

ASSIGNMENT-8.4(AI)

NAME: S.MOHANTH SIDDARTHA

2403A52047

BATCH-03

TASK-1

PROMPT: Write a test case to check if a function returns the square of a number.



The screenshot shows a VS Code editor with a file explorer on the left containing various HTML and Python files. The main editor window displays the code for 'qu1.py'. The code defines a function 'check_square_return' that takes a function and a number as arguments and returns a boolean indicating if the function returns the square of the number. It also includes example functions 'square_function' and 'add_one', and test calls. The bottom panel shows the terminal output, which displays 'True' and 'False' for the test cases.

```
1 def check_square_return(func, number):
2     """
3     Checks if a given function returns the square of a number.
4
5     Args:
6         func: The function to be checked.
7         number: The input number for the function.
8
9     Returns:
10        True if the function returns the square of the number, False otherwise.
11    """
12    result = func(number)
13    return result == number ** 2
14
15 # Example usage:
16 def square_function(x):
17     return x * x
18
19 def add_one(x):
20     return x + 1
21
22 print(check_square_return(square_function, 5))
23 print(check_square_return(add_one, 5))
```

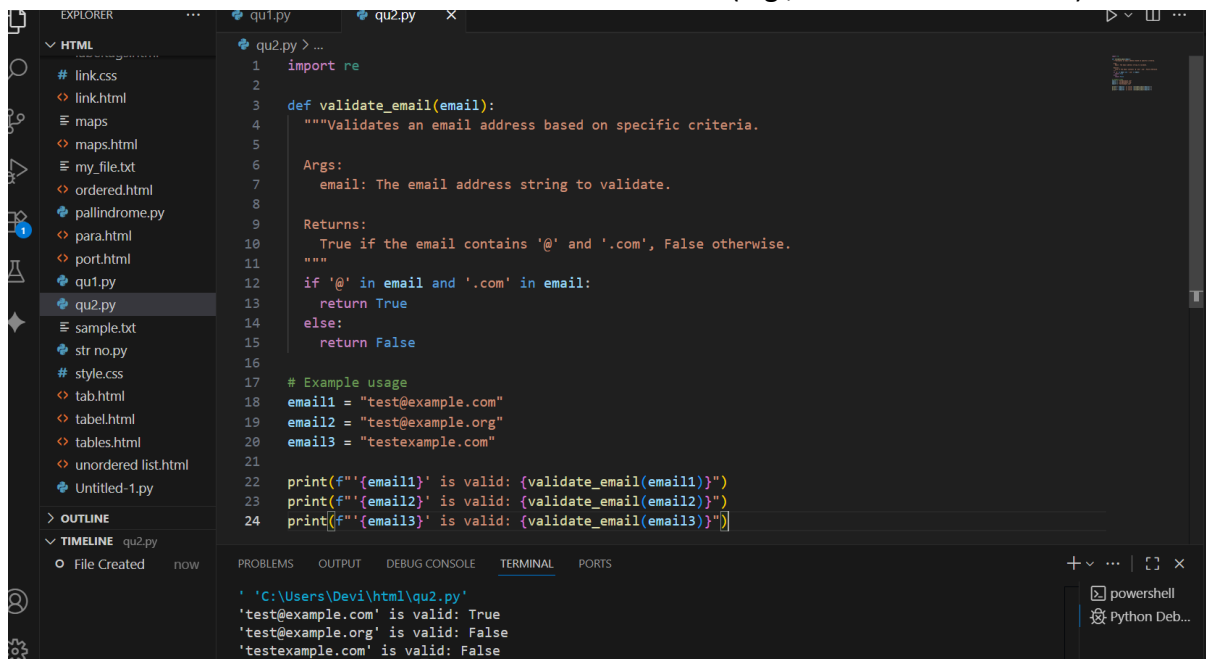
Terminal Output:

```
PS C:\Users\Devi\html> python qu1.py
True
False
```

