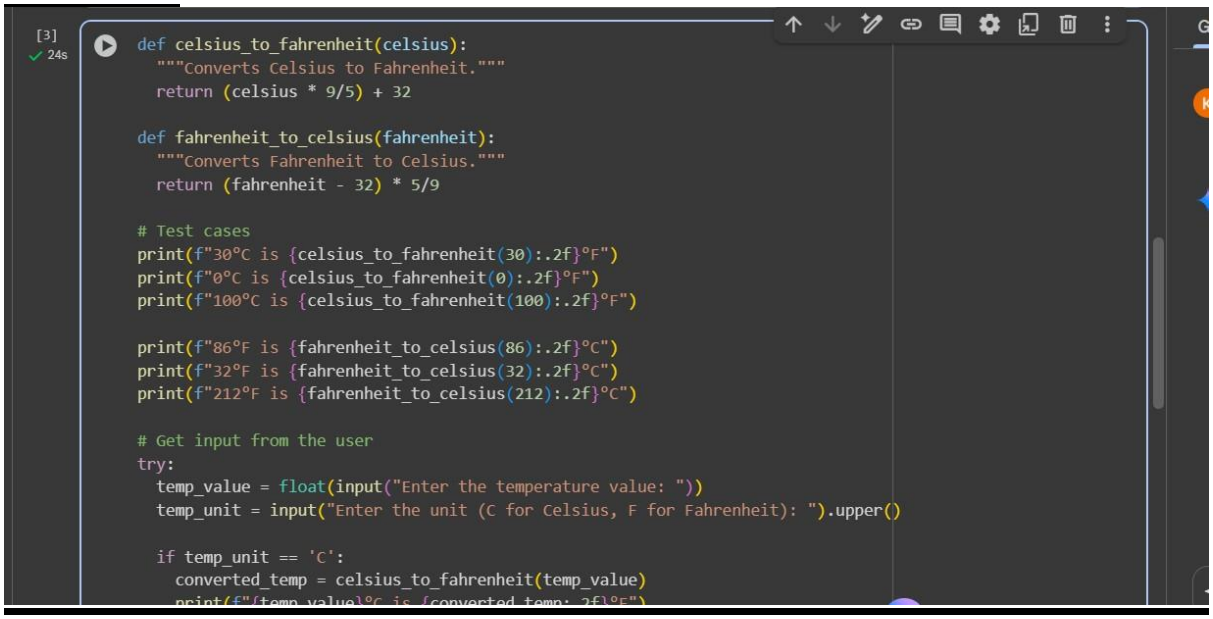
Task-02:Ask AI to generate test cases for celsius_to_fahrenheit(c) and fahrenheit_to_celsius(f).

