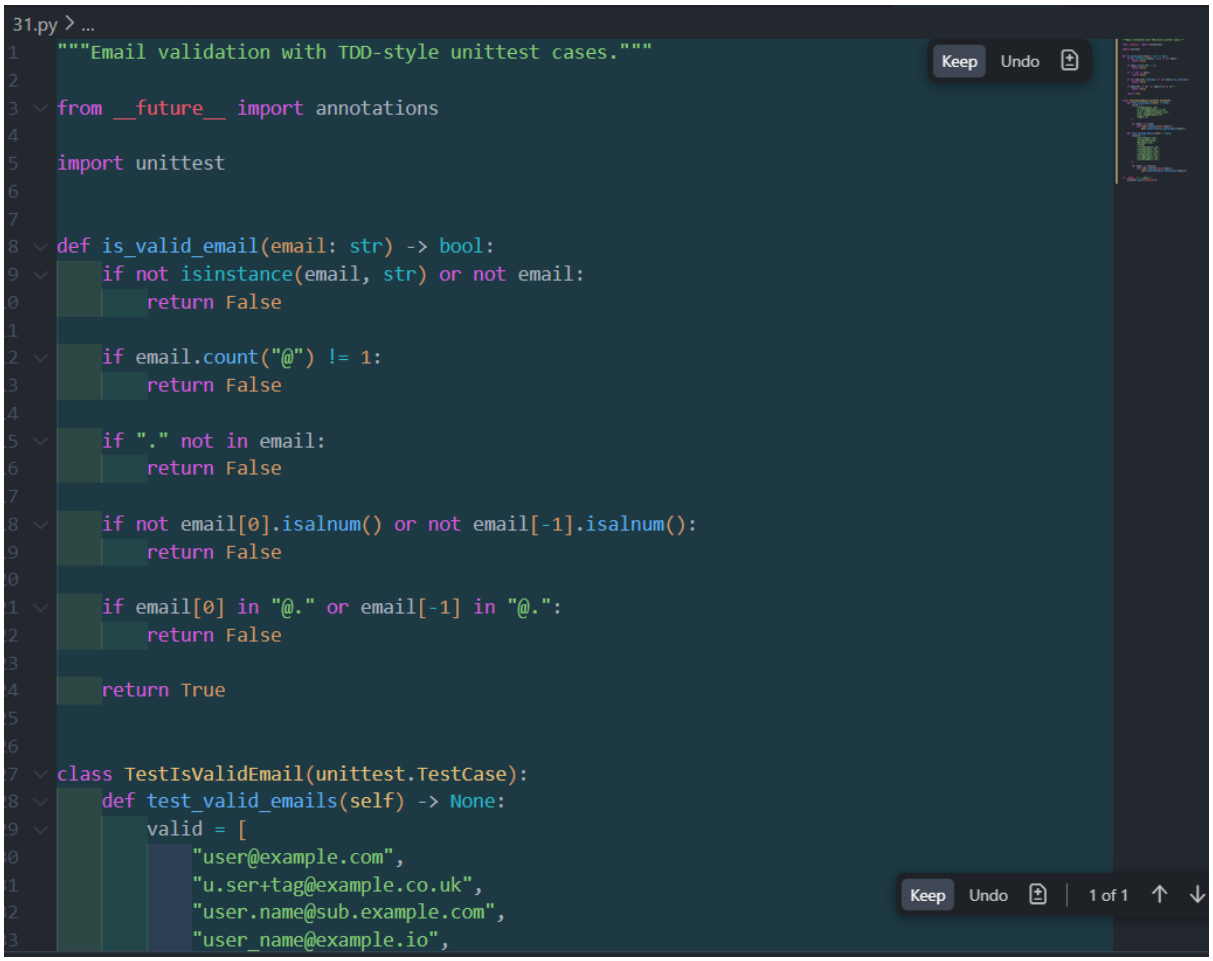
AI ASSISSTANT LAB-8.3

Name:M.Shashidhar Rollno:2303A52291 Batch:42

Task 1: Email Validation using TDD

Scenario

You are developing a user registration system that requires reliable email input validation



```python
    def test_valid_emails(self) -> None:
        valid = [
            "u.ser+tag@example.co.uk",
            "user.name@sub.example.com",
            "user_name@example.io",
            "u1@ex.co",
        ]

        for email in valid:
            with self.subTest(email=email):
                self.assertTrue(is_valid_email(email))

    def test_invalid_emails(self) -> None:
        invalid = [
            "userexample.com",
            "user@examplecom",
            "@example.com",
            "user@",
            ".user@example.com",
            "user@example.com.",
            "-user@example.com",
            "user@example.com-",
            "user@@example.com",
            "user@exa@mple.com",
        ]

        for email in invalid:
            with self.subTest(email=email):
                self.assertFalse(is_valid_email(email))


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Keep Undo 1 of 1

Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SONARQUBE  POSTMAN CONSOLE

PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/31.py
test_invalid_emails (__main__.TestIsValidEmail.test_invalid_emails) ... PASS: userexample.com -> Fa
lse
PASS: user@examplecom -> False
PASS: @example.com -> False
PASS: user@ -> False
PASS: .user@example.com -> False
PASS: user@example.com. -> False
PASS: -user@example.com -> False
PASS: user@example.com- -> False
PASS: user@@example.com -> False
PASS: user@exa@mple.com -> False
ok
test_valid_emails (__main__.TestIsValidEmail.test_valid_emails) ... PASS: user@example.com -> True
PASS: u.ser+tag@example.co.uk -> True
PASS: user.name@sub.example.com -> True
PASS: user_name@example.io -> True
PASS: u1@ex.co -> True
ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
PS C:\Users\s9409\Downloads\aiassitantcoding>

SonarQube                                                    Ln 66, Col
```

