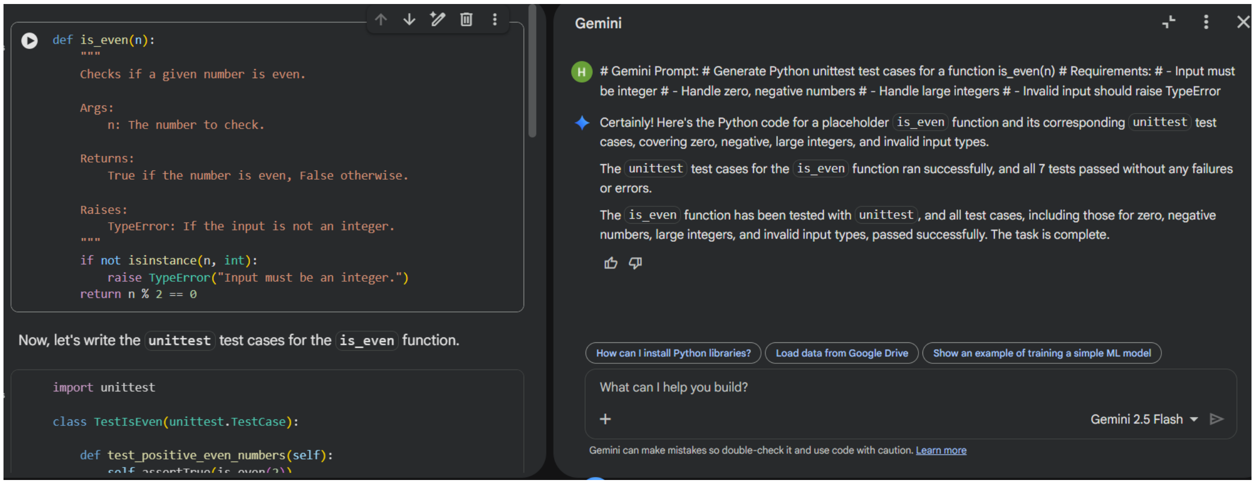
ASSIGNMENT – 8.2

Thumma Hasini | 2303A52076 | Batch – 37

Task 1 – Test-Driven Development for Even/Odd Number Validator

Use AI tools to first generate test cases for a function is_even(n) and then implement the function so that it satisfies all generated tests.

Prompt: Generate Python unittest test cases for a function is_even(n) with integer validation, zero, negative, large integers, and invalid input handling.



```
def is_even(n):
    """
    Checks if a given number is even.

    Args:
        n: The number to check.

    Returns:
        True if the number is even, False otherwise.

    Raises:
        TypeError: If the input is not an integer.
    """
    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")
    return n % 2 == 0
```

Now, let's write the `unittest` test cases for the `is_even` function.

```
import unittest

class TestIsEven(unittest.TestCase):

    def test_positive_even_numbers(self):
        self.assertTrue(is_even(2))
```

Gemini

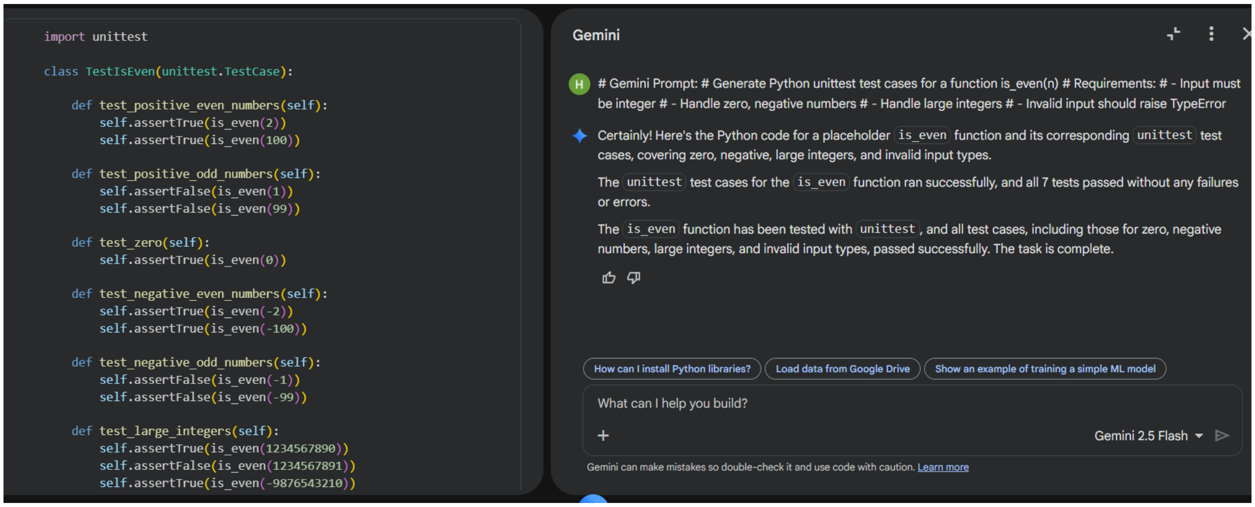
Gemini Prompt: # Generate Python unittest test cases for a function is_even(n) # Requirements: # - Input must be integer # - Handle zero, negative numbers # - Handle large integers # - Invalid input should raise TypeError

Certainly! Here's the Python code for a placeholder `is_even` function and its corresponding `unittest` test cases, covering zero, negative, large integers, and invalid input types.

The `unittest` test cases for the `is_even` function ran successfully, and all 7 tests passed without any failures or errors.

The `is_even` function has been tested with `unittest`, and all test cases, including those for zero, negative numbers, large integers, and invalid input types, passed successfully. The task is complete.

How can I install Python libraries?
Load data from Google Drive
Show an example of training a simple ML model
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more



```
import unittest

class TestIsEven(unittest.TestCase):

    def test_positive_even_numbers(self):
        self.assertTrue(is_even(2))
        self.assertTrue(is_even(100))

    def test_positive_odd_numbers(self):
        self.assertFalse(is_even(1))
        self.assertFalse(is_even(99))

    def test_zero(self):
        self.assertTrue(is_even(0))

    def test_negative_even_numbers(self):
        self.assertTrue(is_even(-2))
        self.assertTrue(is_even(-100))

    def test_negative_odd_numbers(self):
        self.assertFalse(is_even(-1))
        self.assertFalse(is_even(-99))

    def test_large_integers(self):
        self.assertTrue(is_even(1234567890))
        self.assertFalse(is_even(1234567891))
        self.assertTrue(is_even(-9876543210))
```

Gemini

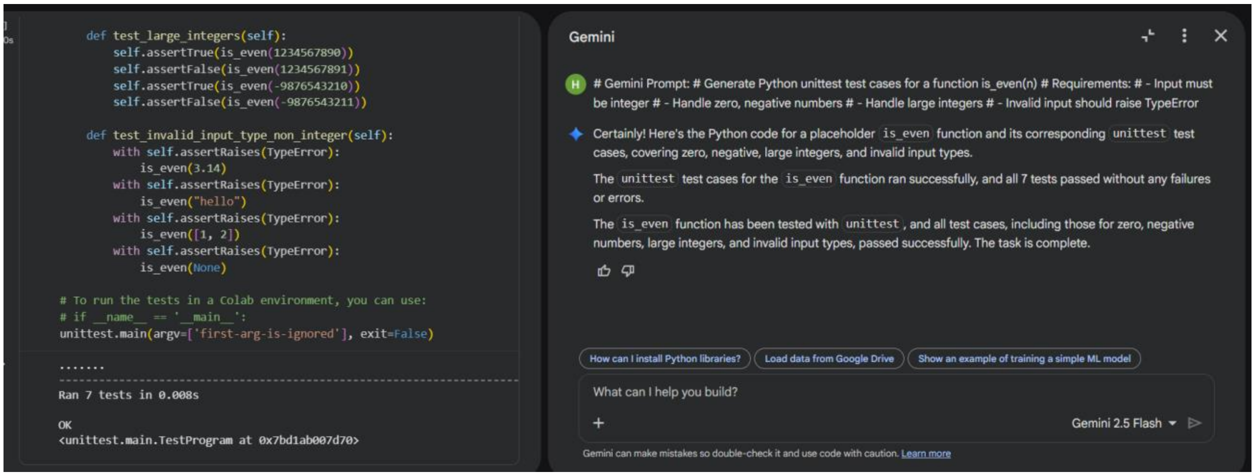
Gemini Prompt: # Generate Python unittest test cases for a function is_even(n) # Requirements: # - Input must be integer # - Handle zero, negative numbers # - Handle large integers # - Invalid input should raise TypeError

Certainly! Here's the Python code for a placeholder `is_even` function and its corresponding `unittest` test cases, covering zero, negative, large integers, and invalid input types.