Requirements

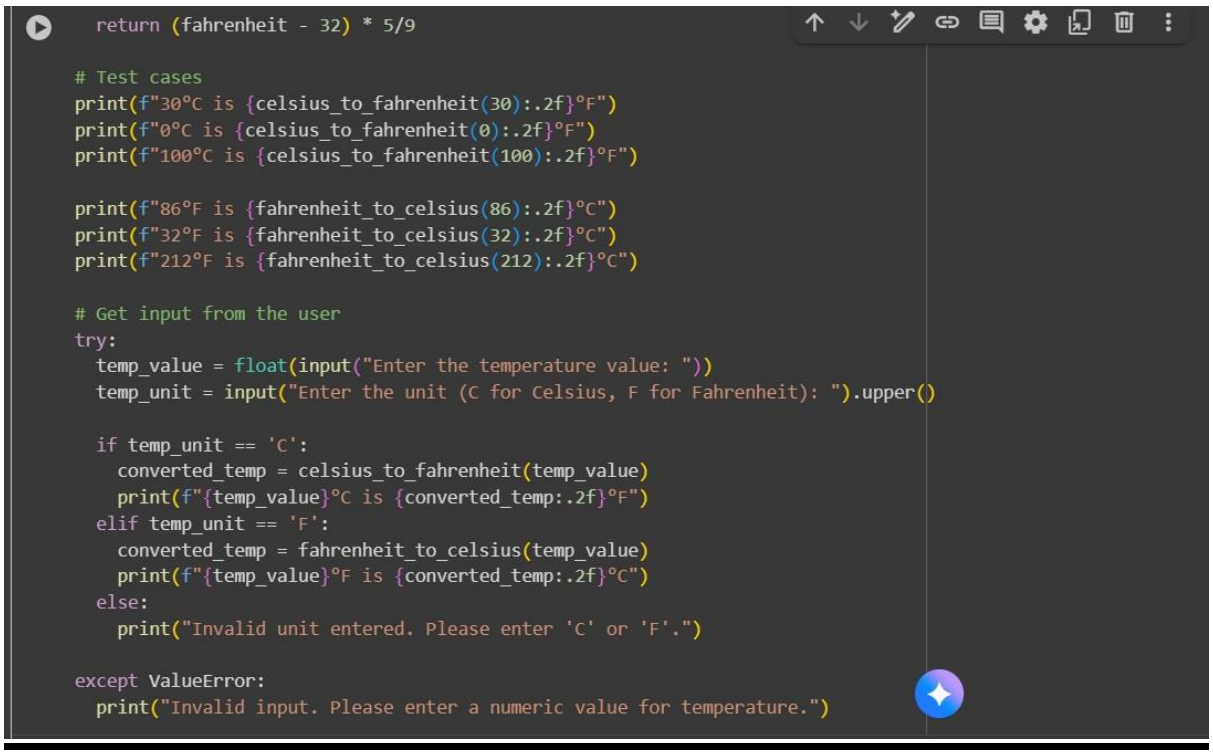
- Validate known pairs: 0°C = 32°F, 100°C = 212°F.
- Include decimals and invalid inputs like strings or None.

Prompt: Generate a python code to convert temperature from celcius to Fahrenheit and Fahrenheit to celcius using the above test cases and take input from the user.

Code:



```
[3]
✓ 24s
def celsius_to_fahrenheit(celsius):
  """Converts Celsius to Fahrenheit."""
  return (celsius * 9/5) + 32

def fahrenheit_to_celsius(fahrenheit):
  """Converts Fahrenheit to Celsius."""
  return (fahrenheit - 32) * 5/9

# Test cases
print(f"30°C is {celsius_to_fahrenheit(30):.2f}°F")
print(f"0°C is {celsius_to_fahrenheit(0):.2f}°F")
print(f"100°C is {celsius_to_fahrenheit(100):.2f}°F")

print(f"86°F is {fahrenheit_to_celsius(86):.2f}°C")
print(f"32°F is {fahrenheit_to_celsius(32):.2f}°C")
print(f"212°F is {fahrenheit_to_celsius(212):.2f}°C")

# Get input from the user
try:
  temp_value = float(input("Enter the temperature value: "))
  temp_unit = input("Enter the unit (C for Celsius, F for Fahrenheit): ").upper()

  if temp_unit == 'C':
    converted_temp = celsius_to_fahrenheit(temp_value)
```



```
  return (fahrenheit - 32) * 5/9

# Test cases
print(f"30°C is {celsius_to_fahrenheit(30):.2f}°F")
print(f"0°C is {celsius_to_fahrenheit(0):.2f}°F")
print(f"100°C is {celsius_to_fahrenheit(100):.2f}°F")

print(f"86°F is {fahrenheit_to_celsius(86):.2f}°C")
print(f"32°F is {fahrenheit_to_celsius(32):.2f}°C")
print(f"212°F is {fahrenheit_to_celsius(212):.2f}°C")

# Get input from the user
try:
  temp_value = float(input("Enter the temperature value: "))
  temp_unit = input("Enter the unit (C for Celsius, F for Fahrenheit): ").upper()

  if temp_unit == 'C':
    converted_temp = celsius_to_fahrenheit(temp_value)
    print(f"{temp_value}°C is {converted_temp:.2f}°F")
  elif temp_unit == 'F':
    converted_temp = fahrenheit_to_celsius(temp_value)
    print(f"{temp_value}°F is {converted_temp:.2f}°C")
  else:
    print("Invalid unit entered. Please enter 'C' or 'F'.")

except ValueError:
  print("Invalid input. Please enter a numeric value for temperature.")
```