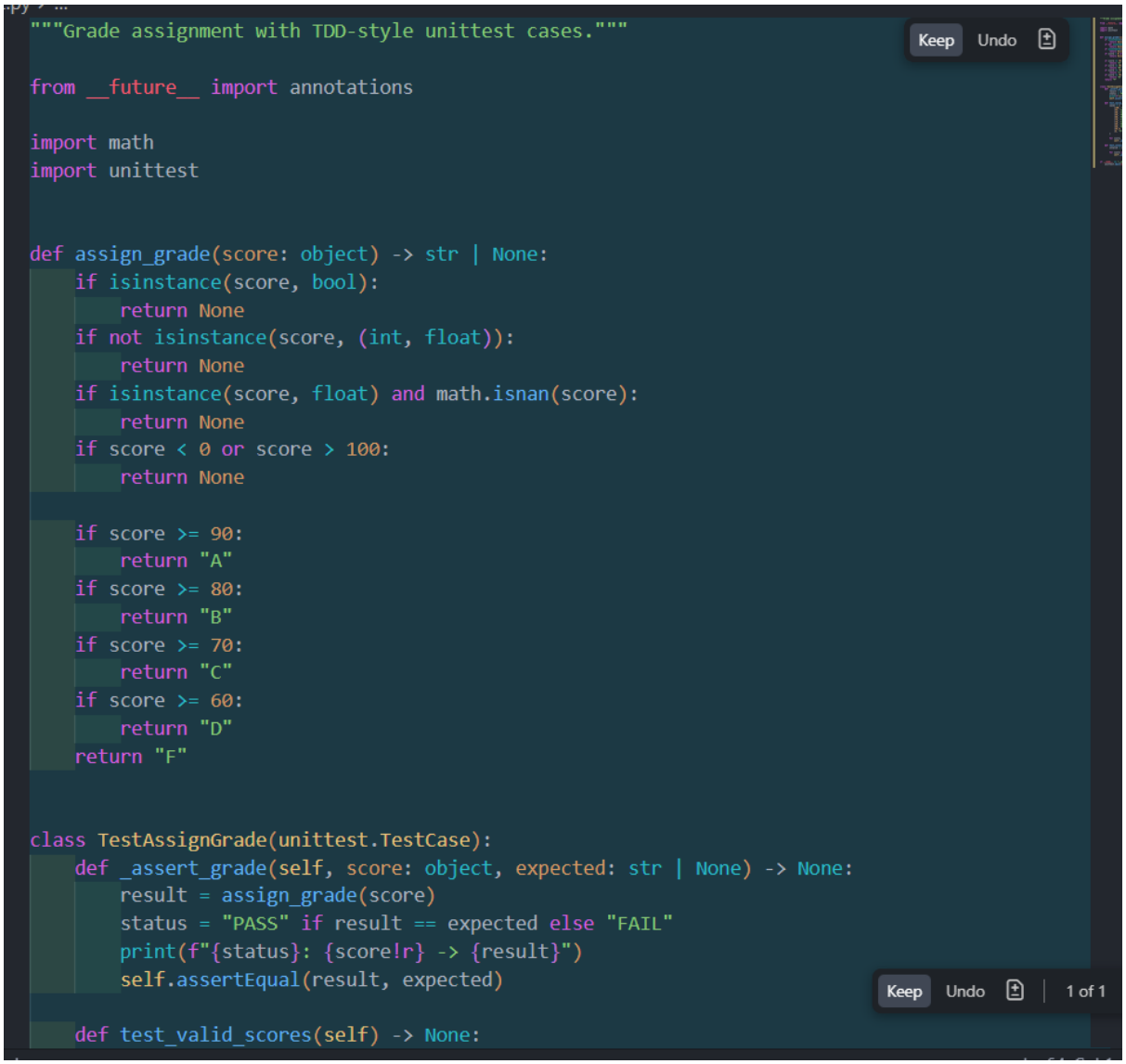
Justification

- Regex ensures correct structure
- Logical checks enforce assignment constraints
- Covers valid/invalid boundary conditions