The `unittest` test cases for the `is_even` function ran successfully, and all 7 tests passed without any failures or errors.

The `is_even` function has been tested with `unittest`, and all test cases, including those for zero, negative numbers, large integers, and invalid input types, passed successfully. The task is complete.

How can I install Python libraries?
Load data from Google Drive
Show an example of training a simple ML model
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more



```
    def test_large_integers(self):
        self.assertTrue(is_even(1234567890))
        self.assertFalse(is_even(1234567891))
        self.assertTrue(is_even(-9876543210))
        self.assertFalse(is_even(-9876543211))

    def test_invalid_input_type_non_integer(self):
        with self.assertRaises(TypeError):
            is_even(3.14)
        with self.assertRaises(TypeError):
            is_even("hello")
        with self.assertRaises(TypeError):
            is_even([1, 2])
        with self.assertRaises(TypeError):
            is_even(None)

# To run the tests in a Colab environment, you can use:
# if __name__ == '__main__':
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.......
----------------------------------------------------------------------
Ran 7 tests in 0.008s

OK
<unittest.main.TestProgram at 0x7bd1ab007d70>
```

Gemini Prompt: # Generate Python unittest test cases for a function is_even(n) # Requirements: # - Input must be integer # - Handle zero, negative numbers # - Handle large integers # - Invalid input should raise TypeError

Certainly! Here's the Python code for a placeholder `is_even` function and its corresponding `unittest` test cases, covering zero, negative, large integers, and invalid input types.

The `unittest` test cases for the `is_even` function ran successfully, and all 7 tests passed without any failures or errors.

The `is_even` function has been tested with `unittest`, and all test cases, including those for zero, negative numbers, large integers, and invalid input types, passed successfully. The task is complete.

How can I install Python libraries?
Load data from Google Drive
Show an example of training a simple ML model
What can I help you build?
+
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more

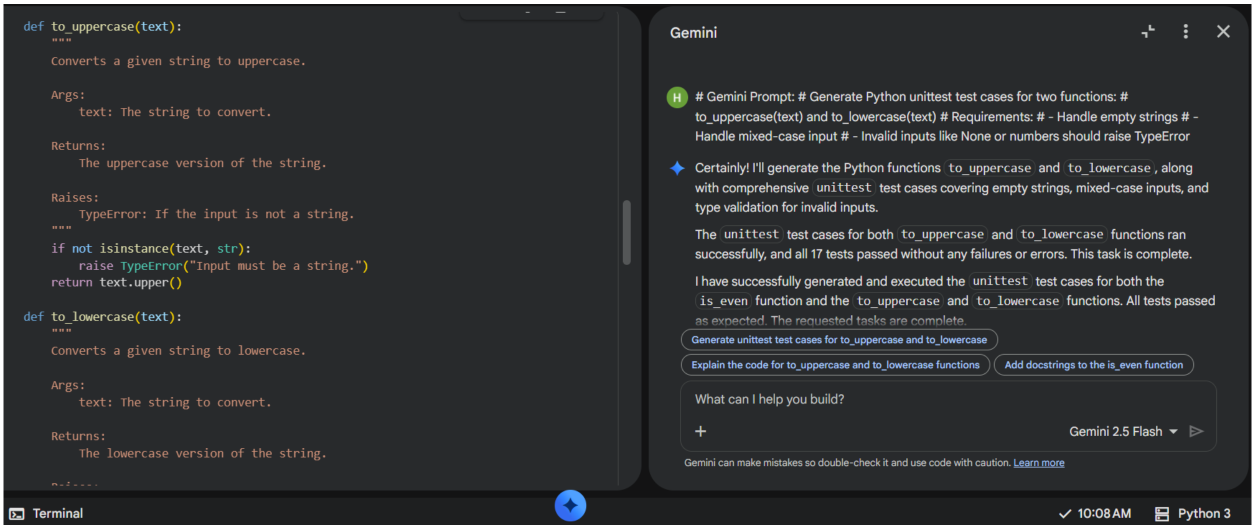
Explanation: In this task, test cases were generated using Gemini following the Test-Driven Development approach. The is_even() function was implemented to satisfy all test conditions, including handling negative numbers, zero, large integers, and raising errors for invalid inputs.

Task 2 – Test-Driven Development for String Case Converter

Ask AI to generate test cases for two functions:

- to_uppercase(text)
- to_lowercase(text)

Prompt: Generate unittest test cases for to_uppercase(text) and to_lowercase(text) functions with empty string, mixed-case, and invalid input handling.



```
def to_uppercase(text):
    """
    Converts a given string to uppercase.

    Args:
        text: The string to convert.

    Returns:
        The uppercase version of the string.

    Raises:
        TypeError: If the input is not a string.
    """
    if not isinstance(text, str):
        raise TypeError("Input must be a string.")
    return text.upper()

def to_lowercase(text):
    """
    Converts a given string to lowercase.

    Args:
        text: The string to convert.

    Returns:
        The lowercase version of the string.
```

Gemini

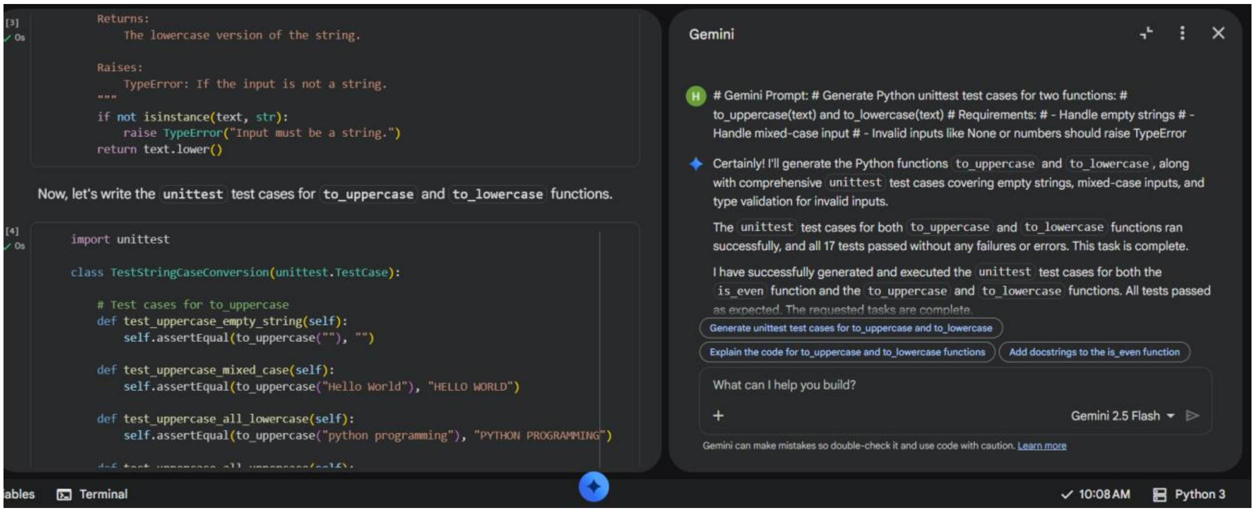
Gemini Prompt: # Generate Python unittest test cases for two functions: # to_uppercase(text) and to_lowercase(text) # Requirements: # - Handle empty strings # - Handle mixed-case input # - Invalid inputs like None or numbers should raise TypeError

Certainly! I'll generate the Python functions `to_uppercase` and `to_lowercase`, along with comprehensive `unittest` test cases covering empty strings, mixed-case inputs, and type validation for invalid inputs.

The `unittest` test cases for both `to_uppercase` and `to_lowercase` functions ran successfully, and all 17 tests passed without any failures or errors. This task is complete.

I have successfully generated and executed the `unittest` test cases for both the `is_even` function and the `to_uppercase` and `to_lowercase` functions. All tests passed as expected. The requested tasks are complete.

