

# AI ASSISTED CODING

## ASSIGNMENT 8.2

Name: Simra Tahseen

RollNo:2503A51L17

Batch NO:24BTCAICSB19

### 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:** `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
```

```
for n, expected in test_cases:
```

```
    result = is_prime(n)
```

```
    if result != expected:
```

```
        print(f"Test failed for n={n}: expected {expected}, got {result}")
```

```
        all_passed = False
```

```
if all_passed:
```

```
    print("All tests passed!")
```

```
if __name__ == "__main__":
```

```
    run_tests()
```

The screenshot shows a code editor window with a dark theme. The file being edited is named 'primen.py'. The code defines a function 'is\_prime(n)' which returns True if n is a prime number and False otherwise. It includes a range of checks from 3 to the square root of n. The code also contains a 'run\_tests()' function that runs AI-generated test cases using edge coverage, comparing results against expected values.

```
1  def is_prime(n):
2      """
3          Returns True if n is a prime number, False otherwise.
4      """
5      if n <= 1:
6          return False
7      if n == 2:
8          return True
9      if n % 2 == 0:
10         return False
11
12     for i in range(3, int(n ** 0.5) + 1, 2):
13         if n % i == 0:
14             return False
15     return True
16
17
18 def run_tests():
19     """
20         Runs AI-generated test cases using edge coverage.
21     """
22     test_cases = [
23         (-5, False),
24         (0, False),
25         (1, False),
26         (2, True),
27         (3, True),
28         (4, False),
29         (17, True),
30         (20, False),
31         (97, True),
32         (100, False),
33         (999_983, True),
34         (1_000_000, False),
35         (1_000_003, True),
36         (999_997, False),
37     ]
38
39     all_passed = True
40
```

```
>Welcome primen.py ●

primen.py > ...
18 def run_tests():
19     all_passed = True
20
21     for i, (input_val, expected) in enumerate(test_cases, 1):
22         result = is_prime(input_val)
23         if result == expected:
24             print(f"Test case {i} passed.")
25         else:
26             print(f"Test case {i} FAILED: is_prime({input_val}) = {result},"
27                 "all_passed = False")
28
29     if all_passed:
30         print("\n✓ All test cases passed!")
31     else:
32         print("\n✗ Some test cases failed.")

33
34
35 def dynamic_check():
36     """
37     Allows the user to enter a number dynamically and checks if it's prime.
38     """
39
40     try:
41         user_input = int(input("\nEnter a number to check if it is prime: "))
42         if is_prime(user_input):
43             print(f"{user_input} is a prime number.")
44         else:
45             print(f"{user_input} is NOT a prime number.")
46     except ValueError:
47         print("Invalid input. Please enter an integer.")

48
49 # Run test cases and then allow dynamic input
50 dynamic_check()
51 run_tests()
```

## **Output :**

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂ ... [ ]
```

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 nonprimes.
- 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.

## **Task #2**

**Prompt:** Ask AI to generate test cases for celsius\_to\_fahrenheit(c) and

fahrenheit\_to\_celsius(f) **Code generated:**

```
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."
```

```
temp.py > ...
1  def celsius_to_fahrenheit(celsius):
2      """Converts Celsius to Fahrenheit."""
3      try:
4          celsius = float(celsius)
5          return (celsius * 9/5) + 32
6      except (ValueError, TypeError):
7          return "Invalid input. Please enter a valid number for Celsius"
8
9  def fahrenheit_to_celsius(fahrenheit):
10     """Converts Fahrenheit to Celsius."""
11     try:
12         fahrenheit = float(fahrenheit)
13         return (fahrenheit - 32) * 5/9
14     except (ValueError, TypeError):
15         return "Invalid input. Please enter a valid number for Fahrenheit"
16
17 if __name__ == "__main__":
18     while True:
19         print("\nTemperature Conversion:")
20         print("1. Celsius to Fahrenheit")
21         print("2. Fahrenheit to Celsius")
22         print("3. Exit")
23
24         choice = input("Enter your choice (1, 2, or 3): ")
25
26         if choice == '1':
27             celsius_input = input("Enter temperature in Celsius: ")
28             result = celsius_to_fahrenheit(celsius_input)
29             print(f"{celsius_input}°C is equal to {result}°F")
30         elif choice == '2':
31             fahrenheit_input = input("Enter temperature in Fahrenheit: ")
32             result = fahrenheit_to_celsius(fahrenheit_input)
33             print(f"{fahrenheit_input}°F is equal to {result}°C")
34         elif choice == '3':
35             print("Exiting program.")
36             break
37         else:
38             print("Invalid choice. Please enter 1, 2, or 3.")
```

## Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + × └ ... | ☰ >

PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted> & C:/ProgramData/anaconda3/python.exe "c:/Users/Gundeti 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  $0^{\circ}\text{C} = 32^{\circ}\text{F}$  and  $100^{\circ}\text{C} = 212^{\circ}\text{F}$
- 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:**

```
import re

def count_words(text):
    """
    Count words in a given string.
    """

    if not isinstance(text, str):
        return 0

    words = re.findall(r'\b\w+\b', text)
    return len(words)

def run_tests():
    """
    Run test cases to validate the count_words function.
    """

    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():  
    """  
    Ask user for input and count the words.  
    """  
  
    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()
```

```
>Welcome  primen.py  temp.py  count.py  X
❶ count.py > ⚡ run_tests
1  import re
2  def count_words(text):
3      """
4      """
5      if not isinstance(text, str):
6          return 0
7      words = re.findall(r'\b\w+\b', text)
8      return len(words)
9  def run_tests():
10      """
11      """
12      test_cases = [
13          ("Hello world", 2),
14          ("Hello world again", 3),
15          ("Hello, world!", 2), [], (" ", 0), ("One-word", 2),
16          ("Wow! This is... amazing.", 4),
17          ("Newlines\nand\ttabs count as spaces", 6),
18          (None, 0),
19          (12345, 0),
20      ]
21      all_passed = True
22      for i, (input_text, expected) in enumerate(test_cases, 1):
23          result = count_words(input_text)
24          if result == expected:
25              print(f"Test case {i} passed.")
26          else:
27              print(f"Test case {i} FAILED: count_words({repr(input_text)}) = {result}, expected {expected}")
28          all_passed = False
29      if all_passed:
30          print("\n✓ All test cases passed!")
31      else:
32          print("\n✗ Some test cases failed.")
33  def dynamic_input():
34      """
35      """
36      user_text = input("\nEnter a sentence to count the words: ")
37      count = count_words(user_text)
38      print(f"Word count: {count}")
39  if __name__ == "__main__":
40      run_tests()
41      dynamic_input()
```

## Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted> & C:/ProgramData/anaconda3/python.exe "c:/Users/Gundeti Hasini\OneDrive\Documents\ai assisted\count.py"
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:**

```
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}) FAILED: got {result}, expected {expected}")
                all_passed = False
            else:
                print(f"✓ Test {i} ({desc}) passed.")
        except Exception as e:
            if isinstance(expected, type) and isinstance(e, expected):
                print(f"✓ Test {i} ({desc}) passed.")

```

```

else:
    print(f"✗ Test {i} ({desc}) FAILED: got exception {e}, expected {expected}")
    all_passed = False

if all_passed:
    print("🎉 All tests passed!")
else:
    print("✗ Some tests failed.")

# To run the tests
run_tests()

```

```

Welcome | primen.py | temp.py | count.py | bank.py •
bank.py > run_tests
1  class BankAccount:
2      """
3      """
4      def __init__(self, initial_balance=0):
5          if initial_balance < 0:
6              raise ValueError("Initial balance cannot be negative.")
7          self.balance = initial_balance
8      def deposit(self, amount):
9          if amount <= 0:
10             raise ValueError("Deposit amount must be positive.")
11             self.balance += amount
12             return self.balance
13     def withdraw(self, amount):
14         if amount <= 0:
15             raise ValueError("Withdrawal amount must be positive.")
16             if amount > self.balance:
17                 raise ValueError("Insufficient funds.")
18                 self.balance -= amount
19                 return self.balance
20     def check_balance(self):
21         return self.balance
22 def run_tests():
23     test_cases = [
24         # (description, function, args, expected result or exception)
25         ("Initial balance 0", lambda: BankAccount().check_balance(), (), 0),
26         ("Deposit 100", lambda: BankAccount().deposit(100), (), 100),
27         ("Withdraw 50 (valid)", lambda: (acct := BankAccount(100)).withdraw(50), (), 50),
28         ("Check balance after deposit", lambda: (acct := BankAccount()).deposit(200) or acct.check_balance(), (), 200),
29         ("Withdraw more than balance", lambda: (acct := BankAccount(50)).withdraw(100), (), ValueError),
30         ("Negative deposit", lambda: BankAccount().deposit(-10), (), ValueError),
31         ("Negative withdrawal", lambda: BankAccount(100).withdraw(-20), (), ValueError)],
32     all_passed = True
33     for i, (desc, func, args, expected) in enumerate(test_cases, 1):
34         try:
35             result = func(*args)
36             if isinstance(expected, type) and issubclass(expected, Exception):
37                 print(f"✗ Test {i} ({desc}) FAILED: Expected exception {expected.__name__}")
38                 all_passed = False
39             elif result == expected:
40                 print(f"✓ Test {i} ({desc}) passed.")
41             else:
42                 print(f"✗ Test {i} ({desc}) FAILED: got {result}, expected {expected}")

