

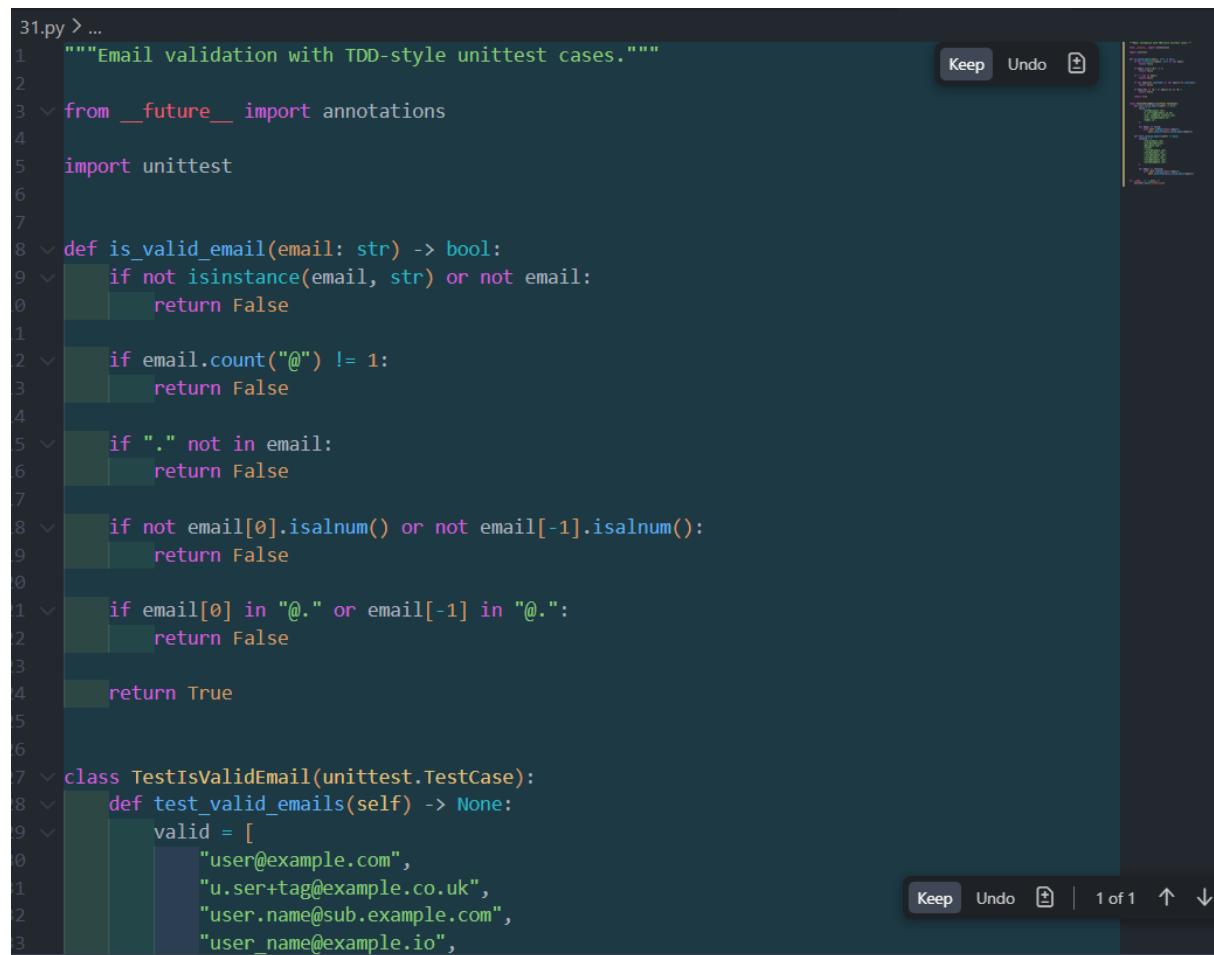
AI ASSISTANT LAB-8.3

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



The screenshot shows a code editor window with Python code for email validation and unit testing. The code is organized into two main sections: a utility function `is_valid_email` and a test class `TestIsValidEmail`.

```
31.py > ...
1 """Email validation with TDD-style unittest cases."""
2
3 from __future__ import annotations
4
5 import unittest
6
7
8 def is_valid_email(email: str) -> bool:
9     if not isinstance(email, str) or not email:
10        return False
11
12    if email.count "@" != 1:
13        return False
14
15    if "." not in email:
16        return False
17
18    if not email[0].isalnum() or not email[-1].isalnum():
19        return False
20
21    if email[0] in "@." or email[-1] in "@.":
22        return False
23
24    return True
25
26
27 class TestIsValidEmail(unittest.TestCase):
28     def test_valid_emails(self) -> None:
29         valid = [
30             "user@example.com",
31             "u.ser+tag@example.co.uk",
32             "user.name@sub.example.com",
33             "user_name@example.io",
34         ]
```

The code editor interface includes tabs for 'Keep' and 'Undo', and a status bar at the bottom indicating '1 of 1'.

```

8     def test_valid_emails(self) -> None:
9         "u.ser+tag@example.co.uk",
10        "user.name@sub.example.com",
11        "user_name@example.io",
12        "u1@ex.co",
13    ]
14
15    for email in valid:
16        with self.subTest(email=email):
17            self.assertTrue(is_valid_email(email))
18
19
20    def test_invalid_emails(self) -> None:
21        invalid = [
22            "userexample.com",
23            "user@examplecom",
24            "@example.com",
25            "user@",
26            ".user@example.com",
27            "user@example.com.",
28            "-user@example.com",
29            "user@example.com-",
30            "user@@example.com",
31            "user@exa@mple.com",
32        ]
33
34        for email in invalid:
35            with self.subTest(email=email):
36                self.assertFalse(is_valid_email(email))
37
38
39 if __name__ == "__main__":
40     unittest.main(verbosity=2)

```

Keep Undo ⌘ | 1 of 1

Output

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SONARQUBE POSTMAN CONSOLE + Py Py

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 -> False
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