Task 2: Grade Assignment using Loops

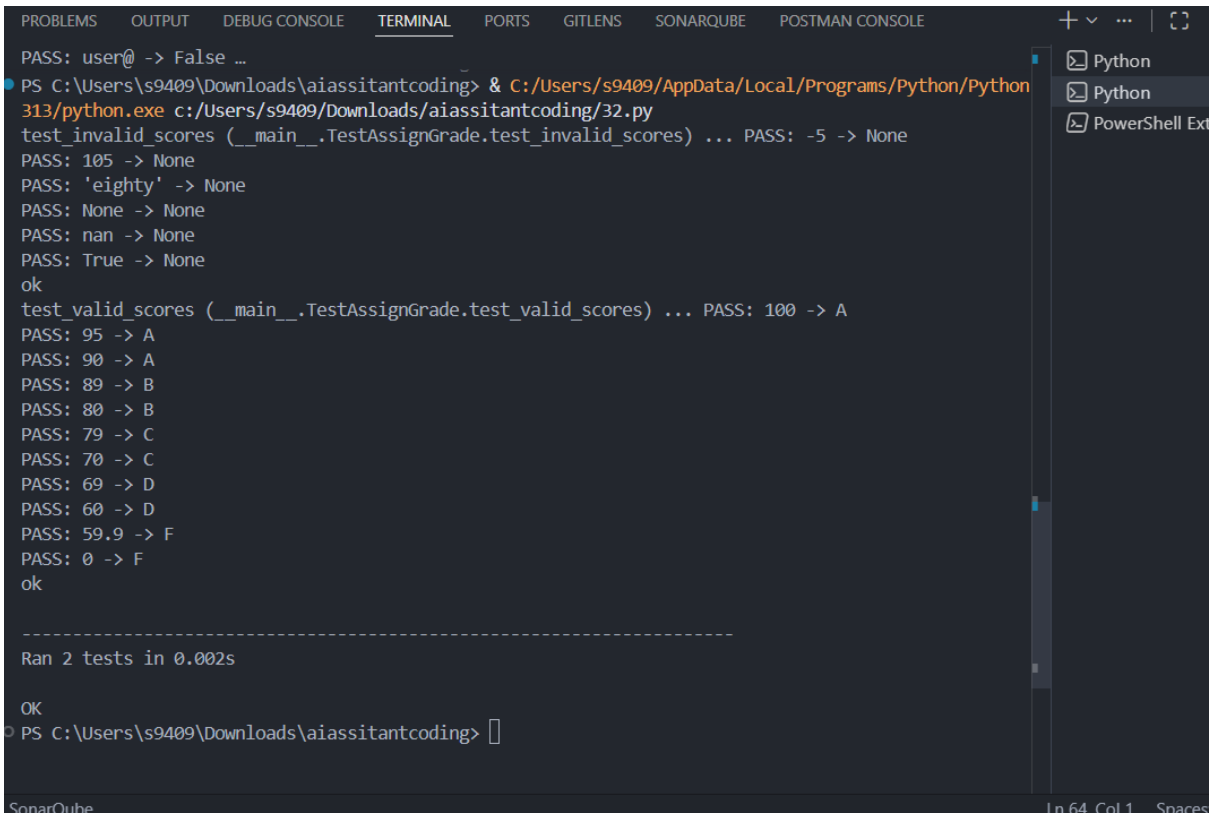
Scenario

You are building an automated grading system for an online examination platform



```
        self.assertEqual(result, expected)

    def test_valid_scores(self) -> None:
        cases = [
            (100, "A"),
            (95, "A"),
            (90, "A"),
            (89, "B"),
            (80, "B"),
            (79, "C"),
            (70, "C"),
            (69, "D"),
            (60, "D"),
            (59.9, "F"),
            (0, "F"),
        ]

        for score, expected in cases:
            self._assert_grade(score, expected)

