

SR UNIVERSITY

AI ASSISTED CODING

ASSIGNMENT - 08

Name: Meer Burhan Ali Hashmi

Htno: 2503A51L44

Batch :20

Lab Objectives:

- To introduce students to test-driven development (TDD) using AI code generation tools.
- To enable the generation of test cases before writing code implementations.
- To reinforce the importance of testing, validation, and error handling.
- To encourage writing clean and reliable code based on AI-generated test expectations.

Lab Outcomes (LOs):

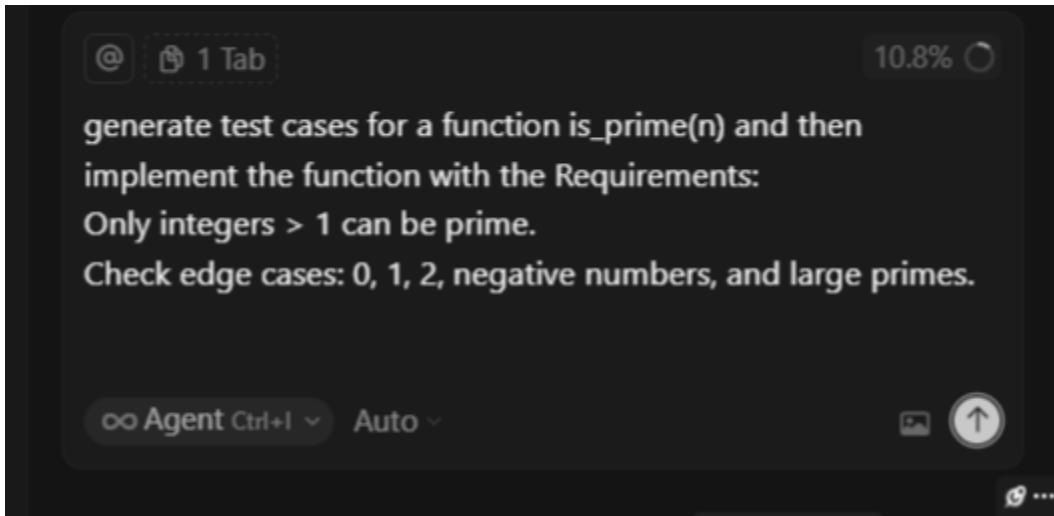
After completing this lab, students will be able to:

- Use AI tools to write test cases for Python functions and classes.
- Implement functions based on test cases in a test-first development style.
- Use unittest or pytest to validate code correctness.
- Analyze the completeness and coverage of AI-generated tests.
- Compare AI-generated and manually written test cases for quality and logic

Task 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.

Entered prompt into the AI:



Given program by the AI:

```
prime_tests.py > ...
1 import unittest
2 import math
3
4 def is_prime(n):
5     """
6         Check if a number is prime.
7
8         Args:
9             n (int): The number to check
10
11        Returns:
12            bool: True if n is prime, False otherwise
13
14        Requirements:
15            - Only integers > 1 can be prime
16            - Handles edge cases: 0, 1, 2, negative numbers, and large primes
17        """
18
19        # Handle non-integer inputs
20        if not isinstance(n, int):
21            return False
22
23        # Handle edge cases: 0, 1, and negative n
24        if n <= 1:
25            return False
26
27        # Implement the prime check logic here
28
29    if n < 2:
30        return False
31
32    for i in range(2, int(n**0.5) + 1):
33        if n % i == 0:
34            return False
35
36    return True
```

```
prime_tests.py > ...
4  def is_prime(n):
5      if n <= 1:
6          return False
7
8      # 2 is the smallest prime number
9      if n == 2:
10         return True
11
12     # Even numbers > 2 are not prime
13     if n % 2 == 0:
14         return False
15
16     # Check odd divisors up to sqrt(n)
17     for i in range(3, int(math.sqrt(n)) + 1, 2):
18         if n % i == 0:
19             return False
20
21     return True
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42 class TestIsPrime(unittest.TestCase):
43     """Test cases for the is_prime function"""
44
```

```
prime_tests.py > ...
41
42 class TestIsPrime(unittest.TestCase):
43     """Test cases for the is_prime function"""
44
45
46     def test_negative_numbers(self):
47         """Test that negative numbers are not prime"""
48         self.assertFalse(is_prime(-1))
49         self.assertFalse(is_prime(-2))
50         self.assertFalse(is_prime(-10))
51         self.assertFalse(is_prime(-100))
52
53     def test_zero_and_one(self):
54         """Test that 0 and 1 are not prime"""
55         self.assertFalse(is_prime(0))
56         self.assertFalse(is_prime(1))
57
58     def test_small_primes(self):
59         """Test small prime numbers"""
60         self.assertTrue(is_prime(2))
61         self.assertTrue(is_prime(3))
62         self.assertTrue(is_prime(5))
63         self.assertTrue(is_prime(7))
64         self.assertTrue(is_prime(11))
```

```
❖ prime_tests.py > ...
42  class TestIsPrime(unittest.TestCase):
57      def test_small_primes(self):
64          self.assertTrue(is_prime(13))
65          self.assertTrue(is_prime(17))
66          self.assertTrue(is_prime(19))
67          self.assertTrue(is_prime(23))
68          self.assertTrue(is_prime(29))
69
70      def test_small_composite_numbers(self):
71          """Test small composite numbers"""
72          self.assertFalse(is_prime(4))
73          self.assertFalse(is_prime(6))
74          self.assertFalse(is_prime(8))
75          self.assertFalse(is_prime(9))
76          self.assertFalse(is_prime(10))
77          self.assertFalse(is_prime(12))
78          self.assertFalse(is_prime(14))
79          self.assertFalse(is_prime(15))
80          self.assertFalse(is_prime(16))
81          self.assertFalse(is_prime(18))
82          self.assertFalse(is_prime(20))
83          self.assertFalse(is_prime(21))
84          self.assertFalse(is_prime(22))
```