In 66 Col

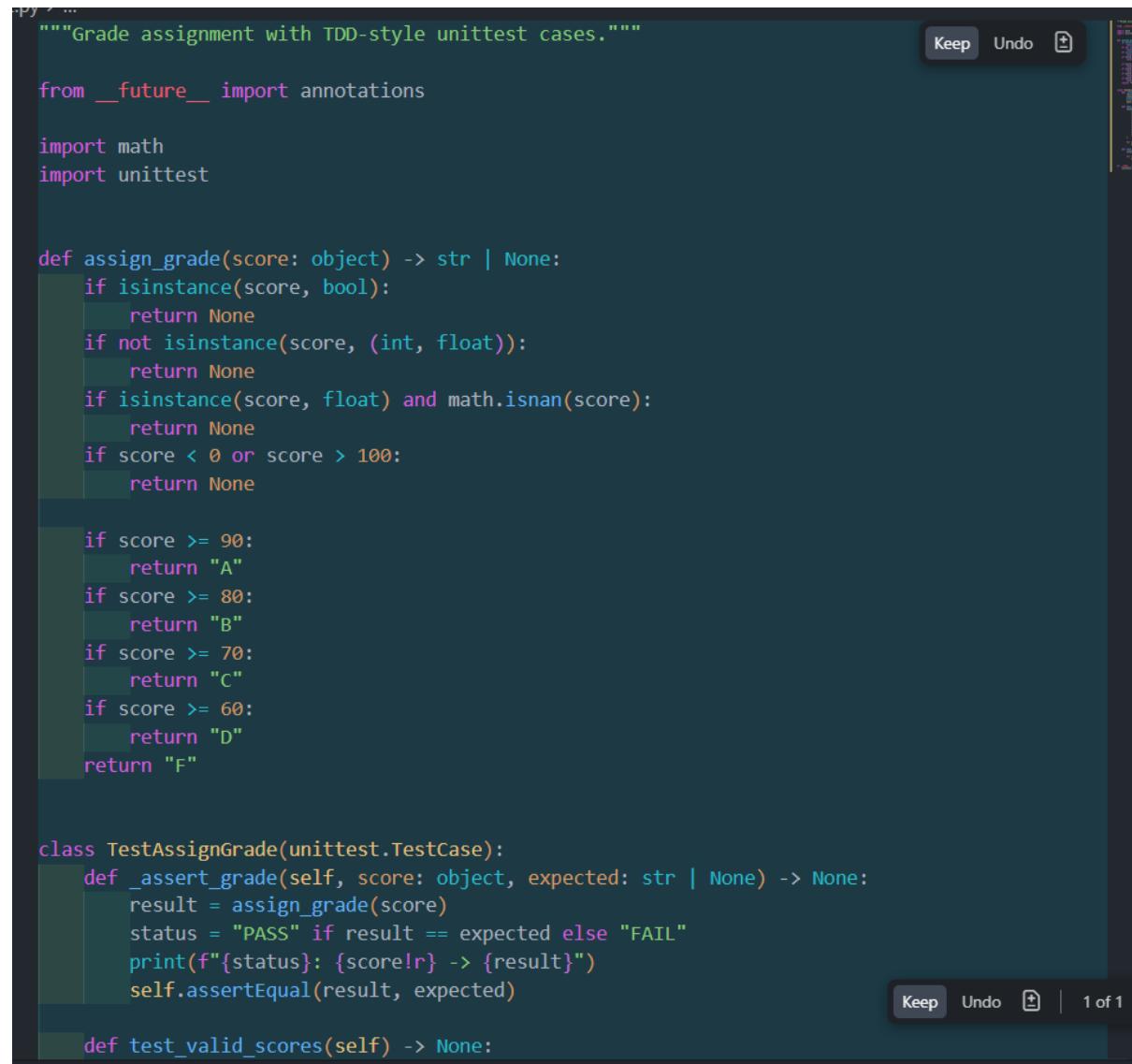
Justification

- 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



The screenshot shows a code editor window with Python code. The code defines a function `assign_grade` that takes a score and returns a grade or None. It includes validation for bool, float, and NaN values, and handles scores from 0 to 100. A `TestAssignGrade` class is defined with a test method `test_valid_scores` using a TDD-style `_assert_grade` helper.

```
py > ...
"""Grade assignment with TDD-style unittest cases."""
from __future__ import annotations

import math
import unittest

def assign_grade(score: object) -> str | None:
    if isinstance(score, bool):
        return None
    if not isinstance(score, (int, float)):
        return None
    if isinstance(score, float) and math.isnan(score):
        return None
    if score < 0 or score > 100:
        return None

    if score >= 90:
        return "A"
    if score >= 80:
        return "B"
    if score >= 70:
        return "C"
    if score >= 60:
        return "D"
    return "F"

class TestAssignGrade(unittest.TestCase):
