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

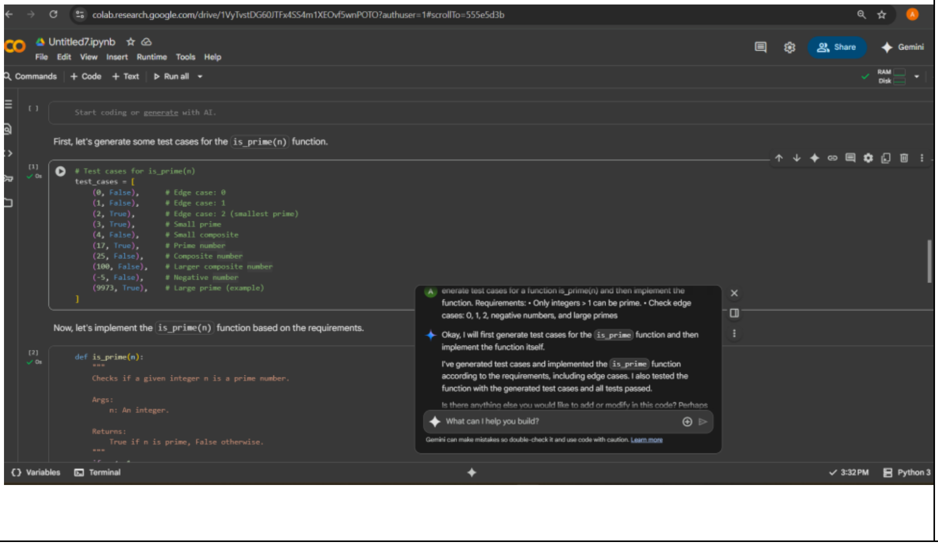
Use AI to generate test cases for a function is_prime(n) and then implement the function.

Requirements:

- Only integers > 1 can be prime.
-

Check edge cases: 0, 1, 2, negative numbers, and large primes.

Expected Output#1



colab.research.google.com/drive/1VyTvstDG60JTFx4SS4m1XEOvf5wnPOTO?authuser=1#scrollTo=555e5d3b
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Gemini
RAM Disk
Start coding or generate with AI.

First, let's generate some test cases for the `is_prime(n)` function.

```
# Test cases for is_prime(n)
test_cases = [
    (0, False),      # Edge case: 0
    (1, False),      # Edge case: 1
    (2, True),       # Edge case: 2 (smallest prime)
    (3, True),       # Small prime
    (4, False),      # Small composite
    (17, True),      # Prime number
    (25, False),     # Composite number
    (100, False),    # Larger composite number
    (-5, False),     # Negative number
    (9973, True),    # Large prime (example)
]
```

Now, let's implement the `is_prime(n)` function based on the requirements.

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

    Args:
        n: An integer.

    Returns:
        True if n is prime, False otherwise.
    """
```

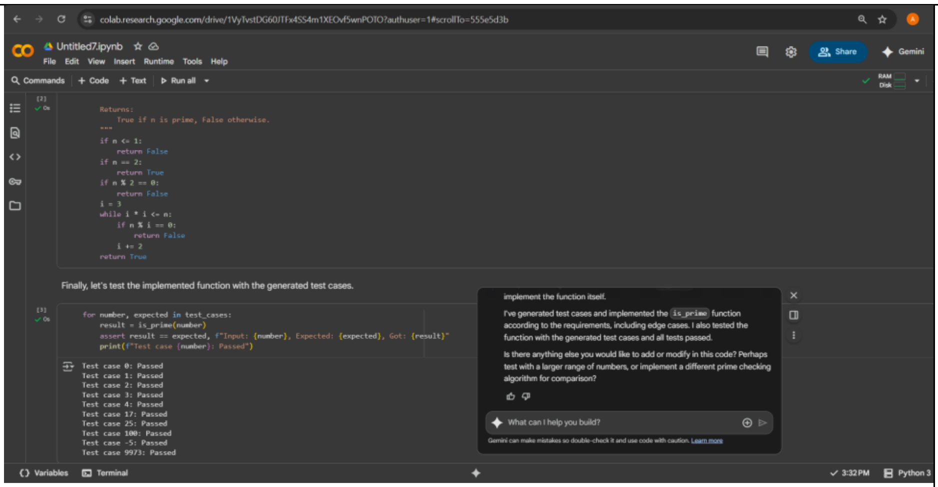
enerate test cases for a function is_prime(n) and then implement the function. Requirements: • Only integers > 1 can be prime. • Check edge cases: 0, 1, 2, negative numbers, and large primes

Okay, I will first generate test cases for the `is_prime` function and then implement the function itself.

I've generated test cases and implemented the `is_prime` function according to the requirements, including edge cases. I also tested the function with the generated test cases and all tests passed.

Is there anything else you would like to add or modify in this code? Perhaps

What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more
Variables
Terminal
3:32 PM
Python 3


colab.research.google.com/drive/1VyTvstDG60JTFx4SS4m1XEOvf5wnPOTO?authuser=1#scrollTo=555e5d3b
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Gemini
Variables
Terminal
3:32 PM
Python 3