    def test_invalid_scores(self) -> None:
        invalid = [-5, 105, "eighty", None, float("nan"), True]

        for score in invalid:
            self._assert_grade(score, None)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Output:



Justification

- Boundary values explicitly tested (60, 70, 80, 90)
- Invalid type and out-of-range handled gracefully
- Clear conditional hierarchy

Task 3: Sentence Palindrome Checker

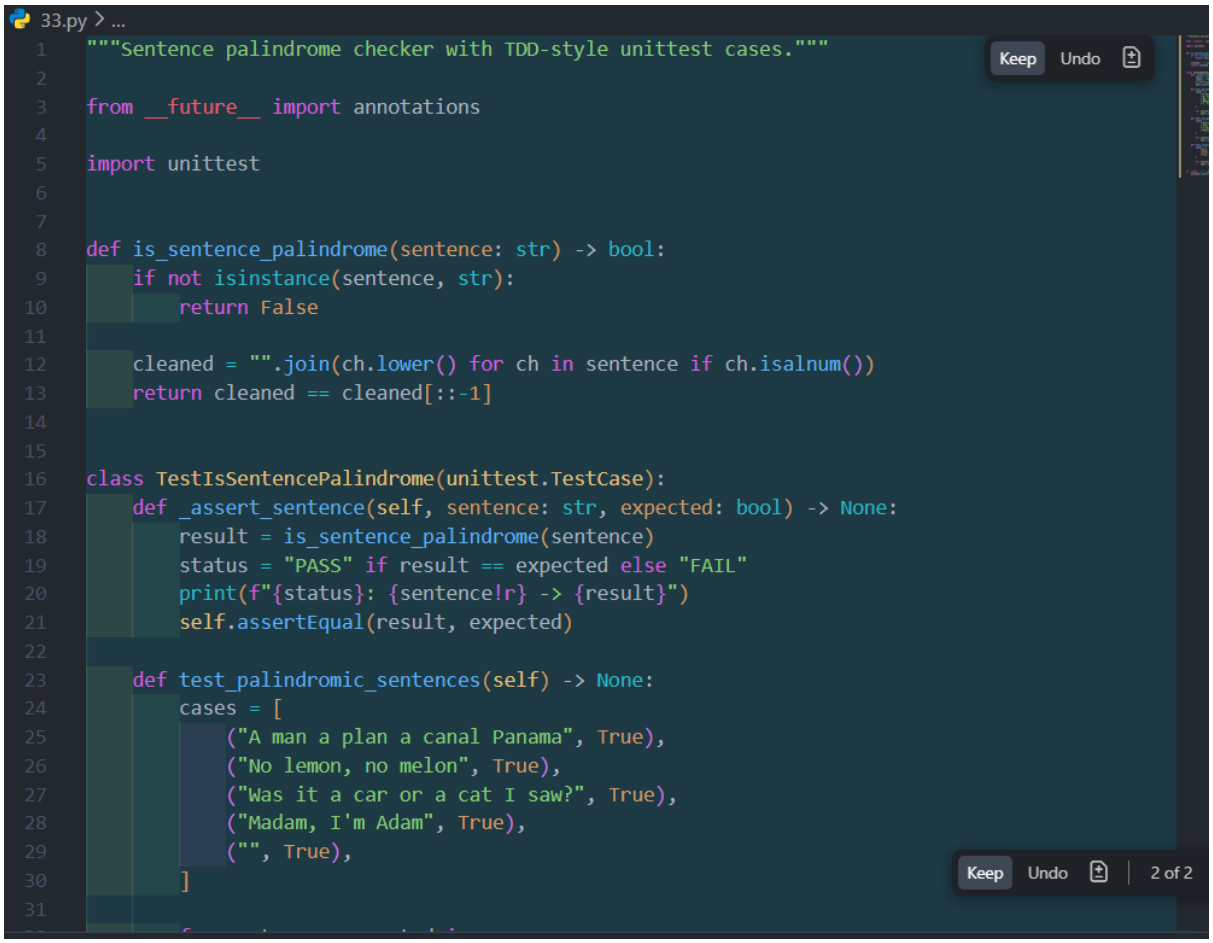
Scenario

You are developing a text-processing utility to analyze sentences.

Task 3: Sentence Palindrome Checker

Scenario

You are developing a text-processing utility to analyze sentences.