Review next file >

```
❖ prime_tests.py > ...
42  class TestIsPrime(unittest.TestCase):
70      def test_small_composite_numbers(self):
85          self.assertFalse(is_prime(24))
86          self.assertFalse(is_prime(25))
87
88      def test_medium_primes(self):
89          """Test medium-sized prime numbers"""
90          self.assertTrue(is_prime(31))
91          self.assertTrue(is_prime(37))
92          self.assertTrue(is_prime(41))
93          self.assertTrue(is_prime(43))
94          self.assertTrue(is_prime(47))
95          self.assertTrue(is_prime(53))
96          self.assertTrue(is_prime(59))
97          self.assertTrue(is_prime(61))
98          self.assertTrue(is_prime(67))
99          self.assertTrue(is_prime(71))
100         self.assertTrue(is_prime(73))
101         self.assertTrue(is_prime(79))
102         self.assertTrue(is_prime(83))
103         self.assertTrue(is_prime(89))
104         self.assertTrue(is_prime(97))
```

Review next file >

```
= even_sum, odd_sum = sum_even_and_odd[1], Untitled-1.py • ❖ prime_tests.py ✘
❖ prime_tests.py > ...
42  class TestIsPrime(unittest.TestCase):
105
106      def test_medium_composite_numbers(self):
107          """Test medium-sized composite numbers"""
108          self.assertFalse(is_prime(30))
109          self.assertFalse(is_prime(32))
110          self.assertFalse(is_prime(33))
111          self.assertFalse(is_prime(34))
112          self.assertFalse(is_prime(35))
113          self.assertFalse(is_prime(36))
114          self.assertFalse(is_prime(38))
115          self.assertFalse(is_prime(39))
116          self.assertFalse(is_prime(40))
117          self.assertFalse(is_prime(42))
118          self.assertFalse(is_prime(44))
119          self.assertFalse(is_prime(45))
120          self.assertFalse(is_prime(46))
121          self.assertFalse(is_prime(48))
122          self.assertFalse(is_prime(49))
123          self.assertFalse(is_prime(50))
124
125      def test_large_primes(self):
126          """Test large prime numbers"""


```

Review next file >

A screenshot of a code editor showing a Python test file named `prime_tests.py`. The file contains a class `TestIsPrime` that inherits from `unittest.TestCase`. It includes several test methods for large prime numbers, each using `assertTrue` or `assertFalse` to check if the `is_prime` function returns the expected result. The code is well-formatted with line numbers and color-coded syntax highlighting.

```
even_sum, odd_sum = sum_even_and_odd([1, Untitled-2 ● ● prime_tests.py X])  
prime_tests.py > ...  
42  class TestIsPrime(unittest.TestCase):  
125      def test_large_primes(self):  
126          # These are known large primes  
127          self.assertTrue(is_prime(101))  
128          self.assertTrue(is_prime(103))  
129          self.assertTrue(is_prime(107))  
130          self.assertTrue(is_prime(109))  
131          self.assertTrue(is_prime(113))  
132          self.assertTrue(is_prime(127))  
133          self.assertTrue(is_prime(131))  
134          self.assertTrue(is_prime(137))  
135          self.assertTrue(is_prime(139))  
136          self.assertTrue(is_prime(149))  
137          self.assertTrue(is_prime(151))  
138          self.assertTrue(is_prime(157))  
139          self.assertTrue(is_prime(163))  
140          self.assertTrue(is_prime(167))  
141          self.assertTrue(is_prime(173))  
142          self.assertTrue(is_prime(179))  
143          self.assertTrue(is_prime(181))  
144          self.assertTrue(is_prime(191))  
145          self.assertTrue(is_prime(193))  
146          self.assertTrue(is_prime(197))  
147  
Review next file >  
Problems Output Debug Console Terminal Ports
```

A screenshot of a code editor showing the continuation of the `prime_tests.py` file. It includes additional test methods for various scenarios:

- `test_very_large_numbers`: Tests large numbers (1000 to 1010) and asserts that they are not prime using `assertFalse`.
- `test_perfect_squares`: Tests perfect squares (4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225) and asserts that they are not prime using `assertFalse`.
- `test_non_integer_inputs`: Tests non-integer inputs (2.0) and asserts that they are not prime using `assertFalse`.

```
prime_tests.py > ...  
42  class TestIsPrime(unittest.TestCase):  
169      def test_very_large_numbers(self):  
170          self.assertFalse(is_prime(1000))  
171          self.assertFalse(is_prime(1001))  
172          self.assertFalse(is_prime(1002))  
173          self.assertFalse(is_prime(1003))  
174          self.assertFalse(is_prime(1004))  
175          self.assertFalse(is_prime(1005))  
176          self.assertFalse(is_prime(1006))  
177          self.assertFalse(is_prime(1007))  
178          self.assertFalse(is_prime(1008))  
179          self.assertFalse(is_prime(1010))  
180  
195      def test_perfect_squares(self):  
196          """Test perfect squares (should not be prime)"""  
197          self.assertFalse(is_prime(4)) # 2^2  
198          self.assertFalse(is_prime(9)) # 3^2  
199          self.assertFalse(is_prime(16)) # 4^2  
200          self.assertFalse(is_prime(25)) # 5^2  
201          self.assertFalse(is_prime(36)) # 6^2  
202          self.assertFalse(is_prime(49)) # 7^2  
203          self.assertFalse(is_prime(64)) # 8^2  
204          self.assertFalse(is_prime(81)) # 9^2  
205          self.assertFalse(is_prime(100)) # 10^2  
206          self.assertFalse(is_prime(121)) # 11^2  
207          self.assertFalse(is_prime(144)) # 12^2  
208          self.assertFalse(is_prime(169)) # 13^2  
209          self.assertFalse(is_prime(196)) # 14^2  
210          self.assertFalse(is_prime(225)) # 15^2  
211  
212      def test_non_integer_inputs(self):  
213          """Test that non-integer inputs return  
214          self.assertFalse(is_prime(2.0))  
Review next file >
```

```
prime_tests.py > ...
42  class TestIsPrime(unittest.TestCase):
227      def test_known_mersenne_primes(self):
234          #  $2^5 - 1 = 31$  (prime)
235          self.assertTrue(is_prime(31))
236          #  $2^7 - 1 = 127$  (prime)
237          self.assertTrue(is_prime(127))
238
239
240 def run_performance_test():
241     """Run a simple performance test on large numbers"""
242     import time
243
244     print("Running performance test...")
245
246     # Test with a Large prime
247     start_time = time.time()
248     result = is_prime(1000003) # This is a known prime
249     end_time = time.time()
250
251     print(f"is_prime(1000003) = {result} (took {end_time - start_time:.6f} seconds)")
252
253     # Test with a large composite
254     start_time = time.time()
255     result = is_prime(1000000) # This is composite ( $1000^2$ )
256     end_time = time.time()
257
258     print(f"is_prime(1000000) = {result} (took {end_time - start_time:.6f} seconds)")
259
260
261 if __name__ == "__main__":
262     # Run the unit tests
263     print("Running unit tests...")
264     unittest.main(verbosity=2, exit=False)
```