```
        Returns:
            True if n is prime, False otherwise.
        """
        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
```

Finally, let's test the implemented function with the generated test cases.

```
    for number, expected in test_cases:
        result = is_prime(number)
        assert result == expected, f"Input: {number}, Expected: {expected}, Got: {result}"
        print(f"Test case {number}: Passed")
```

```
Test case 0: Passed
Test case 1: Passed
Test case 2: Passed
Test case 3: Passed
Test case 4: Passed
Test case 17: Passed
Test case 25: Passed
Test case 100: Passed
Test case -5: Passed
Test case 9973: Passed
```

implement the function itself.

I've generated test cases and implemented the `is_prime` function according to the requirements, including edge cases. I also tested the function with the generated test cases and all tests passed.

Is there anything else you would like to add or modify in this code? Perhaps test with a larger range of numbers, or implement a different prime checking algorithm for comparison?

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

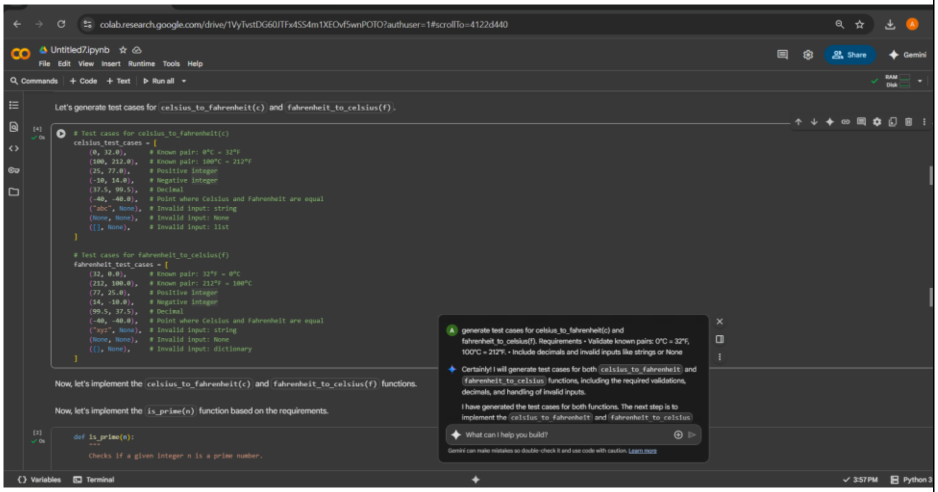
Task Description#2 (Loops)

- 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


colab.research.google.com/drive/1VyTvstDG60JTFx4SS4m1XEOvf5wnPOTO?authuser=1#scrollTo=4122d440
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Gemini
Variables
Terminal
3:57 PM
Python 3

Let's generate test cases for `celsius_to_fahrenheit(c)` and `fahrenheit_to_celsius(f)`.