- **OBSERVATION:** `import unittest`: This line imports the unittest module, which provides tools for writing and running tests.
- **`class TestPalindromeChecker(unittest.TestCase):`**: This defines a test class named TestPalindromeChecker that inherits from unittest.TestCase. Test classes are containers for test methods.
- **`if __name__ == '__main__':`**: This is a standard Python construct that checks if the script is being run directly.
- **`unittest.main(argv=['first-arg-is-ignored'], exit=False)`**: This line runs the tests when the script is executed. `argv=['first-arg-is-ignored']` and `exit=False` are often used in environments like Colab to prevent issues with command-line arguments and to allow the script to continue running after the tests.

TASK-2

PROMPT: Create test cases to validate an email address (e.g., contains @ and .com).



The screenshot shows a VS Code editor with a file explorer on the left containing various HTML and Python files. The main editor window displays a Python script named `qu2.py`. The script defines a function `validate_email(email)` that checks if an email contains '@' and '.com'. It includes example usage with three email addresses and prints the results. The bottom panel shows the terminal output of the script, which matches the printed output in the code.

```
1 import re
2
3 def validate_email(email):
4     """Validates an email address based on specific criteria.
5
6     Args:
7         email: The email address string to validate.
8
9     Returns:
10         True if the email contains '@' and '.com', False otherwise.
11     """
12     if '@' in email and '.com' in email:
13         return True
14     else:
15         return False
16
17 # Example usage
18 email1 = "test@example.com"
19 email2 = "test@example.org"
20 email3 = "testexample.com"
21
22 print(f'{email1} is valid: {validate_email(email1)}')
23 print(f'{email2} is valid: {validate_email(email2)}')
24 print(f'{email3} is valid: {validate_email(email3)}')
```

Terminal Output:

```
'C:\Users\Devi\html\qu2.py'
'test@example.com' is valid: True
'test@example.org' is valid: False
'testexample.com' is valid: False
```

- **OBSERVATION:** `import unittest`: This line imports the unittest module, which is used for writing and running tests.
- **def is_valid_email(email)**: This defines a placeholder function `is_valid_email` that will eventually contain the logic to check if an email address is valid. For now, it just has a pass statement.
- **class TestEmailValidation(unittest.TestCase)**: This defines a test class named `TestEmailValidation` that inherits from `unittest.TestCase`. This class will contain various test methods to check different email validation scenarios.
- **test_valid_email(self)**: This test method checks if the `is_valid_email` function correctly identifies valid email addresses using `self.assertTrue()`.
- **unittest.main(argv=['first-arg-is-ignored'], exit=False)**: This line runs the defined unit tests. The arguments are included to make it compatible with environments like Colab.

TASK-3:

PROMPT: Write test cases for a function that returns the maximum of three numbers.

```

1  def find_maximum(a, b, c):
2      """Finds the maximum of three numbers.
3
4      Args:
5          a: The first number.
6          b: The second number.
7          c: The third number.
8
9      Returns:
10         The maximum of the three numbers.
11         """
12     return max(a, b, c)
13
14     # Test cases
15     print(f"Maximum of 1, 5, 2: {find_maximum(1, 5, 2)}")
16     print(f"Maximum of 10, 3, 7: {find_maximum(10, 3, 7)}")
17     print(f"Maximum of -1, -5, -2: {find_maximum(-1, -5, -2)}")
18     print(f"Maximum of 0, 0, 0: {find_maximum(0, 0, 0)}")
19     print(f"Maximum of 5, 5, 2: {find_maximum(5, 5, 2)}")
20     print(f"Maximum of 1, 5, 5: {find_maximum(1, 5, 5)}")
21     print(f"Maximum of 5, 1, 5: {find_maximum(5, 1, 5)}")