Review next file >

```
prime_tests.py > ...
240     def run_performance_test():
241
242         print(f"is_prime(1000000) = {result} (took {end_time - start_time:.6f} seconds)")
243
244
245 if __name__ == "__main__":
246     # Run the unit tests
247     print("Running unit tests...")
248     unittest.main(verbosity=2, exit=False)
249
250
251     # Run performance test
252     print("\n" + "*50")
253     run_performance_test()
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
```

Output:

```
def run_performance_test():
    Problems Output Debug Console Terminal Ports
test_edge_case_two (__main__.TestIsPrime.test_edge_case_two)
Test the special case of 2 (smallest prime) ... ok
test_known_mersenne_primes (__main__.TestIsPrime.test_known_mersenne_primes)
Test some known Mersenne primes ( $2^p - 1$  where p is prime) ... ok
test_large_composite_numbers (__main__.TestIsPrime.test_large_composite_numbers)
Test large composite numbers ... ok
test_large_primes (__main__.TestIsPrime.test_large_primes)
Test large prime numbers ... ok
test_medium_composite_numbers (__main__.TestIsPrime.test_medium_composite_numbers)
Test medium-sized composite numbers ... ok
test_medium_primes (__main__.TestIsPrime.test_medium_primes)
Test medium-sized prime numbers ... ok
test_negative_numbers (__main__.TestIsPrime.test_negative_numbers)
Test that negative numbers are not prime ... ok
test_non_integer_inputs (__main__.TestIsPrime.test_non_integer_inputs)
Test that non-integer inputs return False ... ok
test_perfect_squares (__main__.TestIsPrime.test_perfect_squares)
Test perfect squares (should not be prime) ... ok
test_small_composite_numbers (__main__.TestIsPrime.test_small_composite_numbers)
Test small composite numbers ... ok
test_small_primes (__main__.TestIsPrime.test_small_primes)
Test small prime numbers ... ok
test_very_large_numbers (__main__.TestIsPrime.test_very_large_numbers)
Test very large numbers (both prime and composite) ... ok
test_zero_and_one (__main__.TestIsPrime.test_zero_and_one)
Test that 0 and 1 are not prime ... ok

-----
Ran 13 tests in 0.005s
OK
-----
Ran 13 tests in 0.005s
...
Later Install Now
```

Observation:

The output displays the results of a Python script named `def run_performance_test()`. The script is likely a unit test suite for a function that determines if a number is prime. The output shows that all **13 tests** passed successfully in **0.005 seconds**.

Task Description#2 (Loops)

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

Requirements

- Validate known pairs: $0^\circ\text{C} = 32^\circ\text{F}$, $100^\circ\text{C} = 212^\circ\text{F}$.

- Include decimals and invalid inputs like strings or None

Given prompt to the AI:

> 1 File Undo All Keep All ⌘E

@Add Context 16.8% ⌂

generate test cases for celsius_to_fahrenheit(c) and fahrenheit_to_celsius(f) with the requirements of Validate known pairs: 0°C = 32°F, 100°C = 212°F. Include decimals and invalid inputs like strings or None

Given program by the AI:

```
temperature_tests.py > celsius_to_fahrenheit
1 import unittest
2
3
4 def celsius_to_fahrenheit(c):
5     """
6         Convert Celsius to Fahrenheit.
7
8     Args:
9         c (float): Temperature in Celsius
10
11    Returns:
12        float: Temperature in Fahrenheit
13
14    Formula: F = (C * 9/5) + 32
15
16    Requirements:
17        - Validate known pairs: 0°C = 32°F, 100°C = 212°F
18        - Handle decimals and invalid inputs like strings or None
19    """
20
21    # Handle invalid inputs
22    if c is None:
23        raise ValueError("Invalid input: None is not a valid temperature")
24
25    # Check for boolean values (True/False are instances of int in Python)
26    if isinstance(c, bool):
27        raise ValueError(f"Invalid input: {c} is not a valid temperature")
28
29    # Check for non-numeric types
30    if not isinstance(c, (int, float)):
31        raise ValueError(f"Invalid input: {c} is not a number")
32
33    # Convert to float to handle both int and float inputs
34    celsius = float(c)
```

```

def celsius_to_fahrenheit(c):
    # Convert to float to handle both int and float inputs
    celsius = float(c)

    # Apply the conversion formula
    fahrenheit = (celsius * 9/5) + 32

    return fahrenheit

def fahrenheit_to_celsius(f):
    """
    Convert Fahrenheit to Celsius.

    Args:
        f (float): Temperature in Fahrenheit

    Returns:
        float: Temperature in Celsius

    Formula: C = (F - 32) * 5/9

    Requirements:
        - Validate known pairs: 32°F = 0°C, 212°F = 100°C
        - Handle decimals and invalid inputs like strings or None
    """
    # Handle invalid inputs
    if f is None:
        raise ValueError("Invalid input: None is not a valid temperature")
    # Check for boolean values (True/False are Review next file > int in Python)
    if isinstance(f, bool):