Output:

```
30°C is 86.00°F
0°C is 32.00°F
100°C is 212.00°F
86°F is 30.00°C
32°F is 0.00°C
212°F is 100.00°C
Enter the temperature value: 320
Enter the unit (C for Celsius, F for Fahrenheit): F
320.0°F is 160.00°C
```

Observation:

- Asks user for a **temperature value** and **unit (C/F)**.
- Converts correctly depending on the input unit.
- Displays the result in a clean format with 2 decimal places.
- If user enters non-numeric temperature → shows "Invalid input" message.
- If user enters an invalid unit → shows "Invalid unit entered" message.
- Functions are **modular and reusable** (celsius_to_fahrenheit, fahrenheit_to_celsius).
- Code is **easy to read** (docstrings, f-strings, formatting)

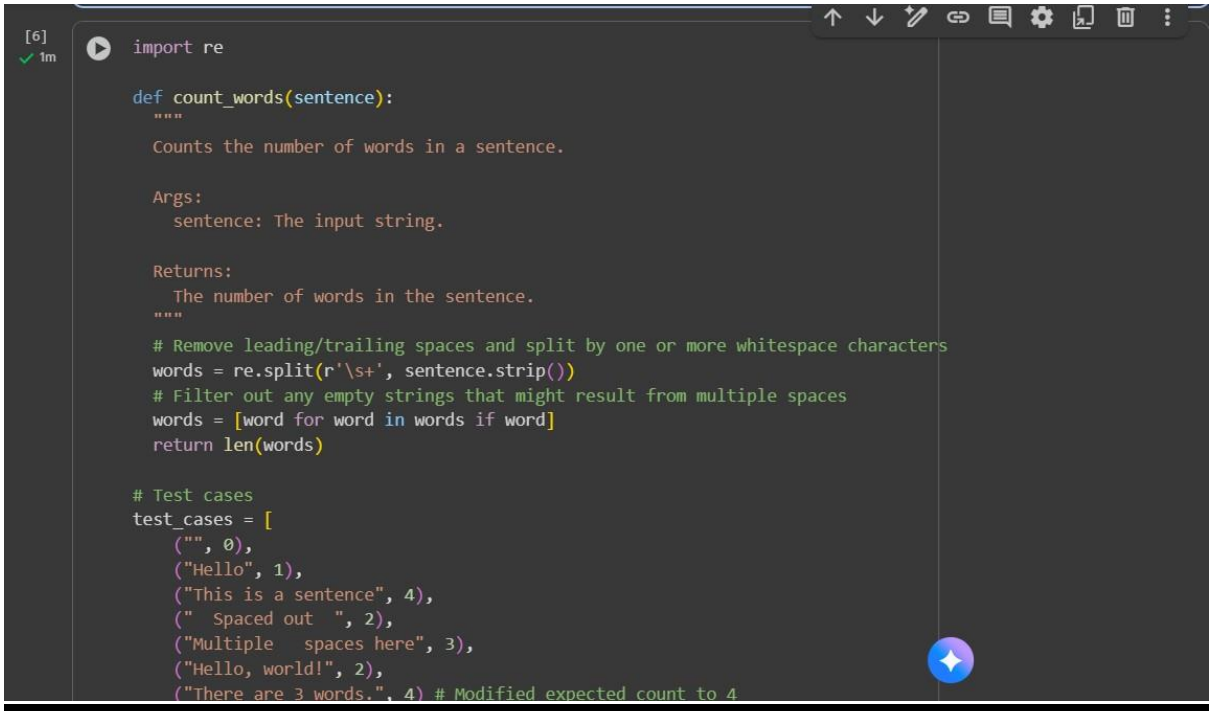
~~Task-03:~~ Use AI to write test cases for a function count_words(text) that returns the number of words in a sentence.