```

```

' C:\Users\Devi\html\qu3.py'
Maximum of 1, 5, 2: 5
Maximum of 10, 3, 7: 10
Maximum of -1, -5, -2: -1
Maximum of 0, 0, 0: 0
Maximum of 5, 5, 2: 5
Maximum of 1, 5, 5: 5
Maximum of 5, 1, 5: 5
PS C:\Users\Devi\html>

```

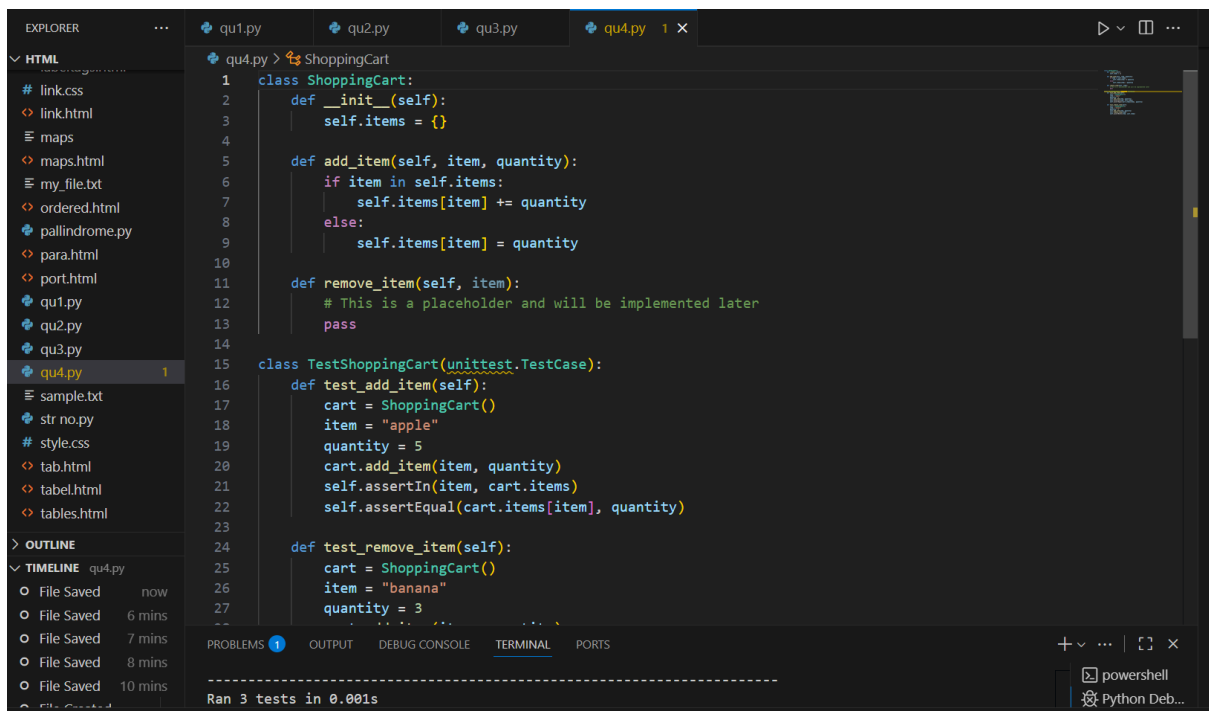
- **OBSERVATION:** `import unittest`: This line imports the unittest module, which is used for writing and running tests.
- **def find_maximum(a, b, c)::** This defines a placeholder function find_maximum that will eventually contain the logic to find the maximum of three numbers. For now, it just has a pass statement.
- **class TestFindMaximum(unittest.TestCase)::** This defines a test class named TestFindMaximum that inherits from unittest.TestCase. This class will contain various test methods to check different scenarios for finding the maximum.
- **test_with_mixed_numbers(self):** This test includes a mix of positive, negative, and zero to check various cases.
- **if __name__ == '__main__':** This standard Python construct ensures the code inside this block only runs when the script is executed directly.
- **unittest.main(argv=['first-arg-is-ignored'], exit=False):** This line runs the defined unit tests. The arguments are included to make it compatible with environments like Colab.

