PROGRAM : B.TECH/CSE

SPECIALIZATION : AIML

COURSE TITLE : AI ASSISTANT CODING

COURSE CODE : 24CS101PC214

SEMESTER : 3RD

NAME OF THE STUDENT : KARNAKANTI GODADEVI

ENROLLMENT NO : 2403A52003

BATCH NO : 01

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.

Expected Output#1

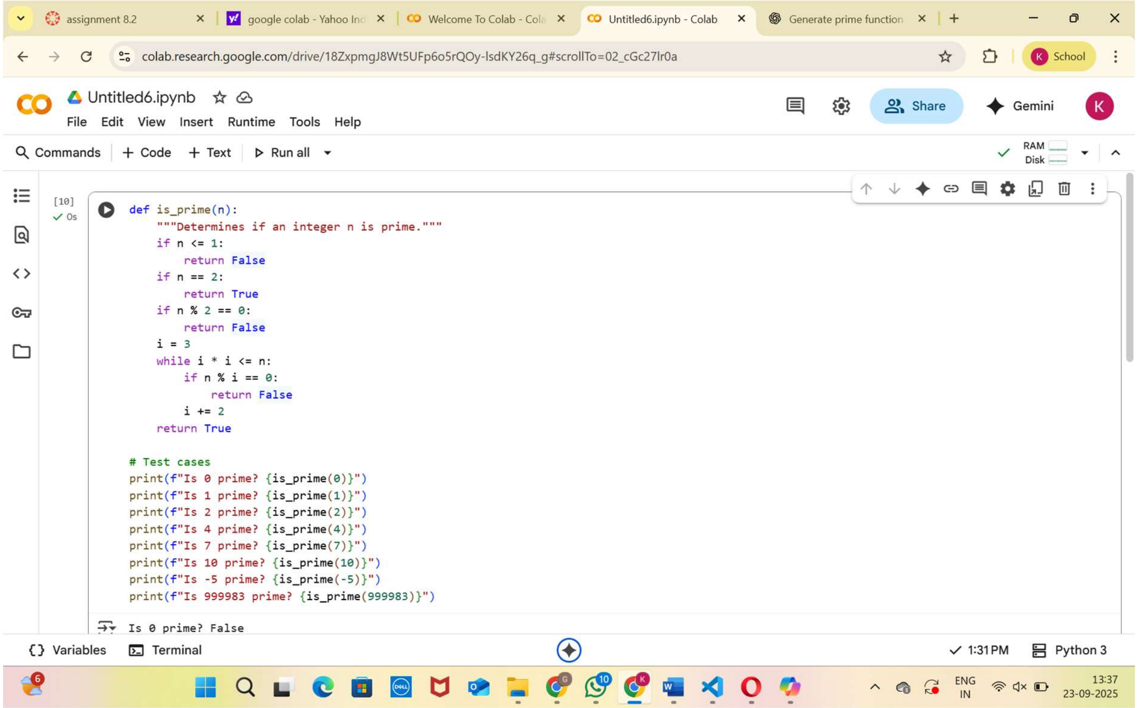
- A working prime checker that passes AI-generated tests using edge coverage.

Prompt:

Generate test cases and implement a function is_prime(n) that determines if an integer n is prime.

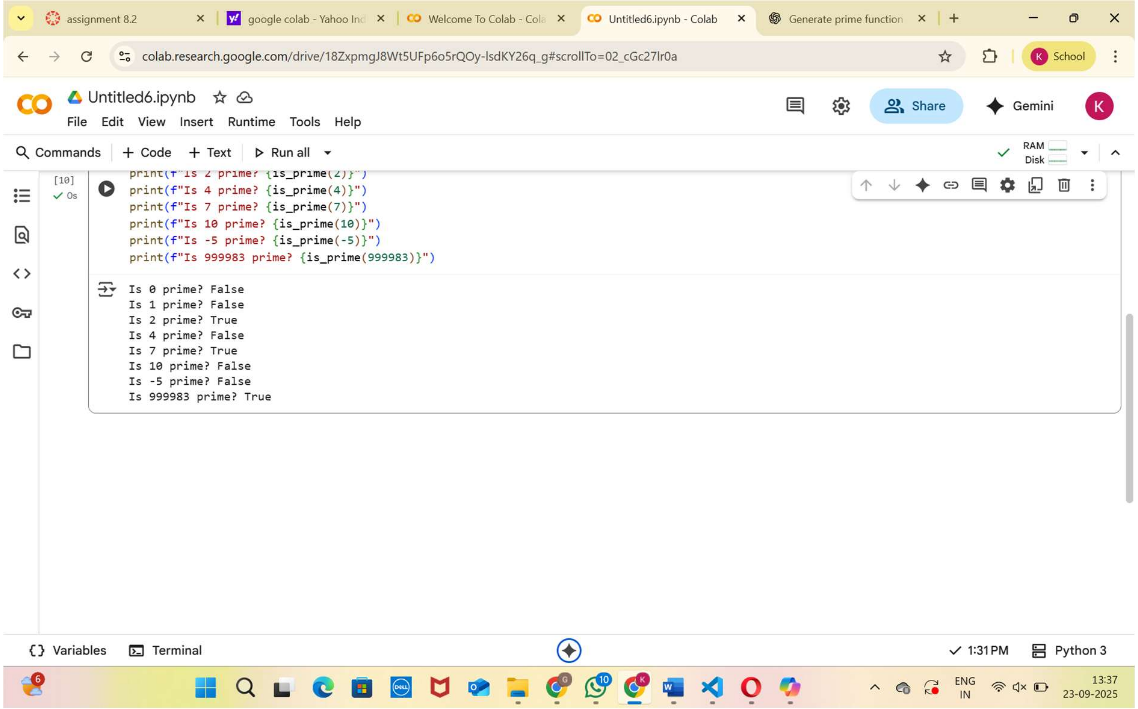
Requirements: Only integers > 1 can be prime,Check edge cases: 0, 1, 2, negative numbers, and large primes.

