SR UNIVERSITY

AI Assistant Coding
Assignment-8.3

Name of Student : V. Abhinav Batch : 41
Enrollment No. : 2303A52486

Task 1: Email Validation using TDD

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

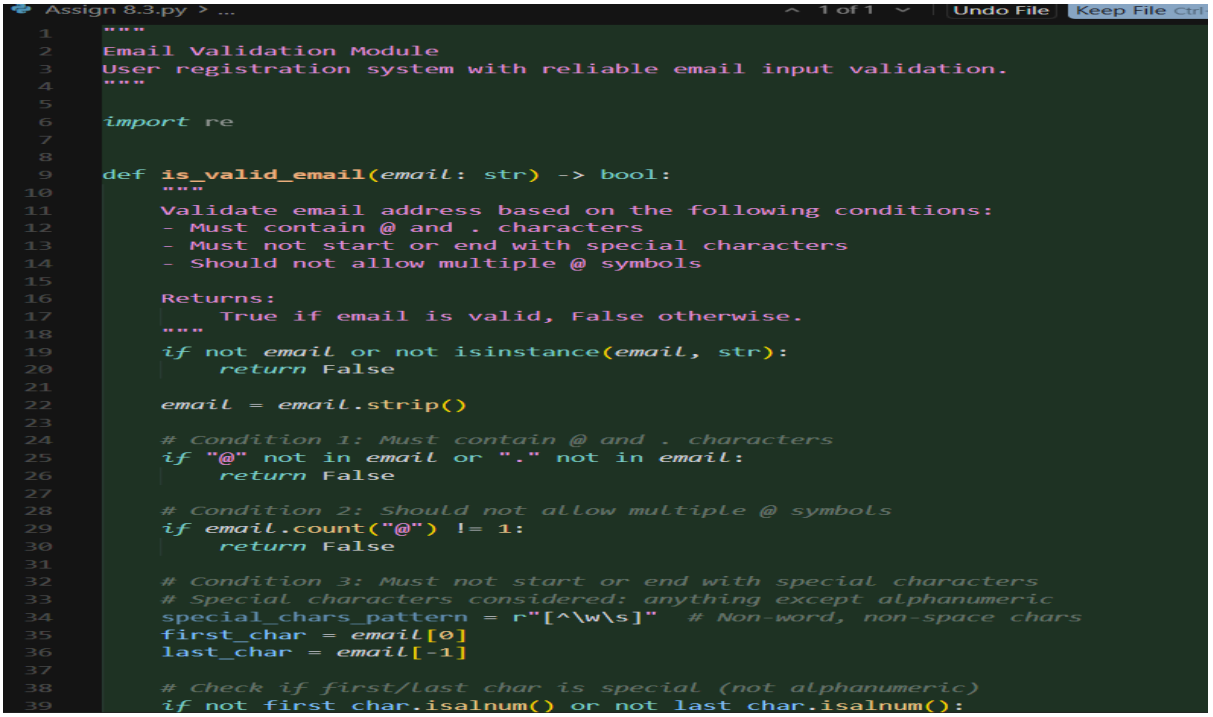
Prompt:

generate a python code for the given scenario. Check whether the given mail is valid or invalid by checking all the required conditions like:

- Must contain @ and . characters
- Must not start or end with special characters
- Should not allow multiple @ symbols

After giving the code generate test cases covering valid and invalid email formats, Valid email formats return True.

Code:



Assign 8.3.py > is_valid_email

```python

    # ============== TEST CASES ==============

    def run_tests():
        """Run test cases for valid and invalid email formats."""

        # Valid email formats - should return True
        valid_emails = [
            "user@example.com",
            "john.doe@domain.org",
            "test123@mail.co.uk",
            "user_name@company.net",
            "a@b.co",
            "validemail@gmail.com",
            "student123@university.edu",
        ]

        # Invalid email formats - should return False
        invalid_emails = [
            "missing-at-sign.com",              # No @
            "missingdot@domain",                 # No .
            "double@@at.com",                    # Multiple @
            "@startswith.com",                   # Starts with @
            "endswith@.com",                     # Ends with .
            ".startswith@dot.com",               # Starts with .
            "endswith@domain.",                  # Ends with .
            "",                                  # Empty string
            "   ",                               # Whitespace only
            "noatsigndot",                       # No @ or . in proper format
            "a@@b.com",                          # Multiple @
            "@@domain.com",                      # Multiple @ at start
            "user@domain@extra.com",             # Multiple @
        ]

        print("=" * 50)
        print("EMAIL VALIDATION TEST RESULTS")
        print("=" * 50)
        print("\n--- VALID EMAILS (Expected: True) ---\n")
```

