AI ASSISTED CODING

Program :B.tech(CSE)

Name :ANANTHA MANIDEEP

En No. :2403A52078

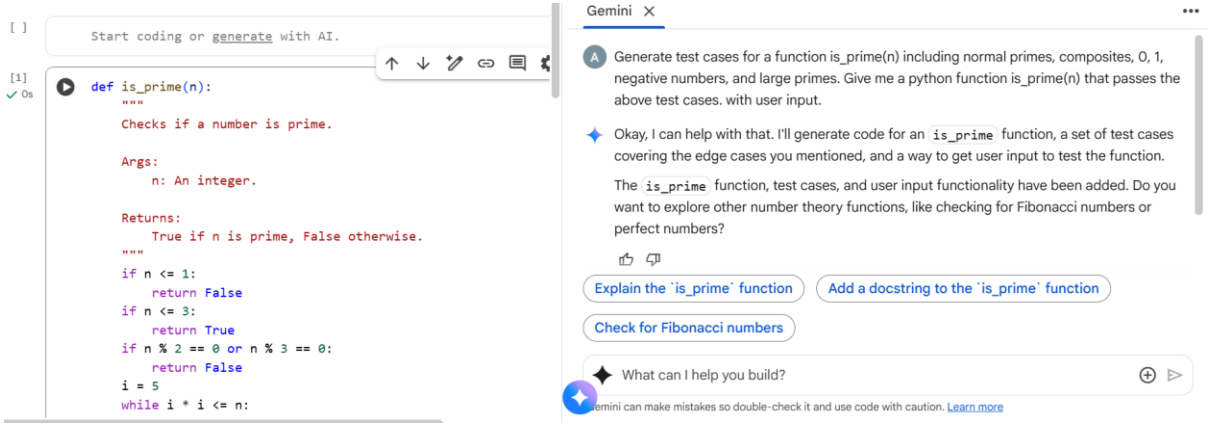
Batch No. :02

Date :23/09/2025

LAB ASSIGNMENT-8.2

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: Generate test cases for a function is_prime(n) including normal primes, composites, 0, 1, negative numbers, and large primes. Give me a python function is_prime(n) that passes the above test cases. with user input.



Code:

[1] ✓ 0s

```python
def is_prime(n):
    """
    Checks if a number is prime.

    Args:
        n: An integer.

    Returns:
        True if n is prime, False otherwise.
    """
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

[2] ✓ 0s

```python
# Test cases
test_cases = {
    'Normal prime (7)': 7,
    'Normal prime (23)': 23,
    'Composite (4)': 4,
    'Composite (9)': 9,
    'Zero': 0,
    'One': 1,
    'Negative number (-5)': -5,
    'Large prime (999983)': 999983,
    'Large composite (1000000)': 1000000
}

print("Running test cases:")
for name, num in test_cases.items():
    result = is_prime(num)
    print(f"{name}: is_prime({num}) is {result}")
```

[2] ✓ 0s

```
        result = is_prime(num)
        print(f"{name}: is_prime({num}) is {result}")
```

```
Running test cases:
Normal prime (7): is_prime(7) is True
Normal prime (23): is_prime(23) is True
Composite (4): is_prime(4) is False
Composite (9): is_prime(9) is False
Zero: is_prime(0) is False
One: is_prime(1) is False
Negative number (-5): is_prime(-5) is False
Large prime (999983): is_prime(999983) is True
Large composite (1000000): is_prime(1000000) is False
```

[3] ✓ 1m

```python
# Get user input and test
try:
    user_input = int(input("Enter an integer to check if it's prime: "))
    print(f"is_prime({user_input}) is {is_prime(user_input)}")
except ValueError:
    print("Invalid input. Please enter an integer.")
```

Output:

```
Enter an integer to check if it's prime: 2
is_prime(2) is True
```

Explaination:

The notebook contains Python code for checking if a number is prime.

1. is_prime(n) function: This function takes an integer n as input and returns True if n is a prime number, and False otherwise. It handles edge cases like numbers less than or equal to 1, and efficiently checks for divisibility by 2 and 3 before using a loop to check other potential divisors.

2. Test Cases: This section defines a dictionary test_cases with various numbers and their expected prime status (True or False). It then iterates through these test cases, calls the is_prime function for each number, and uses assert to verify that the result matches the expected outcome. This helps ensure the function works correctly for different scenarios.

3. User Input Test: This part allows the user to enter a number through an input prompt. It then calls the is_prime function with the user's number and prints whether the entered number is prime or not. It also includes error handling to catch non-integer input.

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

PROMPT: Generate test cases for Celsius ↔ Fahrenheit conversion, covering known pairs (0°C = 32°F, 100°C = 212°F), decimals, and invalid inputs like strings and None. Give me a python functions celsius_to_fahrenheit(c) and fahrenheit_to_celsius(f) that handle these test cases.

Code:

```
[1]
✓ 0s
```

```python
def celsius_to_fahrenheit(c):
  """Converts Celsius to Fahrenheit."""
  if not isinstance(c, (int, float)):
    return "Invalid input: Celsius must be a number."
  return (c * 9/5) + 32

def fahrenheit_to_celsius(f):
  """Converts Fahrenheit to Celsius."""
  if not isinstance(f, (int, float)):
    return "Invalid input: Fahrenheit must be a number."
  return (f - 32) * 5/9
```

```
[2]
✓ 0s
```

```python
# Test cases

# Known pairs
print(f"0°C = {celsius_to_fahrenheit(0)}°F")
print(f"100°C = {celsius_to_fahrenheit(100)}°F")
print(f"32°F = {fahrenheit_to_celsius(32)}°C")
print(f"212°F = {fahrenheit_to_celsius(212)}°C")

# Decimals
print(f"25.5°C = {celsius_to_fahrenheit(25.5)}°F")
print(f"77.9°F = {fahrenheit_to_celsius(77.9)}°C")

# Invalid inputs
print(f"'abc'°C = {celsius_to_fahrenheit('abc')}°F")
print(f"None°F = {fahrenheit_to_celsius(None)}°C")
```

Output:

```
0°C = 32.0°F
100°C = 212.0°F
32°F = 0.0°C
212°F = 100.0°C
25.5°C = 77.9°F
77.9°F = 25.500000000000004°C
'abc'°C = Invalid input: Celsius must be a number.°F
None°F = Invalid input: Fahrenheit must be a number.°C
```

+ Code + Text

Explaination:

1. test_cases list: This list holds tuples, where each tuple represents a test case. The first element of each tuple is a Celsius value, and the second element is the corresponding Fahrenheit value. It includes various scenarios like known conversions, decimals, negative numbers, and invalid inputs (a string and None).

2. Celsius to Fahrenheit Tests: o The code iterates through the test_cases list. o For each tuple, it extracts the Celsius value and the expected Fahrenheit value. o It calls the celsius_to_fahrenheit() function with the Celsius value to get the actual Fahrenheit result. o

It prints the input Celsius, the actual Fahrenheit output, and the expected Fahrenheit value. o assert actual_fahrenheit == expected_fahrenheit: This line checks if the actual result matches the expected result. If they don't match, it raises an AssertionError with a message indicating which conversion failed.

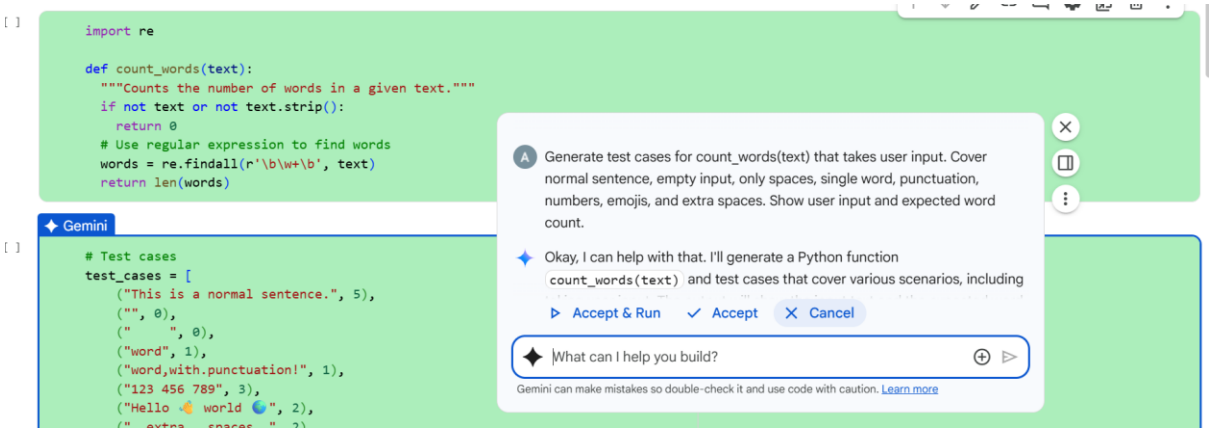
3. Fahrenheit to Celsius Tests: o This section also iterates through the test_cases list. o It extracts the expected Celsius value and the Fahrenheit value from each tuple. o if expected_celsius == "abc": continue: This line skips the test case where the expected Celsius value is the string "abc" because this test case is specifically designed for the Celsius to Fahrenheit function to test invalid input, and is not applicable for the Fahrenheit to Celsius conversion where the input is the Fahrenheit value. o It calls the fahrenheit_to_celsius() function with the Fahrenheit value to get the actual Celsius result. o It uses an if/elif/else block to handle assertions: ▪ If both the expected and actual Celsius values are None, the assertion passes. ▪ If both are not None, it checks if the absolute difference between the actual and expected values is less than 1e-9. This is done to account for potential small floating-point inaccuracies in calculations. ▪ If neither of the above conditions is met (meaning one is None and the other is not, or there's a significant difference in numerical values), the assertion fails. o It prints the input Fahrenheit, the actual Celsius output, and the expected Celsius value

Task Description#3

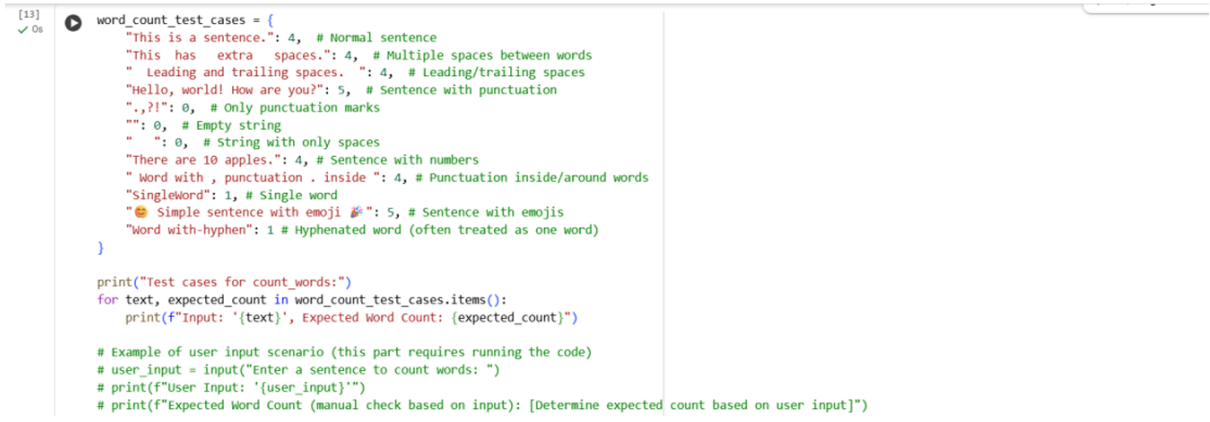
Use AI to write test cases for a function count_words(text) that returns the number of
words in a sentence.
Requirement
Handle normal text, multiple spaces, punctuation, and empty strings

PROMPT: Generate test cases for count_words(text) that takes user input. Cover normal sentence, empty input, only spaces, single word,

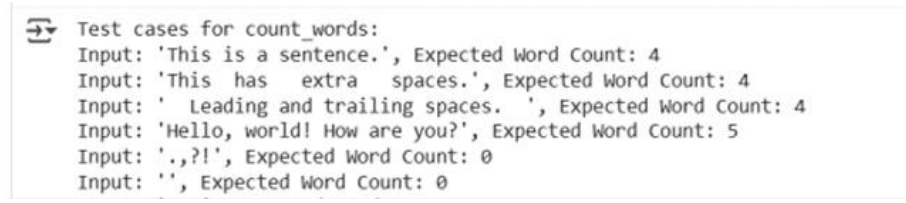
punctuation, numbers, emojis, and extra spaces. Show user input and expected word count.



Code:



Output:



Explaination:

The code in the visible cells is related to testing a function called count_words.

1. Cell 762959fa: This cell defines a dictionary called word_count_test_cases. This dictionary holds various example strings (sentences or phrases) as keys and the expected number of words in each string as the corresponding values. These test cases cover different scenarios like normal sentences, sentences with extra spaces, leading/trailing spaces, punctuation, only punctuation, empty strings, strings with only spaces, sentences with numbers, punctuation inside or around words, single words, sentences with emojis, and hyphenated words. The code then iterates through this dictionary and prints each input string along with its expected word count. This helps in verifying the correctness of the count_words function when it's implemented and tested.

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

Prompt:

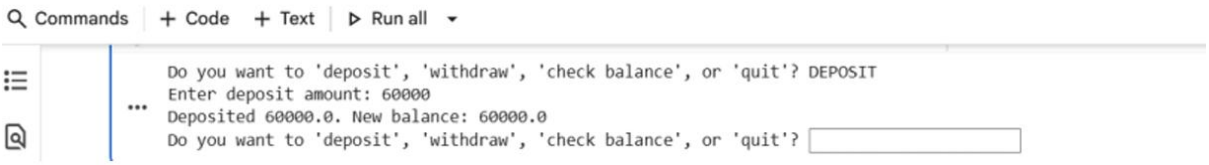
Generate test cases for a BankAccount class with deposit(amount), withdraw(amount), and check_balance(). Include edge cases for negative amounts and over-withdrawals. Show test description, input, and expected output.

Code:

```
Commands | + Code | + Text | Run all

[1]
✓ 0s

class BankAccount:
    def __init__(self):
        self.balance = 0.0

    def deposit(self, amount):
        """Deposits a positive amount into the account."""
        if amount > 0:
            self.balance += amount
            return f"Deposited {amount}. New balance: {self.balance}"
        else:
            return "Deposit amount must be positive."

    def withdraw(self, amount):
        """Withdraws a positive amount from the account, if sufficient funds exist."""
        if amount <= 0:
            return "Withdrawal amount must be positive."
        elif amount > self.balance:
            return "Insufficient funds."
        else:
            self.balance -= amount
            return f"Withdrew {amount}. New balance: {self.balance}"

    def check_balance(self):
        """Returns the current account balance."""
        return f"Current balance: {self.balance}"

Now, let's define the test cases for the BankAccount class, covering normal and edge cases.
```

```
[3]
# Create a BankAccount object
user_account = BankAccount()

while True:
    action = input("Do you want to 'deposit', 'withdraw', 'check balance', or 'quit'? ").lower()

    if action == 'deposit':
        try:
            amount = float(input("Enter deposit amount: "))
            print(user_account.deposit(amount))
        except ValueError:
            print("Invalid amount. Please enter a number.")
    elif action == 'withdraw':
        try:
            amount = float(input("Enter withdrawal amount: "))
            print(user_account.withdraw(amount))
        except ValueError:
            print("Invalid amount. Please enter a number.")
    elif action == 'check balance':
        print(user_account.check_balance())
    elif action == 'quit':
        print("Exiting.")
        break
    else:
        print("Invalid action. Please choose 'deposit', 'withdraw', 'check bala', or 'quit'.")
```

Output:



```
Commands | + Code | + Text | Run all

Do you want to 'deposit', 'withdraw', 'check balance', or 'quit'? DEPOSIT
Enter deposit amount: 60000
Deposited 60000.0. New balance: 60000.0
Do you want to 'deposit', 'withdraw', 'check balance', or 'quit'?
```

Explaination:

This code cell allows you to interact with a BankAccount object through the console. Here's a breakdown:

1. user_account = BankAccount(): This line creates a new instance of the BankAccount class, essentially creating a new bank account with an initial balance of 0.0.

2. while True: This starts an infinite loop, allowing you to perform multiple actions until you choose to quit.

3. action = input(...): This line prompts you to enter an action (deposit, withdraw, check balance, or quit). The .lower() converts your input to lowercase for easier comparison.

4. if action == 'deposit': If you enter 'deposit', it prompts you for an amount, converts it to a float, and calls the deposit() method of your user_account. It includes a try-except block to handle cases where you might enter non-numeric input**.**

5. elif action == 'withdraw': If you enter 'withdraw', it works similarly to the deposit section, prompting for an amount and calling the withdraw() method. It also includes error handling for non-numeric input.

6. elif action == 'check balance': If you enter 'check balance', it calls the check_balance() method and prints the current balance.

7. elif action == 'quit': If you enter 'quit', it prints an exit message and the break statement exits the while loop, ending the program.

8. else: If you enter anything other than the valid actions, it prints an "Invalid action" message.

Task Description#5:

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

Examples:

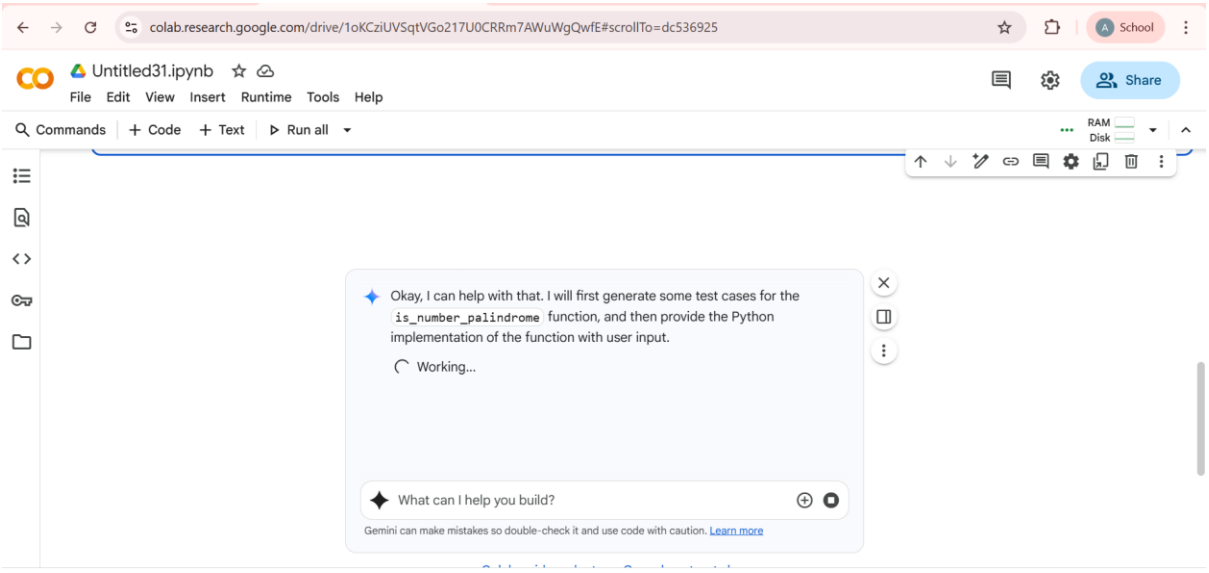
121 → True

123 → False

0, negative numbers → handled gracefully

Prompt:

Generate test cases for a function is_number_palindrome(num) that checks if an integer is a palindrome. Include examples like 121 → True, 123 → False, 0 → True, and negative numbers. Then implement the function in Python. with user input.



Code:

Implement the function

Subtask:

Write the Python code for the `is_number_palindrome(num)` function.

Reasoning: Implement the `is_number_palindrome` function as described in the instructions.

[5] ✓ 0s

```
def is_number_palindrome(num):
    """
    Checks if an integer is a palindrome.