    def _assert_grade(self, score: object, expected: str | None) -> None:
        result = assign_grade(score)
        status = "PASS" if result == expected else "FAIL"
        print(f"{status}: {score!r} -> {result}")
        self.assertEqual(result, expected)

    def test_valid_scores(self) -> None:
```

```
self.assertEqual(result, expected)

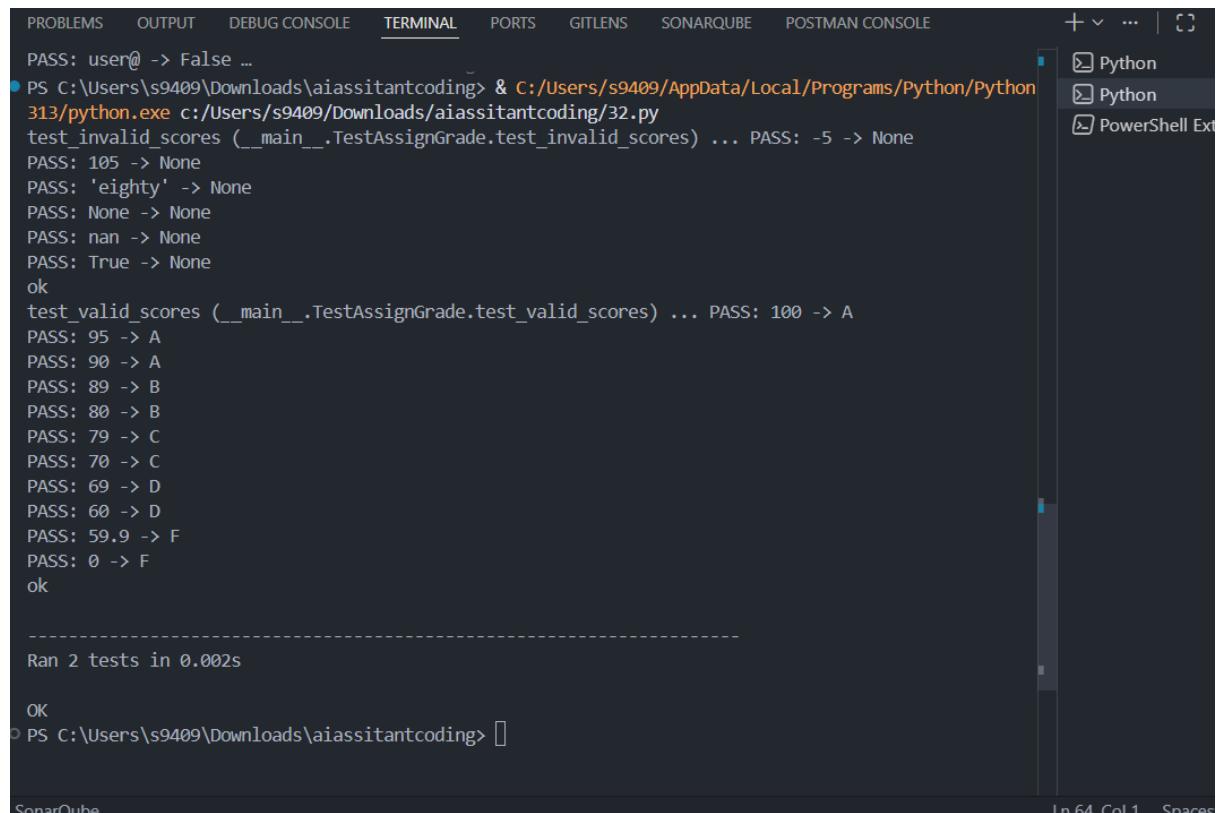
def test_valid_scores(self) -> None:
    cases = [
        (100, "A"),
        (95, "A"),
        (90, "A"),
        (89, "B"),
        (80, "B"),
        (79, "C"),
        (70, "C"),
        (69, "D"),
        (60, "D"),
        (59.9, "F"),
        (0, "F"),
    ]
    for score, expected in cases: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



The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the execution of a Python test script. The output shows various test cases being run, with each case's result (PASS or FAIL) and the corresponding grade or status message. The test cases cover valid scores from 0 to 100 and invalid inputs like negative numbers, strings, and non-finite floating-point values.