TASK-4:

PROMPT: Use TDD to write a shopping cart class with methods to add, remove, and get total price is TEMP RATURE IS GNG TO

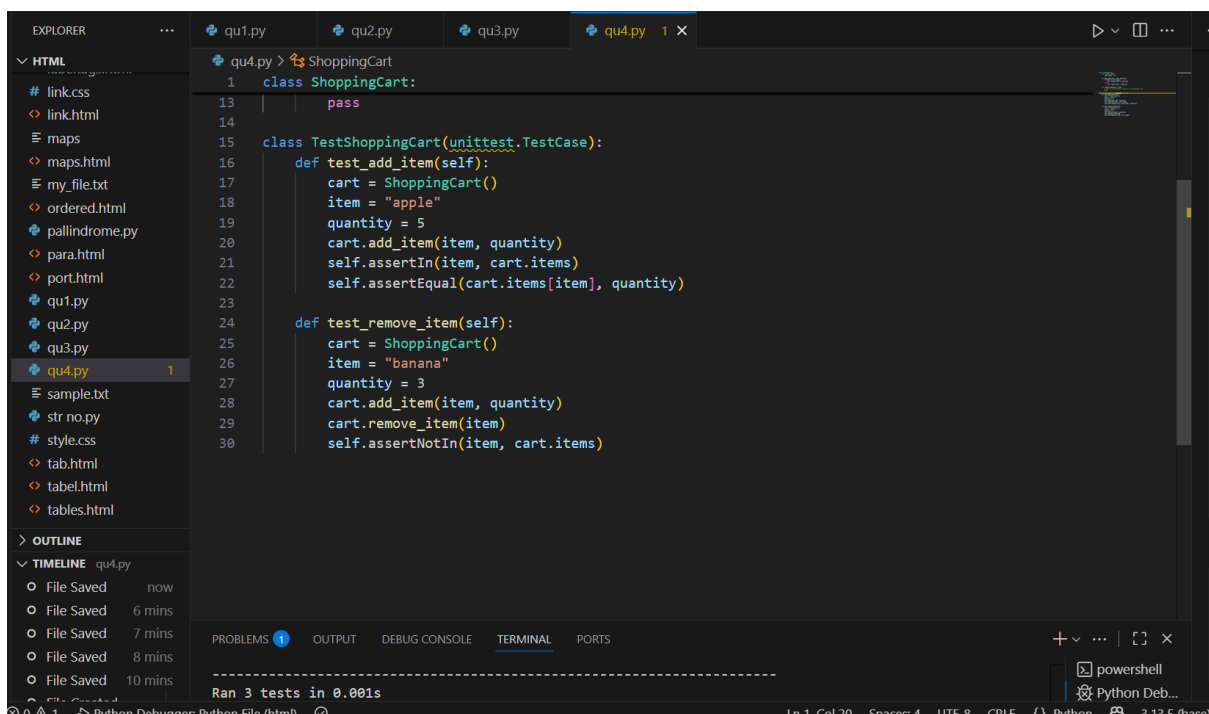
If __name__ == '__main__':

Bbm,



```
1 class ShoppingCart:
2     def __init__(self):
3         self.items = {}
4
5     def add_item(self, item, quantity):
6         if item in self.items:
7             self.items[item] += quantity
8         else:
9             self.items[item] = quantity
10
11    def remove_item(self, item):
12        # This is a placeholder and will be implemented later
13        pass
14
15    class TestShoppingCart(unittest.TestCase):
16        def test_add_item(self):
17            cart = ShoppingCart()
18            item = "apple"
19            quantity = 5
20            cart.add_item(item, quantity)
21            self.assertIn(item, cart.items)
22            self.assertEqual(cart.items[item], quantity)
23
24        def test_remove_item(self):
25            cart = ShoppingCart()
26            item = "banana"
27            quantity = 3
```