```

```

41 def fahrenheit_to_celsius(f):
42     # Check for non-numeric types
43     if not isinstance(f, (int, float)):
44         raise ValueError(f"Invalid input: {f} is not a number")
45
46     # Convert to float to handle both int and float inputs
47     fahrenheit = float(f)
48
49     # Apply the conversion formula
50     celsius = (fahrenheit - 32) * 5/9
51
52     return celsius
53
54
55
56 class TestTemperatureConversion(unittest.TestCase):
57     """Test cases for temperature conversion functions"""
58
59     def test_celsius_to_fahrenheit_known_pairs(self):
60         """Test known temperature conversion pairs"""
61         # 0°C = 32°F
62         self.assertAlmostEqual(celsius_to_fahrenheit(0), 32.0, places=5)
63
64         # 100°C = 212°F
65         self.assertAlmostEqual(celsius_to_fahrenheit(100), 212.0, places=5)
66
67         # -40°C = -40°F (special case where both scales are equal)
68         self.assertAlmostEqual(celsius_to_fahrenheit(-40), -40.0, places=5)
69
70     def test_fahrenheit_to_celsius_known_pairs(self):
71         """Test known temperature conversion pairs"""
72         # 32°F = 0°C
73         self.assertAlmostEqual(fahrenheit_to_celsius(32), 0.0, places=5)

```

```

❖ temperature_tests.py > ⓘ celsius_to_fahrenheit
78   class TestTemperatureConversion(unittest.TestCase):
79       def test_fahrenheit_to_celsius_known_pairs(self):
80           # -40°F = -40°C (special case where both scales are equal)
81           self.assertAlmostEqual(fahrenheit_to_celsius(-40), -40.0, places=5)
82
83       def test_celsius_to_fahrenheit_round_trip(self):
84           """Test that converting C->F->C returns original value"""
85           test_temperatures = [-50, -20, 0, 20, 37, 50, 100, 150]
86
87           for celsius in test_temperatures:
88               fahrenheit = celsius_to_fahrenheit(celsius)
89               back_to_celsius = fahrenheit_to_celsius(fahrenheit)
90               self.assertAlmostEqual(back_to_celsius, celsius, places=5,
91                                     msg=f"Round trip failed for {celsius}°C")
92
93       def test_fahrenheit_to_celsius_round_trip(self):
94           """Test that converting F->C->F returns original value"""
95           test_temperatures = [-58, -4, 32, 68, 98.6, 122, 212, 302]
96
97           for fahrenheit in test_temperatures:
98               celsius = fahrenheit_to_celsius(fahrenheit)
99               back_to_fahrenheit = celsius_to_fahrenheit(celsius)
100              self.assertAlmostEqual(back_to_fahrenheit, fahrenheit, places=5,
101                                     msg=f"Round trip failed for {fahrenheit}°F")
102
103       def test_celsius_to_fahrenheit_decimal_inputs(self):
104           """Test Celsius to Fahrenheit conversion with decimal inputs"""
105           # Test various decimal inputs
106           test_cases = [
107               (36.5, 97.7),      # Body temperature
108               (25.5, 77.9),      # Room temperature
109               (-18.5, -1.3),     # Freezer temperi Review next file >
110               (0.1, 32.18),       # Just above freezing
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130

```

```

❖ temperature_tests.py > ⓘ celsius_to_fahrenheit
78   class TestTemperatureConversion(unittest.TestCase):
79       def test_celsius_to_fahrenheit_decimal_inputs(self):
80           (37.2, 98.96),    # Fever temperature
81           (100.5, 212.9),   # Boiling point + 0.5
82           (-273.15, -459.67) # Absolute zero
83       ]
84
85       for celsius, expected_fahrenheit in test_cases:
86           with self.subTest(celsius=celsius):
87               result = celsius_to_fahrenheit(celsius)
88               self.assertAlmostEqual(result, expected_fahrenheit, places=2)
89
90       def test_fahrenheit_to_celsius_decimal_inputs(self):
91           """Test Fahrenheit to Celsius conversion with decimal inputs"""
92           # Test various decimal inputs
93           test_cases = [
94               (98.6, 37.0),      # Body temperature
95               (77.0, 25.0),      # Room temperature
96               (0.0, -17.78),     # Freezing point in F
97               (32.0, 0.0),       # Freezing point
98               (212.0, 100.0),    # Boiling point
99               (451.0, 232.78),   # Paper burning point
100              (-459.67, -273.15) # Absolute zero
101          ]
102
103          for fahrenheit, expected_celsius in test_cases:
104              with self.subTest(fahrenheit=fahrenheit):
105                  result = fahrenheit_to_celsius(fahrenheit)
106                  self.assertAlmostEqual(result, expected_celsius, places=2)
107
108
109
110       def test_celsius_to_fahrenheit_integer_inputs(self):
111           """Test Celsius to Fahrenheit conversion with integer inputs"""
112           test_cases = [