Generate unittest test cases for to_uppercase and to_lowercase
Explain the code for to_uppercase and to_lowercase functions
Add docstrings to the is_even function
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
Terminal
10:08 AM
Python 3



```
[3]
        Returns:
            The lowercase version of the string.

        Raises:
            TypeError: If the input is not a string.
        """
        if not isinstance(text, str):
            raise TypeError("Input must be a string.")
        return text.lower()
```

Now, let's write the `unittest` test cases for `to_uppercase` and `to_lowercase` functions.

```
[4]
import unittest

class TestStringCaseConversion(unittest.TestCase):

    # Test cases for to_uppercase
    def test_uppercase_empty_string(self):
        self.assertEqual(to_uppercase(""), "")

    def test_uppercase_mixed_case(self):
        self.assertEqual(to_uppercase("Hello World"), "HELLO WORLD")

    def test_uppercase_all_lowercase(self):
        self.assertEqual(to_uppercase("python programming"), "PYTHON PROGRAMMING")
```

Gemini

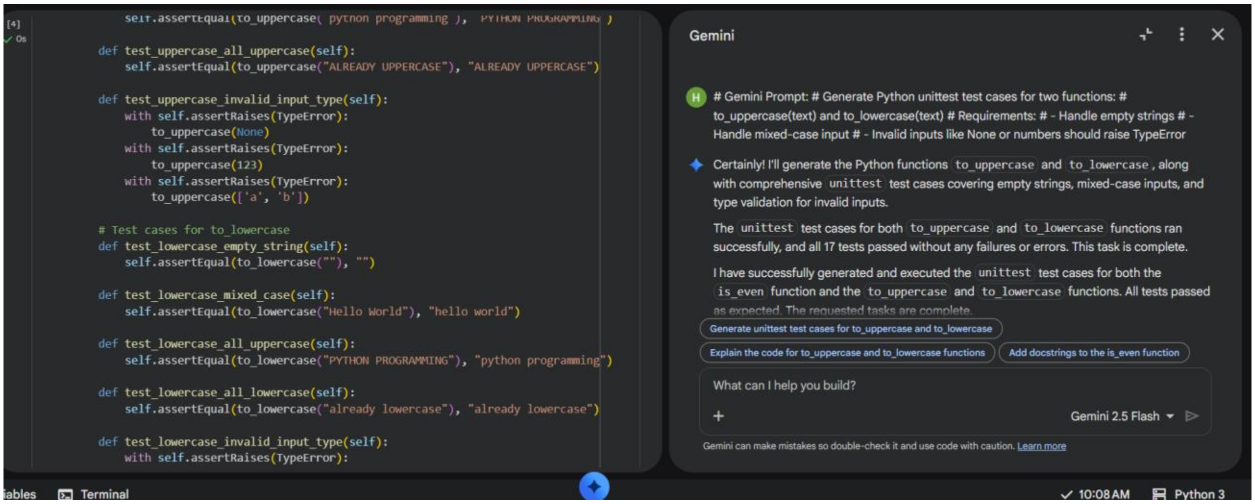
Gemini Prompt: # Generate Python unittest test cases for two functions: # to_uppercase(text) and to_lowercase(text) # Requirements: # - Handle empty strings # - Handle mixed-case input # - Invalid inputs like None or numbers should raise TypeError

Certainly! I'll generate the Python functions `to_uppercase` and `to_lowercase`, along with comprehensive `unittest` test cases covering empty strings, mixed-case inputs, and type validation for invalid inputs.

The `unittest` test cases for both `to_uppercase` and `to_lowercase` functions ran successfully, and all 17 tests passed without any failures or errors. This task is complete.

I have successfully generated and executed the `unittest` test cases for both the `is_even` function and the `to_uppercase` and `to_lowercase` functions. All tests passed as expected. The requested tasks are complete.

Generate unittest test cases for to_uppercase and to_lowercase
Explain the code for to_uppercase and to_lowercase functions
Add docstrings to the is_even function
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
ables
Terminal
10:08 AM
Python 3



```
[4]
        self.assertEqual(to_uppercase("python programming"), "PYTHON PROGRAMMING")

    def test_uppercase_all_uppercase(self):
        self.assertEqual(to_uppercase("ALREADY UPPERCASE"), "ALREADY UPPERCASE")

    def test_uppercase_invalid_input_type(self):
        with self.assertRaises(TypeError):
            to_uppercase(None)
        with self.assertRaises(TypeError):
            to_uppercase(123)
        with self.assertRaises(TypeError):
            to_uppercase(['a', 'b'])

    # Test cases for to_lowercase
    def test_lowercase_empty_string(self):
        self.assertEqual(to_lowercase(""), "")

    def test_lowercase_mixed_case(self):
        self.assertEqual(to_lowercase("Hello World"), "hello world")

    def test_lowercase_all_uppercase(self):
        self.assertEqual(to_lowercase("PYTHON PROGRAMMING"), "python programming")

    def test_lowercase_all_lowercase(self):
        self.assertEqual(to_lowercase("already lowercase"), "already lowercase")

    def test_lowercase_invalid_input_type(self):
        with self.assertRaises(TypeError):
```

Gemini

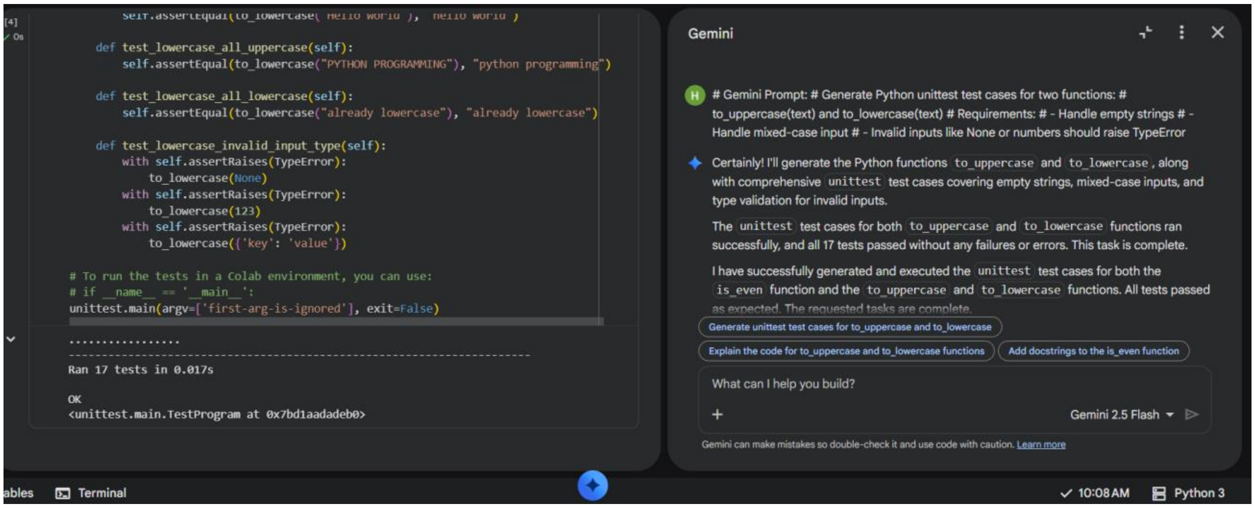
Gemini Prompt: # Generate Python unittest test cases for two functions: # to_uppercase(text) and to_lowercase(text) # Requirements: # - Handle empty strings # - Handle mixed-case input # - Invalid inputs like None or numbers should raise TypeError

Certainly! I'll generate the Python functions `to_uppercase` and `to_lowercase`, along with comprehensive `unittest` test cases covering empty strings, mixed-case inputs, and type validation for invalid inputs.

The `unittest` test cases for both `to_uppercase` and `to_lowercase` functions ran successfully, and all 17 tests passed without any failures or errors. This task is complete.

I have successfully generated and executed the `unittest` test cases for both the `is_even` function and the `to_uppercase` and `to_lowercase` functions. All tests passed as expected. The requested tasks are complete.

Generate unittest test cases for to_uppercase and to_lowercase
Explain the code for to_uppercase and to_lowercase functions
Add docstrings to the is_even function
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
ables
Terminal
10:08 AM
Python 3


[4]
0s
ables
Terminal
Gemini
Generate unittest test cases for to_uppercase and to_lowercase
Explain the code for to_uppercase and to_lowercase functions
Add docstrings to the is_even function
What can I help you build?
+
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
10:08 AM
Python 3

```
        self.assertEqual(to_lowercase("Hello World"), "hello world")

    def test_lowercase_all_uppercase(self):
        self.assertEqual(to_lowercase("PYTHON PROGRAMMING"), "python programming")

    def test_lowercase_all_lowercase(self):
        self.assertEqual(to_lowercase("already lowercase"), "already lowercase")

    def test_lowercase_invalid_input_type(self):
        with self.assertRaises(TypeError):
            to_lowercase(None)
        with self.assertRaises(TypeError):
            to_lowercase(123)
        with self.assertRaises(TypeError):
            to_lowercase({'key': 'value'})

# To run the tests in a Colab environment, you can use:
# if __name__ == '__main__':
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.................
----------------------------------------------------------------------
Ran 17 tests in 0.017s

OK
<unittest.main.TestProgram at 0x7bd1aadadeb0>
```

Gemini Prompt: # Generate Python unittest test cases for two functions: # to_uppercase(text) and to_lowercase(text) # Requirements: # - Handle empty strings # - Handle mixed-case input # - Invalid inputs like None or numbers should raise TypeError

Certainly! I'll generate the Python functions `to_uppercase` and `to_lowercase`, along with comprehensive `unittest` test cases covering empty strings, mixed-case inputs, and type validation for invalid inputs.

