SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
ProgramName:B. Tech		**Assignment Type: Lab**	**AcademicYear:**2025-2026
CourseCoordinatorName		Venkataramana Veeramsetty	
Instructor(s)Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week4 - Tuesday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	
AssignmentNumber:8.2(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*ExpectedTime to complete*
1	Lab 8: Test-Driven Development with AI – Generating and Working with Test Cases **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.	Week4 - Wednesday

- 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

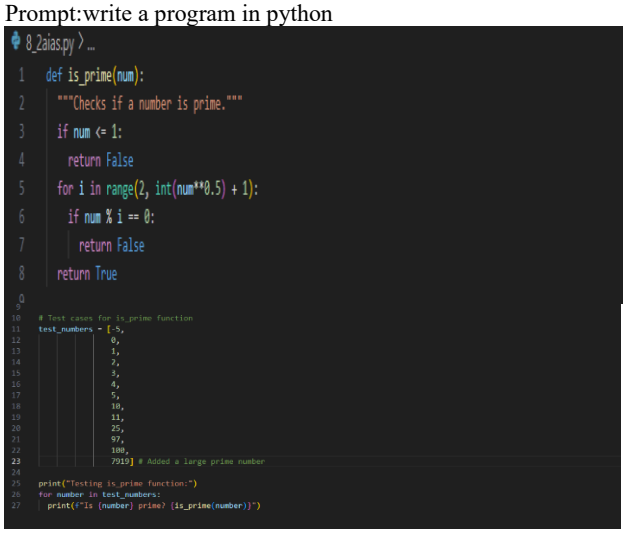
Task Description#1

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.
Prompt:write a program in python



Expected Output#1

- A working prime checker that passes AI-generated tests using edge coverage.

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°C = 32°F, 100°C = 212°F.
- Include decimals and invalid inputs like strings or None

```python
8_2aias.py > ...
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 for celsius_to_fahrenheit
test_celsius = [0, 100, 25.5, -10, "abc", None]

print("Testing celsius_to_fahrenheit function:")
for celsius in test_celsius:
    try:
        fahrenheit = celsius_to_fahrenheit(celsius)
        print(f"{celsius}°C is equal to {fahrenheit}°F")
    except (TypeError, ValueError) as e:
        print(f"Invalid input for celsius_to_fahrenheit: {celsius} - {e}")
# Test cases for fahrenheit_to_celsius
test_fahrenheit = [32, 212, 77.9, 14, "xyz", None]

print("\nTesting fahrenheit_to_celsius function:")
for fahrenheit in test_fahrenheit:
    try:
        celsius = fahrenheit_to_celsius(fahrenheit)
        print(f"{fahrenheit}°F is equal to {celsius}°C")
    except (TypeError, ValueError) as e:
        print(f"Invalid input for fahrenheit_to_celsius: {fahrenheit} - {e}")
```

Expected Output#2

Dual conversion functions with complete test coverage and safe type handling

```
Testing celsius_to_fahrenheit function:
0°C is equal to 32.0°F
100°C is equal to 212.0°F
25.5°C is equal to 77.9°F
-10°C is equal to 14.0°F
Invalid input for celsius_to_fahrenheit: abc - unsupported operand type(s) for /: 'str' and 'int'
Invalid input for celsius_to_fahrenheit: None - unsupported operand type(s) for *: 'NoneType' and 'int'

Testing fahrenheit_to_celsius function:
32°F is equal to 0.0°C
212°F is equal to 100.0°C
77.9°F is equal to 25.500000000000004°C
14°F is equal to -10.0°C
Invalid input for fahrenheit_to_celsius: xyz - unsupported operand type(s) for -: 'str' and 'int'
Invalid input for fahrenheit_to_celsius: None - unsupported operand type(s) for -: 'NoneType' and 'int'
PS C:\Users\HARSHINI\Downloads\aiac> & C:/Users/HARSHINI/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/HARSHINI/Downloads/aiac/8_2aias.py
Testing celsius_to_fahrenheit function:
0°C is equal to 32.0°F
100°C is equal to 212.0°F
25.5°C is equal to 77.9°F
-10°C is equal to 14.0°F
Invalid input for celsius_to_fahrenheit: abc - unsupported operand type(s) for /: 'str' and 'int'
Invalid input for celsius_to_fahrenheit: None - unsupported operand type(s) for *: 'NoneType' and 'int'

Testing fahrenheit_to_celsius function:
32°F is equal to 0.0°C
212°F is equal to 100.0°C
77.9°F is equal to 25.500000000000004°C
14°F is equal to -10.0°C
Invalid input for fahrenheit_to_celsius: xyz - unsupported operand type(s) for -: 'str' and 'int'
Invalid input for fahrenheit_to_celsius: None - unsupported operand type(s) for -: 'NoneType' and 'int'
PS C:\Users\HARSHINI\Downloads\aiac>
```