Code&Output:



```
def is_prime(n):
    """Determines if an integer n is prime."""
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True

# Test cases
print(f"Is 0 prime? {is_prime(0)}")
print(f"Is 1 prime? {is_prime(1)}")
print(f"Is 2 prime? {is_prime(2)}")
print(f"Is 4 prime? {is_prime(4)}")
print(f"Is 7 prime? {is_prime(7)}")
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is -5 prime? {is_prime(-5)}")
print(f"Is 999983 prime? {is_prime(999983)}")
```

```
Is 0 prime? False
```



```
print(f"Is 2 prime? {is_prime(2)}")
print(f"Is 4 prime? {is_prime(4)}")
print(f"Is 7 prime? {is_prime(7)}")
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is -5 prime? {is_prime(-5)}")
print(f"Is 999983 prime? {is_prime(999983)}")
```

```
Is 0 prime? False
Is 1 prime? False
Is 2 prime? True
Is 4 prime? False
Is 7 prime? True
Is 10 prime? False
Is -5 prime? False
Is 999983 prime? True
```

Observation:

This code defines a Python function called is_prime that checks if a given integer is a prime number, along with several test cases.

Here's a breakdown of the code:

1. **Function Definition**:

  - def is_prime(n): defines the function is_prime that takes one argument, n.

2. **Base Cases**:
   - if n <= 1: checks if the number is less than or equal to 1. Prime numbers are defined as integers greater than 1, so it returns False for these cases.
   - if n == 2: checks if the number is 2. 2 is the only even prime number, so it returns True.
   - if n % 2 == 0: checks if the number is even (divisible by 2). Since 2 is handled in the previous step, any other even number is not prime, so it returns False.
3. **Checking for Divisors**:
   - i = 3 initializes a variable i to 3. This is the first odd number to check as a potential divisor.
   - while i * i <= n: This loop continues as long as the square of i is less than or equal to n. We only need to check for divisors up to the square root of n because if n has a divisor larger than its square root, it must also have a divisor smaller than its square root.
   - if n % i == 0: checks if n is divisible by i with no remainder. If it is, n has a divisor other than 1 and itself, so it's not prime, and the function returns False.
   - i += 2 increments i by 2. This ensures that we only check odd numbers as potential divisors, as even divisors (other than 2, which is already checked) would have made n not prime in the earlier check.
4. **Returning True**:
   - If the loop completes without finding any divisors, it means n is not divisible by any number from 3 up to its square root (excluding even numbers), so it must be prime. The function then returns True.
5. **Test Cases**:
   - The lines starting with print(f"Is...") are test cases that call the is_prime function with different inputs (including 0, 1, 2, small numbers, negative numbers, and a large prime) and print whether the function returns True or False