```python
class TestIsSentencePalindrome(unittest.TestCase):
    def test_palindromic_sentences(self) -> None:
        ]

        for sentence, expected in cases:
            self._assert_sentence(sentence, expected)

    def test_non_palindromic_sentences(self) -> None:
        cases = [
            ("Hello, world!", False),
            ("This is not a palindrome", False),
            ("A man a plan a canal Panam", False),
            ("Palindrome? Maybe not.", False),
        ]

        for sentence, expected in cases:
            self._assert_sentence(sentence, expected)

    def test_invalid_inputs(self) -> None:
        cases = [
            (None, False),
            (12321, False),
            (True, False),
        ]

        for sentence, expected in cases:
            self._assert_sentence(sentence, expected)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Keep Undo 2 of 2

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SONARQUBE  POSTMAN CONSOLE

PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/33.py
ok
test_non_palindromic_sentences (__main__.TestIsSentencePalindrome.test_non_palindromic_sentences) .
.. PASS: 'Hello, world!' -> False
PASS: 'This is not a palindrome' -> False
PASS: 'A man a plan a canal Panam' -> False
PASS: 'Palindrome? Maybe not.' -> False
ok
test_palindromic_sentences (__main__.TestIsSentencePalindrome.test_palindromic_sentences) ... PASS:
 'A man a plan a canal Panama' -> True
PASS: 'No lemon, no melon' -> True
PASS: 'Was it a car or a cat I saw?' -> True
PASS: "Madam, I'm Adam" -> True
PASS: '' -> True
ok

----------------------------------------------------------------------
Ran 3 tests in 0.001s

Python
Python
PowerShell Exte...