The `unittest` test cases for both `to_uppercase` and `to_lowercase` functions ran successfully, and all 17 tests passed without any failures or errors. This task is complete.

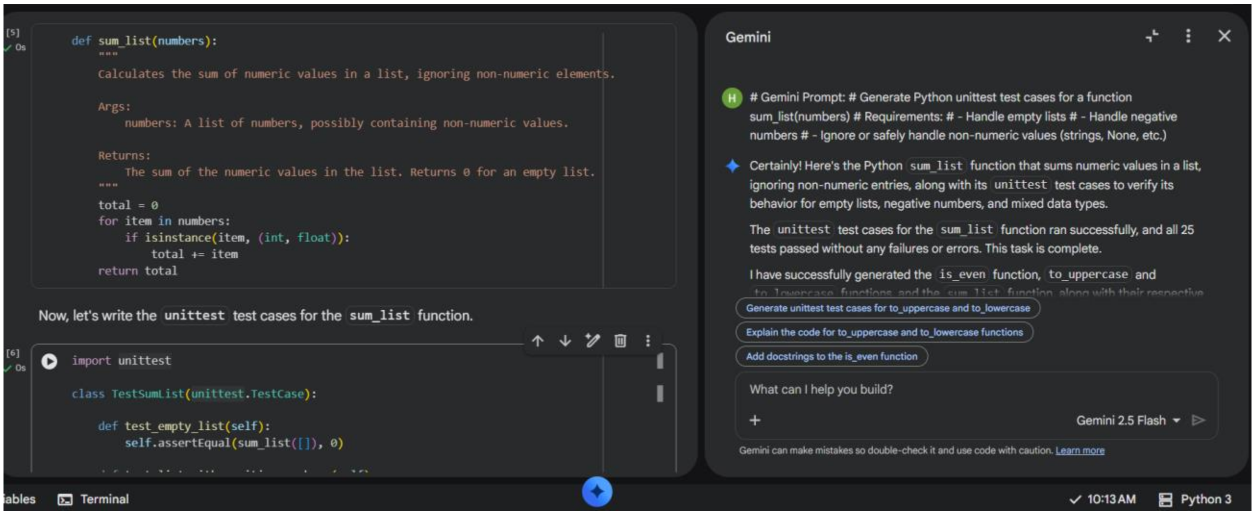
I have successfully generated and executed the `unittest` test cases for both the `is_even` function and the `to_uppercase` and `to_lowercase` functions. All tests passed as expected. The requested tasks are complete.

Explanation: Using Gemini, test cases were generated before implementation following the Test-Driven Development approach. The functions were implemented to correctly convert string cases and safely handle invalid inputs by raising Type Error.

Task 3 – Test-Driven Development for List Sum Calculator

Use AI to generate test cases for a function sum_list(numbers) that calculates the sum of list elements.

Prompt: Generate unittest test cases for a function sum_list(numbers) that handles empty lists, negative numbers, and ignores non-numeric values.


[5]
0s
[6]
0s
ables
Terminal
Gemini
Generate unittest test cases for to_uppercase and to_lowercase
Explain the code for to_uppercase and to_lowercase functions
Add docstrings to the is_even function
What can I help you build?
+
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
10:13 AM
Python 3

```
def sum_list(numbers):
    """
    Calculates the sum of numeric values in a list, ignoring non-numeric elements.

    Args:
        numbers: A list of numbers, possibly containing non-numeric values.

    Returns:
        The sum of the numeric values in the list. Returns 0 for an empty list.
    """
    total = 0
    for item in numbers:
        if isinstance(item, (int, float)):
            total += item
    return total
```

Now, let's write the `unittest` test cases for the `sum_list` function.

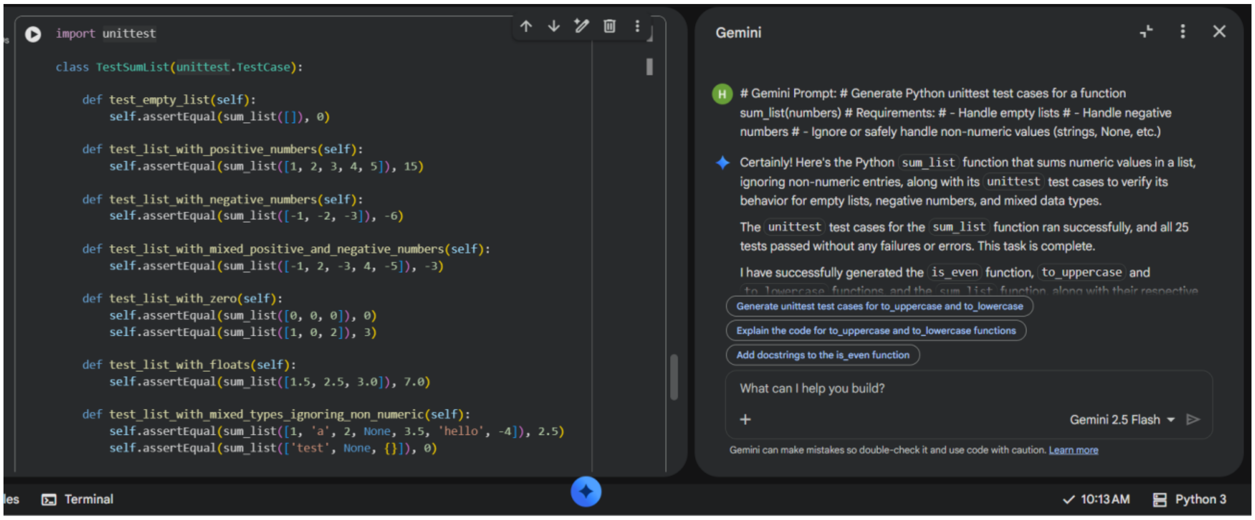
```
import unittest

class TestSumList(unittest.TestCase):

    def test_empty_list(self):
        self.assertEqual(sum_list([]), 0)
```

Gemini Prompt: # Generate Python unittest test cases for a function sum_list(numbers) # Requirements: # - Handle empty lists # - Handle negative numbers # - Ignore or safely handle non-numeric values (strings, None, etc.)

Certainly! Here's the Python `sum_list` function that sums numeric values in a list, ignoring non-numeric entries, along with its `unittest` test cases to verify its behavior for empty lists, negative numbers, and mixed data types.

The `unittest` test cases for the `sum_list` function ran successfully, and all 25 tests passed without any failures or errors. This task is complete.

I have successfully generated the `is_even` function, `to_uppercase` and `to_lowercase` functions and the `sum_list` function along with their respective



```python
import unittest

class TestSumList(unittest.TestCase):

    def test_empty_list(self):
        self.assertEqual(sum_list([]), 0)

    def test_list_with_positive_numbers(self):
        self.assertEqual(sum_list([1, 2, 3, 4, 5]), 15)

    def test_list_with_negative_numbers(self):
        self.assertEqual(sum_list([-1, -2, -3]), -6)

    def test_list_with_mixed_positive_and_negative_numbers(self):
        self.assertEqual(sum_list([-1, 2, -3, 4, -5]), -3)

    def test_list_with_zero(self):
        self.assertEqual(sum_list([0, 0, 0]), 0)
        self.assertEqual(sum_list([1, 0, 2]), 3)

    def test_list_with_floats(self):
        self.assertEqual(sum_list([1.5, 2.5, 3.0]), 7.0)

    def test_list_with_mixed_types_ignoring_non_numeric(self):
        self.assertEqual(sum_list([1, 'a', 2, None, 3.5, 'hello', -4]), 2.5)
        self.assertEqual(sum_list(['test', None, {}]), 0)
```