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

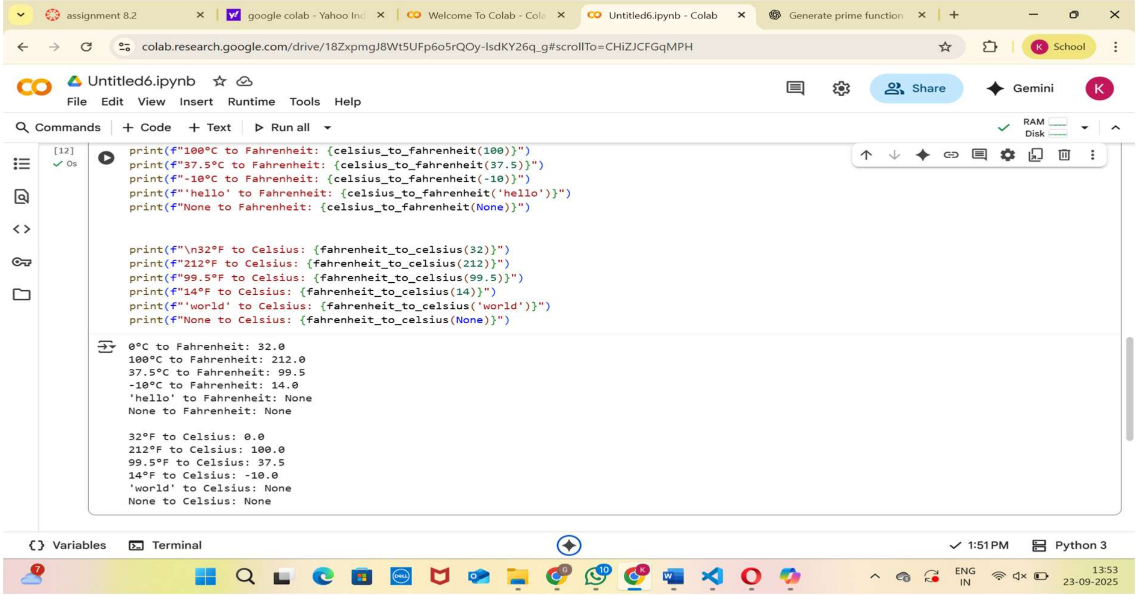
Expected Output#2

Dual conversion functions with complete test coverage and safe type handling

Prompt:

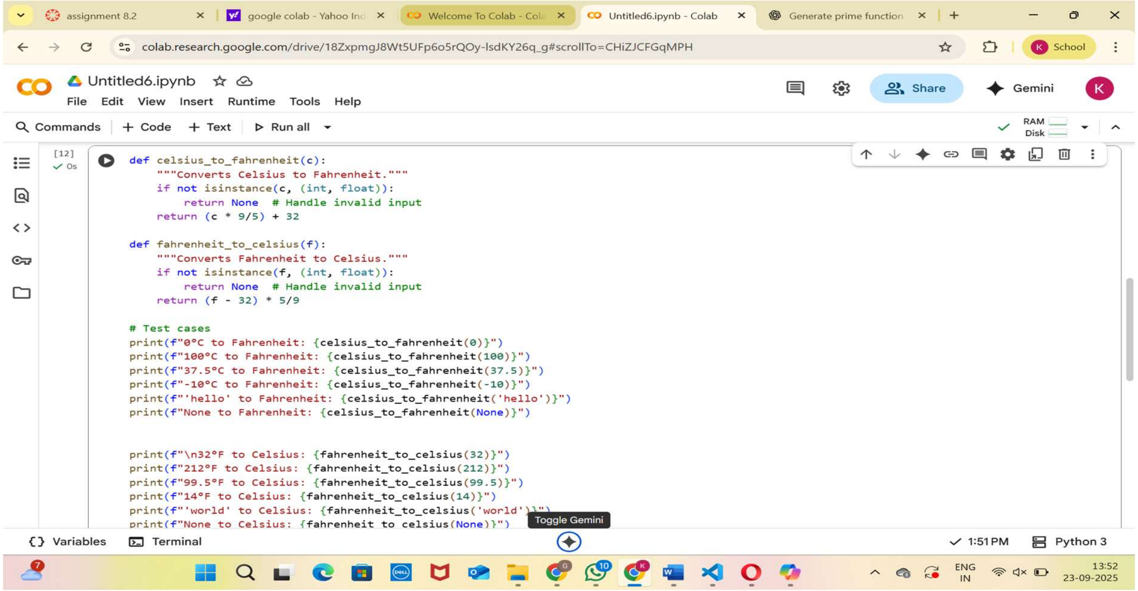
Generate test cases and implement two functions: celsius_to_fahrenheit(c) fahrenheit_to_celsius(f)Requirements • Validate known pairs: 0°C = 32°F, 100°C = 212°F. • Include decimals and invalid inputs like strings or None.

