

Assignment 8.4 Ai Assisted Coding

Htno:2303A51100

Btno:05

Task 1: Developing a Utility Function Using TDD

Scenario

You are working on a small utility library for a larger software system. One of the required functions should calculate the square of a given number, and correctness is critical because other modules depend on it.

Task Description

Following the Test Driven Development (TDD) approach:

1. First, write unit test cases to verify that a function correctly returns the square of a number for multiple inputs.
2. After defining the test cases, use GitHub Copilot or Cursor AI to generate the function implementation so that all tests pass.

Ensure that the function is written only after the tests are created.

Expected Outcome

- A separate test file and implementation file
- Clearly written test cases executed before implementation
- AI-assisted function implementation that passes all tests •

Demonstration of the TDD cycle: test → fail → implement → pass

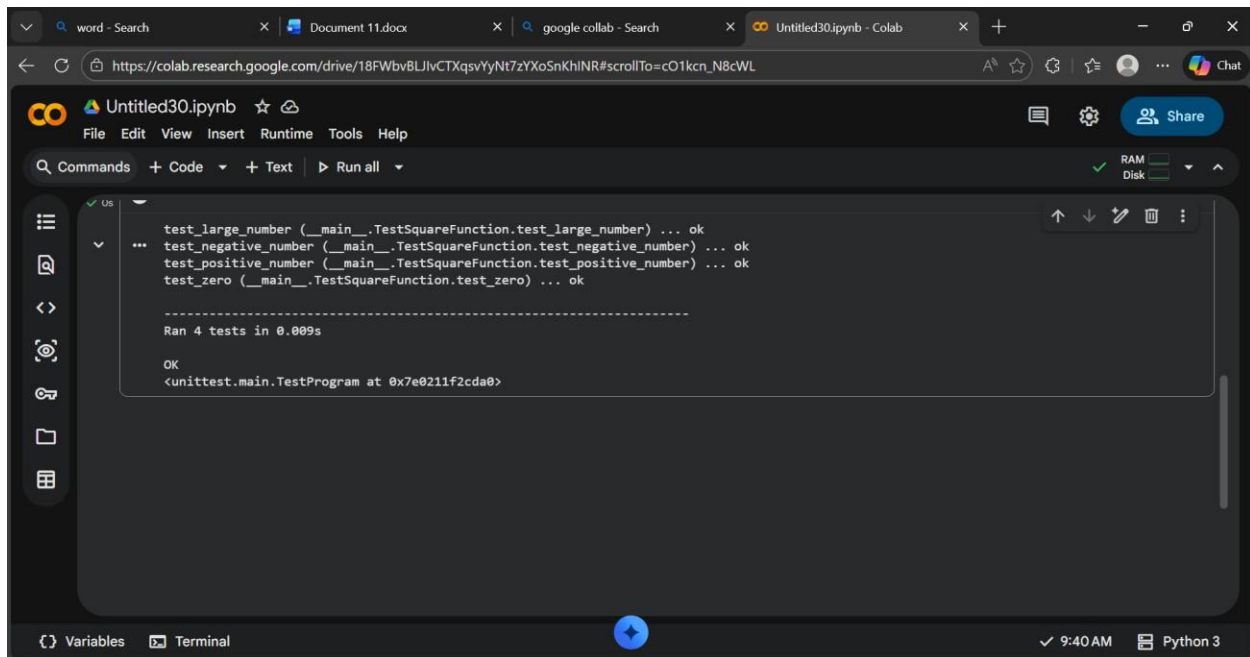
Code:

The image displays two sequential screenshots of a Google Colab notebook titled 'Untitled30.ipynb'. The browser tabs at the top include 'word - Search', 'Document 11.docx', 'google colab - Search', and 'Untitled30.ipynb - Colab'. The notebook interface shows a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all'. The left sidebar contains icons for file management and a 'Variables' panel. The main code area is divided into three cells:

- Cell [1]: Imports `unittest` and defines a `TestSquareFunction` class with four test methods: `test_positive_number`, `test_negative_number`, `test_zero`, and `test_large_number`. Each method uses `self.assertEqual` to verify the output of the `square` function.
- Cell [2]: Defines the `square` function, which returns `n * n`.
- Cell [3]: Executes `unittest.main(argv=[''], verbosity=2, exit=False)` to run the tests.