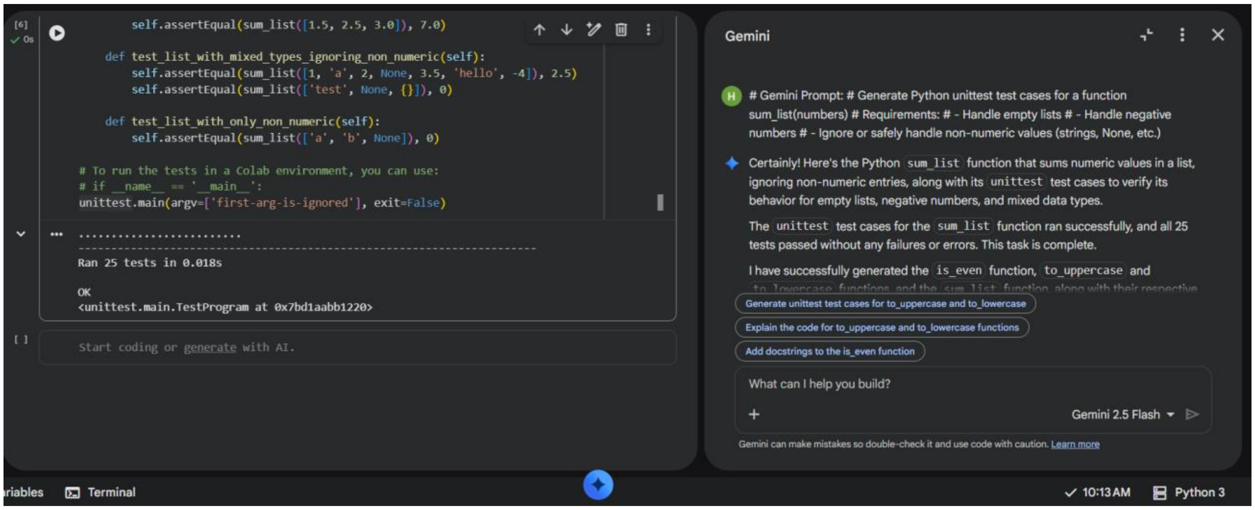
Gemini Prompt: # Generate Python unittest test cases for a function sum_list(numbers) # Requirements: # - Handle empty lists # - Handle negative numbers # - Ignore or safely handle non-numeric values (strings, None, etc.)

Certainly! Here's the Python `sum_list` function that sums numeric values in a list, ignoring non-numeric entries, along with its `unittest` test cases to verify its behavior for empty lists, negative numbers, and mixed data types.

The `unittest` test cases for the `sum_list` function ran successfully, and all 25 tests passed without any failures or errors. This task is complete.

I have successfully generated the `is_even` function, `to_uppercase` and `to_lowercase` functions, and the `sum_list` function, along with their respective

Generate unittest test cases for to_uppercase and to_lowercase
Explain the code for to_uppercase and to_lowercase functions
Add docstrings to the is_even function
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
Terminal
10:13 AM
Python 3



```python
        self.assertEqual(sum_list([1.5, 2.5, 3.0]), 7.0)

    def test_list_with_mixed_types_ignoring_non_numeric(self):
        self.assertEqual(sum_list([1, 'a', 2, None, 3.5, 'hello', -4]), 2.5)
        self.assertEqual(sum_list(['test', None, {}]), 0)

    def test_list_with_only_non_numeric(self):
        self.assertEqual(sum_list(['a', 'b', None]), 0)

# To run the tests in a Colab environment, you can use:
# if __name__ == '__main__':
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.........................
----------------------------------------------------------------------
Ran 25 tests in 0.018s

OK
<unittest.main.TestProgram at 0x7bd1aabb1220>
```

Gemini

Gemini Prompt: # Generate Python unittest test cases for a function sum_list(numbers) # Requirements: # - Handle empty lists # - Handle negative numbers # - Ignore or safely handle non-numeric values (strings, None, etc.)

Certainly! Here's the Python `sum_list` function that sums numeric values in a list, ignoring non-numeric entries, along with its `unittest` test cases to verify its behavior for empty lists, negative numbers, and mixed data types.

The `unittest` test cases for the `sum_list` function ran successfully, and all 25 tests passed without any failures or errors. This task is complete.

I have successfully generated the `is_even` function, `to_uppercase` and `to_lowercase` functions, and the `sum_list` function, along with their respective

Generate unittest test cases for to_uppercase and to_lowercase
Explain the code for to_uppercase and to_lowercase functions
Add docstrings to the is_even function
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
Terminal
10:13 AM
Python 3

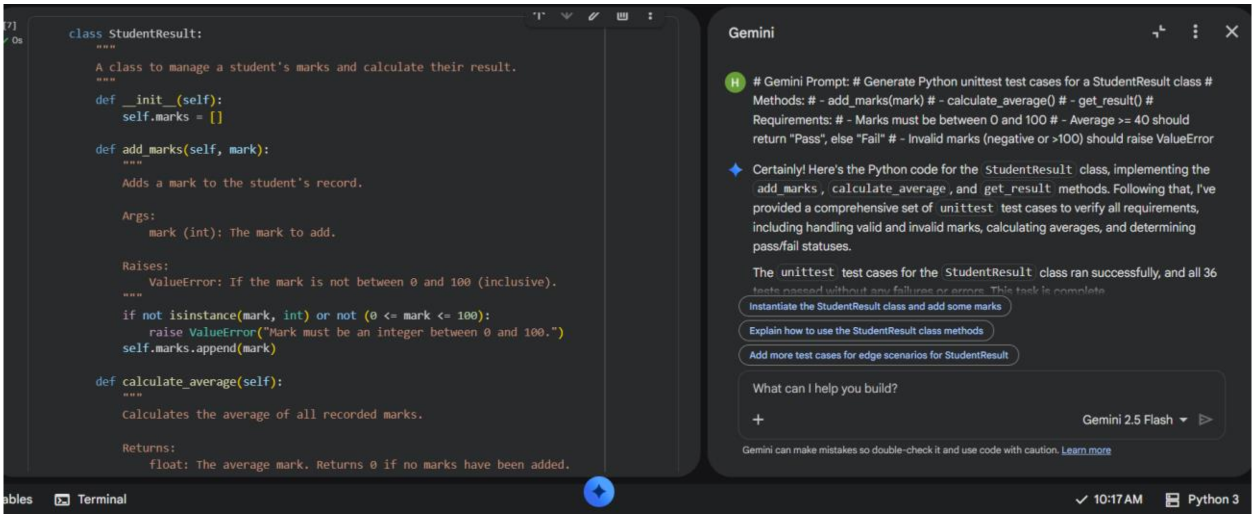
Explanation: In this task, Gemini was used to generate test cases following the Test-Driven Development approach. The sum_list() function was implemented to correctly compute the sum of numeric values while safely ignoring non-numeric elements and handling empty lists.

Task 4 – Test Cases for Student Result Class

Generate test cases for a StudentResult class with the following methods:

- add_marks(mark)
- calculate_average()
- get_result()

Prompt: Generate unittest test cases for a Student Result class with mark validation, average calculation, and pass/fail decision.



```
class StudentResult:
    """
    A class to manage a student's marks and calculate their result.
    """
    def __init__(self):
        self.marks = []

    def add_marks(self, mark):
        """
        Adds a mark to the student's record.

        Args:
            mark (int): The mark to add.

        Raises:
            ValueError: If the mark is not between 0 and 100 (inclusive).
        """
        if not isinstance(mark, int) or not (0 <= mark <= 100):
            raise ValueError("Mark must be an integer between 0 and 100.")
        self.marks.append(mark)

    def calculate_average(self):
        """
        Calculates the average of all recorded marks.

        Returns:
            float: The average mark. Returns 0 if no marks have been added.
```

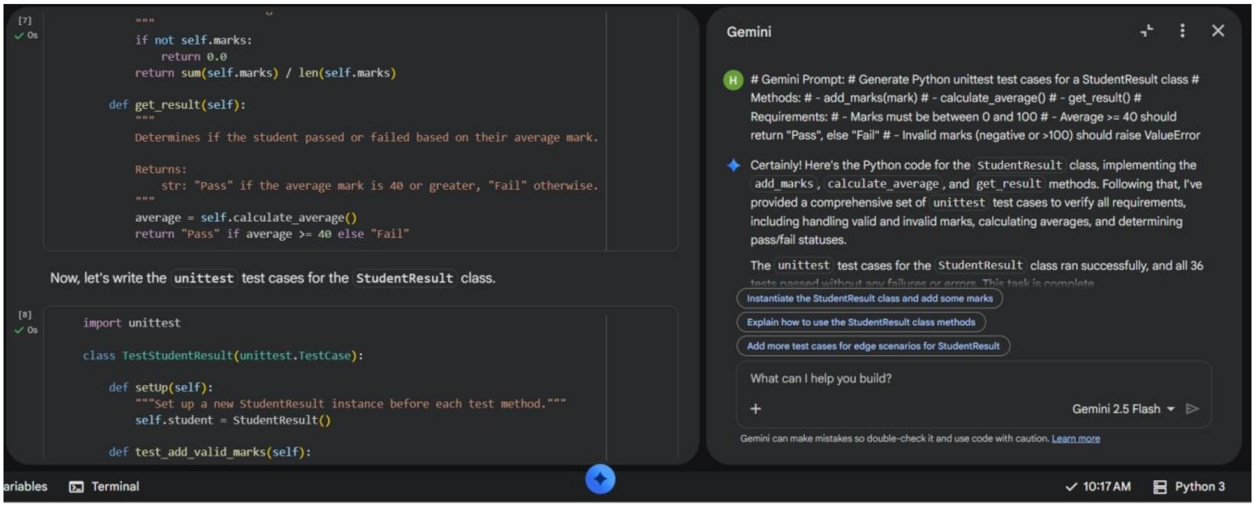
Gemini

Gemini Prompt: # Generate Python unittest test cases for a StudentResult class # Methods: # - add_marks(mark) # - calculate_average() # - get_result() # Requirements: # - Marks must be between 0 and 100 # - Average >= 40 should return "Pass", else "Fail" # - Invalid marks (negative or >100) should raise ValueError