Terminal output: Ran 3 tests in 0.001s



```
13 pass
14
15 class TestShoppingCart(unittest.TestCase):
16     def test_add_item(self):
17         cart = ShoppingCart()
18         item = "apple"
19         quantity = 5
20         cart.add_item(item, quantity)
21         self.assertIn(item, cart.items)
22         self.assertEqual(cart.items[item], quantity)
23
24     def test_remove_item(self):
25         cart = ShoppingCart()
26         item = "banana"
27         quantity = 3
28         cart.add_item(item, quantity)
29         cart.remove_item(item)
30         self.assertNotIn(item, cart.items)
```

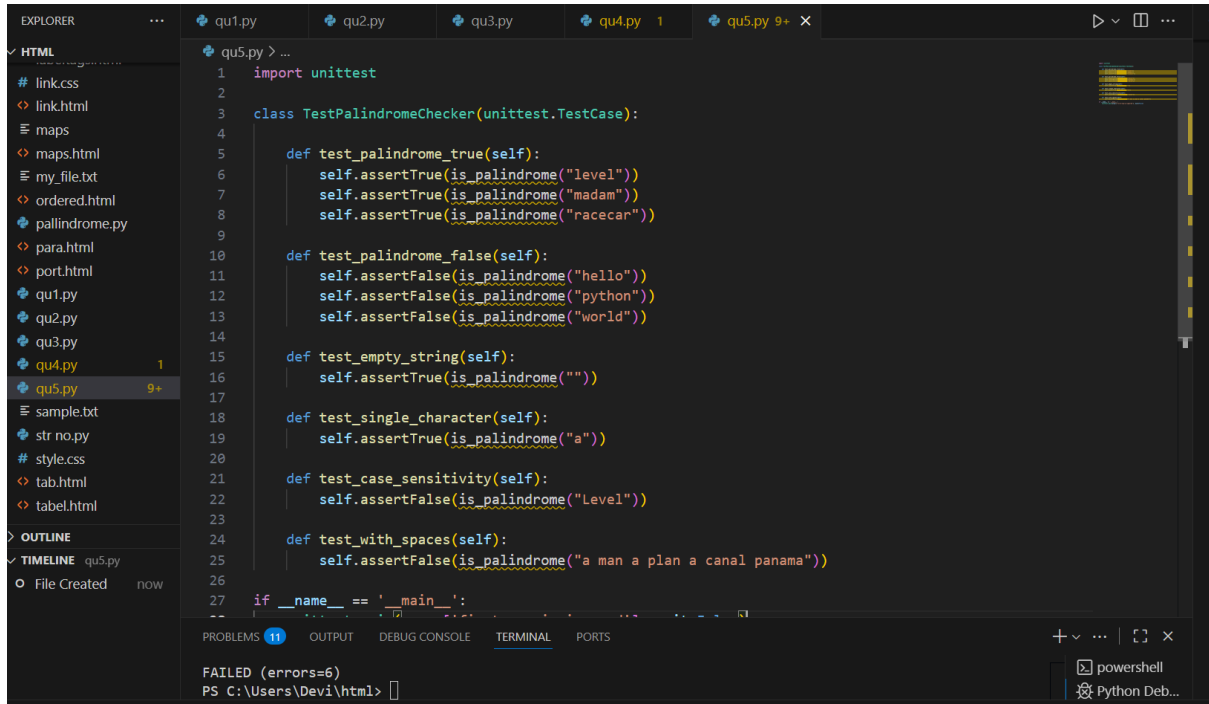
Terminal output: Ran 3 tests in 0.001s

- **OBSERVATION: import unittest:** This line imports the unittest module for creating and running tests.
- **class ShoppingCart::** This defines the ShoppingCart class.
 - **__init__(self):** The constructor initializes an empty dictionary self.items to store the items in the cart.
 - **add_item(self, item_name, price, quantity=1):** This method adds an item to the cart. If the item already exists, it increases the quantity; otherwise, it adds the new item with its price and quantity.