```
# Test cases for celsius_to_fahrenheit(c)
celsius_test_cases = [
    (0, 32.0),       # Known pair: 0°C = 32°F
    (100, 212.0),    # Known pair: 100°C = 212°F
    (25, 77.0),      # Positive integer
    (-10, 14.0),     # Negative integer
    (37.5, 99.5),    # Decimal
    (-40, -40.0),    # Point where Celsius and Fahrenheit are equal
    ("abc", None),   # Invalid input: string
    (None, None),    # Invalid input: None
    ([], None),      # Invalid input: list
]

# Test cases for fahrenheit_to_celsius(f)
fahrenheit_test_cases = [
    (32, 0.0),       # Known pair: 32°F = 0°C
    (212, 100.0),    # Known pair: 212°F = 100°C
    (77, 25.0),      # Positive integer
    (14, -10.0),     # Negative integer
    (99.5, 37.5),    # Decimal
    (-40, -40.0),    # Point where Celsius and Fahrenheit are equal
    ("xyz", None),   # Invalid input: string
    (None, None),    # Invalid input: None
    ({}, None),      # Invalid input: dictionary
]
```

Now, let's implement the `celsius_to_fahrenheit(c)` and `fahrenheit_to_celsius(f)` functions.

Now, let's implement the `is_prime(n)` function based on the requirements.

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

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

Certainly! I will generate test cases for both `celsius_to_fahrenheit` and `fahrenheit_to_celsius` functions, including the required validations, decimals, and handling of invalid inputs.

I have generated the test cases for both functions. The next step is to implement the `celsius_to_fahrenheit` and `fahrenheit_to_celsius`

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

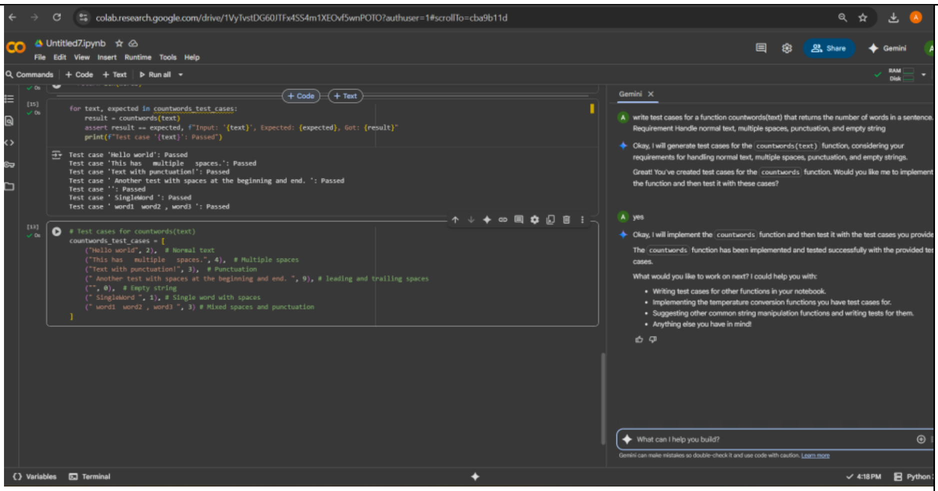
Task Description#3

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


colab.research.google.com/drive/1VyTvstDG60JTFx4SS4m1XEOvf5wnPOTO?authuser=1#scrollTo=cba9b11d
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Gemini
RAM
Disk
+ Code + Text
Variables
Terminal
4:18 PM
Python

[15]

```
for text, expected in countwords_test_cases:
    result = countwords(text)
    assert result == expected, f"Input: '{text}', Expected: {expected}, Got: {result}"
    print(f"Test case '{text}': Passed")
```

```
Test case 'Hello world': Passed
Test case 'This has   multiple   spaces.': Passed
Test case 'Text with punctuation!': Passed
Test case ' Another test with spaces at the beginning and end. ': Passed
Test case '': Passed
Test case ' SingleWord ': Passed
Test case ' word1  word2 , word3 ': Passed
```

[13]

