AI ASSITED CODING

ENROLLMENT NUMBER:2403A51289

NAME OF THE STUDENT:

METTU SEELVA VARSHITH REDDY

BATCH :01

ASSIGNMENT :8.2

Description 1:

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 function that finds whether a user defined number is Prime or not .Generate the test cases for it and check the code as per the test cases

TEST CASES:

[32]
✓ 0s

```
test_cases = [
    (2, True),
    (3, True),
    (5, True),
    (7, True),
    (11, True),
    (13, True),
    (4, False),
    (6, False),
    (8, False),
    (9, False),
    (10, False),
    (12, False),
    (0, False),
    (1, False),
    (17, True),
    (19, True),
    (20, False),
    (21, False),
    (23, True),
    (25, False),
]
```

Code:

[33]
✓ 0s

```
results = []
for number, expected_output in test_cases:
  actual_output = is_prime_greater_than_one(number)
  results.append({
      'input': number,
      'expected_output': expected_output,
      'actual_output': actual_output,
      'passed': actual_output == expected_output
  })

for result in results:
  print(
      f"Input: {result['input']}, Expected: {result['expected_output']},"
      f" Actual: {result['actual_output']}, Passed:"
      f" {result['passed']}"
  )
```

```
Please provide a number greater than 1 to check for primality.
Please provide a number greater than 1 to check for primality.
Input: 2, Expected: True, Actual: True, Passed: True
Input: 3, Expected: True, Actual: True, Passed: True
Input: 5, Expected: True, Actual: True, Passed: True
Input: 7, Expected: True, Actual: True, Passed: True
Input: 11, Expected: True, Actual: True, Passed: True
Input: 13, Expected: True, Actual: True, Passed: True
Input: 4, Expected: False, Actual: False, Passed: True
Input: 6, Expected: False, Actual: False, Passed: True
Input: 8, Expected: False, Actual: False, Passed: True
Input: 9, Expected: False, Actual: False, Passed: True
Input: 10, Expected: False, Actual: False, Passed: True
Input: 12, Expected: False, Actual: False, Passed: True
```

OUTPUT:

```
Please provide a number greater than 1 to check for primality.
Please provide a number greater than 1 to check for primality.
Input: 2, Expected: True, Actual: True, Passed: True
Input: 3, Expected: True, Actual: True, Passed: True
Input: 5, Expected: True, Actual: True, Passed: True
Input: 7, Expected: True, Actual: True, Passed: True
Input: 11, Expected: True, Actual: True, Passed: True
Input: 13, Expected: True, Actual: True, Passed: True
Input: 4, Expected: False, Actual: False, Passed: True
Input: 6, Expected: False, Actual: False, Passed: True
Input: 8, Expected: False, Actual: False, Passed: True
Input: 9, Expected: False, Actual: False, Passed: True
Input: 10, Expected: False, Actual: False, Passed: True
Input: 12, Expected: False, Actual: False, Passed: True
Input: 0, Expected: False, Actual: False, Passed: True
Input: 1, Expected: False, Actual: False, Passed: True
Input: 17, Expected: True, Actual: True, Passed: True
Input: 19, Expected: True, Actual: True, Passed: True
Input: 20, Expected: False, Actual: False, Passed: True
Input: 21, Expected: False, Actual: False, Passed: True
Input: 23, Expected: True, Actual: True, Passed: True
Input: 25, Expected: False, Actual: False, Passed: True
```

Observation:

I have asked the gemini first to generate the test cases then it has implemented all those given test cases as and resulted as true or false based upon the number that we are giving