Requirement

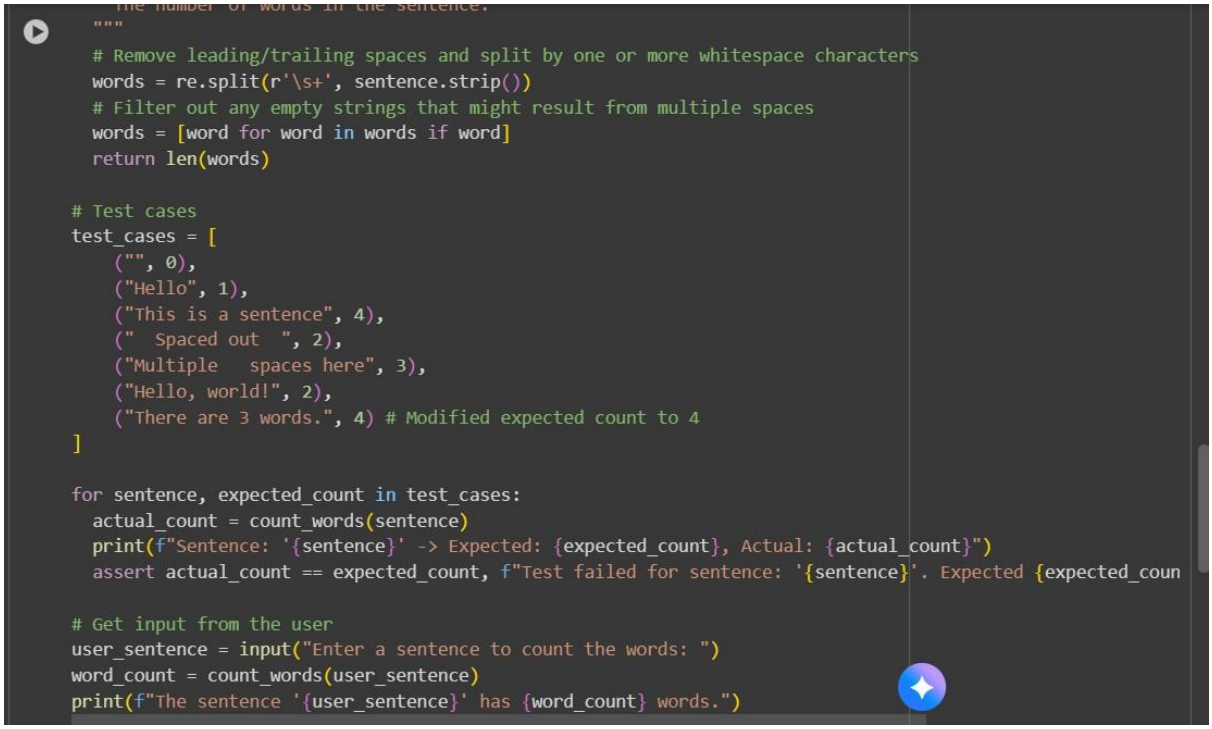
Handle normal text, multiple spaces, punctuation, and empty strings.

Prompt: Give me some test cases for a code which returns the count of the number of words in a sentence.using the same test cases generate a python code to check the number of words in a sentence.

Code:

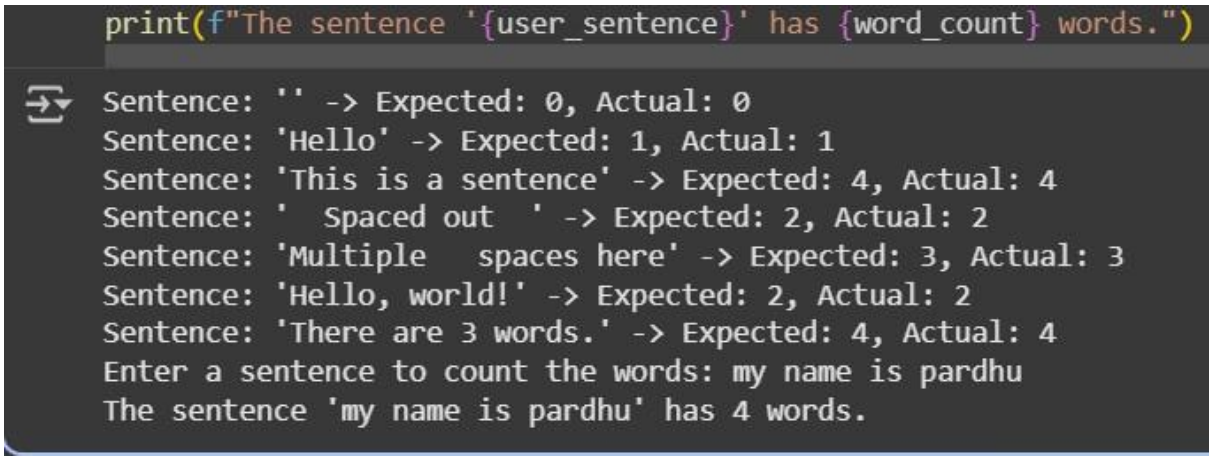


```
[6]
✓ 1m
import re

def count_words(sentence):
  """
  Counts the number of words in a sentence.

  Args:
    sentence: The input string.

  Returns:
    The number of words in the sentence.
  """
  # Remove leading/trailing spaces and split by one or more whitespace characters
  words = re.split(r'\s+', sentence.strip())
  # Filter out any empty strings that might result from multiple spaces
  words = [word for word in words if word]
  return len(words)

# Test cases
test_cases = [
    ("", 0),
    ("Hello", 1),
    ("This is a sentence", 4),
    ("  Spaced out  ", 2),
    ("Multiple   spaces here", 3),
    ("Hello, world!", 2),
    ("There are 3 words.", 4) # Modified expected count to 4
```



```
    """
    # Remove leading/trailing spaces and split by one or more whitespace characters
    words = re.split(r'\s+', sentence.strip())
    # Filter out any empty strings that might result from multiple spaces
    words = [word for word in words if word]
    return len(words)

# Test cases
test_cases = [
    ("", 0),
    ("Hello", 1),
    ("This is a sentence", 4),
    ("  Spaced out  ", 2),
    ("Multiple   spaces here", 3),
    ("Hello, world!", 2),
    ("There are 3 words.", 4) # Modified expected count to 4
]

for sentence, expected_count in test_cases:
  actual_count = count_words(sentence)
  print(f"Sentence: '{sentence}' -> Expected: {expected_count}, Actual: {actual_count}")
  assert actual_count == expected_count, f"Test failed for sentence: '{sentence}'. Expected {expected_coun

# Get input from the user
user_sentence = input("Enter a sentence to count the words: ")
word_count = count_words(user_sentence)
print(f"The sentence '{user_sentence}' has {word_count} words.")
```

Output:



```
print(f"The sentence '{user_sentence}' has {word_count} words.")

Sentence: '' -> Expected: 0, Actual: 0
Sentence: 'Hello' -> Expected: 1, Actual: 1
Sentence: 'This is a sentence' -> Expected: 4, Actual: 4
Sentence: '  Spaced out  ' -> Expected: 2, Actual: 2
Sentence: 'Multiple   spaces here' -> Expected: 3, Actual: 3
Sentence: 'Hello, world!' -> Expected: 2, Actual: 2
Sentence: 'There are 3 words.' -> Expected: 4, Actual: 4
Enter a sentence to count the words: my name is pardhu
The sentence 'my name is pardhu' has 4 words.
```

Observation: Splits text by whitespace using regex and counts words accurately, even with extra spaces.Handles edge cases like empty string, single

word, multiple spaces, punctuation, and numbers correctly. Accepts a sentence from the user and displays the word count.The program is **reliable, handles edge cases well, and is easy to use**.