```
PASS: user@ -> False ...
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python 313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/32.py
test_invalid_scores (__main__.TestAssignGrade.test_invalid_scores) ... PASS: -5 -> None
PASS: 105 -> None
PASS: 'eighty' -> None
PASS: None -> None
PASS: nan -> None
PASS: True -> None
ok
test_valid_scores (__main__.TestAssignGrade.test_valid_scores) ... PASS: 100 -> A
PASS: 95 -> A
PASS: 90 -> A
PASS: 89 -> B
PASS: 80 -> B
PASS: 79 -> C
PASS: 70 -> C
PASS: 69 -> D
PASS: 60 -> D
PASS: 59.9 -> F
PASS: 0 -> F
ok

-----
Ran 2 tests in 0.002s

OK
PS C:\Users\s9409\Downloads\aiassitantcoding> []
```

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

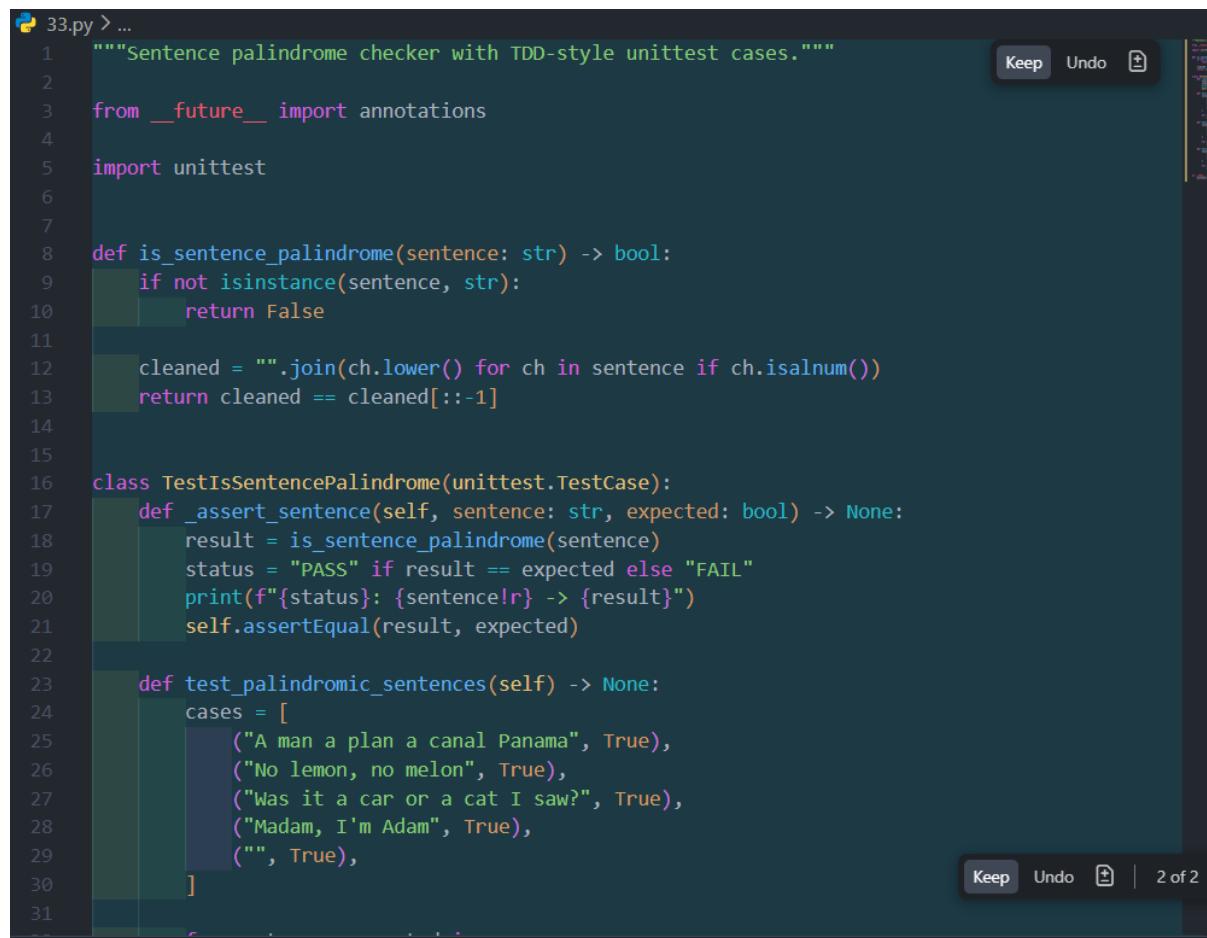
Scenario

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.



The screenshot shows a code editor window with Python code for a sentence palindrome checker. The code includes a docstring, imports for annotations and unittest, and a function to check if a sentence is a palindrome. It also contains a test class with an assertion method and a test method for multiple sentences. The code is syntax-highlighted, and the editor interface includes tabs for 'Keep' and 'Undo'.