The status bar at the bottom indicates '9:40 AM' and 'Python 3'.

Output:



The screenshot shows a Google Colab notebook interface. The top bar includes a search bar and several open tabs: 'word - Search', 'Document 11.docx', 'google colab - Search', and 'Untitled30.ipynb - Colab'. The notebook's address bar shows a URL from 'https://colab.research.google.com'. The notebook title is 'Untitled30.ipynb'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all'. The main code area contains a Python test script using unittest. The test script defines a class 'TestSquareFunction' with methods 'test_large_number', 'test_negative_number', 'test_positive_number', and 'test_zero'. The test results show that all four tests passed successfully, with a total execution time of 0.009s. The bottom status bar indicates 'Variables', 'Terminal', '9:40 AM', and 'Python 3'.

```
test_large_number (__main__.TestSquareFunction.test_large_number) ... ok
test_negative_number (__main__.TestSquareFunction.test_negative_number) ... ok
test_positive_number (__main__.TestSquareFunction.test_positive_number) ... ok
test_zero (__main__.TestSquareFunction.test_zero) ... ok

Ran 4 tests in 0.009s

OK
<unittest.main.TestProgram at 0x7e0211f2cda0>
```

Task 2: Email Validation for a User Registration System

Scenario

You are developing the backend of a user registration system. One requirement is to validate user email addresses before storing them in the database.

Task Description

Apply Test Driven Development by:

1. Writing unit test cases that define valid and invalid email formats (e.g., missing @, missing domain, incorrect structure).
2. Using AI assistance to implement the `validate_email()` function based strictly on the behavior described by the test cases.

The implementation should be driven entirely by the test expectations.

Expected Outcome

- Well-defined unit tests using unittest or pytest
- An AI-generated email validation function
- All test cases passing successfully

- Clear alignment between test cases and function behavior Code:

The image shows two screenshots of a Google Colab notebook, illustrating the process of writing tests and then implementing a function to pass them.

Top Screenshot: The notebook is titled "Untitled30.ipynb". The code cell [4] contains the following Python code:

```
import unittest

# ----- TEST CASES (WRITTEN BEFORE FUNCTION) -----
class TestEmailValidation(unittest.TestCase):

    def test_valid_email(self):
        self.assertTrue(validate_email("user@example.com"))

    def test_missing_at_symbol(self):
        self.assertFalse(validate_email("userexample.com"))

    def test_missing_domain(self):
        self.assertFalse(validate_email("user@"))

    def test_missing_username(self):
        self.assertFalse(validate_email("@example.com"))

    def test_invalid_structure(self):
        self.assertFalse(validate_email("user@com"))

    def test_email_with_numbers(self):
        self.assertTrue(validate_email("user123@gmail.com"))
```

The bottom status bar shows "9:46 AM" and "Python 3".

Bottom Screenshot: The notebook is still titled "Untitled30.ipynb". The code cell [5] contains the following Python code:

```
import re

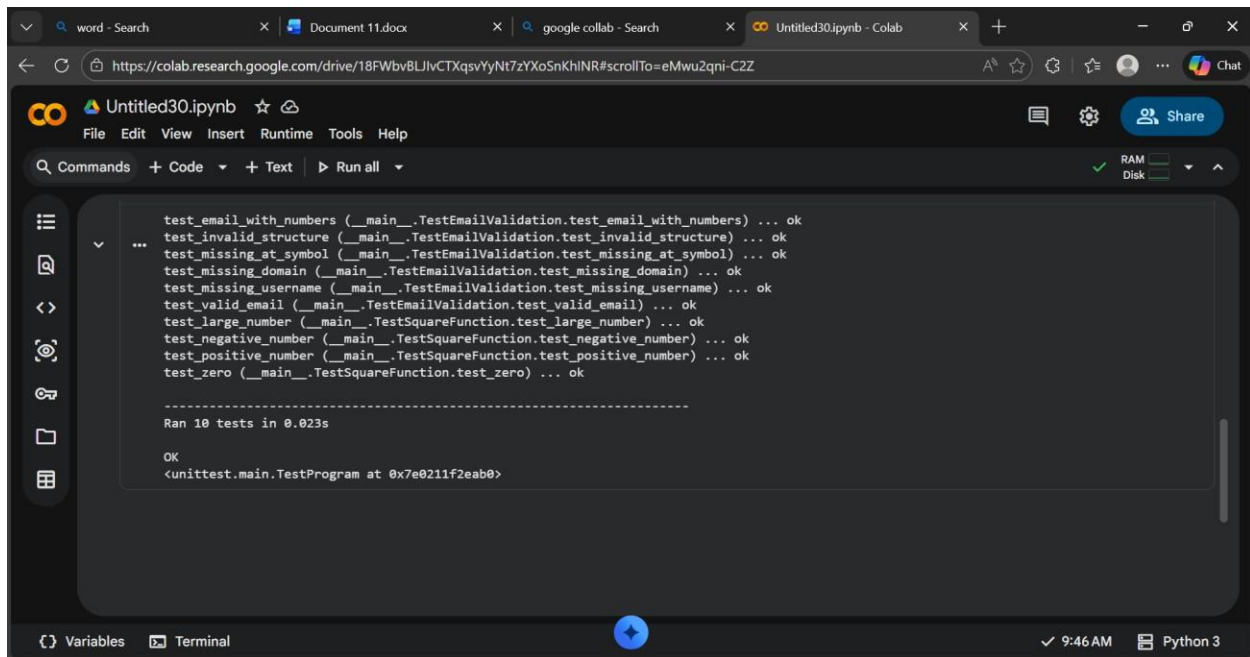
# ----- IMPLEMENTATION (AFTER TESTS) -----
def validate_email(email):
    pattern = r'^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
    return re.match(pattern, email) is not None
```