Description 2

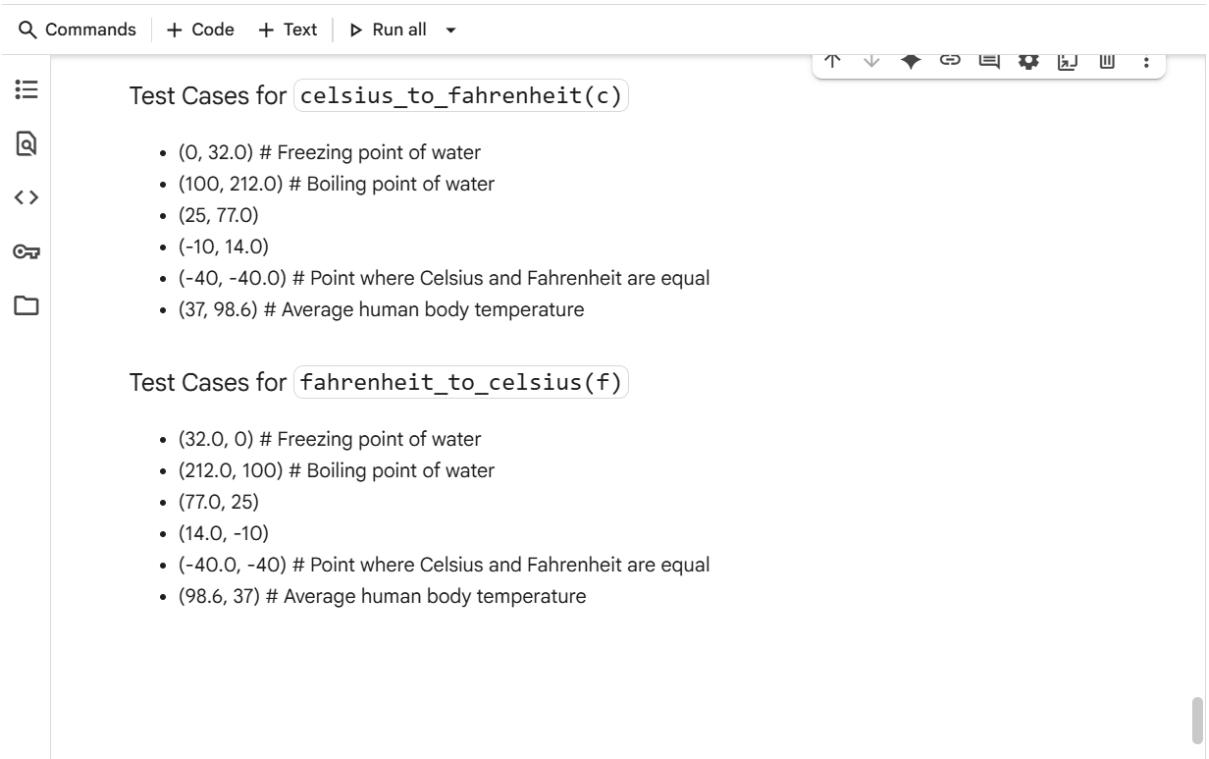
Ask AI to generate test cases for celsius_to_fahrenheit(c) and fahrenheit_to_celsius(f).

Requirements

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

PROMPT:

generate the test cases to convert from celsius_to_fahrenheit(c) and fahrenheit_to_celsius(f).Don't give any code only test cases



PROMPT:

enerate a python function that validates above generated test cases

[39]
✓ 0s

```python
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
```

[40]
✓ 0s

```python
celsius_test_cases = [
    (0, 32.0),  # Freezing point of water
    (100, 212.0), # Boiling point of water
    (25, 77.0),
    (-10, 14.0),
    (-40, -40.0), # Point where Celsius and Fahrenheit are equal
    (37, 98.6) # Average human body temperature
]

print("Testing celsius_to_fahrenheit:")
validate_test_cases(celsius_test_cases, celsius_to_fahrenheit)
```

```python
fahrenheit_test_cases = [
    (32.0, 0), # Freezing point of water
    (212.0, 100), # Boiling point of water
    (77.0, 25),
    (14.0, -10),
    (-40.0, -40), # Point where Celsius and Fahrenheit are equal
    (98.6, 37) # Average human body temperature
]

print("\nTesting fahrenheit_to_celsius:")
validate_test_cases(fahrenheit_test_cases, fahrenheit_to_celsius)
```

```
Testing fahrenheit_to_celsius:
Input: 32.0, Expected: 0, Actual: 0.0, Passed: True
Input: 212.0, Expected: 100, Actual: 100.0, Passed: True
Input: 77.0, Expected: 25, Actual: 25.0, Passed: True
Input: 14.0, Expected: -10, Actual: -10.0, Passed: True
Input: -40.0, Expected: -40, Actual: -40.0, Passed: True
Input: 98.6, Expected: 37, Actual: 37.0, Passed: True
True
```

Observation:

Task Description -3

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

PROMPT:

Test Cases for Counting Words

Here are test cases for a function that counts the number of words in a string:

- ("Hello world", 2)
- ("This is a sentence.", 4)
- (" One two three ", 3) # Leading/trailing and multiple spaces
- ("", 0) # Empty string
- ("SingleWord", 1)
- (" ", 0) # Only spaces
- ("Hello,world!", 1) # Punctuation attached to words
- ("Hello, world!", 2) # Punctuation with spaces
- (" Hello world ", 2) # Mixed spacing

PROMPT:

generate a python code to count number of words in a given string where string is given

by user dynamically

```python
import re

def count_words(text):
  """Counts the number of words in a string."""
  # Use regular expression to find words (sequences of non-whitespace characters)
  words = re.findall(r'\S+', text)
  return len(words)

# Get input from the user dynamically
user_string = input("Enter a string or sentence: ")

# Call the function with user input
word_count = count_words(user_string)

# Print the result
print(f"The number of words in the string is: {word_count}")
```

```
Enter a string or sentence: hello world
The number of words in the string is: 2
```

OUTPUT:

```
Enter a string or sentence: this is a sentence
The number of words in the string is: 4
```

Task Description:

Generate test cases for a BankAccount class with:

Methods:

deposit(amount)

withdraw(amount)

check_balance()

PROMPT:

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

```
deposit_test_cases = [
    (100.0, 50.0, 150.0),  # Positive deposit
    (0.0, 100.0, 100.0),  # Positive deposit to zero balance
    (50.0, 0.0, 50.0),    # Depositing zero
    (75.0, -20.0, 75.0),   # Depositing negative amount (should not change balance)
    (1000.0, 250.0, 1250.0), # Another positive deposit
]
```

Generate test cases for `withdraw`

Subtask:

Create test cases for the `withdraw` method, including withdrawing valid amounts, withdrawing amounts exceeding the balance, and potentially edge cases like withdrawing zero or negative amounts.

Reasoning: Create a list of tuples to store the test cases for the withdraw method, including different scenarios as specified in the instructions.

[47] ✓ 0s

```
withdraw_test_cases = [
    (100.0, 50.0, 50.0),  # Valid withdrawal
    (100.0, 100.0, 0.0),  # Valid withdrawal equal to balance
    (50.0, 100.0, 50.0),  # Withdrawal exceeding balance
    (100.0, 0.0, 100.0),  # Withdrawing zero
    (100.0, -20.0, 100.0), # Withdrawing a negative amount
]
```

Generate test cases for `check balance`

Subtask:

Create simple test cases to check the balance after a series of deposits and withdrawals.

Reasoning: Create a list of test cases for checking the balance after a sequence of operations, including deposits, withdrawals, and combinations, as well as edge cases like no operations.

[48] ✓ 0s

```
check_balance_test_cases = [
    ([('deposit', 100.0), ('withdraw', 50.0)], 50.0),  # Deposit then withdraw
    ([('deposit', 200.0), ('deposit', 50.0)], 250.0), # Multiple deposits
    ([('withdraw', 50.0),)], 0.0), # Withdrawal exceeding initial zero balance
    ([], 0.0), # No operations
    ([('deposit', 1000.0), ('withdraw', 200.0), ('withdraw', 300.0), ('deposit', 50.0)], 550.0), # Mixed
    ([('deposit', 0.0), ('withdraw', 0.0)], 0.0), # Zero deposit and withdrawal
    ([('deposit', -100.0), ('withdraw', -50.0)], 0.0), # Negative deposit and withdrawal
]
```

PROMPT:

```python
deposit_test_results = []
for initial_balance, deposit_amount, expected_balance in deposit_test_cases:
    account = BankAccount(initial_balance)
    account.deposit(deposit_amount)
    actual_balance = account.check_balance()
    deposit_test_results.append({
        'initial_balance': initial_balance,
        'deposit_amount': deposit_amount,
        'expected_balance': expected_balance,
        'actual_balance': actual_balance,
        'passed': actual_balance == expected_balance
    })

withdraw_test_results = []
for initial_balance, withdraw_amount, expected_balance in withdraw_test_cases:
    account = BankAccount(initial_balance)
    account.withdraw(withdraw_amount)
    actual_balance = account.check_balance()
    withdraw_test_results.append({
        'initial_balance': initial_balance,
        'withdraw_amount': withdraw_amount,
        'expected_balance': expected_balance,
        'actual_balance': actual_balance,
        'passed': actual_balance == expected_balance
    })

check_balance_test_results = []
for operations_list, expected_final_balance in check_balance_test_cases:
    account = BankAccount(0.0)
    for operation, amount in operations_list:
```