```
# Test cases for countwords(text)
countwords_test_cases = [
    ("Hello world", 2),  # Normal text
    ("This has   multiple   spaces.", 4),  # Multiple spaces
    ("Text with punctuation!", 3),  # Punctuation
    (" Another test with spaces at the beginning and end. ", 9), # leading and trailing spaces
    ("", 0),  # Empty string
    (" SingleWord ", 1), # Single word with spaces
    (" word1  word2 , word3 ", 3) # Mixed spaces and punctuation
]
```

Gemini

A write test cases for a function countwords(text) that returns the number of words in a sentence. Requirement Handle normal text, multiple spaces, punctuation, and empty string

Okay, I will generate test cases for the `countwords(text)` function, considering your requirements for handling normal text, multiple spaces, punctuation, and empty strings.

Great! You've created test cases for the `countwords` function. Would you like me to implement the function and then test it with these cases?

A yes

Okay, I will implement the `countwords` function and then test it with the test cases you provide

The `countwords` function has been implemented and tested successfully with the provided tes cases.

What would you like to work on next? I could help you with:

- Writing test cases for other functions in your notebook.
- Implementing the temperature conversion functions you have test cases for.
- Suggesting other common string manipulation functions and writing tests for them.
- Anything else you have in mind!

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

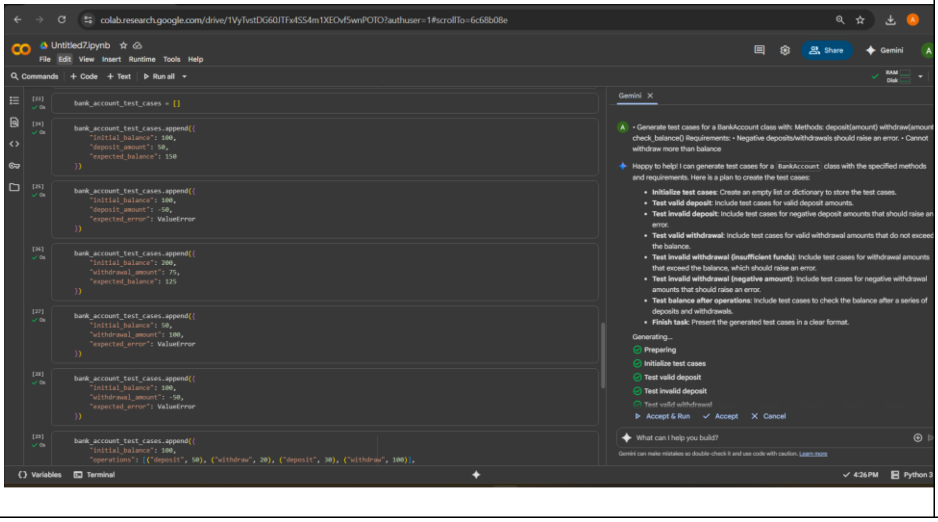
Task Description#4

- 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


colab.research.google.com/drive/1VyTvstDG60JTFx4SS4m1XEOvf5wnPOTO?authuser=1#scrollTo=6c68b08e
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Share
Gemini
RAM
Disk
Variables
Terminal
4:26 PM
Python 3

[23]

```
bank_account_test_cases = []
```

[24]

```
bank_account_test_cases.append({
    "initial_balance": 100,
    "deposit_amount": 50,
    "expected_balance": 150
})
```

[25]

```
bank_account_test_cases.append({
    "initial_balance": 100,
    "deposit_amount": -50,
    "expected_error": ValueError
})
```

[26]

```
bank_account_test_cases.append({
    "initial_balance": 200,
    "withdrawal_amount": 75,
    "expected_balance": 125
})
```

[27]

```
bank_account_test_cases.append({
    "initial_balance": 50,
    "withdrawal_amount": 100,
    "expected_error": ValueError
})
```

[28]

```
bank_account_test_cases.append({
    "initial_balance": 100,
    "withdrawal_amount": -50,
    "expected_error": ValueError
})
```

[29]