Certainly! Here's the Python code for the `StudentResult` class, implementing the `add_marks`, `calculate_average`, and `get_result` methods. Following that, I've provided a comprehensive set of `unittest` test cases to verify all requirements, including handling valid and invalid marks, calculating averages, and determining pass/fail statuses.

The `unittest` test cases for the `StudentResult` class ran successfully, and all 36 tests passed without any failures or errors. This task is complete

[7]
0s
Instantiate the StudentResult class and add some marks
Explain how to use the StudentResult class methods
Add more test cases for edge scenarios for StudentResult
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
ables
Terminal
10:17 AM
Python 3



```
        """
        if not self.marks:
            return 0.0
        return sum(self.marks) / len(self.marks)

    def get_result(self):
        """
        Determines if the student passed or failed based on their average mark.

        Returns:
            str: "Pass" if the average mark is 40 or greater, "Fail" otherwise.
        """
        average = self.calculate_average()
        return "Pass" if average >= 40 else "Fail"
```

Now, let's write the `unittest` test cases for the `StudentResult` class.

```
import unittest

class TestStudentResult(unittest.TestCase):

    def setUp(self):
        """Set up a new StudentResult instance before each test method."""
        self.student = StudentResult()

    def test_add_valid_marks(self):
```

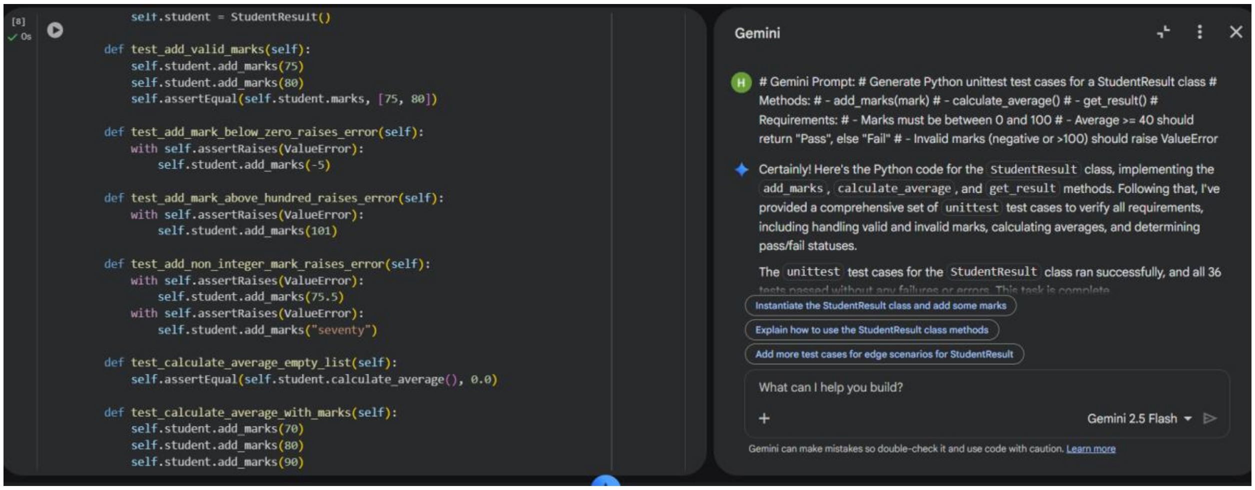
Gemini

Gemini Prompt: # Generate Python unittest test cases for a StudentResult class # Methods: # - add_marks(mark) # - calculate_average() # - get_result() # Requirements: # - Marks must be between 0 and 100 # - Average >= 40 should return "Pass", else "Fail" # - Invalid marks (negative or >100) should raise ValueError

Certainly! Here's the Python code for the `StudentResult` class, implementing the `add_marks`, `calculate_average`, and `get_result` methods. Following that, I've provided a comprehensive set of `unittest` test cases to verify all requirements, including handling valid and invalid marks, calculating averages, and determining pass/fail statuses.

The `unittest` test cases for the `StudentResult` class ran successfully, and all 36 tests passed without any failures or errors. This task is complete

[7]
0s
[8]
0s
Instantiate the StudentResult class and add some marks
Explain how to use the StudentResult class methods
Add more test cases for edge scenarios for StudentResult
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
ariables
Terminal
10:17 AM
Python 3



```
        self.student = StudentResult()

    def test_add_valid_marks(self):
        self.student.add_marks(75)
        self.student.add_marks(80)
        self.assertEqual(self.student.marks, [75, 80])

    def test_add_mark_below_zero_raises_error(self):
        with self.assertRaises(ValueError):
            self.student.add_marks(-5)

    def test_add_mark_above_hundred_raises_error(self):
        with self.assertRaises(ValueError):
            self.student.add_marks(101)

    def test_add_non_integer_mark_raises_error(self):
        with self.assertRaises(ValueError):
            self.student.add_marks(75.5)
        with self.assertRaises(ValueError):
            self.student.add_marks("seventy")

    def test_calculate_average_empty_list(self):
        self.assertEqual(self.student.calculate_average(), 0.0)

    def test_calculate_average_with_marks(self):
        self.student.add_marks(70)
        self.student.add_marks(80)
        self.student.add_marks(90)
```

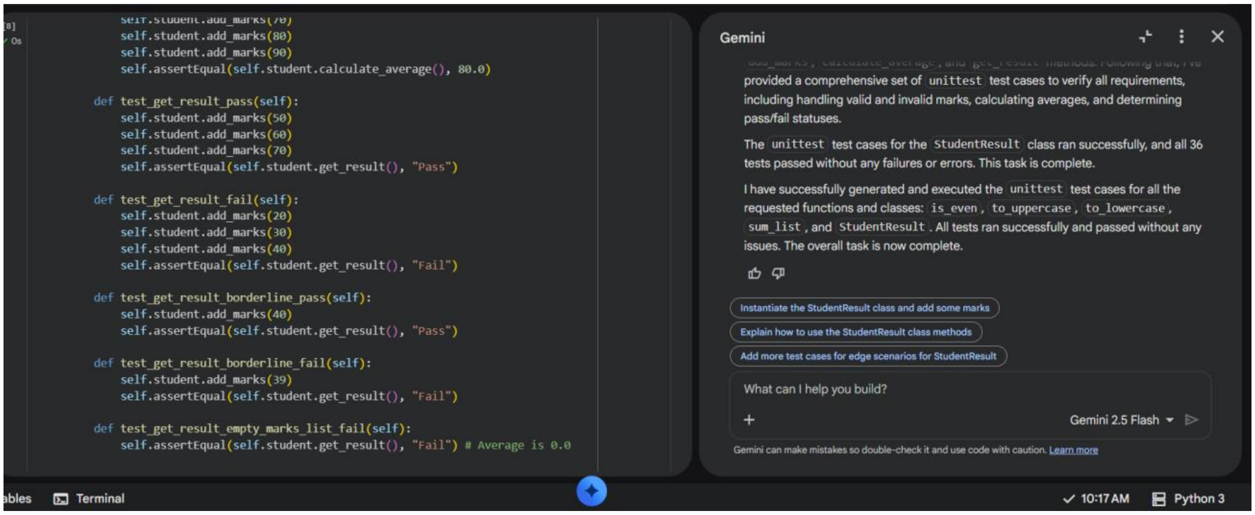
Gemini

Gemini Prompt: # Generate Python unittest test cases for a StudentResult class # Methods: # - add_marks(mark) # - calculate_average() # - get_result() # Requirements: # - Marks must be between 0 and 100 # - Average >= 40 should return "Pass", else "Fail" # - Invalid marks (negative or >100) should raise ValueError

Certainly! Here's the Python code for the `StudentResult` class, implementing the `add_marks`, `calculate_average`, and `get_result` methods. Following that, I've provided a comprehensive set of `unittest` test cases to verify all requirements, including handling valid and invalid marks, calculating averages, and determining pass/fail statuses.