```

```

temperature_tests.py > celsius_to_fahrenheit
78  class TestTemperatureConversion(unittest.TestCase):
160      def test_celsius_to_fahrenheit_integer_inputs(self):
161          test_cases = [
162              (20, 68.0),
163              (37, 98.6),
164              (100, 212.0),
165              (-20, -4.0),
166              (-40, -40.0),
167              (50, 122.0)
168          ]
169
170
171      for celsius, expected_fahrenheit in test_cases:
172          with self.subTest(celsius=celsius):
173              result = celsius_to_fahrenheit(celsius)
174              self.assertAlmostEqual(result, expected_fahrenheit, places=1)
175
176
177  def test_fahrenheit_to_celsius_integer_inputs(self):
178      """Test Fahrenheit to Celsius conversion with integer inputs"""
179      test_cases = [
180          (32, 0.0),
181          (68, 20.0),
182          (98, 36.67),
183          (212, 100.0),
184          (-4, -20.0),
185          (-40, -40.0),
186          (122, 50.0)
187      ]
188
189      for fahrenheit, expected_celsius in test_cases:
190          with self.subTest(fahrenheit=fahrenheit):
191              result = fahrenheit_to_celsius(fahrenheit)
192              self.assertAlmostEqual(result, expected_celsius, places=1)
193
```

```

temperature_tests.py > celsius_to_fahrenheit
78  class TestTemperatureConversion(unittest.TestCase):
194      def test_celsius_to_fahrenheit_invalid_string_inputs(self):
195          """Test that string inputs raise ValueError for Celsius to Fahrenheit"""
196          invalid_inputs = ["25", "hello", "25.5", "", "abc123", "25°C"]
197
198          for invalid_input in invalid_inputs:
199              with self.subTest(input=invalid_input):
200                  with self.assertRaises(ValueError):
201                      celsius_to_fahrenheit(invalid_input)
202
203  def test_fahrenheit_to_celsius_invalid_string_inputs(self):
204      """Test that string inputs raise ValueError for Fahrenheit to Celsius"""
205      invalid_inputs = ["77", "hello", "77.0", "", "abc123", "77°F"]
206
207          for invalid_input in invalid_inputs:
208              with self.subTest(input=invalid_input):
209                  with self.assertRaises(ValueError):
210                      fahrenheit_to_celsius(invalid_input)
211
212  def test_celsius_to_fahrenheit_none_input(self):
213      """Test that None input raises ValueError for Celsius to Fahrenheit"""
214      with self.assertRaises(ValueError):
215          celsius_to_fahrenheit(None)
216
217  def test_fahrenheit_to_celsius_none_input(self):
218      """Test that None input raises ValueError for Fahrenheit to Celsius"""
219      with self.assertRaises(ValueError):
220          fahrenheit_to_celsius(None)
221
222  def test_celsius_to_fahrenheit_other_invalid_inputs(self):
223      """Test other invalid input types for Celsius to Fahrenheit"""
224      invalid_inputs = [True, False, [], {}, None]
225
```

```

# temperature_tests.py > ⓘ celsius_to_fahrenheit
78  class TestTemperatureConversion(unittest.TestCase):
79      def test_fahrenheit_to_celsius_other_invalid_inputs(self):
80          """Test other invalid input types for Fahrenheit to Celsius"""
81          invalid_inputs = [True, False, [], {}, [77], {"temp": 77}]
82
83          for invalid_input in invalid_inputs:
84              with self.subTest(input=invalid_input):
85                  with self.assertRaises(ValueError):
86                      fahrenheit_to_celsius(invalid_input)
87
88
89      def test_celsius_to_fahrenheit_extreme_values(self):
90          """Test Celsius to Fahrenheit conversion with extreme values"""
91          # Very cold temperatures
92          self.assertAlmostEqual(celsius_to_fahrenheit(-273.15), -459.67, places=2) # Absolute zero
93          self.assertAlmostEqual(celsius_to_fahrenheit(-200), -328.0, places=1)
94          self.assertAlmostEqual(celsius_to_fahrenheit(-100), -148.0, places=1)
95
96
97          # Very hot temperatures
98          self.assertAlmostEqual(celsius_to_fahrenheit(1000), 1832.0, places=1)
99          self.assertAlmostEqual(celsius_to_fahrenheit(5000), 9032.0, places=1)
100
101
102      def test_fahrenheit_to_celsius_extreme_values(self):
103          """Test Fahrenheit to Celsius conversion with extreme values"""
104          # Very cold temperatures
105          self.assertAlmostEqual(fahrenheit_to_celsius(-459.67), -273.15, places=2) # Absolute zero
106          self.assertAlmostEqual(fahrenheit_to_celsius(-328), -200.0, places=1)
107          self.assertAlmostEqual(fahrenheit_to_celsius(-148), -100.0, places=1)
108
109
110          # Very hot temperatures
111          self.assertAlmostEqual(fahrenheit_to_celsius(1832), 1000.0, places=1)
112          self.assertAlmostEqual(fahrenheit_to_celsius(9032), 5000.0, places=1)
113
114
115      def test_celsius_to_fahrenheit_zero_and_negative_zero(self):
116          """Test handling of zero and negative zero"""
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193

```

```

# temperature_tests.py > ⓘ celsius_to_fahrenheit
178  class TestTemperatureConversion(unittest.TestCase):
179      def test_celsius_to_fahrenheit_zero_and_negative_zero(self):
180          """Test handling of zero and negative zero"""
181          self.assertAlmostEqual(celsius_to_fahrenheit(0), 32.0, places=5)
182          self.assertAlmostEqual(celsius_to_fahrenheit(-0), 32.0, places=5)
183          self.assertAlmostEqual(celsius_to_fahrenheit(0.0), 32.0, places=5)
184          self.assertAlmostEqual(celsius_to_fahrenheit(-0.0), 32.0, places=5)
185
186
187      def test_fahrenheit_to_celsius_zero_and_negative_zero(self):
188          """Test handling of zero and negative zero"""
189          self.assertAlmostEqual(fahrenheit_to_celsius(32), 0.0, places=5)
190          self.assertAlmostEqual(fahrenheit_to_celsius(32.0), 0.0, places=5)
191
192
193      def test_celsius_to_fahrenheit_precision(self):
194          """Test precision of Celsius to Fahrenheit conversion"""
195          # Test with high precision decimal
196          result = celsius_to_fahrenheit(25.123456789)
197          expected = 77.2222222202
198          self.assertAlmostEqual(result, expected, places=8)
199
200
201      def test_fahrenheit_to_celsius_precision(self):
202          """Test precision of Fahrenheit to Celsius conversion"""
203          # Test with high precision decimal
204          result = fahrenheit_to_celsius(77.2222222202)
205          expected = 25.123456789
206          self.assertAlmostEqual(result, expected, places=8)
207
208
209
210      def run_temperature_demo():
211          """Run a demonstration of temperature conversions"""
212          print("Temperature Conversion Demo")
213          print("=" * 40)
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293

```

```
temperature_tests.py > celsius_to_fahrenheit
59 def run_temperature_demo():
60     # Common temperature conversions
61     temperatures_c = [-40, -20, 0, 20, 25, 37, 100]
62     temperatures_f = [-40, -4, 32, 68, 77, 98.6, 212]
63
64     print("Celsius to Fahrenheit:")
65     for c in temperatures_c:
66         f = celsius_to_fahrenheit(c)
67         print(f"\t{c:3}°C = {f:6.1f}°F")
68
69     print("\nFahrenheit to Celsius:")
70     for f in temperatures_f:
71         c = fahrenheit_to_celsius(f)
72         print(f"\t{f:6.1f}°F = {c:3.1f}°C")
73
74     print("\nRound-trip conversions:")
75     test_temp = 25.5
76     f_temp = celsius_to_fahrenheit(test_temp)
77     c_temp = fahrenheit_to_celsius(f_temp)
78     print(f"\t{test_temp}°C -> {f_temp:.1f}°F -> {c_temp:.1f}°C")
79
80     # Test error handling
81     print("\nError handling examples:")
82     try:
83         celsius_to_fahrenheit("25")
84     except ValueError as e:
85         print(f"\tError: {e}")
86
87     try:
88         fahrenheit_to_celsius(None)
89     except ValueError as e:
90         print(f"\tError: {e}")
```