```
33.py > ...
1 """Sentence palindrome checker with TDD-style unittest cases."""
2
3 from __future__ import annotations
4
5 import unittest
6
7
8 def is_sentence_palindrome(sentence: str) -> bool:
9     if not isinstance(sentence, str):
10         return False
11
12     cleaned = "".join(ch.lower() for ch in sentence if ch.isalnum())
13     return cleaned == cleaned[::-1]
14
15
16 class TestIsSentencePalindrome(unittest.TestCase):
17     def _assert_sentence(self, sentence: str, expected: bool) -> None:
18         result = is_sentence_palindrome(sentence)
19         status = "PASS" if result == expected else "FAIL"
20         print(f"{status}: {sentence!r} -> {result}")
21         self.assertEqual(result, expected)
22
23     def test_palindromic_sentences(self) -> None:
24         cases = [
25             ("A man a plan a canal Panama", True),
26             ("No lemon, no melon", True),
27             ("Was it a car or a cat I saw?", True),
28             ("Madam, I'm Adam", True),
29             ("", True),
30         ]
31
```

A screenshot of the Visual Studio Code interface. The main area shows Python test code for determining if sentences are palindromes. The code uses the `unittest` module. It defines three test functions: `test_palindromic_sentences`, `test_non_palindromic_sentences`, and `test_invalid_inputs`. Each test iterates over a list of cases, asserting that the function returns the expected boolean value. The code includes imports for `unittest` and `assertion`. At the bottom, there's a conditional block to run the tests with `verbosity=2`. The status bar at the bottom right shows "Keep Undo ⌘W | 2 of 2 ↑".

```
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)
```

Output:

A screenshot of the Visual Studio Code interface focusing on the Terminal tab. The terminal window displays the output of running the Python test script. It shows the command `python.exe` being run with the file path `C:/Users/s9409/Downloads/aiassitantcoding/33.py`. The output details the execution of three test cases: `test_non_palindromic_sentences` and `test_palindromic_sentences`. Each case lists several test assertions with their results (e.g., PASS or FAIL). The terminal also shows the message "Ran 3 tests in 0.001s". The status bar at the bottom right indicates the current line is 59, column 1, with 4 spaces and UTF-8 encoding.

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/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
```

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.

```

30.py   X  sample.txt  31.py  32.py  33.py  34.py  X  ⌂  ⌂  ⌂
34.py > ...
"""
ShoppingCart class with TDD-style unittest cases."""
from __future__ import annotations
import unittest

class ShoppingCart:
    def __init__(self) -> None:
        self._items: list[tuple[str, float]] = []

    def add_item(self, name: str, price: float) -> bool:
        if not isinstance(name, str) or not name:
            return False
        if not isinstance(price, (int, float)) or price < 0:
            return False

        self._items.append((name, float(price)))
        return True

    def remove_item(self, name: str) -> bool:
        if not isinstance(name, str) or not name:
            return False

        for index, (item_name, _) in enumerate(self._items):
            if item_name == name:
                del self._items[index]
                return True

        return False