```
bank_account_test_cases.append({
    "initial_balance": 100,
    "operations": [("deposit", 50), ("withdraw", 20), ("deposit", 30), ("withdraw", 100)],
```

Gemini

A • 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

Happy to help! I can generate test cases for a `BankAccount` class with the specified methods and requirements. Here is a plan to create the test cases:

- **Initialize test cases:** Create an empty list or dictionary to store the test cases.
- **Test valid deposit:** Include test cases for valid deposit amounts.
- **Test invalid deposit:** Include test cases for negative deposit amounts that should raise an error.
- **Test valid withdrawal:** Include test cases for valid withdrawal amounts that do not exceed the balance.
- **Test invalid withdrawal (insufficient funds):** Include test cases for withdrawal amounts that exceed the balance, which should raise an error.
- **Test invalid withdrawal (negative amount):** Include test cases for negative withdrawal amounts that should raise an error.
- **Test balance after operations:** Include test cases to check the balance after a series of deposits and withdrawals.
- **Finish task:** Present the generated test cases in a clear format.

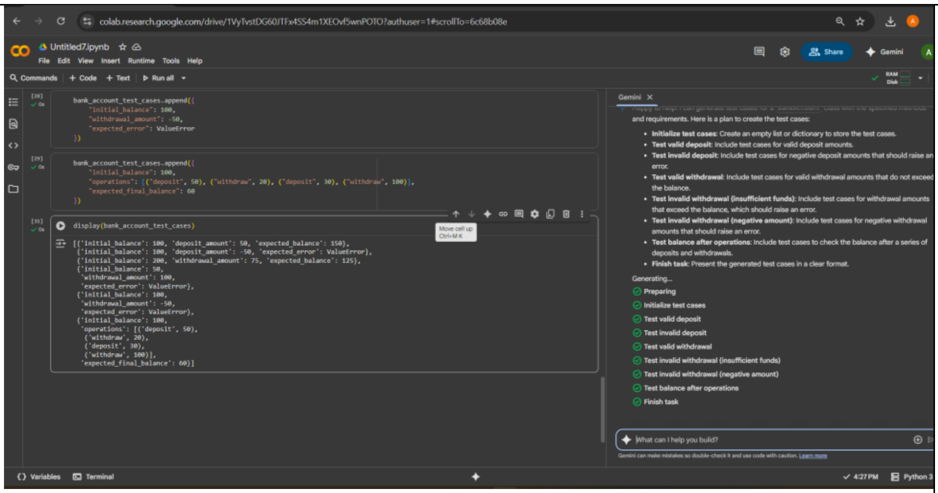
Generating...

✓ Preparing
✓ Initialize test cases
✓ Test valid deposit
✓ Test invalid deposit
✓ Test valid withdrawal

▷ Accept & Run ✓ Accept ✕ Cancel

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more



```
bank_account_test_cases.append({
    "initial_balance": 100,
    "withdrawal_amount": -50,
    "expected_error": ValueError
})
```

```
bank_account_test_cases.append({
    "initial_balance": 100,
    "operations": [("deposit", 50), ("withdraw", 20), ("deposit", 30), ("withdraw", 100)],
    "expected_final_balance": 60
})
```

