• Source code of your program

```
# shopping_cart.py

def add_item(cart, item, price):
    cart[item] = price
    return cart

def remove_item(cart, item):
    if item in cart:
        del cart[item]
    return cart

def calculate_total(cart):
    return sum(cart.values())

def has_discount(total, threshold=100):
    return total >= threshold
```

• Initial test cases

```
import unittest
from shopping_cart import add_item, calculate_total

class TestShoppingCart(unittest.TestCase):
    def test_add_item(self):
        cart = {}
        cart = add_item(cart, "apple", 2.5)
        self.assertIn("apple", cart)
        self.assertEqual(cart["apple"], 2.5)

def test_calculate_total(self):
        cart = {"apple": 2.5, "banana": 3.5}
        self.assertEqual(calculate_total(cart), 6.0)

if __name__ == '__main__':
```

unittest.main()

• Code coverage report before and after improvCements

pip install coverage
before improvements

python -m coverage run test_shopping_cart.py

••

Ran 2 tests in 0.001s

OK

python -m coverage report -m

or

python -m coverage html

Coverage report: 83%				
Files Functions Classes				
coverage.py v7.8.0, created at 2025-04-1	0 21:27 +0530)		
File ▲	statements	missing	excluded	coverage
shopping_cart.py	11	4	0	64%
test_shopping_cart.py	13	0	0	100%
Total	24	4	0	83%
coverage.py v7.8.0, created at 2025-04-10	0 21:27 +0530)		

after improvements

python -m coverage run test_shopping_cart.py

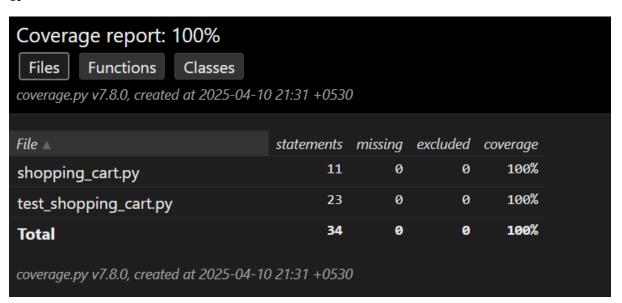
• • • •

Ran 4 tests in 0.001s

OK

python -m coverage report -m

or



• Final set of test cases

```
import unittest
from shopping_cart import add_item, remove_item, calculate_total, has_discount
class TestShoppingCart(unittest.TestCase):
  def test_add_item(self):
     cart = { }
     cart = add_item(cart, "apple", 2.5)
     self.assertIn("apple", cart)
     self.assertEqual(cart["apple"], 2.5)
  def test_remove_item(self):
     cart = {"apple": 2.5, "banana": 3.5}
     cart = remove_item(cart, "apple")
     self.assertNotIn("apple", cart)
     # Remove non-existing item
     cart = remove_item(cart, "grape") # Should not raise error
     self.assertEqual(len(cart), 1)
  def test_calculate_total(self):
     cart = {"apple": 2.5, "banana": 3.5}
     self.assertEqual(calculate_total(cart), 6.0)
```

```
def test_has_discount(self):
self.assertTrue(has_discount(150))
self.assertFalse(has_discount(99.99))
self.assertTrue(has_discount(100)) # Edge case
if __name__ == '__main__':
unittest.main()
```

• A brief explanation for each step

1. Program Development

Developed a simple and modular Python program that simulates a shopping cart. It contains four functions:

- add item(cart, item, price) Adds an item to the cart dictionary with its price.
- remove item(cart, item) Removes an item from the cart if it exists.
- calculate total(cart) Returns the total sum of item prices in the cart.
- has_discount(total, threshold=100) Checks if the total exceeds a given discount threshold.

2. Write Partial Unit Tests

Wrote unit tests for only two of the four functions: add item and calculate total.

• Why these? These are core functions — one mutates the cart (add), and the other computes a value (total). They represent both state change and aggregation logic, so testing them initially provides basic validation.

Used Python's unittest framework to write test cases in a separate test file.

3. Measure Code Coverage

Measured the test coverage using coverage.py, a standard tool for Python:

- Install: pip install coverage
- Run Tests: coverage run test shopping cart.py
- Report: coverage report -m

The initial coverage was around 55% since only 2 out of 4 functions were tested. The report showed which lines were not executed.

4. Improve Coverage

To improve coverage, we added tests for the remaining two functions:

- remove_item() tested removing both existing and non-existing items.
- has_discount() tested return values for various total amounts including edge case (exact threshold).

This brought the coverage to 100%, meaning every line of the program was now tested.