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.

```python
import re

def count_words(text):
    """Counts the number of words in a sentence."""
    if not text:
        return 0
    # Use regex to split by spaces and punctuation, then filter out empty strings
    words = re.split(r'\W+', text)
    words = [word for word in words if word]
    return len(words)
# Test cases for count_words function
test_sentences = [
    "This is a normal sentence.",
    "This sentence has   multiple spaces.",
    "This sentence has punctuation!",
    "  leading and trailing spaces.  ",
    "",
    "One word.",
    "Sentence with numbers 123.",
    "Sentence-with-hyphenated-words."
]

print("Testing count_words function:")
for sentence in test_sentences:
    word_count = count_words(sentence)
    print(f"'{sentence}' -> Word count: {word_count}")
```

Expected Output#3

Accurate word count with robust test case validation.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Testing count_words function:
Sentence: 'This is a normal sentence.' -> Word count: 5
Sentence: 'This sentence has   multiple spaces.' -> Word count: 5
Sentence: 'This sentence has punctuation!' -> Word count: 4
Sentence: '  Leading and trailing spaces.  ' -> Word count: 4
Sentence: '' -> Word count: 0
Sentence: 'One word.' -> Word count: 2
Sentence: 'Sentence with numbers 123.' -> Word count: 4
Sentence: 'Sentence-with-hyphenated-words.' -> Word count: 4
PS C:\Users\HARSHINI\Downloads\aiac>
```

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.

```
2aias.py > ...
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 cannot be negative")
        self.balance += amount
        print(f"Deposited {amount}. New balance is {self.balance}")

    def withdraw(self, amount):
        if amount < 0:
            raise ValueError("Withdrawal amount cannot be negative")
        if amount > self.balance:
            raise ValueError("Insufficient funds")
        self.balance -= amount
        print(f"Withdrew {amount}. New balance is {self.balance}")

    def check_balance(self):
        return self.balance
# Test initial balance
account1 = BankAccount(100)
print(f"Initial balance: {account1.check_balance()}")

# Test valid deposit
account1.deposit(50)
print(f"Balance after deposit: {account1.check_balance()}")

# Test valid withdrawal
account1.withdraw(30)
print(f"Balance after withdrawal: {account1.check_balance()}")

# Test withdrawal of exact balance
account1.withdraw(120)
print(f"Balance after withdrawing exact amount: {account1.check_balance()}")
# Test invalid deposit (negative amount)
try:
    account1.deposit(-20)
except ValueError as e:
    print(f"Error depositing negative amount: {e}")

# Test invalid withdrawal (negative amount)
try:
    account1.withdraw(-40)
except ValueError as e:
    print(f"Error withdrawing negative amount: {e}")

# Test invalid withdrawal (insufficient funds)
try:
    account1.withdraw(50) # Current balance is 0
except ValueError as e:
    print(f"Error withdrawing more than balance: {e}")

# Test initial balance with negative amount
try:
    account2 = BankAccount(-50)
except ValueError as e:
    print(f"Error with negative initial balance: {e}")
```

Expected Output#4

- AI-generated test suite with a robust class that handles all test cases.

```
Error withdrawing more than balance: Insufficient funds
Error with negative initial balance: Initial balance cannot be negative
PS C:\Users\HARSHINI\Downloads\aiac> & C:/Users/HARSHINI/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/HARSHINI/Downloads/aiac/8_2aias.py
Initial balance: 100
Deposited 50. New balance is 150
Balance after deposit: 150
Withdrew 30. New balance is 120
Balance after withdrawal: 120
Withdrew 120. New balance is 0
Balance after withdrawing exact amount: 0
Error depositing negative amount: Deposit amount cannot be negative
Error withdrawing negative amount: Withdrawal amount cannot be negative
Error withdrawing more than balance: Insufficient funds
Error with negative initial balance: Initial balance cannot be negative
```

-
-

Task Description#5

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

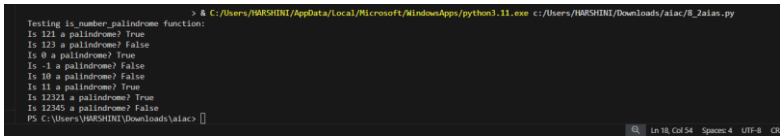
```python
def is_number_palindrome(num):
    """Checks if an integer reads the same backward."""
    # Handle negative numbers and 0
    if num < 0:
        return False
    if num == 0:
        return True

    # Convert number to string to check for palindrome
    num_str = str(num)
    return num_str == num_str[::-1]
# Test cases for is_number_palindrome function
test_numbers = [121, 123, 0, -1, 10, 11, 12321, 12345]

print("Testing is_number_palindrome function:")
for number in test_numbers:
    is_palindrome = is_number_palindrome(number)
    print(f"Is {number} a palindrome? {is_palindrome}")
```

Expected Output#5

- Number-based palindrome checker function validated against test cases.



```
> & C:/Users/HARSHINI/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/HARSHINI/Downloads/aiac/8_2aias.py
Testing is_number_palindrome function:
Is 121 a palindrome? True
Is 123 a palindrome? False
Is 0 a palindrome? True
Is -1 a palindrome? False
Is 10 a palindrome? False
Is 11 a palindrome? True
Is 12321 a palindrome? True
Is 12345 a palindrome? False
PS C:\Users\HARSHINI\Downloads\aiac>
```

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria	Max Marks
Task #1	0.5
Task #2	0.5
Task #3	0.5
Task #4	0.5
Task #5	0.5
Total	**2.5 Marks**