```

```

1 Welcome | 2 primen.py | 3 temp.py | 4 count.py | 5 bank.py | ●
6 bank.py > 7 run_tests()
8
9 def run_tests():
10     all_passed = False
11     except Exception as e:
12         if isinstance(expected, type) and isinstance(e, expected):
13             print(f"✓ Test {i} ({desc}) passed (raised {expected.__name__}).")
14         else:
15             print(f"✗ Test {i} ({desc}) FAILED: raised {e.__class__.__name__} ({e})")
16             all_passed = False
17     if all_passed:
18         print("\n* All test cases passed!")
19     else:
20         print("\n⚠ Some test cases failed.")
21
22 def dynamic_input():
23     account = BankAccount()
24     while True:
25         print("\nChoose an option:")
26         print("1. Deposit")
27         print("2. Withdraw")
28         print("3. Check Balance")
29         print("4. Exit")
30         choice = input("Enter your choice: ")
31         try:
32             if choice == "1":
33                 amt = float(input("Enter deposit amount: "))
34                 account.deposit(amt)
35                 print(f"✓ Deposited {amt}. Current balance: {account.check_balance()}")
36             elif choice == "2":
37                 amt = float(input("Enter withdrawal amount: "))
38                 account.withdraw(amt)
39                 print(f"✗ Withdrew {amt}. Current balance: {account.check_balance()}")
40             elif choice == "3":
41                 print(f"⌚ Current balance: {account.check_balance()}")
42             elif choice == "4":
43                 print("Exiting...")
44                 break
45             else:
46                 print("Invalid choice. Try again.")
47         except Exception as e:
48             print(f"⚠ Error: {e}")
49
50 if __name__ == "__main__":
51     dynamic_input()
52     run_tests()

```

## Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Gundeti Hasini\OneDrive\Documents\ai assisted> & C:/ProgramData/anaconda3/python.exe "c:/Users/Gundeti 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:**

```
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("\nAll test cases passed!")  
    dynamic_input()
```

```
❶ Welcome ❷ primen.py ❸ temp.py ❹ count.py ❺ bank.py ❻ palin.py •
```

```
# palin.py > ⚡ run_tests
1 def is_number_palindrome(num):
2     """
3     """
4     if not isinstance(num, int):
5         raise ValueError("Input must be an integer.")
6     if num < 0:
7         return False # Negative numbers are not palindromes
8     return str(num) == str(num)[::-1]
9
10 def run_tests():
11     test_cases = [
12         (121, True),
13         (123, False), (0, True), (-121, False),
14         (9, True), (1221, True), (1001, True),
15         (100, False), (12321, True), ("121", ValueError)]
16     all_passed = True
17     for i, (input_val, expected) in enumerate(test_cases, 1):
18         try:
19             result = is_number_palindrome(input_val)
20             if isinstance(expected, type) and issubclass(expected, Exception):
21                 print(f" Test {i} FAILED: Expected {expected.__name__} for input {input_val}")
22                 all_passed = False
23             elif result == expected:
24                 print(f" Test {i} passed.")
25             else:
26                 print(f" Test {i} FAILED: input={input_val}, got={result}, expected={expected}")
27                 all_passed = False
28         except Exception as e:
29             if isinstance(expected, type) and isinstance(e, expected):
30                 print(f"Test {i} passed (raised {expected.__name__}).")
31             else:
32                 print(f" Test {i} FAILED: input={input_val}, raised {e.__class__.__name__} ({e})")
33                 all_passed = False
34     if all_passed:
35         print("\n🎉 All test cases passed!")
36     else:
37         print("\n Some test cases failed.")
38
39 def dynamic_input():
40     while True:
41         try:
42             user_input = input("\nEnter an integer to check palindrome (or 'exit' to quit): ")
43             if user_input.lower() == "exit":
44                 print("Exiting...")
45                 break
46             num = int(user_input)
47             result = is_number_palindrome(num)
48             print(f" {num} is a palindrome? {result}")
49         except ValueError as e:
50             print(f"⚠ Error: {e}")
51     if __name__ == "__main__":
52         run_tests()
53         dynamic_input()
```

```
❶ Welcome ❷ primen.py ❸ temp.py ❹ count.py ❺ bank.py ❻ palin.py •
```

```
# palin.py > ⚡ run_tests
37 def dynamic_input():
38     if user_input.lower() == "exit":
39         print("Exiting...")
40         break
41     num = int(user_input)
42     result = is_number_palindrome(num)
43     print(f" {num} is a palindrome? {result}")
44     except ValueError as e:
45         print(f"⚠ Error: {e}")
46     if __name__ == "__main__":
47         run_tests()
48         dynamic_input()
```

## Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python
```

```
🎉 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.