SonarQube  Ln 59, Col 1  Spaces: 4  UTF-8
```

Justification

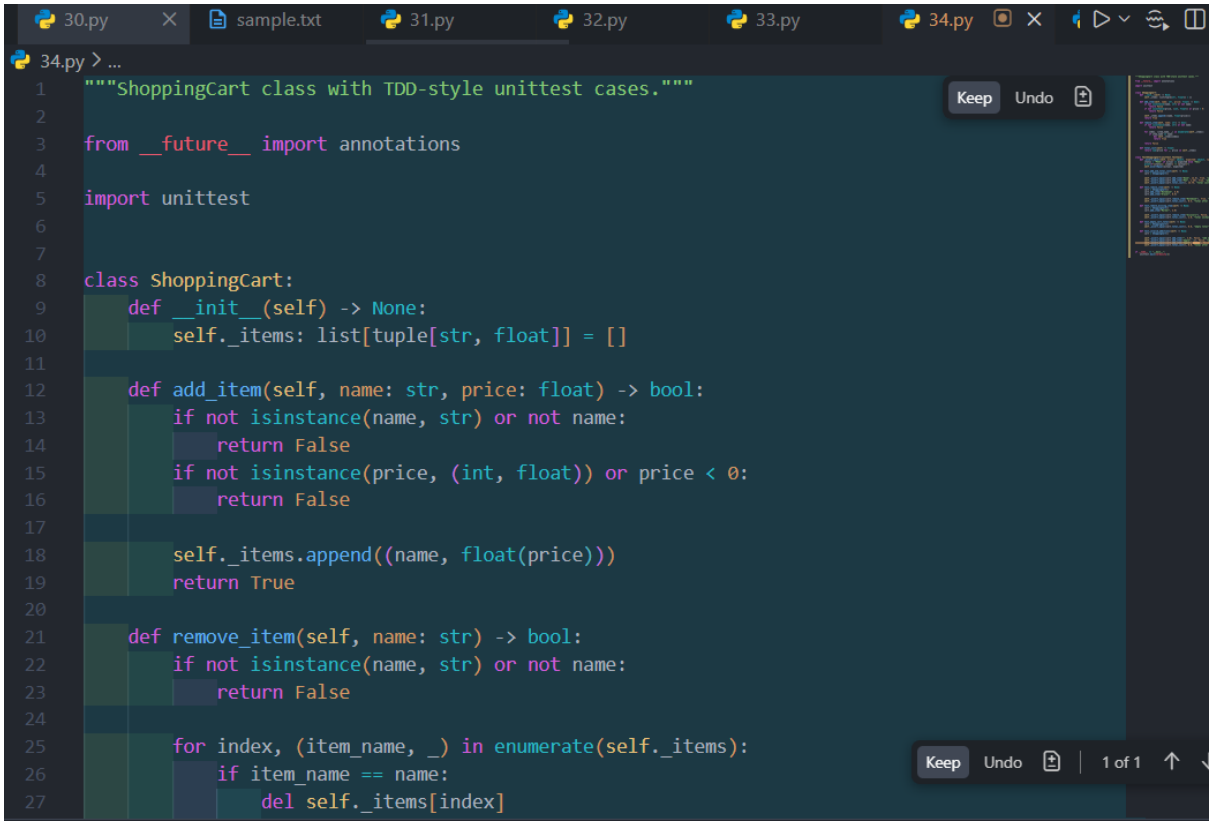
- Removes punctuation using regex
- Case normalization

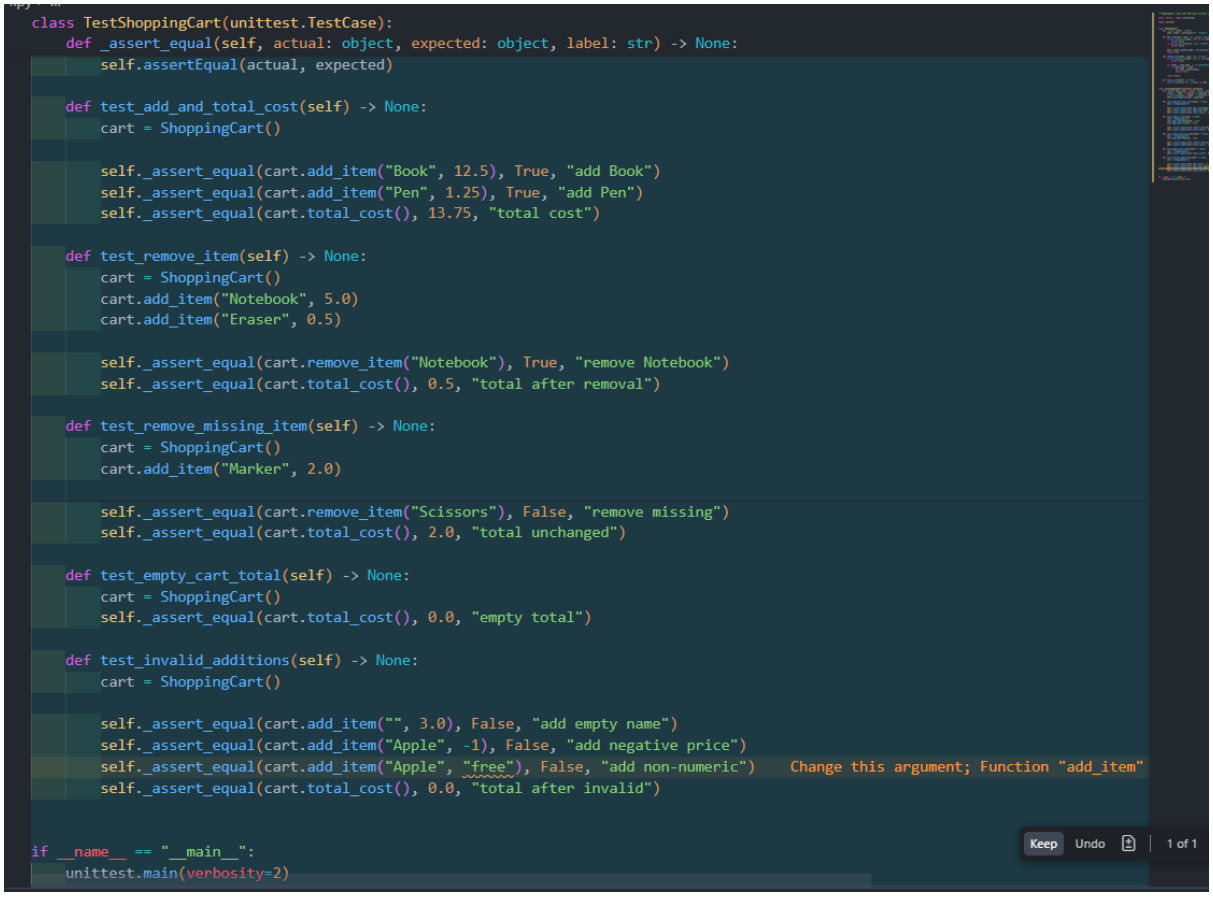