The code cell [6] contains the following Python code:

```
unittest.main(argv=[''], verbosity=2, exit=False)
```

The bottom status bar shows "9:46 AM" and "Python 3".

Output:



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, the file name 'Untitled30.ipynb', and a 'Share' button. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. A search bar and a 'Run all' button are also present. The main area displays the output of a test run, showing 10 tests passing. The tests are grouped into two categories: email validation and square function tests. The email validation tests include 'test_email_with_numbers', 'test_invalid_structure', 'test_missing_at_symbol', 'test_missing_domain', 'test_missing_username', and 'test_valid_email'. The square function tests include 'test_large_number', 'test_negative_number', 'test_positive_number', and 'test_zero'. The output shows that all tests passed successfully, with a total of 10 tests in 0.023s. The bottom status bar shows 'Variables', 'Terminal', and the time '9:46 AM' along with 'Python 3'.

```
test_email_with_numbers (__main__.TestEmailValidation.test_email_with_numbers) ... ok
test_invalid_structure (__main__.TestEmailValidation.test_invalid_structure) ... ok
test_missing_at_symbol (__main__.TestEmailValidation.test_missing_at_symbol) ... ok
test_missing_domain (__main__.TestEmailValidation.test_missing_domain) ... ok
test_missing_username (__main__.TestEmailValidation.test_missing_username) ... ok
test_valid_email (__main__.TestEmailValidation.test_valid_email) ... ok
test_large_number (__main__.TestSquareFunction.test_large_number) ... ok
test_negative_number (__main__.TestSquareFunction.test_negative_number) ... ok
test_positive_number (__main__.TestSquareFunction.test_positive_number) ... ok
test_zero (__main__.TestSquareFunction.test_zero) ... ok

-----
Ran 10 tests in 0.023s

OK
<unittest.main.TestProgram at 0x7e0211f2eab0>
```

Task 3: Decision Logic Development Using TDD

Scenario

In a grading or evaluation module, a function is required to determine the maximum value among three inputs. Accuracy is essential, as incorrect results could affect downstream decision logic.

Task Description

Using the TDD methodology:

1. Write test cases that describe the expected output for different combinations of three numbers.
2. Prompt GitHub Copilot or Cursor AI to implement the function logic based on the written tests.

Avoid writing any logic before test cases are completed.

Expected Outcome

- Comprehensive test cases covering normal and edge cases
- AI-generated function implementation
- Passing test results demonstrating correctness

- Evidence that logic was derived from tests, not assumptions Code:

The image displays two screenshots of a Google Colab notebook, illustrating the Test-Driven Development (TDD) process for implementing a function.