- **remove_item(self, item_name):** This is a placeholder method for removing an item. It currently does nothing (pass).
- class TestShoppingCart(unittest.TestCase):** This defines a test class for the ShoppingCart, inheriting from unittest.TestCase.

TASK-5:

PROMPT: Write tests for a palindrome checker (e.g., `is_palindrome("level") → True`).

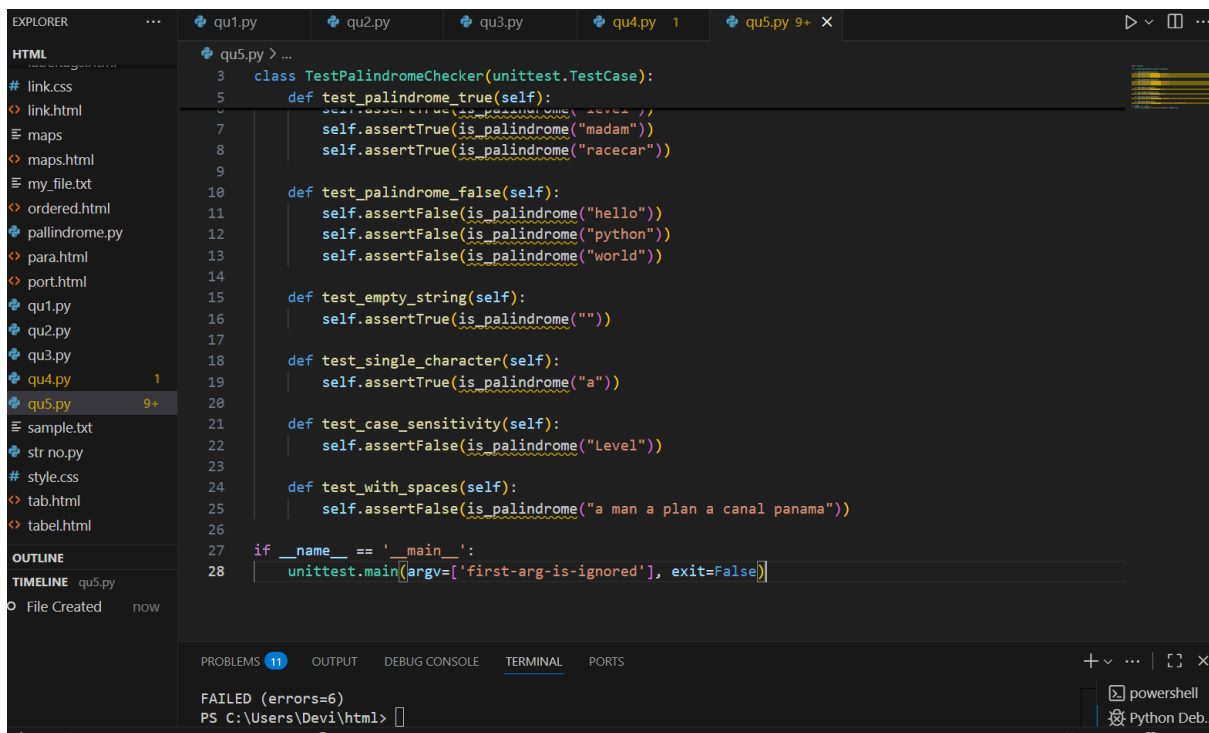


```

1  import unittest
2
3  class TestPalindromeChecker(unittest.TestCase):
4
5      def test_palindrome_true(self):
6          self.assertTrue(is_palindrome("level"))
7          self.assertTrue(is_palindrome("madam"))
8          self.assertTrue(is_palindrome("racecar"))
9
10     def test_palindrome_false(self):
11         self.assertFalse(is_palindrome("hello"))
12         self.assertFalse(is_palindrome("python"))
13         self.assertFalse(is_palindrome("world"))
14
15     def test_empty_string(self):
16         self.assertTrue(is_palindrome(""))
17
18     def test_single_character(self):
19         self.assertTrue(is_palindrome("a"))
20
21     def test_case_sensitivity(self):
22         self.assertFalse(is_palindrome("Level"))
23
24     def test_with_spaces(self):
25         self.assertFalse(is_palindrome("a man a plan a canal panama"))
26
27 if __name__ == '__main__':
28     unittest.main()