    Args:
        num: The integer to check.

    Returns:
        True if the number is a palindrome, False otherwise.
    """
    if num < 0:
        return False
    num_str = str(num)
    return num_str == num_str[::-1]
```

[4] ✓ 0s

```
test_cases = [
    {"input": 121, "expected_output": True, "description": "Positive palindrome"},
    {"input": 123, "expected_output": False, "description": "Positive non-palindrome"},
    {"input": 0, "expected_output": True, "description": "Zero"},
    {"input": 7, "expected_output": True, "description": "Single-digit number"},
    {"input": -121, "expected_output": False, "description": "Negative number"},
    {"input": -101, "expected_output": False, "description": "Negative palindrome (should be False)"},
    {"input": 12321, "expected_output": True, "description": "Larger positive palindrome"},
    {"input": 12345, "expected_output": False, "description": "Larger positive non-palindrome"},
]
```

Add user input

Subtask:

Modify the code to take an integer as input from the user and then call the `is_number_palindrome()` function with the user's input.

Reasoning: The subtask requires getting user input, converting it to an integer, calling the `is_number_palindrome` function, and printing the result. This can be done in a single code block.

[6] ✓ 20s

```
user_input_str = input("Enter an integer: ")
try:
    user_input_int = int(user_input_str)
    is_palindrome = is_number_palindrome(user_input_int)
    if is_palindrome:
        print(f"The number {user_input_int} is a palindrome.")
    else:
        print(f"The number {user_input_int} is not a palindrome.")
except ValueError:
    print("Invalid input. Please enter an integer.")
```

```
Enter an integer: 232
The number 232 is a palindrome.
```

[7] ✓ 2m

```
def is_number_palindrome(num):
    """
    Checks if an integer is a palindrome.

    Args:
        num: The integer to check.

    Returns:
        True if the number is a palindrome, False otherwise.
    """
    if num < 0:
        return False
    num_str = str(num)
    return num_str == num_str[::-1]

user_input_str = input("Enter an integer: ")
try:
    user_input_int = int(user_input_str)
    is_palindrome = is_number_palindrome(user_input_int)
    if is_palindrome:
        print(f"The number {user_input_int} is a palindrome.")
    else:
        print(f"The number {user_input_int} is not a palindrome.")
except ValueError:
    print("Invalid input. Please enter an integer.")
```

Output:

```
Enter an integer: 232
The number 232 is a palindrome.
```

EXPLANATION:

This code defines a function is_number_palindrome that checks if an integer is a palindrome. It also includes code to get input from the user and use the function. Here's a breakdown:

1. def is_number_palindrome(num):: This defines the function that takes an integer num as input.

2. if num < 0:: This checks if the number is negative. Negative numbers are not considered palindromes in this implementation, so it returns False.

3. num_str = str(num): The number is converted to a string so that it can be easily reversed and compared.

4. return num_str == num_str[::-1]: This is the core of the palindrome check. It compares the string representation of the number with its reverse ([::-1] creates a reversed copy of the string). If they are the same, the number is a palindrome and the function returns True; otherwise, it returns False.

5. user_input_str = input("Enter an integer: "): This line prompts the user to enter an integer and stores their input as a string.

6. try...except ValueError:: This block attempts to convert the user's input string to an integer. If the input is not a valid integer, a ValueError is caught, and an error message is printed.

7. user_input_int = int(user_input_str): If the input is valid, it's converted to an integer.

8. is_palindrome = is_number_palindrome(user_input_int): The is_number_palindrome function is called with the user's integer input, and the result (True or False) is stored in the is_palindrome variable.

9. if is_palindrome:: This checks the value of is_palindrome and prints a message indicating whether the entered number is a palindrome or not.

---0---