```

```
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")

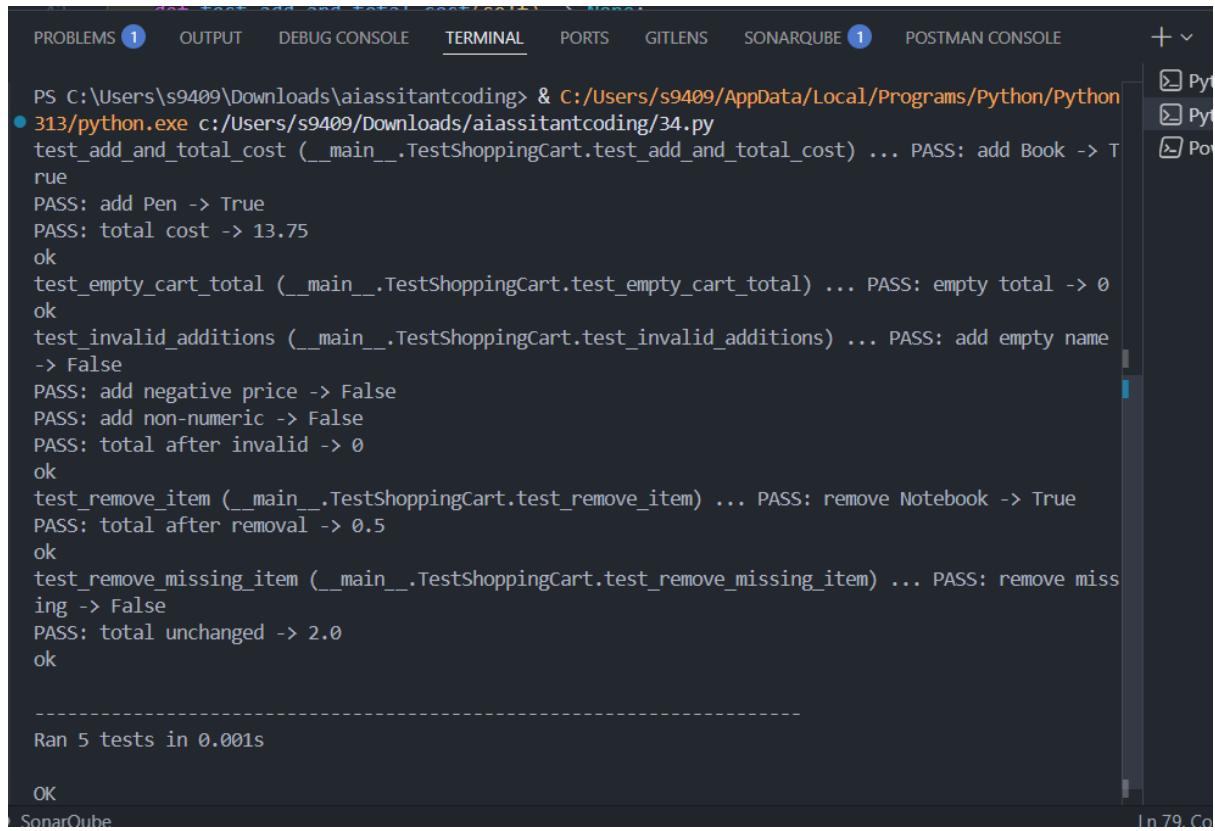
    def test_empty_cart_total(self) -> None:
        cart = ShoppingCart()
        self._assert_equal(cart.total_cost(), 0.0, "empty total")

    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)
```