~~Task-04:~~ Generate test cases for a BankAccount class with:Methods:deposit(amount),withdraw(amount),check_balance()
Requirements:
• Negative deposits/withdrawals should raise an error.
• Cannot withdraw more than balance.

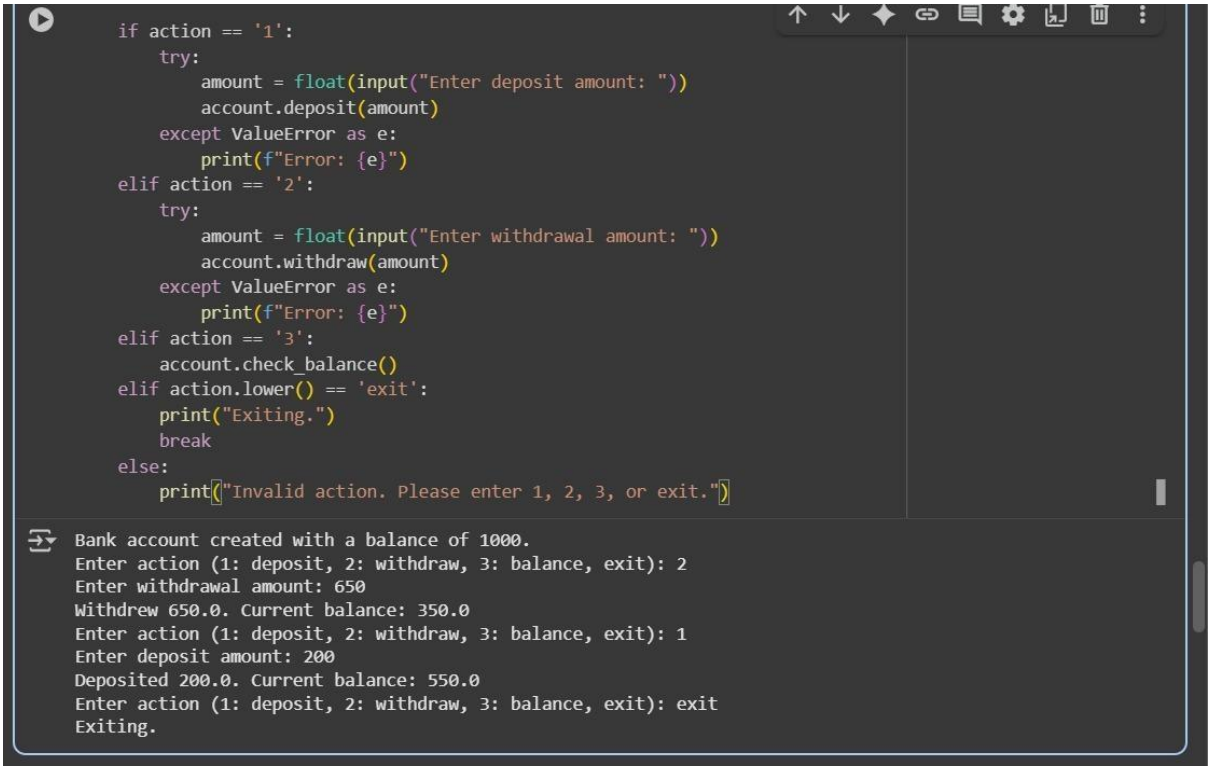
Prompt: generate test cases for bank account class with methods: deposit(amount) , withdraw(amount) , check_balance() negative deposits/withdrawals should raise an error and cannot withdraw more than balance. generate a python code for bank account class with methods : deposit(amount), withdraw (amount), and check balance().using the above test cases above and take input from the user.