Top Screenshot: The notebook shows the initial test cases being written. The code includes the following:

```
[7] import unittest

# ----- TEST CASES FIRST (TDD) -----
class TestMaxOfThree(unittest.TestCase):

    def test_normal_numbers(self):
        self.assertEqual(max_of_three(2, 8, 5), 8)

    def test_first_is_largest(self):
        self.assertEqual(max_of_three(10, 3, 6), 10)

    def test_negative_numbers(self):
        self.assertEqual(max_of_three(-1, -5, -3), -1)

    def test_all_equal(self):
        self.assertEqual(max_of_three(4, 4, 4), 4)

    def test_two_equal_largest(self):
        self.assertEqual(max_of_three(7, 7, 2), 7)
```

Bottom Screenshot: The notebook shows the implementation of the function after the tests are written. The code includes the following:

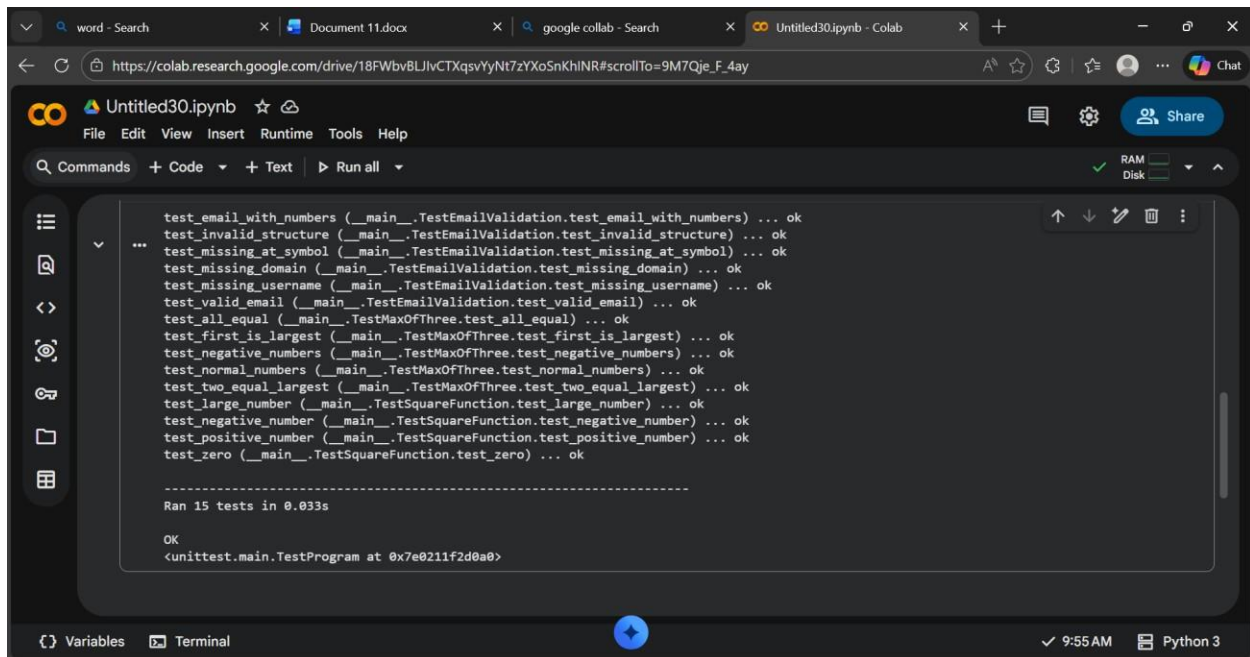
```
[8] #AI-Generated Implementation

# ----- IMPLEMENTATION (AFTER TESTS) -----
def max_of_three(a, b, c):
    return max(a, b, c)

[9] #Run Tests

unittest.main(argv=[''], verbosity=2, exit=False)
```

Output:



The screenshot shows a Google Colab notebook titled 'Untitled30.ipynb'. The code cell contains 15 unit tests using the unittest module. The tests are organized into two groups: email validation tests and mathematical function tests. All tests passed successfully, as indicated by the '... ok' status for each. The output shows 'Ran 15 tests in 0.033s' and 'OK'.

```
test_email_with_numbers (__main__.TestEmailValidation.test_email_with_numbers) ... ok
test_invalid_structure (__main__.TestEmailValidation.test_invalid_structure) ... ok
test_missing_at_symbol (__main__.TestEmailValidation.test_missing_at_symbol) ... ok
test_missing_domain (__main__.TestEmailValidation.test_missing_domain) ... ok
test_missing_username (__main__.TestEmailValidation.test_missing_username) ... ok
test_valid_email (__main__.TestEmailValidation.test_valid_email) ... ok
test_all_equal (__main__.TestMaxOfThree.test_all_equal) ... ok
test_first_is_largest (__main__.TestMaxOfThree.test_first_is_largest) ... ok
test_negative_numbers (__main__.TestMaxOfThree.test_negative_numbers) ... ok
test_normal_numbers (__main__.TestMaxOfThree.test_normal_numbers) ... ok
test_two_equal_largest (__main__.TestMaxOfThree.test_two_equal_largest) ... ok
test_large_number (__main__.TestSquareFunction.test_large_number) ... ok
test_negative_number (__main__.TestSquareFunction.test_negative_number) ... ok
test_positive_number (__main__.TestSquareFunction.test_positive_number) ... ok
test_zero (__main__.TestSquareFunction.test_zero) ... ok

-----
Ran 15 tests in 0.033s

OK
<unittest.main.TestProgram at 0x7e0211f2d0a0>
```

Task 4: Shopping Cart Development with AI-Assisted TDD

Scenario

You are building a simple shopping cart module for an e-commerce application.

The cart must support adding items, removing items, and calculating the total price accurately.

Task Description

Follow a test-driven approach:

1. Write unit tests for each required behavior:

- o Adding an item
- o Removing an item
- o Calculating the total price

2. After defining all tests, use AI tools to generate the ShoppingCart class and its methods so that the tests pass.

Focus on behavior-driven testing rather than implementation details.

Expected Outcome

- Unit tests defining expected shopping cart behavior

- AI-generated class implementation
- All tests passing successfully
- Clear demonstration of TDD applied to a class-based design Code:

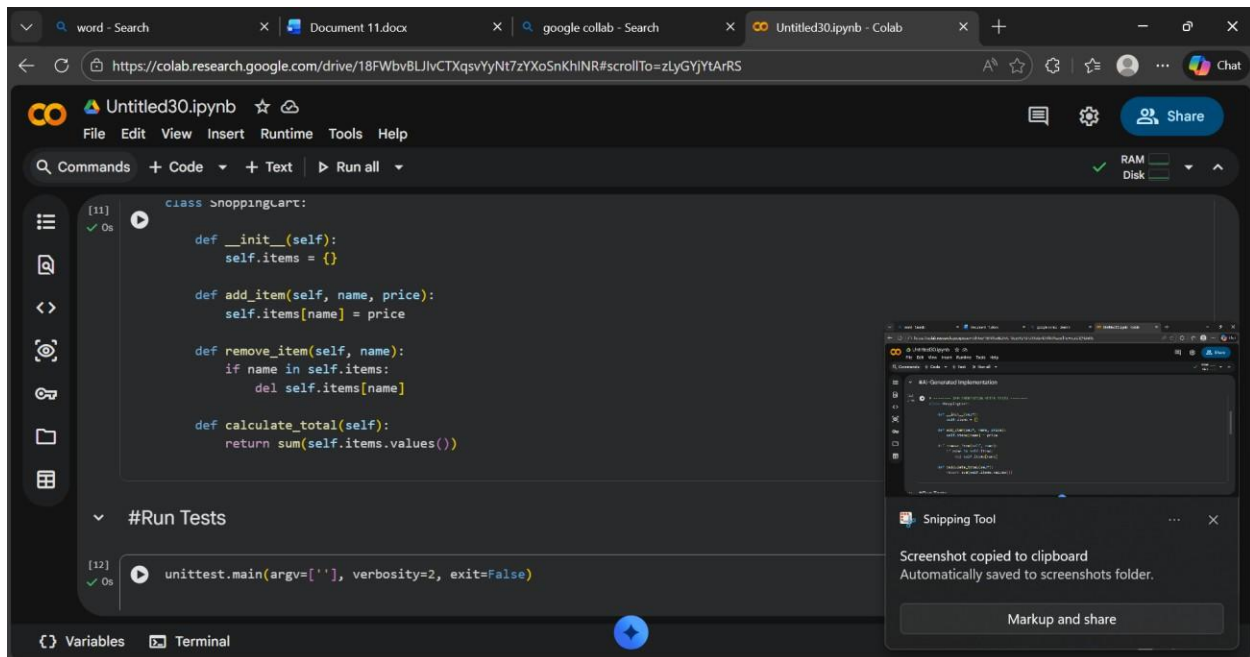
The image displays two sequential screenshots of a Google Colab notebook titled 'Untitled30.ipynb', illustrating the Test-Driven Development (TDD) process for a class-based design.