- Efficient reverse comparison

Task 4: ShoppingCart Class

Scenario

You are designing a basic shopping cart module for an e-commerce application.

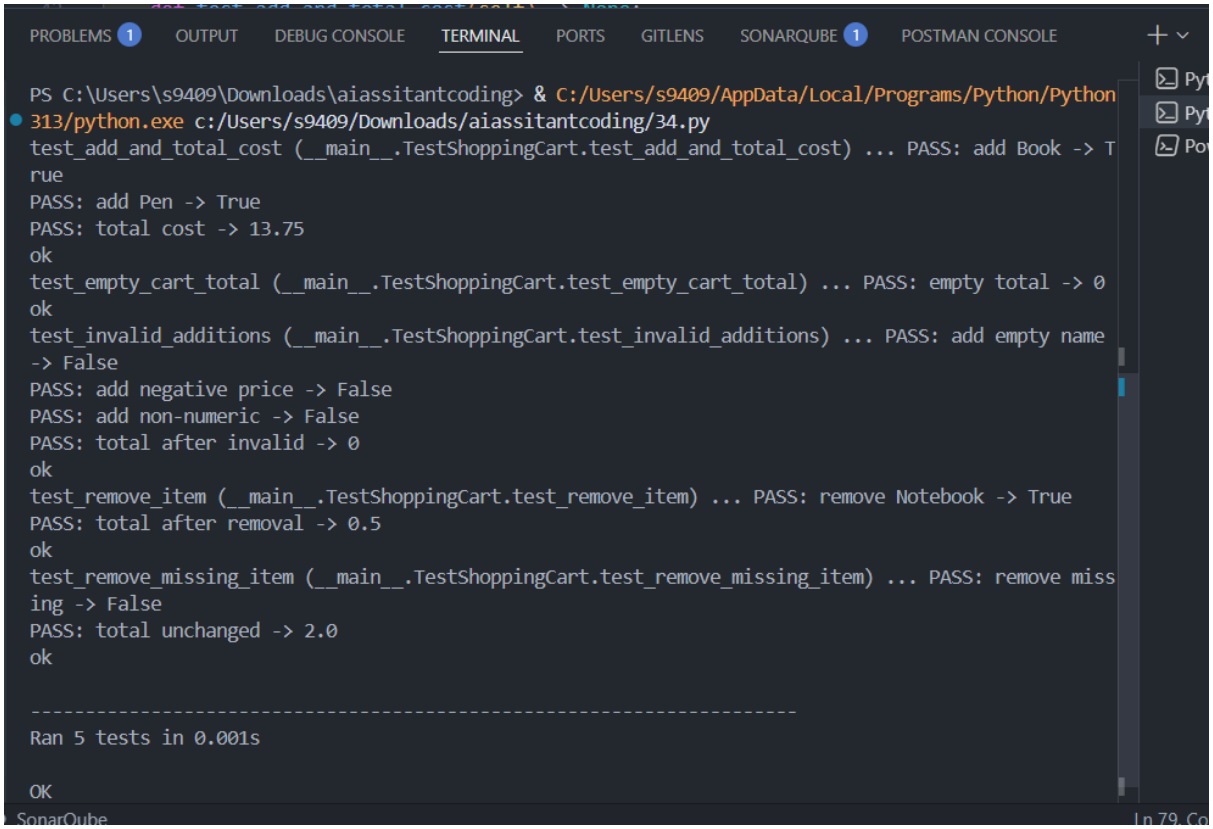