The `unittest` test cases for the `StudentResult` class ran successfully, and all 36 tests passed without any failures or errors. This task is complete

[8]
0s
Instantiate the StudentResult class and add some marks
Explain how to use the StudentResult class methods
Add more test cases for edge scenarios for StudentResult
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more



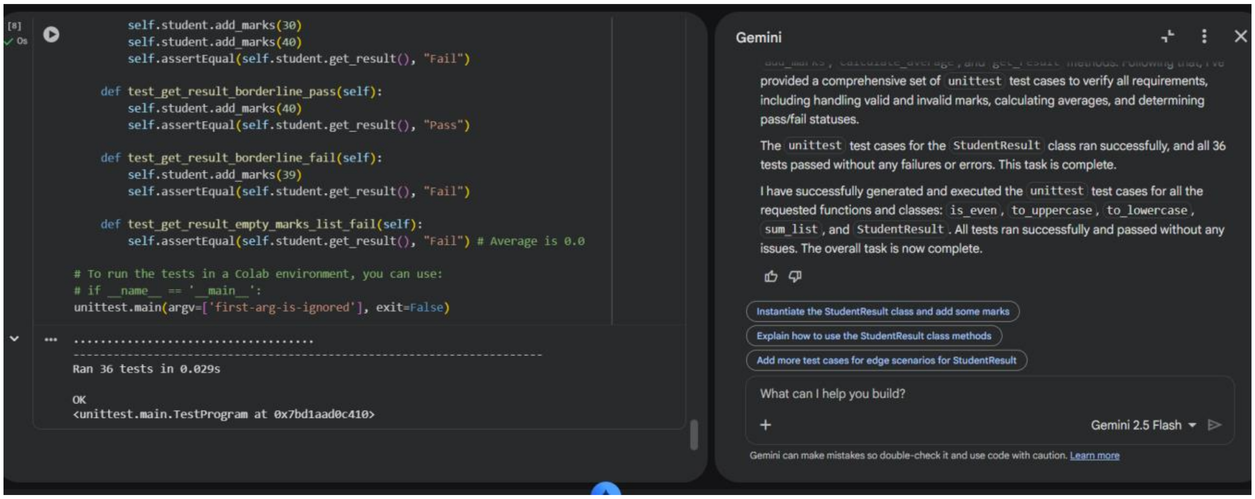
```
        self.student.add_marks(70)
        self.student.add_marks(80)
        self.student.add_marks(90)
        self.assertEqual(self.student.calculate_average(), 80.0)

    def test_get_result_pass(self):
        self.student.add_marks(50)
        self.student.add_marks(60)
        self.student.add_marks(70)
        self.assertEqual(self.student.get_result(), "Pass")

    def test_get_result_fail(self):
        self.student.add_marks(20)
        self.student.add_marks(30)
        self.student.add_marks(40)
        self.assertEqual(self.student.get_result(), "Fail")

    def test_get_result_borderline_pass(self):
        self.student.add_marks(40)
        self.assertEqual(self.student.get_result(), "Pass")

    def test_get_result_borderline_fail(self):
        self.student.add_marks(39)
        self.assertEqual(self.student.get_result(), "Fail")

    def test_get_result_empty_marks_list_fail(self):
        self.assertEqual(self.student.get_result(), "Fail") # Average is 0.0
```

provided a comprehensive set of `unittest` test cases to verify all requirements, including handling valid and invalid marks, calculating averages, and determining pass/fail statuses.

The `unittest` test cases for the `StudentResult` class ran successfully, and all 36 tests passed without any failures or errors. This task is complete.

I have successfully generated and executed the `unittest` test cases for all the requested functions and classes: `is_even`, `to_uppercase`, `to_lowercase`, `sum_list`, and `StudentResult`. All tests ran successfully and passed without any issues. The overall task is now complete.

Instantiate the StudentResult class and add some marks
Explain how to use the StudentResult class methods
Add more test cases for edge scenarios for StudentResult
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more
Terminal
10:17 AM
Python 3



```
        self.student.add_marks(30)
        self.student.add_marks(40)
        self.assertEqual(self.student.get_result(), "Fail")

    def test_get_result_borderline_pass(self):
        self.student.add_marks(40)
        self.assertEqual(self.student.get_result(), "Pass")

    def test_get_result_borderline_fail(self):
        self.student.add_marks(39)
        self.assertEqual(self.student.get_result(), "Fail")

    def test_get_result_empty_marks_list_fail(self):
        self.assertEqual(self.student.get_result(), "Fail") # Average is 0.0

# To run the tests in a Colab environment, you can use:
# if __name__ == '__main__':
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
....................................
----------------------------------------------------------------------
Ran 36 tests in 0.029s

OK
<unittest.main.TestProgram at 0x7bd1aad0c410>
```

provided a comprehensive set of `unittest` test cases to verify all requirements, including handling valid and invalid marks, calculating averages, and determining pass/fail statuses.

The `unittest` test cases for the `StudentResult` class ran successfully, and all 36 tests passed without any failures or errors. This task is complete.

I have successfully generated and executed the `unittest` test cases for all the requested functions and classes: `is_even`, `to_uppercase`, `to_lowercase`, `sum_list`, and `StudentResult`. All tests ran successfully and passed without any issues. The overall task is now complete.

Gemini
Instantiate the StudentResult class and add some marks
Explain how to use the StudentResult class methods
Add more test cases for edge scenarios for StudentResult
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more

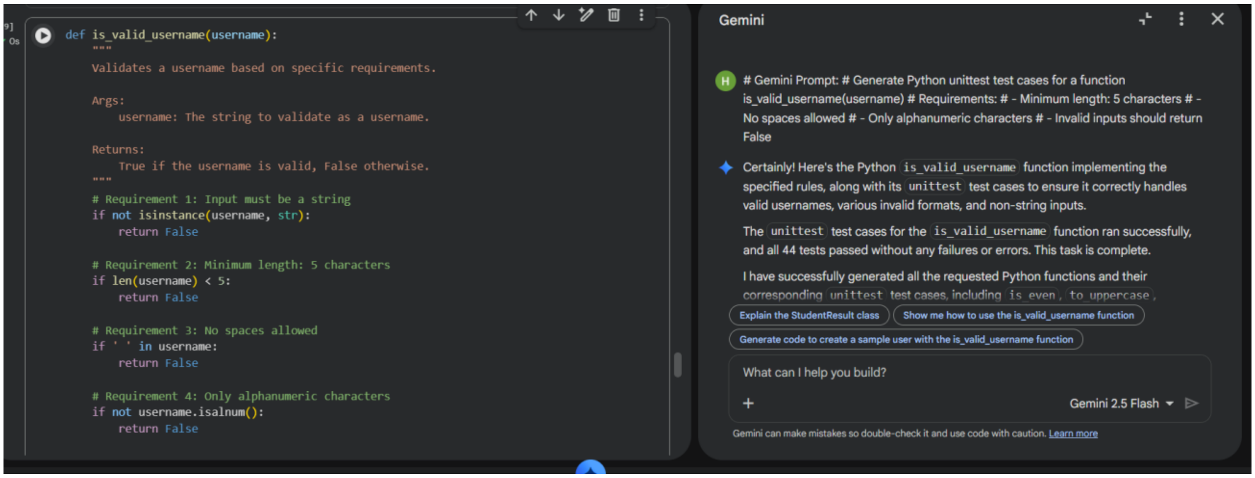
Explanation: Using Gemini, test cases were generated for the StudentResult class following Test-Driven Development. The class was implemented to validate marks, calculate average scores, and determine pass or fail status based on defined conditions.