Top Screenshot (Test Cases):

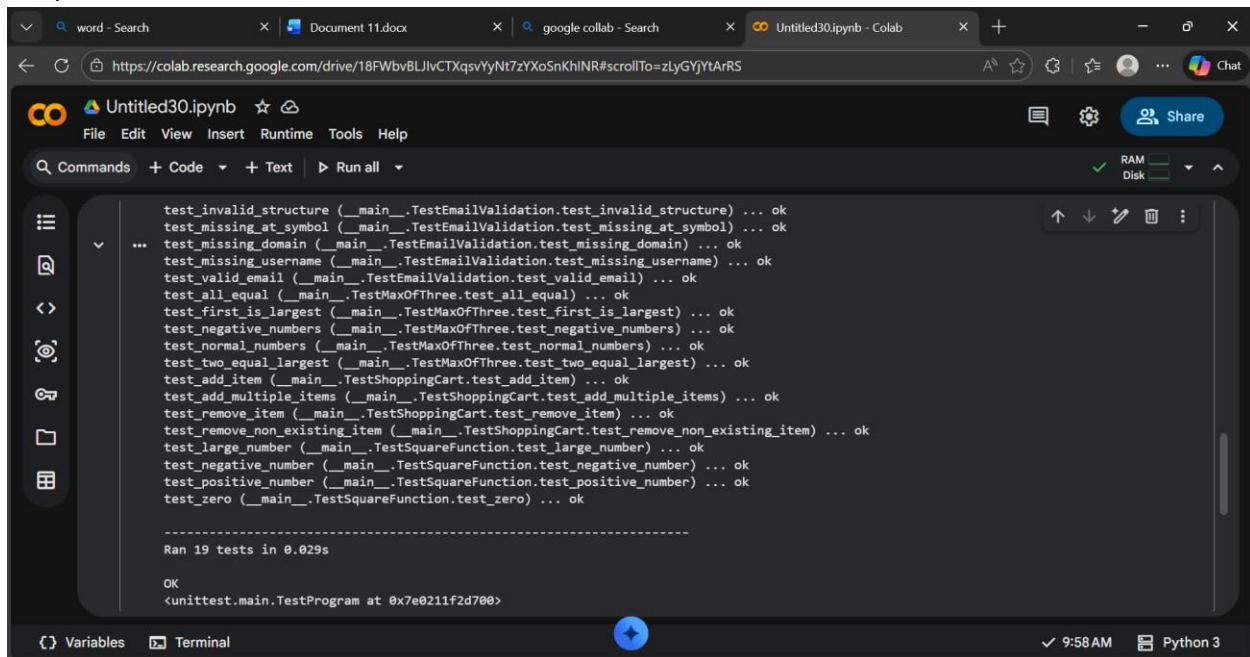
- The notebook is at cell [10].
- The code defines a `TestShoppingCart` class inheriting from `unittest.TestCase`.
- Three test methods are implemented:
 - `test_add_item(self)`: Adds a 'Book' for 100 and asserts the total is 100.
 - `test_add_multiple_items(self)`: Adds a 'Book' for 100 and a 'Pen' for 20, asserting the total is 120.
 - `test_remove_item(self)`: Adds a 'Book' for 100 and then removes it, asserting the total is 0.
- The status bar shows '9:58 AM' and 'Python 3'.

Bottom Screenshot (Implementation):

- The notebook is at cell [11].
- The code is titled '#AI-Generated Implementation'.
- The implementation defines the `ShoppingCart` class with the following methods:
 - `__init__(self)`: Initializes `self.items` as an empty dictionary.
 - `add_item(self, name, price)`: Adds an item to the dictionary.
 - `remove_item(self, name)`: Removes an item from the dictionary if it exists.
 - `calculate_total(self)`: Returns the sum of all item prices.
- The status bar shows '9:58 AM' and 'Python 3'.



Output:



Task 5: String Validation Module Using TDD

Scenario

You are working on a text-processing module where a function is required to identify whether a given string is a palindrome. The function must handle different cases and inputs reliably.