Code&Output:



```
print(f"100°C to Fahrenheit: {celsius_to_fahrenheit(100)}")
print(f"37.5°C to Fahrenheit: {celsius_to_fahrenheit(37.5)}")
print(f"-10°C to Fahrenheit: {celsius_to_fahrenheit(-10)}")
print(f"'hello' to Fahrenheit: {celsius_to_fahrenheit('hello')}")
print(f"None to Fahrenheit: {celsius_to_fahrenheit(None)}")


print(f"\n32°F to Celsius: {fahrenheit_to_celsius(32)}")
print(f"212°F to Celsius: {fahrenheit_to_celsius(212)}")
print(f"99.5°F to Celsius: {fahrenheit_to_celsius(99.5)}")
print(f"14°F to Celsius: {fahrenheit_to_celsius(14)}")
print(f"'world' to Celsius: {fahrenheit_to_celsius('world')}")
print(f"None to Celsius: {fahrenheit_to_celsius(None)}")
```

```
0°C to Fahrenheit: 32.0
100°C to Fahrenheit: 212.0
37.5°C to Fahrenheit: 99.5
-10°C to Fahrenheit: 14.0
'hello' to Fahrenheit: None
None to Fahrenheit: None

32°F to Celsius: 0.0
212°F to Celsius: 100.0
99.5°F to Celsius: 37.5
14°F to Celsius: -10.0
'world' to Celsius: None
None to Celsius: None
```



```
def celsius_to_fahrenheit(c):
    """Converts Celsius to Fahrenheit."""
    if not isinstance(c, (int, float)):
        return None  # Handle invalid input
    return (c * 9/5) + 32

def fahrenheit_to_celsius(f):
    """Converts Fahrenheit to Celsius."""
    if not isinstance(f, (int, float)):
        return None  # Handle invalid input
    return (f - 32) * 5/9

# Test cases
print(f"0°C to Fahrenheit: {celsius_to_fahrenheit(0)}")
print(f"100°C to Fahrenheit: {celsius_to_fahrenheit(100)}")
print(f"37.5°C to Fahrenheit: {celsius_to_fahrenheit(37.5)}")
print(f"-10°C to Fahrenheit: {celsius_to_fahrenheit(-10)}")
print(f"'hello' to Fahrenheit: {celsius_to_fahrenheit('hello')}")
print(f"None to Fahrenheit: {celsius_to_fahrenheit(None)}")


print(f"\n32°F to Celsius: {fahrenheit_to_celsius(32)}")
print(f"212°F to Celsius: {fahrenheit_to_celsius(212)}")
print(f"99.5°F to Celsius: {fahrenheit_to_celsius(99.5)}")
print(f"14°F to Celsius: {fahrenheit_to_celsius(14)}")
print(f"'world' to Celsius: {fahrenheit_to_celsius('world')}")
print(f"None to Celsius: {fahrenheit_to_celsius(None)}")
```

Observation:

This code defines two Python functions for converting temperatures between Celsius and Fahrenheit, and includes test cases to demonstrate their usage and handling of different inputs.

Here's a breakdown:

1. celsius_to_fahrenheit(c) Function:
   - def celsius_to_fahrenheit(c): defines the function that takes a Celsius temperature c as input.
   - if not isinstance(c, (int, float)): return None checks if the input c is not an integer or a float. If it's not a valid number type, the function returns None.
   - return (c * 9/5) + 32 performs the conversion from Celsius to Fahrenheit using the standard formula and returns the result.
2. fahrenheit_to_celsius(f) Function:
   - def fahrenheit_to_celsius(f): defines the function that takes a Fahrenheit temperature f as input.
   - if not isinstance(f, (int, float)): return None checks if the input f is not an integer or a float. If it's not a valid number type, the function returns None.
   - return (f - 32) * 5/9 performs the conversion from Fahrenheit to Celsius using the standard formula and returns the result.
3. Test Cases:
   - The print statements demonstrate the usage of both functions with various inputs:
     - Known conversion pairs (0°C = 32°F, 100°C = 212°F, 32°F = 0°C, 212°F = 100°C).
     - Inputs with decimal values.
     - Negative temperature inputs.
     - Invalid inputs like strings ('hello', 'world') and None, to show how the input validation is handled (returning None).

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.

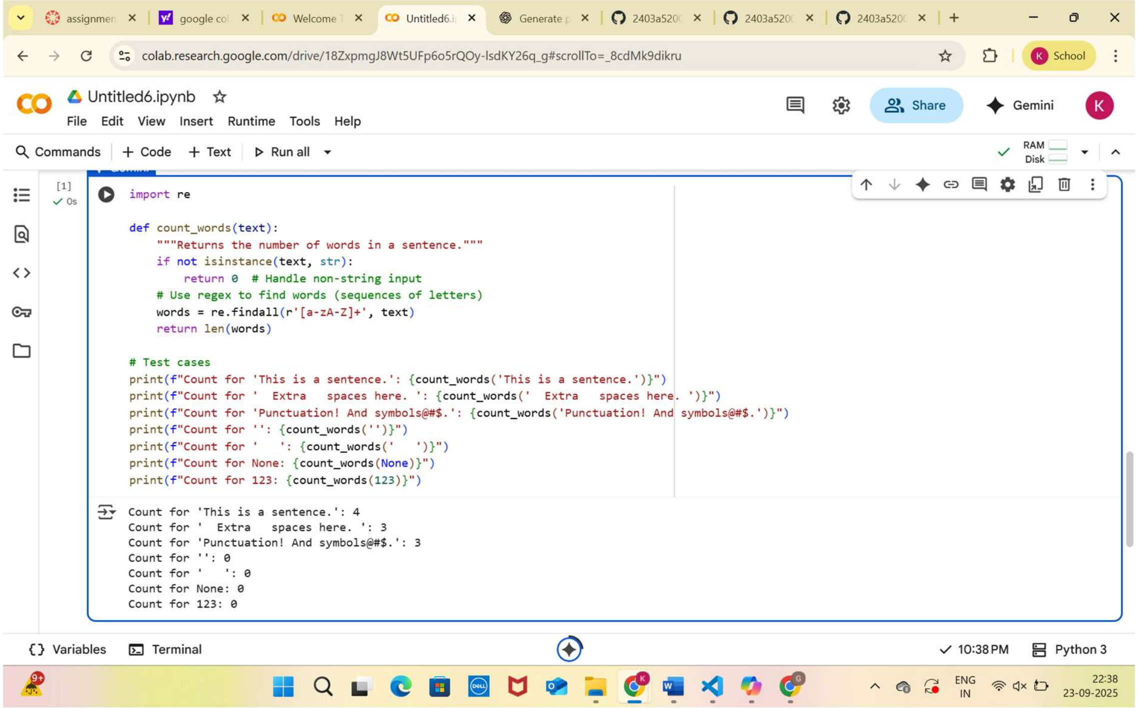
Expected Output#3

Accurate word count with robust test case validation**.**

Prompt:

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

Code&Output:


Untitled6.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
Variables Terminal
10:38 PM Python 3

```
import re

def count_words(text):
    """Returns the number of words in a sentence."""
    if not isinstance(text, str):
        return 0  # Handle non-string input
    # Use regex to find words (sequences of letters)
    words = re.findall(r'[a-zA-Z]+', text)
    return len(words)

# Test cases
print(f"Count for 'This is a sentence.': {count_words('This is a sentence.')}")
print(f"Count for '  Extra   spaces here. ': {count_words('  Extra   spaces here. ')}")
print(f"Count for 'Punctuation! And symbols@#$.': {count_words('Punctuation! And symbols@#$.')}")
print(f"Count for '': {count_words('')}")
print(f"Count for '   ': {count_words('   ')}")
print(f"Count for None: {count_words(None)}")
print(f"Count for 123: {count_words(123)}")
```

```
Count for 'This is a sentence.': 4
Count for '  Extra   spaces here. ': 3
Count for 'Punctuation! And symbols@#$.': 3
Count for '': 0
Count for '   ': 0
Count for None: 0
Count for 123: 0
```

Observation:

This code defines a Python function called count_words that counts the number of words in a given text string.

Here's how it works:

1. Function Definition:

   - def count_words(text): defines the function count_words that takes one argument, text.

2. Input Validation:
   - if not isinstance(text, str): return 0 checks if the input text is actually a string. If it's not (e.g., if it's a number, None, or another data type), the function returns 0 because it cannot count words in non-string input.

3. Word Counting using Regular Expressions:
   - import re imports the regular expression module in Python.
   - re.findall(r'[a-zA-Z]+', text) uses a regular expression to find all sequences of one or more letters (both lowercase a-z and uppercase A-Z) within the input text. This effectively finds what are considered "words" in this context, ignoring numbers, punctuation, and spaces.
   - words = ... stores the list of found words in the variable words.
   - return len(words) returns the number of items (words) found in the words list.

4. Test Cases:
   - The lines starting with print(f"Count for...") demonstrate how to use the count_words function with various inputs:
     - A normal sentence.
     - Text with extra spaces.
     - Text with punctuation and symbols.
     - An empty string.
     - A string with only spaces.
     - None and an integer, to show the input validation in action.

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.