Output:



```
PS C:\Users\s9409\Downloads\aiassistantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Python313/python.exe c:/Users/s9409/Downloads/aiassistantcoding/34.py
test_add_and_total_cost (__main__.TestShoppingCart.test_add_and_total_cost) ... PASS: add Book -> True
PASS: add Pen -> True
PASS: total cost -> 13.75
ok
test_empty_cart_total (__main__.TestShoppingCart.test_empty_cart_total) ... PASS: empty total -> 0
ok
test_invalid_additions (__main__.TestShoppingCart.test_invalid_additions) ... PASS: add empty name -> False
PASS: add negative price -> False
PASS: add non-numeric -> False
PASS: total after invalid -> 0
ok
test_remove_item (__main__.TestShoppingCart.test_remove_item) ... PASS: remove Notebook -> True
PASS: total after removal -> 0.5
ok
test_remove_missing_item (__main__.TestShoppingCart.test_remove_missing_item) ... PASS: remove missing -> False
PASS: total unchanged -> 2.0
ok

-----
Ran 5 tests in 0.001s

OK
```

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} -> {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),
        ]
```

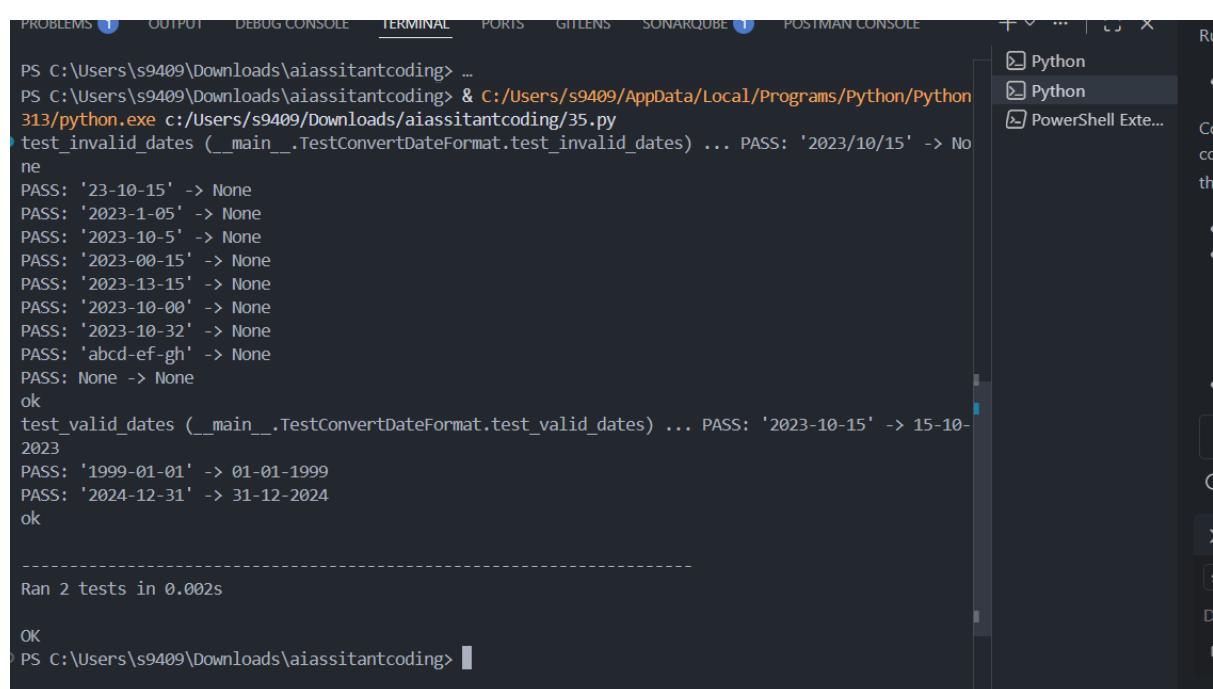
Keep Undo ⌂ 1 of 1 ↑

```
        ("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:



The screenshot shows a terminal window in VS Code displaying the output of a Python unit test. The test cases for invalid dates pass, showing 'None' as the result. The valid date test also passes, showing the correct transformation. The test runner summary at the bottom indicates 2 tests ran in 0.002s and the status 'OK'.

```
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' -> None
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>
```

Justification

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