Task Description

Using Test Driven Development:

1. Write test cases for a palindrome checker covering:

o Simple palindromes

o Non-palindromes o

Case variations

2. Use GitHub Copilot or Cursor AI to generate the `is_palindrome()` function based on the test case expectations.

The function should be implemented only after tests are written.

Expected Outcome

- Clearly written test cases defining expected behavior
 - AI-assisted implementation of the palindrome checker
 - All test cases passing successfully • Evidence of TDD methodology applied correctly
- Code:

The image displays two sequential screenshots of a Google Colab notebook titled 'Untitled30.ipynb'. The browser tabs at the top include 'word - Search', 'Document 11.docx', 'google colab - Search', and 'Untitled30.ipynb - Colab'. The notebook interface shows a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all'. The left sidebar contains icons for file management and search. The main code area is divided into cells, each with a status indicator (e.g., '[13] ✓ Os').

Top Screenshot: The first cell contains the following code:

```
[13] ✓ Os
import unittest

# ----- TEST CASES FIRST (TDD) -----
class TestPalindrome(unittest.TestCase):

    def test_simple_palindrome(self):
        self.assertTrue(is_palindrome("madam"))

    def test_not_palindrome(self):
        self.assertFalse(is_palindrome("hello"))

    def test_case_insensitive(self):
        self.assertTrue(is_palindrome("Madam"))

    def test_with_spaces(self):
        self.assertTrue(is_palindrome("nurses run"))

    def test_single_character(self):
        self.assertTrue(is_palindrome("a"))
```

Bottom Screenshot: The notebook has been scrolled down. The first cell now shows the implementation of the `is_palindrome` function:

```
[13] ✓ Os
self.assertTrue(is_palindrome("nurses run"))

def test_single_character(self):
    self.assertTrue(is_palindrome("a"))

#Ai Implemented Code

[14] ✓ Os
# ----- IMPLEMENTATION AFTER TESTS -----
def is_palindrome(s):
    s = s.replace(" ", "").lower()
    return s == s[::-1]

#Run Tests

[15] ✓ Os
unittest.main(argv=[''], verbosity=2, exit=False)
```

The bottom screenshot also shows the 'Variables' and 'Terminal' tabs at the bottom of the notebook interface.

Output:

word - Search x Document 11.docx x google colab - Search x Untitled30.ipynb - Colab x +

https://colab.research.google.com/drive/18FWbvBLJlvCTXqsvYyNt7zYXoSnKhINR#scrollTo=LpQRy_SmCH9E

Untitled30.ipynb ☆ Saving...

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

RAM Disk

```
test_all_equal (__main__.TestMaxOfThree.test_all_equal) ... ok
test_first_is_largest (__main__.TestMaxOfThree.test_first_is_largest) ... ok
test_negative_numbers (__main__.TestMaxOfThree.test_negative_numbers) ... ok
test_normal_numbers (__main__.TestMaxOfThree.test_normal_numbers) ... ok
test_two_equal_largest (__main__.TestMaxOfThree.test_two_equal_largest) ... ok
test_case_insensitive (__main__.TestPalindrome.test_case_insensitive) ... ok
test_not_palindrome (__main__.TestPalindrome.test_not_palindrome) ... ok
test_simple_palindrome (__main__.TestPalindrome.test_simple_palindrome) ... ok
test_single_character (__main__.TestPalindrome.test_single_character) ... ok
test_with_spaces (__main__.TestPalindrome.test_with_spaces) ... ok
test_add_item (__main__.TestShoppingCart.test_add_item) ... ok
test_add_multiple_items (__main__.TestShoppingCart.test_add_multiple_items) ... ok
test_remove_item (__main__.TestShoppingCart.test_remove_item) ... ok
test_remove_non_existing_item (__main__.TestShoppingCart.test_remove_non_existing_item) ... ok
test_large_number (__main__.TestSquareFunction.test_large_number) ... ok
test_negative_number (__main__.TestSquareFunction.test_negative_number) ... ok
test_positive_number (__main__.TestSquareFunction.test_positive_number) ... ok
test_zero (__main__.TestSquareFunction.test_zero) ... ok

-----
Ran 24 tests in 0.032s

OK
<unittest.main.TestProgram at 0x7e0211f3cc80>
```

Variables Terminal

✓ 10:05 AM Python 3