Review next file >

```
325
326
327 if __name__ == "__main__":
328     # Run the unit tests
329     print("Running temperature conversion unit tests...")
330     unittest.main(verbosity=2, exit=False)
331
332     # Run demonstration
333     print("\n" + "*"*60)
334     run_temperature_demo()
335
```

Output:

```
=====
Temperature Conversion Demo
=====
Celsius to Fahrenheit:
-40°C = -40.0°F
-20°C = -4.0°F
0°C = 32.0°F
20°C = 68.0°F
25°C = 77.0°F
37°C = 98.6°F
100°C = 212.0°F

Fahrenheit to Celsius:
-40.0°F = -40.0°C
-4.0°F = -20.0°C
32.0°F = 0.0°C
68.0°F = 20.0°C
77.0°F = 25.0°C
98.6°F = 37.0°C
212.0°F = 100.0°C

Round-trip conversions:
25.5°C -> 77.9°F -> 25.5°C

Error handling examples:
Error: Invalid input: 25 is not a number
Error: Invalid input: None is not a valid temperature
PS C:\Users\musta\cursor>
```

Observation:

The output displays the Temperature Conversion Demo script. This program is a simple utility that demonstrates temperature conversions between Celsius and Fahrenheit

Task Description#3

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.

Given prompt to

write test cases for a function count_words(text) that returns the number of words in a sentence add the requirement
Requirement Handle normal text, multiple spaces, punctuation, and empty strings.

AI:

Given code by AI:

```
❷ count_words_short.py > ...
1  import unittest
2
3
4  def count_words(text):
5      """Count words in text, handling spaces, punctuation, and empty strings."""
6      if not text or not isinstance(text, str):
7          return 0
8
9      # Split by whitespace and filter valid words (containing alphanumeric chars)
10     words = [word.strip('.,!?:;()[]{}') for word in text.split()]
11     return len([word for word in words if word and any(c.isalnum() for c in word)])
12
13
14 class TestCountWords(unittest.TestCase):
15     """Test cases for count_words function"""
16
17     def test_basic_cases(self):
18         """Test basic functionality"""
19         self.assertEqual(count_words("Hello world"), 2)
20         self.assertEqual(count_words("The quick brown fox"), 4)
21         self.assertEqual(count_words(""), 0)
22         self.assertEqual(count_words("    "), 0)
23         self.assertEqual(count_words(None), 0)
24         self.assertEqual(count_words(123), 0)
25
26     def test_spaces_and_punctuation(self):
27         """Test multiple spaces and punctuation"""
28         self.assertEqual(count_words("Hello    world"), 2)
29         self.assertEqual(count_words("Hello, world!"), 2)
30         self.assertEqual(count_words("Hello, world!"), 2)
31         self.assertEqual(count_words("Hello, world!"), 2)
32         self.assertEqual(count_words("    "), 0)
```

```
⌚ count_words_short.py > ...
14 class TestCountWords(unittest.TestCase):
15     def test_special_cases(self):
16         """Test special cases"""
17         self.assertEqual(count_words("I'm happy"), 2)
18         self.assertEqual(count_words("state-of-the-art"), 1)
19         self.assertEqual(count_words("Hello 世界"), 2)
20         self.assertEqual(count_words("I have 5 apples"), 4)
21         self.assertEqual(count_words("Just had an amazing day! #summer #fun"), 7)
22         self.assertEqual(count_words("don't"), 1)
23         self.assertEqual(count_words("123"), 1)
24
25
26     def demo():
27         """Quick demo of the function"""
28         test_cases = [
29             "Hello world",
30             "The quick brown fox",
31             "Python is awesome!",
32             "This is a test sentence",
33             "Hello, how are you?",
34             "I'm happy today!",
35             "state-of-the-art technology",
36             "Hello 世界",
37             "",
38             " ",
39             "123",
40             None
41         ]
42
43         print("Word Count Demo: ")
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
```

⌚ count_words_short.py X ▶ ⓘ ...

undo all Ctrl+Shift+Z Keep all Ctrl+Shift+Y

```
even_sum, odd_sum = sum_even_and_odd([1, Untitled-2] • ⌚ Untitled-1.py • ⌚ count_words_tests.py • ⌚ count_words_short.py X ▶ ⓘ ...)
⌚ count_words_short.py > ...
46 def demo():
47
48     print("Word Count Demo:")
49     for text in test_cases:
50         count = count_words(text)
51         print(f'{text}' + " -> " + str(count) + " words")
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
```

⌚ count_words_short.py X ▶ ⓘ ...

Undo Ctrl+N Keep Ctrl+Shift+Y

Output:

```
Problems Output Debug Console Terminal Ports + ... ^  
=====Word Count Demo:  
'Hello world' -> 2 words  
'The quick brown fox' -> 4 words  
'Python is awesome!' -> 3 words  
'This is a test sentence' -> 5 words  
'Hello, how are you?' -> 4 words  
'I'm happy today!' -> 3 words  
'state-of-the-art technology' -> 2 words  
'Hello 世界' -> 2 words  
'' -> 0 words  
' ' -> 0 words  
'123' -> 1 words  
'None' -> 0 words  
PS C:\Users\musta\.cursor>  
  
'None' -> 0 words  
PS C:\Users\musta\.cursor>  
  
'None' -> 0 words
```

Problems Output Debug Console Terminal Ports + ⋮ ×

'None' -> 0 words
PS C:\Users\musta\.cursor>

'None' -> 0 words
'None' -> 0 words
'None' -> 0 words
'None' -> 0 words
PS C:\Users\musta\.cursor>

Observation:

The output of a script that appears to be counting words from an input. The output shows multiple lines, each with the same result: 'None' -> 0 words

Task Description#4

- Generate test cases for a BankAccount class with:

Methods:

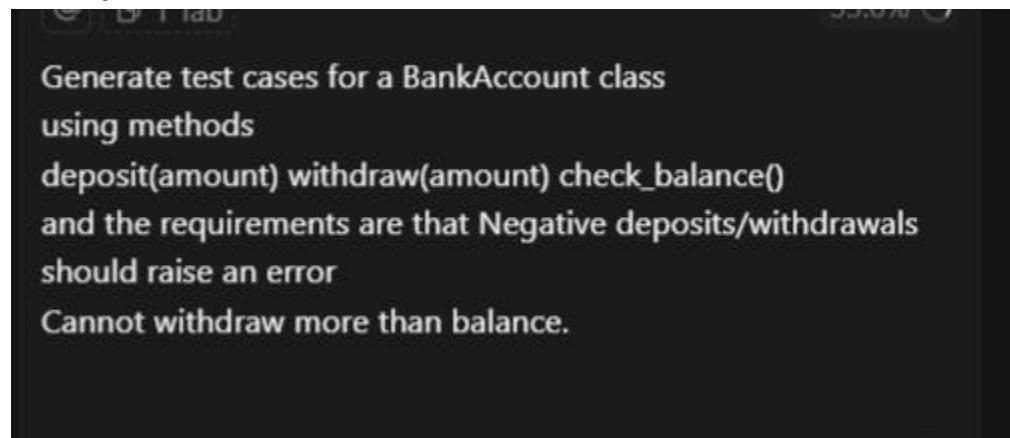
deposit(amount)

withdraw(amount)

check_balance()

Requirements:

- Negative deposits/withdrawals should raise an error.
- Cannot withdraw more than balance

Given prompt to the AI:

Given program by the AI:

```
1 import unittest
2
3
4 class BankAccount:
5     def __init__(self, initial_balance=0.0):
6         if initial_balance < 0:
7             raise ValueError("Initial balance cannot be negative")
8         self._balance = float(initial_balance)
9
10    def deposit(self, amount):
11        if not isinstance(amount, (int, float)):
12            raise TypeError("Amount must be a number")
13        if amount <= 0:
14            raise ValueError("Deposit amount must be positive")
15        self._balance += float(amount)
16        return self._balance
17
18    def withdraw(self, amount):
19        if not isinstance(amount, (int, float)):
20            raise TypeError("Amount must be a number")
21        if amount <= 0:
22            raise ValueError("Withdrawal amount must be positive")
23        if amount > self._balance:
24            raise ValueError("Insufficient funds")
25        self._balance -= float(amount)
26        return self._balance
27
28    def check_balance(self):
29        return self._balance
30
31
32 class TestBankAccount(unittest.TestCase):
33     def test_initial_balance_default(self):
34         pass
```

```
bank_account_tests.py >  TestBankAccount
class TestBankAccount(unittest.TestCase):

    def test_initial_balance_positive(self):
        acct = BankAccount(100)
        self.assertEqual(acct.check_balance(), 100.0)

    def test_initial_balance_negative_raises(self):
        with self.assertRaises(ValueError):
            BankAccount(-1)

    def test_deposit_positive(self):
        acct = BankAccount()
        acct.deposit(50)
        self.assertEqual(acct.check_balance(), 50.0)
        acct.deposit(25.5)
        self.assertEqual(acct.check_balance(), 75.5)

    def test_deposit_zero_or_negative_raises(self):
        acct = BankAccount()
        with self.assertRaises(ValueError):
            acct.deposit(0)
        with self.assertRaises(ValueError):
            acct.deposit(-10)

    def test_deposit_non_numeric_raises(self):
        acct = BankAccount()
        for bad in [None, "10", [], {}]:
            with self.assertRaises(TypeError):
                acct.deposit(bad)

    def test_withdraw_success(self):
        acct = BankAccount() ^ 1 / 1 Undo all Ctrl+Shift+ Keep all Ctrl+
        acct.withdraw(40) < 4 / 4 files >
```

```
_account_tests.py >  TestBankAccount
class TestBankAccount(unittest.TestCase):
    def test_withdraw_success(self):
        acct.withdraw(10.5)
        self.assertEqual(acct.check_balance(), 49.5)

    def test_withdraw_zero_or_negative_raises(self):
        acct = BankAccount(100)
        with self.assertRaises(ValueError):
            acct.withdraw(0)
        with self.assertRaises(ValueError):
            acct.withdraw(-5)

    def test_withdraw_more_than_balance_raises(self):
        acct = BankAccount(50)
        with self.assertRaises(ValueError):
            acct.withdraw(50.01)
        with self.assertRaises(ValueError):
            acct.withdraw(100)

    def test_withdraw_non_numeric_raises(self):
        acct = BankAccount(100)
        for bad in [None, "10", [], {}]:
            with self.assertRaises(TypeError):
                acct.withdraw(bad)

    def test_sequence_of_operations(self):
        acct = BankAccount(100)
        acct.deposit(50)
        acct.withdraw(30)
        acct.deposit(20)
        self.assertEqual(acct.check_balance(), 100.0) ^ 1 / 1 Undo all Ctrl+Shift+ Keep all Ctrl+
        def test_floating_point_precision(self): < 4 / 4 files >
```

```
91
92     def test_sequence_of_operations(self):
93         acct = BankAccount(100)
94         acct.deposit(50)
95         acct.withdraw(30)
96         acct.deposit(20)
97         self.assertEqual(acct.check_balance(), 140.0)
98
99     def test_floating_point_precision(self):
100        acct = BankAccount(0)
101        acct.deposit(0.1)
102        acct.deposit(0.2)
103        self.assertAlmostEqual(acct.check_balance(), 0.3, places=7)
104
105
106 if __name__ == "__main__":
107     print("Running BankAccount unit tests...")
108     unittest.main(verbosity=2)
```