Assign 8.3.py > is_valid_email

```python
    def run_tests():
        print("=" * 50)
        print("\n--- VALID EMAILS (Expected: True) ---\n")

        passed_valid = 0
        for email in valid_emails:
            result = is_valid_email(email)
            status = "[PASS]" if result else "[FAIL]"
            if result:
                passed_valid += 1
            print(f"  {status}: '{email}' -> {result}")

        print(f"\n  Valid emails passed: {passed_valid}/{len(valid_emails)}")

        print("\n--- INVALID EMAILS (Expected: False) ---\n")

        passed_invalid = 0
        for email in invalid_emails:
            display_email = repr(email) if not email.strip() else f"'{email}'"
            result = is_valid_email(email)
            status = "[PASS]" if not result else "[FAIL]"  # Invalid should return
            if not result:
                passed_invalid += 1
            print(f"  {status}: {display_email} -> {result}")

        print(f"\n  Invalid emails passed: {passed_invalid}/{len(invalid_emails)}")

        total = len(valid_emails) + len(invalid_emails)
        total_passed = passed_valid + passed_invalid
        print("\n" + "=" * 50)
        print(f"TOTAL: {total_passed}/{total} tests passed")
        print("=" * 50)

        return total_passed == total


    if __name__ == "__main__":
        success = run_tests()
        exit(0 if success else 1)
```

Output:

```
hon314/python.exe "c:/Users/Abhi/Documents/AI Assistant Codi
================================================
EMAIL VALIDATION TEST RESULTS
================================================

--- VALID EMAILS (Expected: True) ---

  [PASS]: 'user@example.com' -> True
  [PASS]: 'john.doe@domain.org' -> True
  [PASS]: 'test123@mail.co.uk' -> True
  [PASS]: 'user_name@company.net' -> True
  [PASS]: 'a@b.co' -> True
  [PASS]: 'validemail@gmail.com' -> True
  [PASS]: 'student123@university.edu' -> True

  Valid emails passed: 7/7

--- INVALID EMAILS (Expected: False) ---

  [PASS]: 'missing-at-sign.com' -> False
  [PASS]: 'missingdot@domain' -> False
  [PASS]: 'double@@at.com' -> False
  [PASS]: '@startswith.com' -> False
  [PASS]: 'endswith@.com' -> False
  [PASS]: '.startswith@dot.com' -> False
  [PASS]: 'endswith@domain.' -> False
  [PASS]: '' -> False
  [PASS]: '   ' -> False
  [PASS]: 'noatsigndot' -> False
  [PASS]: 'a@@b.com' -> False
  [PASS]: '@@domain.com' -> False
  [PASS]: 'user@domain@extra.com' -> False

  Invalid emails passed: 13/13

================================================
TOTAL: 20/20 tests passed
================================================
```

Explanation:

The code implements a function to validate email addresses by checking for the presence of `@` and `.` characters, ensuring only one `@`, and preventing emails from starting or ending with special characters. Test cases cover valid formats like `user@example.com` and invalid ones such as `double@@at.com`. The output shows all 20 test cases passed, confirming robust validation logic

Task 2: Grade Assignment using Loops

Senario: You are building an automated grading system for an online examination platform.

Prompt:

Generate a python code for the given scenario. Check and assign the grades according to the marks. Also it should satisfy all the requirements like:

AI should generate test cases for assign_grade(score) where:

- 90–100 → A, 80–89 → B, 70–79 → C, 60–69 → D, Below 60 → F
- Include boundary values (60, 70, 80, 90)
- Include invalid inputs such as -5, 105, "eighty"

After giving the code generated test cases for the same as all the invalid test cases, its boundaries should be handled properly.