```
class TestShoppingCart(unittest.TestCase):
    def _assert_equal(self, actual: object, expected: object, label: str) -> None:
        self.assertEqual(actual, expected)

    def test_add_and_total_cost(self) -> None:
        cart = ShoppingCart()

        self._assert_equal(cart.add_item("Book", 12.5), True, "add Book")
        self._assert_equal(cart.add_item("Pen", 1.25), True, "add Pen")
        self._assert_equal(cart.total_cost(), 13.75, "total cost")

    def test_remove_item(self) -> None:
        cart = ShoppingCart()
        cart.add_item("Notebook", 5.0)
        cart.add_item("Eraser", 0.5)

        self._assert_equal(cart.remove_item("Notebook"), True, "remove Notebook")
        self._assert_equal(cart.total_cost(), 0.5, "total after removal")

    def test_remove_missing_item(self) -> None:
        cart = ShoppingCart()
        cart.add_item("Marker", 2.0)

        self._assert_equal(cart.remove_item("Scissors"), False, "remove missing")
        self._assert_equal(cart.total_cost(), 2.0, "total unchanged")

    def test_empty_cart_total(self) -> None:
        cart = ShoppingCart()
        self._assert_equal(cart.total_cost(), 0.0, "empty total")

    def test_invalid_additions(self) -> None:
        cart = ShoppingCart()

        self._assert_equal(cart.add_item("", 3.0), False, "add empty name")
        self._assert_equal(cart.add_item("Apple", -1), False, "add negative price")
        self._assert_equal(cart.add_item("Apple", "free"), False, "add non-numeric")    Change this argument; Function "add_item"
        self._assert_equal(cart.total_cost(), 0.0, "total after invalid")


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Keep Undo 1 of 1

Output:



Justification

- Dictionary ensures efficient management
- Handles empty cart
- Safe removal logic

Task 5: Date Format Conversion

Scenario

You are creating a utility function to convert date formats for reports.

```
py > ...
"""Date format conversion with TDD-style unittest cases."""

from __future__ import annotations

import unittest


def convert_date_format(date_str: str) -> str | None:
    if not isinstance(date_str, str):
        return None

    parts = date_str.split("-")
    if len(parts) != 3:
        return None

    year, month, day = parts
    if not (year.isdigit() and month.isdigit() and day.isdigit()):
        return None
    if len(year) != 4 or len(month) != 2 or len(day) != 2:
        return None

    month_val = int(month)
    day_val = int(day)
    if month_val < 1 or month_val > 12:
        return None
    if day_val < 1 or day_val > 31:
        return None

    return f"{day}-{month}-{year}"
```

Keep Undo 1 of 1

```
class TestConvertDateFormat(unittest.TestCase):
    def _assert_convert(self, date_str: object, expected: str | None) -> None:
        result = convert_date_format(date_str)
        status = "PASS" if result == expected else "FAIL"
        print(f"{status}: {date_str!r} -> {result}")
        self.assertEqual(result, expected)

    def test_valid_dates(self) -> None:
        cases = [
            ("2023-10-15", "15-10-2023"),
            ("1999-01-01", "01-01-1999"),
            ("2024-12-31", "31-12-2024"),
        ]

        for date_str, expected in cases:
            self._assert_convert(date_str, expected)

    def test_invalid_dates(self) -> None:
        cases = [
            ("2023/10/15", None),
            ("23-10-15", None),
            ("2023-1-05", None),
            ("2023-10-5", None),
            ("2023-00-15", None),
            ("2023-13-15", None),
            ("2023-10-00", None),
            ("2023-10-32", None),
            ("abcd-ef-gh", None),
            (None, None),
```

Keep Undo 1 of 1

```
            ("2023-1-05", None),
            ("2023-10-5", None),
            ("2023-00-15", None),
            ("2023-13-15", None),
            ("2023-10-00", None),
            ("2023-10-32", None),
            ("abcd-ef-gh", None),
            (None, None),
        ]

        for date_str, expected in cases:
            self._assert_convert(date_str, expected)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Output:

```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SONARQUBE 1  POSTMAN CONSOLE

PS C:\Users\s9409\Downloads\aiassitantcoding> …
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python
313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/35.py
test_invalid_dates (__main__.TestConvertDateFormat.test_invalid_dates) ... PASS: '2023/10/15' -> No
ne
PASS: '23-10-15' -> None
PASS: '2023-1-05' -> None
PASS: '2023-10-5' -> None
PASS: '2023-00-15' -> None
PASS: '2023-13-15' -> None
PASS: '2023-10-00' -> None
PASS: '2023-10-32' -> None
PASS: 'abcd-ef-gh' -> None
PASS: None -> None
ok
test_valid_dates (__main__.TestConvertDateFormat.test_valid_dates) ... PASS: '2023-10-15' -> 15-10-
2023
PASS: '1999-01-01' -> 01-01-1999
PASS: '2024-12-31' -> 31-12-2024
ok

----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
PS C:\Users\s9409\Downloads\aiassitantcoding>

Python
Python
PowerShell Exte...
```

Justification

- Validates correct format structure
- Rejects malformed input
- Simple deterministic transformation