Undo Ctrl+N Keep Ctrl+Shift

Output:

```
PS C:\Users\musta\.cursor> & C:/Users/musta/AppData/Local/Programs/Python/Python313/python.exe c:/Users/musta/.cursor/bank_account_tests.py
Running BankAccount unit tests...
test_deposit_non_numeric_raises (__main__.TestBankAccount.test_deposit_non_numeric_raises) ... ok
test_deposit_positive (__main__.TestBankAccount.test_deposit_positive) ... ok
test_deposit_zero_or_negative_raises (__main__.TestBankAccount.test_deposit_zero_or_negative_raises) ... ok
test_floating_point_precision (__main__.TestBankAccount.test_floating_point_precision) ... ok
test_initial_balance_default (__main__.TestBankAccount.test_initial_balance_default) ... ok
test_initial_balance_negative_raises (__main__.TestBankAccount.test_initial_balance_negative_raises) ... ok
test_initial_balance_positive (__main__.TestBankAccount.test_initial_balance_positive) ... ok
test_sequence_of_operations (__main__.TestBankAccount.test_sequence_of_operations) ... ok
test_withdraw_more_than_balance_raises (__main__.TestBankAccount.test_withdraw_more_than_balance_raises) ... ok
test_withdraw_non_numeric_raises (__main__.TestBankAccount.test_withdraw_non_numeric_raises) ... ok
test_withdraw_success (__main__.TestBankAccount.test_withdraw_success) ... ok
test_withdraw_zero_or_negative_raises (__main__.TestBankAccount.test_withdraw_zero_or_negative_raises) ... ok
test_withdraw_zero_or_negative_raises (__main__.TestBankAccount.test_withdraw_zero_or_negative_raises) ... ok
-----
-----
Ran 12 tests in 0.004s

OK
OK
OK
PS C:\Users\musta\.cursor> []
```

Observation:

All 12 unit tests for the BankAccount class executed successfully with OK status. The results confirm that deposit, withdrawal, balance validation, and precision features are working correctly without errors.

Task Description#5

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

Given prompt to AI:

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

Examples:

121 → True

123 → False

0, negative numbers → handled gracefully

Given program by the AI:

```
* palindrome_tests.py > ...
1 import unittest
2
3
4 def is_number_palindrome(num):
5     """
6         Return True if the integer reads the same backward as forward.
7         Gracefully handle 0 and negative numbers.
8         - Non-integers return False
9         - Negative numbers are not palindromes (by conventional definition)
10        - Boolean values are not valid inputs
11    """
12    # Reject booleans explicitly (bool is subclass of int)
13    if isinstance(num, bool):
14        return False
15    if not isinstance(num, int):
16        return False
17    # Handle 0 explicitly
18    if num == 0:
19        return True
20    # Negative numbers -> not palindrome
21    if num < 0:
22        return False
23    # Numbers ending with 0 cannot be palindrome unless the number is 0
24    if num % 10 == 0:
25        return False
26    # Reverse half the number to avoid overflow and extra work
27    reversed_half = 0
28    while num > reversed_half:
29        reversed_half = reversed_half * 10 + num % 10
30        num //= 10
31    # For odd digit counts
32    return num == reversed_half
```

```
class TestIsNumberPalindrome(unittest.TestCase):
    def test_examples(self):
        self.assertTrue(is_number_palindrome(121))
        self.assertFalse(is_number_palindrome(123))

    def test_zero(self):
        self.assertTrue(is_number_palindrome(0))

    def test_single_digits(self):
        for d in range(10):
            self.assertTrue(is_number_palindrome(d))

    def test_negative_numbers(self):
        self.assertFalse(is_number_palindrome(-1))
        self.assertFalse(is_number_palindrome(-121))
        self.assertFalse(is_number_palindrome(-101))

    def test_trailing_zeros(self):
        self.assertFalse(is_number_palindrome(10))
        self.assertFalse(is_number_palindrome(100))
        self.assertFalse(is_number_palindrome(1000))

    def test_even_digit_palindromes(self):
        self.assertTrue(is_number_palindrome(1221))
        self.assertTrue(is_number_palindrome(3443))
        self.assertTrue(is_number_palindrome(9009))
        self.assertTrue(is_number_palindrome(1001))

    def test_odd_digit_palindromes(self):
        self.assertTrue(is_number_palindrome(12321))
        self.assertFalse(is_number_palindrome(123)) # 123 is not a palindrome
```

```
palindrome_tests.py > ...
35 class TestIsNumberPalindrome(unittest.TestCase):
36     def test_odd_digit_palindromes(self):
37         self.assertTrue(is_number_palindrome(12321))
38         self.assertTrue(is_number_palindrome(1234321))

39     def test_non_palindromes(self):
40         self.assertFalse(is_number_palindrome(12))
41         self.assertFalse(is_number_palindrome(1231))
42         self.assertFalse(is_number_palindrome(123421))
43         self.assertFalse(is_number_palindrome(1002))

44     def test_large_numbers(self):
45         self.assertTrue(is_number_palindrome(123454321))
46         self.assertTrue(is_number_palindrome(10000000000000001))
47         self.assertFalse(is_number_palindrome(1234567890123456780))

48     def test_invalid_inputs(self):
49         self.assertFalse(is_number_palindrome(12.21))
50         self.assertFalse(is_number_palindrome("121"))
51         self.assertFalse(is_number_palindrome(None))
52         self.assertFalse(is_number_palindrome(True))
53         self.assertFalse(is_number_palindrome([1,2,1]))

54     if __name__ == "__main__":
55         print("Running palindrome unit tests...")
56         unittest.main(verbose=2)
```

Output:

```
PS C:\Users\musta\.cursor> & C:/Users/musta/AppData/Local/Programs/Python/Python313/python.exe c:/Users/musta/.cursor/palindrome_tests.py
Running palindrome unit tests...
test_even_digit_palindromes (__main__.TestIsNumberPalindrome.test_even_digit_palindromes) ... ok
test_examples (__main__.TestIsNumberPalindrome.test_examples) ... ok
test_invalid_inputs (__main__.TestIsNumberPalindrome.test_invalid_inputs) ... ok
test_large_numbers (__main__.TestIsNumberPalindrome.test_large_numbers) ... ok
test_negative_numbers (__main__.TestIsNumberPalindrome.test_negative_numbers) ... ok
test_non_palindromes (__main__.TestIsNumberPalindrome.test_non_palindromes) ... ok
test_odd_digit_palindromes (__main__.TestIsNumberPalindrome.test_odd_digit_palindromes) ... ok
test_single_digits (__main__.TestIsNumberPalindrome.test_single_digits) ... ok
test_trailing_zeros (__main__.TestIsNumberPalindrome.test_trailing_zeros) ... ok
test_zero (__main__.TestIsNumberPalindrome.test_zero) ... ok

-----
Ran 10 tests in 0.002s

OK
PS C:\Users\musta\.cursor>
```

Observation:

All 10 unit tests for the palindrome program passed successfully with OK. The implementation handles valid, invalid, and edge cases correctly.