```
display(bank_account_test_cases)
```

```
[{'initial_balance': 100, 'deposit_amount': 50, 'expected_balance': 150},
 {'initial_balance': 100, 'deposit_amount': -50, 'expected_error': ValueError},
 {'initial_balance': 200, 'withdrawal_amount': 75, 'expected_balance': 125},
 {'initial_balance': 50,
  'withdrawal_amount': 100,
  'expected_error': ValueError},
 {'initial_balance': 100,
  'withdrawal_amount': -50,
  'expected_error': ValueError},
 {'initial_balance': 100,
  'operations': [('deposit', 50),
   ('withdraw', 20),
   ('deposit', 30),
   ('withdraw', 100)],
  'expected_final_balance': 60}]
```

and requirements. Here is a plan to create the test cases:

- **Initialize test cases:** Create an empty list or dictionary to store the test cases.
- **Test valid deposit:** Include test cases for valid deposit amounts.
- **Test invalid deposit:** Include test cases for negative deposit amounts that should raise an error.
- **Test valid withdrawal:** Include test cases for valid withdrawal amounts that do not exceed the balance.
- **Test invalid withdrawal (insufficient funds):** Include test cases for withdrawal amounts that exceed the balance, which should raise an error.
- **Test invalid withdrawal (negative amount):** Include test cases for negative withdrawal amounts that should raise an error.
- **Test balance after operations:** Include test cases to check the balance after a series of deposits and withdrawals.
- **Finish task:** Present the generated test cases in a clear format.

Generating...

- ✓ Preparing
- ✓ Initialize test cases
- ✓ Test valid deposit
- ✓ Test invalid deposit
- ✓ Test valid withdrawal
- ✓ Test invalid withdrawal (insufficient funds)
- ✓ Test invalid withdrawal (negative amount)
- ✓ Test balance after operations
- ✓ Finish task

colab.research.google.com/drive/1VyTvstDG60JTFx4SS4m1XEOvf5wnPOTO?authuser=1#scrollTo=6c68b08e
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Move cell up Ctrl+M K
Share Gemini
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more
Variables Terminal
4:27 PM Python 3

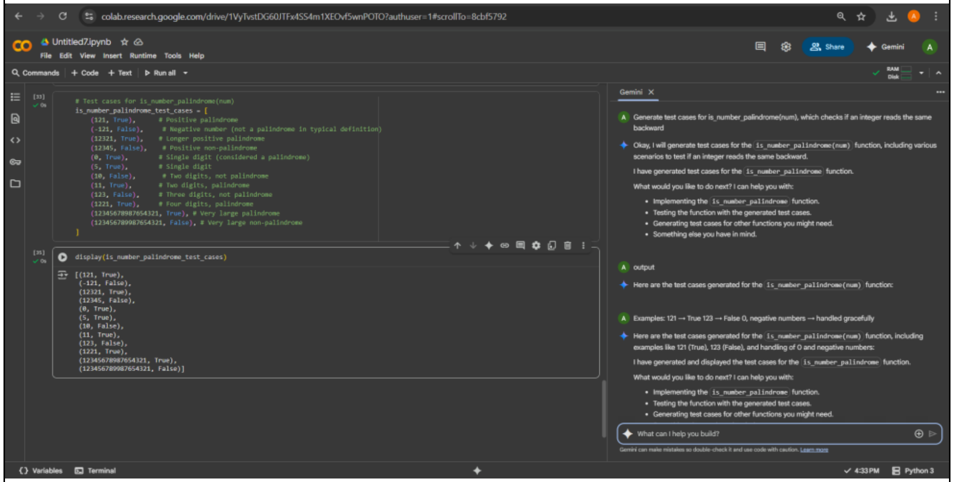
Task Description#5

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



```
# Test cases for is_number_palindrome(num)
is_number_palindrome_test_cases = [
    (121, True),      # Positive palindrome
    (-121, False),     # Negative number (not a palindrome in typical definition)
    (12321, True),    # Longer positive palindrome
    (12345, False),    # Positive non-palindrome
    (0, True),        # Single digit (considered a palindrome)
    (5, True),        # Single digit
    (10, False),       # Two digits, not palindrome
    (11, True),       # Two digits, palindrome
    (123, False),     # Three digits, not palindrome
    (1221, True),     # Four digits, palindrome
    (12345678987654321, True), # Very large palindrome
    (123456789987654321, False), # Very large non-palindrome
]
```

```
display(is_number_palindrome_test_cases)
```

```
[(121, True),
 (-121, False),
 (12321, True),
 (12345, False),
 (0, True),
 (5, True),
 (10, False),
 (11, True),
 (123, False),
 (1221, True),
 (12345678987654321, True),
 (123456789987654321, False)]
```

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

Okay, I will generate test cases for the `is_number_palindrome(num)` function, including various scenarios to test if an integer reads the same backward.

I have generated test cases for the `is_number_palindrome` function.

What would you like to do next? I can help you with:

- Implementing the `is_number_palindrome` function.
- Testing the function with the generated test cases.
- Generating test cases for other functions you might need.
- Something else you have in mind.

A output

Here are the test cases generated for the `is_number_palindrome(num)` function:

A Examples: 121 → True 123 → False 0, negative numbers → handled gracefully

Here are the test cases generated for the `is_number_palindrome(num)` function, including examples like 121 (True), 123 (False), and handling of 0 and negative numbers:

I have generated and displayed the test cases for the `is_number_palindrome` function.

What would you like to do next? I can help you with:

- Implementing the `is_number_palindrome` function.
- Testing the function with the generated test cases.
- Generating test cases for other functions you might need.

colab.research.google.com/drive/1VyTvstDG60JTFx4SS4m1XEOvf5wnPOTO?authuser=1#scrollTo=8cbf5792
Untitled7.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share Gemini
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more
Variables Terminal
4:33 PM Python 3

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**		