```

FAILED (errors=6)
PS C:\Users\Devi\html>



```

3  class TestPalindromeChecker(unittest.TestCase):
4
5      def test_palindrome_true(self):
6          self.assertTrue(is_palindrome("level"))
7          self.assertTrue(is_palindrome("madam"))
8          self.assertTrue(is_palindrome("racecar"))
9
10     def test_palindrome_false(self):
11         self.assertFalse(is_palindrome("hello"))
12         self.assertFalse(is_palindrome("python"))
13         self.assertFalse(is_palindrome("world"))
14
15     def test_empty_string(self):
16         self.assertTrue(is_palindrome(""))
17
18     def test_single_character(self):
19         self.assertTrue(is_palindrome("a"))
20
21     def test_case_sensitivity(self):
22         self.assertFalse(is_palindrome("Level"))
23
24     def test_with_spaces(self):
25         self.assertFalse(is_palindrome("a man a plan a canal panama"))
26
27 if __name__ == '__main__':
28     unittest.main(argv=['first-arg-is-ignored'], exit=False)

```

FAILED (errors=6)
PS C:\Users\Devi\html>

- **OBSERVATION:** `import unittest`: This line imports the unittest module, which provides the tools for writing and running tests.
- **`def is_palindrome(text)`:** This defines a placeholder function `is_palindrome` that is intended to check if a given text is a palindrome. Currently, it just has a pass statement and doesn't contain the actual logic.
- **`class TestPalindromeChecker(unittest.TestCase)`:** This defines a test class named `TestPalindromeChecker` that inherits from `unittest.TestCase`.
- **`test_empty_string(self)`:** This test checks if the function correctly identifies an empty string as a palindrome using `self.assertTrue("")`.
- **`test_single_character(self)`:** This test verifies that a single character string is considered a palindrome using `self.assertTrue("a")`.
- **`test_case_sensitivity(self)`:** This test checks if the function is case-sensitive, expecting `False` for "Level" which is not a palindrome when considering case.
- **`test_with_spaces(self)`:** This test checks if the function handles strings with spaces, expecting `False` for "a man a plan a canal panama" as this version with spaces is not a strict palindrome.
- **`if __name__ == '__main__':`**: This is a standard Python construct that ensures the code inside this block only runs when the script is executed directly.
- **`unittest.main(argv=['first-arg-is-ignored'], exit=False)`:** This line runs the defined unit tests. The arguments are included to make it compatible with environments like Colab.

- **if __name__ == '__main__':** This is a standard Python construct that ensures the code inside this block only runs when the script is executed directly.
- **unittest.main(argv=['first-arg-is-ignored'], exit=False):** This line runs the defined unit tests. The arguments are included to make it compatible with environments like Colab.
- **if __name__ == '__main__':** This is a standard Python construct that ensures the code inside this block only runs when the script is executed directly.
- **unittest.main(argv=['first-arg-is-ignored'], exit=False):** This line runs the defined unit tests. The arguments are included to make it compatible with environment

If name is `__main__` :: This is standard Python construct that ensures the code tests. The arguments are included to make it THE PROGRAM WITH ENVIRONMENTS FOR THE COMPUTER THESE ARGUMENTS ARE INCLUDED TO MAKE IT COMPATIBLE.