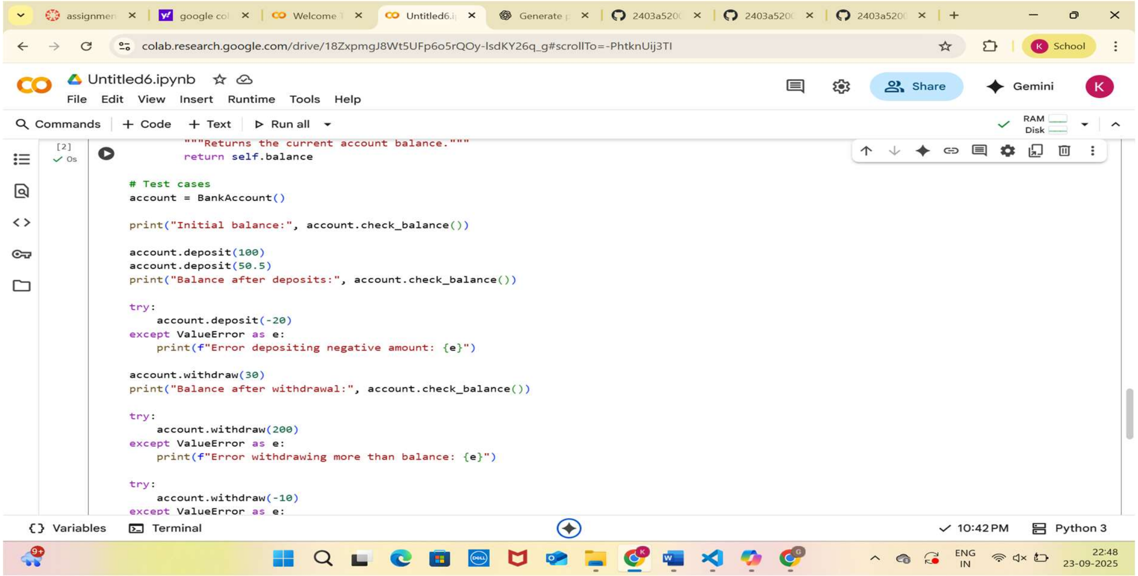
Expected Output#4

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

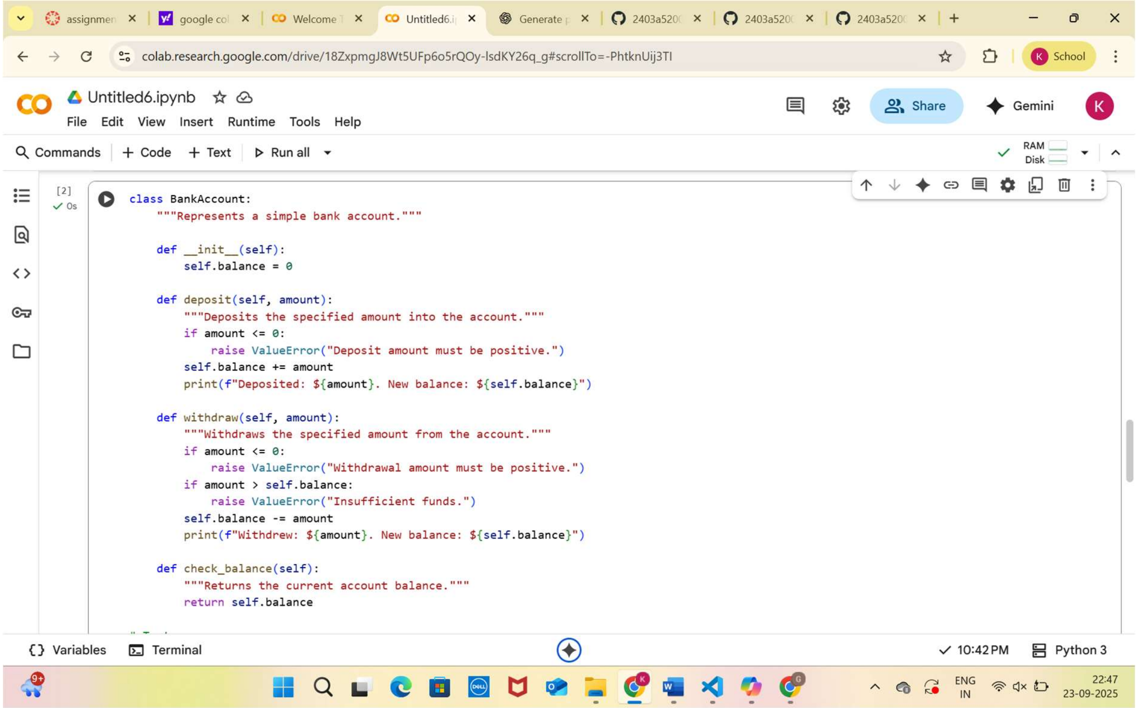
Prompt:

Generate test cases and implement a BankAccount class with the following methods: deposit(amount) withdraw(amount) check_balance() Requirements: • Negative deposits/withdrawals should raise an error. • Cannot withdraw more than balance.