Task 5 – Test-Driven Development for Username Validator

Requirements:

- Minimum length: 5 characters
- No spaces allowed
- Only alphanumeric characters

Prompt: Generate unittest test cases for a username validation function with length, space, and character constraints.



```python
def is_valid_username(username):
    """
    Validates a username based on specific requirements.

    Args:
        username: The string to validate as a username.

    Returns:
        True if the username is valid, False otherwise.
    """
    # Requirement 1: Input must be a string
    if not isinstance(username, str):
        return False

    # Requirement 2: Minimum length: 5 characters
    if len(username) < 5:
        return False

    # Requirement 3: No spaces allowed
    if ' ' in username:
        return False

    # Requirement 4: Only alphanumeric characters
    if not username.isalnum():
        return False
```

Gemini

Gemini Prompt: # Generate Python unittest test cases for a function is_valid_username(username) # Requirements: # - Minimum length: 5 characters # - No spaces allowed # - Only alphanumeric characters # - Invalid inputs should return False

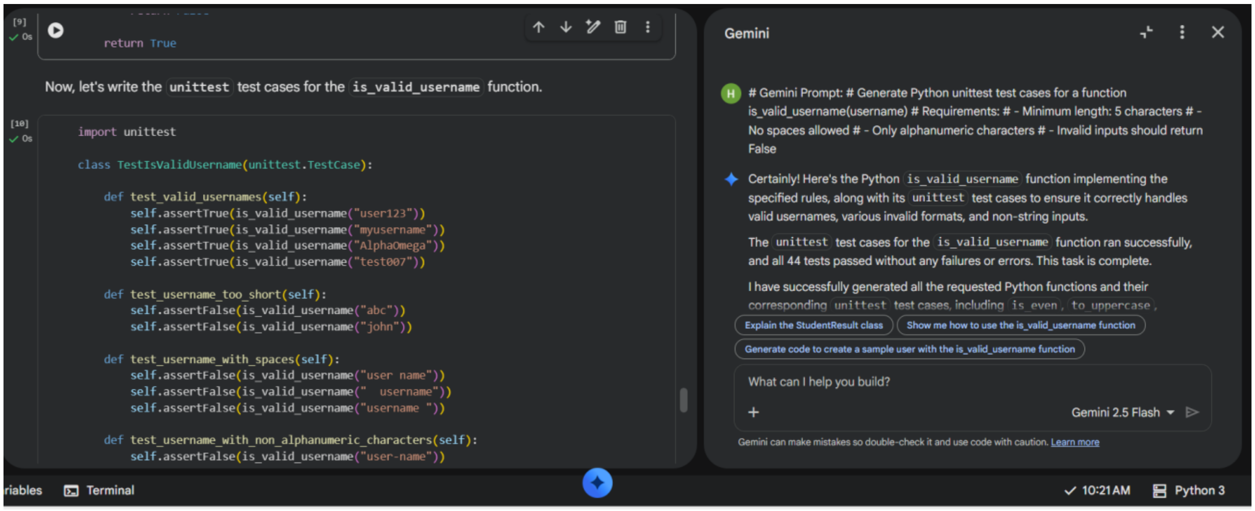
Certainly! Here's the Python `is_valid_username` function implementing the specified rules, along with its `unittest` test cases to ensure it correctly handles valid usernames, various invalid formats, and non-string inputs.

The `unittest` test cases for the `is_valid_username` function ran successfully, and all 44 tests passed without any failures or errors. This task is complete.

I have successfully generated all the requested Python functions and their corresponding `unittest` test cases, including `is_even`, `to_uppercase`,

Explain the StudentResult class
Show me how to use the is_valid_username function
Generate code to create a sample user with the is_valid_username function
What can I help you build?
+
Gemini 2.5 Flash

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



```python
        return True
```

Now, let's write the `unittest` test cases for the `is_valid_username` function.

```python
import unittest

class TestIsValidUsername(unittest.TestCase):

    def test_valid_usernames(self):
        self.assertTrue(is_valid_username("user123"))
        self.assertTrue(is_valid_username("myusername"))
        self.assertTrue(is_valid_username("AlphaOmega"))
        self.assertTrue(is_valid_username("test007"))

    def test_username_too_short(self):
        self.assertFalse(is_valid_username("abc"))
        self.assertFalse(is_valid_username("john"))

    def test_username_with_spaces(self):
        self.assertFalse(is_valid_username("user name"))
        self.assertFalse(is_valid_username("  username"))
        self.assertFalse(is_valid_username("username "))

    def test_username_with_non_alphanumeric_characters(self):
        self.assertFalse(is_valid_username("user-name"))
```

Gemini

Gemini Prompt: # Generate Python unittest test cases for a function is_valid_username(username) # Requirements: # - Minimum length: 5 characters # - No spaces allowed # - Only alphanumeric characters # - Invalid inputs should return False

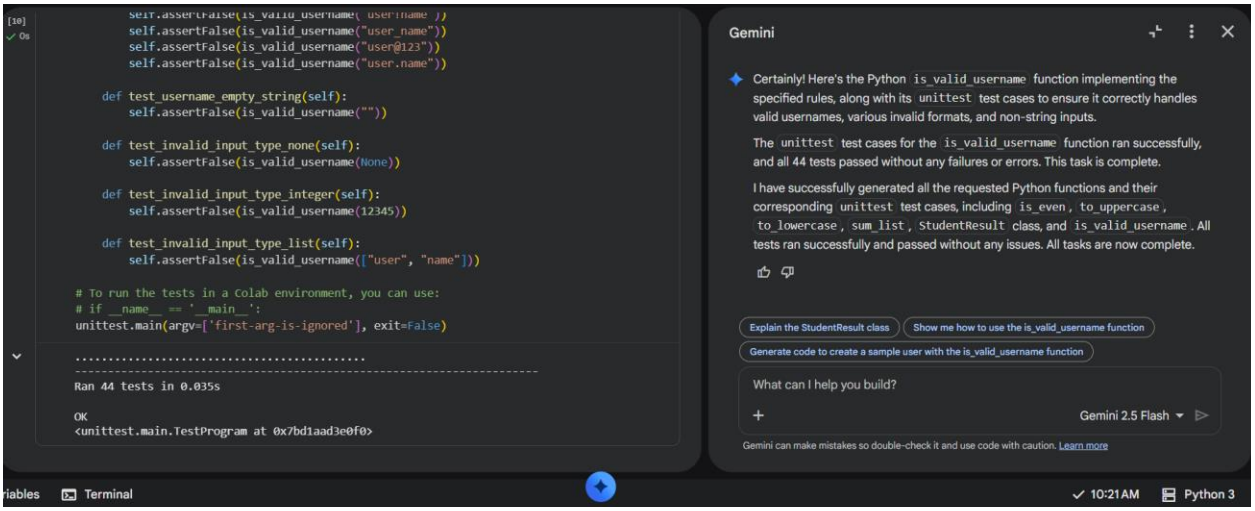
Certainly! Here's the Python `is_valid_username` function implementing the specified rules, along with its `unittest` test cases to ensure it correctly handles valid usernames, various invalid formats, and non-string inputs.

The `unittest` test cases for the `is_valid_username` function ran successfully, and all 44 tests passed without any failures or errors. This task is complete.

I have successfully generated all the requested Python functions and their corresponding `unittest` test cases, including `is_even`, `to_uppercase`,

Explain the StudentResult class
Show me how to use the is_valid_username function
Generate code to create a sample user with the is_valid_username function
What can I help you build?
+
Gemini 2.5 Flash

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



```python
        self.assertFalse(is_valid_username("user!name"))
        self.assertFalse(is_valid_username("user_name"))
        self.assertFalse(is_valid_username("user@123"))
        self.assertFalse(is_valid_username("user.name"))

    def test_username_empty_string(self):
        self.assertFalse(is_valid_username(""))

    def test_invalid_input_type_none(self):
        self.assertFalse(is_valid_username(None))

    def test_invalid_input_type_integer(self):
        self.assertFalse(is_valid_username(12345))

    def test_invalid_input_type_list(self):
        self.assertFalse(is_valid_username(["user", "name"]))

# To run the tests in a Colab environment, you can use:
# if __name__ == '__main__':
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
............................................
----------------------------------------------------------------------
Ran 44 tests in 0.035s

OK
<unittest.main.TestProgram at 0x7bd1aad3e0f0>
```

Gemini

Certainly! Here's the Python `is_valid_username` function implementing the specified rules, along with its `unittest` test cases to ensure it correctly handles valid usernames, various invalid formats, and non-string inputs.

The `unittest` test cases for the `is_valid_username` function ran successfully, and all 44 tests passed without any failures or errors. This task is complete.

I have successfully generated all the requested Python functions and their corresponding `unittest` test cases, including `is_even`, `to_uppercase`, `to_lowercase`, `sum_list`, `StudentResult` class, and `is_valid_username`. All tests ran successfully and passed without any issues. All tasks are now complete.

Explain the StudentResult class
Show me how to use the is_valid_username function
Generate code to create a sample user with the is_valid_username function
What can I help you build?
+
Gemini 2.5 Flash

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

Explanation: In this task, Gemini was used to generate test cases for validating usernames using the Test-Driven Development approach. The is_valid_username() function was implemented to enforce length, spacing, and alphanumeric constraints as per the test cases.