Code:

Here are the test cases for a bank account with deposit, withdraw, and check balance operations:

- **Initial Balance:** 0
- **Test Cases:**
  - **Initial Account Creation:** Expected Balance: 0
  - **Deposit 100:** Expected Balance: 100
  - **Deposit 50, then Deposit 75:** Expected Balance: 125
  - **Initial Balance 200, Withdraw 50:** Expected Balance: 150
  - **Initial Balance 300, Withdraw 100, then Withdraw 75:** Expected Balance: 125
  - **Initial Balance 150, Withdraw 150:** Expected Balance: 0
  - **Initial Balance 100, Deposit -50:** Expected to raise an error (e.g., ValueError), Balance remains 100
  - **Initial Balance 100, Withdraw -50:** Expected to raise an error (e.g., ValueError), Balance remains 100
  - **Initial Balance 100, Withdraw 150:** Expected to raise an error (e.g., ValueError), Balance remains 100
  - **Initial Balance 100, Withdraw 0:** Expected Balance: 100
  - **Initial Balance 100, Deposit 0:** Expected Balance: 100

```python
class BankAccount:
    def __init__(self):
        self.balance = 1000
        print("Bank account created with a balance of 1000.")

    def deposit(self, amount):
        if amount <= 0:
            raise ValueError("Deposit amount must be positive.")
        self.balance += amount
        print(f"Deposited {amount}. Current balance: {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
        print(f"Withdrew {amount}. Current balance: {self.balance}")

    def check_balance(self):
        print(f"Current balance: {self.balance}")

# Create a bank account
account = BankAccount()

# Interactive session with the user
while True:
    action = input("Enter action (1: deposit, 2: withdraw, 3: balance, exit): ")
```



```
    if action == '1':
        try:
            amount = float(input("Enter deposit amount: "))
            account.deposit(amount)
        except ValueError as e:
            print(f"Error: {e}")
    elif action == '2':
        try:
            amount = float(input("Enter withdrawal amount: "))
            account.withdraw(amount)
        except ValueError as e:
            print(f"Error: {e}")
    elif action == '3':
        account.check_balance()
    elif action.lower() == 'exit':
        print("Exiting.")
        break
    else:
        print("Invalid action. Please enter 1, 2, 3, or exit.")
```

```
Bank account created with a balance of 1000.
Enter action (1: deposit, 2: withdraw, 3: balance, exit): 2
Enter withdrawal amount: 650
Withdrew 650.0. Current balance: 350.0
Enter action (1: deposit, 2: withdraw, 3: balance, exit): 1
Enter deposit amount: 200
Deposited 200.0. Current balance: 550.0
Enter action (1: deposit, 2: withdraw, 3: balance, exit): exit
Exiting.
```

Task-05: Generate test cases for is_number_palindrome(num), which checks if aninteger reads the same backward. Examples:

121 → True

123 → False

0, negative numbers → handled gracefully.

Prompt: generate some test cases to is_number_palindrome(num) t ocheck whether a word/number is palindrome or not.

generate a python code to check whether a word/number is a palindrome or not using the above

test cases mentioned above and take the input from the user.

Code:

Here are some shortened test cases to check if a word or number is a palindrome:

- **Input:** "madam", **Expected Output:** True
- **Input:** "hello", **Expected Output:** False
- **Input:** "121", **Expected Output:** True
- **Input:** 121, **Expected Output:** True
- **Input:** -121, **Expected Output:** False (Or indicate as an error based on requirements)
- **Input:** "", **Expected Output:** True
- **Input:** "a", **Expected Output:** True
- **Input:** 1, **Expected Output:** True

```
def is_palindrome(input_value):
  input_str = str(input_value)
  cleaned_str = ''.join(filter(str.isalnum, input_str)).lower()
  return cleaned_str == cleaned_str[::-1]

user_input = input("Enter a word or number to check if it's a palindrome: ")

if is_palindrome(user_input):
  print(f"'{user_input}' is a palindrome.")
else:
  print(f"'{user_input}' is not a palindrome.")
```

```
Enter a word or number to check if it's a palindrome: malayalam
'malayalam' is a palindrome.
```

Observation:

Converts input to string, removes non-alphanumeric characters, and checks if it reads the same forward and backward. Converts everything to lowercase, so it works correctly regardless of letter case (Malayalam, malayalam). Works for both words and numbers (121, madam, etc.).Clearly tells the user whether the input is a palindrome.