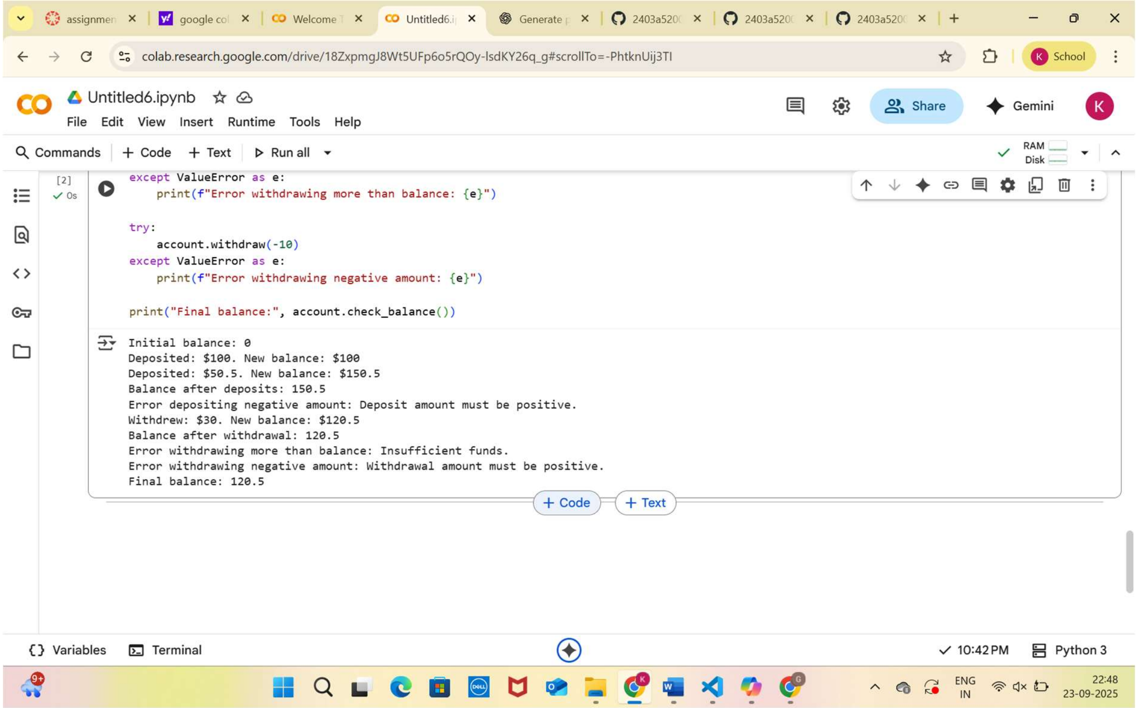
Code&Output:



```
        """Returns the current account balance."""
        return self.balance

# Test cases
account = BankAccount()

print("Initial balance:", account.check_balance())

account.deposit(100)
account.deposit(50.5)
print("Balance after deposits:", account.check_balance())

try:
    account.deposit(-20)
except ValueError as e:
    print(f"Error depositing negative amount: {e}")

account.withdraw(30)
print("Balance after withdrawal:", account.check_balance())

try:
    account.withdraw(200)
except ValueError as e:
    print(f"Error withdrawing more than balance: {e}")

try:
    account.withdraw(-10)
except ValueError as e:
```



```
class BankAccount:
    """Represents a simple bank account."""

    def __init__(self):
        self.balance = 0

    def deposit(self, amount):
        """Deposits the specified amount into the account."""
        if amount <= 0:
            raise ValueError("Deposit amount must be positive.")
        self.balance += amount
        print(f"Deposited: ${amount}. New balance: ${self.balance}")

    def withdraw(self, amount):
        """Withdraws the specified amount from the account."""
        if amount <= 0:
            raise ValueError("Withdrawal amount must be positive.")
        if amount > self.balance:
            raise ValueError("Insufficient funds.")
        self.balance -= amount
        print(f"Withdrew: ${amount}. New balance: ${self.balance}")

    def check_balance(self):
        """Returns the current account balance."""
        return self.balance
```



```
except ValueError as e:
    print(f"Error withdrawing more than balance: {e}")

try:
    account.withdraw(-10)
except ValueError as e:
    print(f"Error withdrawing negative amount: {e}")

print("Final balance:", account.check_balance())
```

```
Initial balance: 0
Deposited: $100. New balance: $100
Deposited: $50.5. New balance: $150.5
Balance after deposits: 150.5
Error depositing negative amount: Deposit amount must be positive.
Withdrew: $30. New balance: $120.5
Balance after withdrawal: 120.5
Error withdrawing more than balance: Insufficient funds.
Error withdrawing negative amount: Withdrawal amount must be positive.
Final balance: 120.5
```

Observation:

This code defines a Python class called BankAccount which simulates a simple bank account with methods for depositing, withdrawing, and checking the balance.

Here's a breakdown of the code:

1. **Class Definition**:
    - class BankAccount: defines a new class named BankAccount.
2. **Constructor (__init__)**:
    - def __init__(self): is the constructor method, called when a new BankAccount object is created.
    - self.balance = 0 initializes the balance attribute of the account to 0. self refers to the instance of the class.
3. **deposit Method**:
    - def deposit(self, amount): defines the method for depositing money. It takes self (the instance) and the amount to deposit as arguments.
    - if amount <= 0: checks if the deposit amount is not positive.
    - raise ValueError("Deposit amount must be positive.") raises a ValueError with a message if the amount is not positive.

  - self.balance += amount adds the valid deposit amount to the account's balance.
  - print(f"Deposited: ${amount}. New balance: ${self.balance}") prints a confirmation message.

4. **withdraw Method**:
   - def withdraw(self, amount): defines the method for withdrawing money. It takes self and the amount to withdraw as arguments.
   - if amount <= 0: checks if the withdrawal amount is not positive.
   - raise ValueError("Withdrawal amount must be positive.") raises a ValueError if the amount is not positive.
   - if amount > self.balance: checks if the withdrawal amount exceeds the current balance.
   - raise ValueError("Insufficient funds.") raises a ValueError if there are insufficient funds.
   - self.balance -= amount subtracts the valid withdrawal amount from the balance.
   - print(f"Withdrew: ${amount}. New balance: ${self.balance}") prints a confirmation message.
5. **check_balance Method**:
   - def check_balance(self): defines the method to check the current balance.
   - return self.balance returns the current value of the balance attribute.
6. **Test Cases**:
   - account = BankAccount() creates a new instance of the BankAccount class.
   - The subsequent print statements and try...except blocks demonstrate how to use the deposit, withdraw, and check_balance methods, including testing the error handling for invalid deposit and withdrawal amounts, and insufficient funds.

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

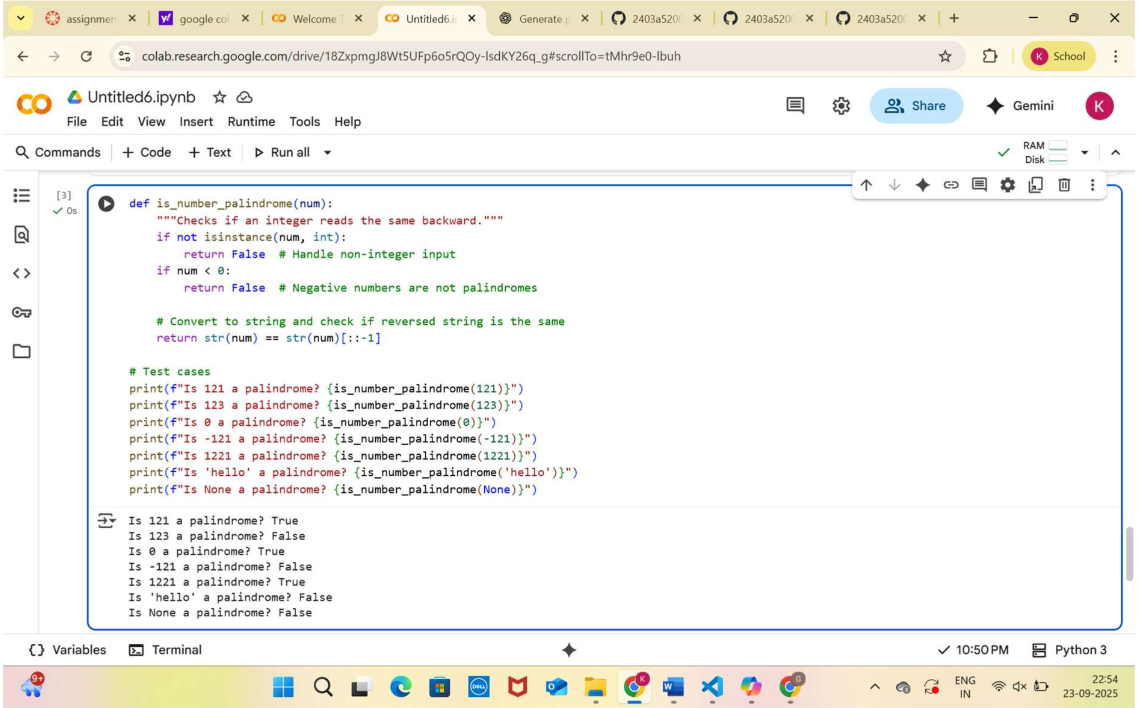
Expected Output#5

• Number-based palindrome checker function validated against test cases\

Prompt:

Generate test cases and implement a function is_number_palindrome(num) that checks if an integer reads the same backward.Examples: 121 → True 123 → False 0, negative numbers → handled gracefully

Code&Output:



```
def is_number_palindrome(num):
    """Checks if an integer reads the same backward."""
    if not isinstance(num, int):
        return False  # Handle non-integer input
    if num < 0:
        return False  # Negative numbers are not palindromes

    # Convert to string and check if reversed string is the same
    return str(num) == str(num)[::-1]

# Test cases
print(f"Is 121 a palindrome? {is_number_palindrome(121)}")
print(f"Is 123 a palindrome? {is_number_palindrome(123)}")
print(f"Is 0 a palindrome? {is_number_palindrome(0)}")
print(f"Is -121 a palindrome? {is_number_palindrome(-121)}")
print(f"Is 1221 a palindrome? {is_number_palindrome(1221)}")
print(f"Is 'hello' a palindrome? {is_number_palindrome('hello')}")
print(f"Is None a palindrome? {is_number_palindrome(None)}")
```

```
Is 121 a palindrome? True
Is 123 a palindrome? False
Is 0 a palindrome? True
Is -121 a palindrome? False
Is 1221 a palindrome? True
Is 'hello' a palindrome? False
Is None a palindrome? False
```

colab.research.google.com/drive/18ZxpmgJ8Wt5UFp6o5rQOy-lsdKY26q_g#scrollTo=tMhr9e0-lbuh
Untitled6.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
RAM Disk
[3] 0s
Variables Terminal
10:50 PM Python 3
ENG IN 22:54 23-09-2025

Observation:

This code defines a function is_number_palindrome that checks if an integer is a palindrome (reads the same forwards and backwards).

Here's a breakdown:

1. **Function Definition**:
    - def is_number_palindrome(num): defines the function that takes one argument, num.
2. **Input Validation**:
    - if not isinstance(num, int): return False checks if the input num is an integer. If not, it returns False.

   - if num < 0: return False checks if the number is negative. Negative numbers are not considered palindromes in this implementation.

3. **Palindrome Check**:
   - str(num) converts the integer to a string.
   - str(num)[::-1] creates a reversed version of the string. The [::-1] is a slicing technique that reverses a sequence.
   - str(num) == str(num)[::-1] compares the original string with the reversed string. If they are equal, the number is a palindrome and the function returns True; otherwise, it returns False.
4. **Test Cases**:
   - The lines starting with print(f"Is...") are test cases that call the is_number_palindrome function with different inputs (including palindromes, non-palindromes, negative numbers, strings, and None) and print the result.