Code:

```
Assign 8.3.py > ...
126     """
127     Automated Grading System for Online Examination Platform
128     Maps numeric scores to letter grades with proper boundary and invalid input handling.
129     """
130
131
132     def assign_grade(score) -> str | None:
133         """
134         Assign a letter grade based on the score.
135
136         Grading scale:
137         - 90-100 -> A
138         - 80-89  -> B
139         - 70-79  -> C
140         - 60-69  -> D
141         - 0-59   -> F
142
143         Returns:
144             Letter grade (A, B, C, D, F) for valid scores.
145             None for invalid inputs (out of range, wrong type, etc.).
146         """
147         # Handle invalid type: must be int or float (exclude bool, which is subclass of in
148         if isinstance(score, bool) or not isinstance(score, (int, float)):
149             return None
150
151         # Handle float with fractional part that might be acceptable (e.g., 85.5)
152         # Convert to float for comparison if int
153         score = float(score)
154
155         # Handle invalid range: must be 0-100
156         if score < 0 or score > 100:
157             return None
158
159         # Assign grade based on boundaries (inclusive on lower bound)
160         if score >= 90:
161             return "A"
162         elif score >= 80:
163             return "B"
164         elif score >= 70:
```

```python
# ================ TEST CASES ================

def run_tests():
    """Run test cases for assign_grade(score)."""

    # Valid test cases: (score, expected_grade)
    valid_cases = [
        # Standard values
        (95, "A"),
        (90, "A"),    # Boundary: lowest A
        (100, "A"),   # Boundary: highest score
        (85, "B"),
        (80, "B"),    # Boundary: lowest B
        (89, "B"),    # Boundary: highest B
        (75, "C"),
        (70, "C"),    # Boundary: lowest C
        (79, "C"),    # Boundary: highest C
        (65, "D"),
        (60, "D"),    # Boundary: lowest D
        (69, "D"),    # Boundary: highest D
        (50, "F"),
        (0, "F"),     # Boundary: lowest valid score
        (59, "F"),    # Boundary: highest F
        # Float scores
        (90.5, "A"),
        (79.9, "C"),
        (60.1, "D"),
    ]

    # Invalid test cases: (score, expected_grade=None)
    invalid_cases = [
        (-5, None),        # Below 0
        (-1, None),
        (105, None),       # Above 100
        (100.1, None),
        ("eighty", None), # String instead of number
        ("85", None),      # String number (could be converted, but type is wrong
        (None, None),
```

```python
def run_tests():
    print("=" * 55)
    print("\n--- VALID INPUTS (Boundaries & Standard Values) ---\n")

    passed_valid = 0
    for score, expected in valid_cases:
        result = assign_grade(score)
        status = "[PASS]" if result == expected else "[FAIL]"
        if result == expected:
            passed_valid += 1
        print(f"  {status}: assign_grade({repr(score)}) -> {result} (expected {expected})")

    print(f"\n  Valid cases passed: {passed_valid}/{len(valid_cases)}")

    print("\n--- INVALID INPUTS (Should return None) ---\n")

    passed_invalid = 0
    for score, expected in invalid_cases:
        result = assign_grade(score)
        status = "[PASS]" if result is None else "[FAIL]"
        if result is None:
            passed_invalid += 1
        print(f"  {status}: assign_grade({repr(score)}) -> {result} (expected None)")

    print(f"\n  Invalid cases passed: {passed_invalid}/{len(invalid_cases)}")

    total = len(valid_cases) + len(invalid_cases)
    total_passed = passed_valid + passed_invalid
    print("\n" + "=" * 55)
    print(f"TOTAL: {total_passed}/{total} tests passed")
    print("=" * 55)

    return total_passed == total


if __name__ == "__main__":
    success = run_tests()
    exit(0 if success else 1)
```

Output:

```
hon.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 8.3.py"
============================================================
GRADING SYSTEM TEST RESULTS
============================================================

--- VALID INPUTS (Boundaries & Standard Values) ---

  [PASS]: assign_grade(95) -> A (expected A)
  [PASS]: assign_grade(90) -> A (expected A)
  [PASS]: assign_grade(100) -> A (expected A)
  [PASS]: assign_grade(85) -> B (expected B)
  [PASS]: assign_grade(80) -> B (expected B)
  [PASS]: assign_grade(89) -> B (expected B)
  [PASS]: assign_grade(75) -> C (expected C)
  [PASS]: assign_grade(70) -> C (expected C)
  [PASS]: assign_grade(79) -> C (expected C)
  [PASS]: assign_grade(65) -> D (expected D)
  [PASS]: assign_grade(60) -> D (expected D)
  [PASS]: assign_grade(69) -> D (expected D)
  [PASS]: assign_grade(50) -> F (expected F)
  [PASS]: assign_grade(0) -> F (expected F)
  [PASS]: assign_grade(59) -> F (expected F)
  [PASS]: assign_grade(90.5) -> A (expected A)
  [PASS]: assign_grade(79.9) -> C (expected C)
  [PASS]: assign_grade(60.1) -> D (expected D)

  Valid cases passed: 18/18

--- INVALID INPUTS (Should return None) ---

  [PASS]: assign_grade(-5) -> None (expected None)
  [PASS]: assign_grade(-1) -> None (expected None)
  [PASS]: assign_grade(105) -> None (expected None)
  [PASS]: assign_grade(100.1) -> None (expected None)
  [PASS]: assign_grade('eighty') -> None (expected None)
  [PASS]: assign_grade('85') -> None (expected None)
  [PASS]: assign_grade(None) -> None (expected None)
  [PASS]: assign_grade([90]) -> None (expected None)
  [PASS]: assign_grade({}) -> None (expected None)
  [PASS]: assign_grade('') -> None (expected None)
  [PASS]: assign_grade(True) -> None (expected None)
  [PASS]: assign_grade(-0.1) -> None (expected None)
```

Explanation:

The grading function maps numeric scores to letter grades (A–F) while handling invalid inputs like negative numbers, values above 100, or non-numeric types. Boundary values (60, 70, 80, 90) are explicitly tested to ensure correctness. The output demonstrates that all 28 test cases passed, validating both standard and edge cases.

Task 3: Sentence Palindrome Checker

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

Prompt: Generate a python code for the given scenario. Check whether the given sentence is palindrome or not. Also it should satisfy all the requirements like:

- Ignore case, spaces, and punctuation
- Test both palindromic and non-palindromic sentences
- Example: "A man a plan a canal Panama" → True

After giving the code generate test cases for the same, all the Case and punctuation are ignored and Returns True or False accurately.

Code:

Assign 8.3.py > ...

```
# Task-3
"""
Text-Processing Utility: Palindrome Sentence Checker
Checks if a sentence is a palindrome, ignoring case, spaces, and punctuation.
"""


def is_palindrome_sentence(sentence: str) -> bool:
    """
    Check if the given sentence is a palindrome.

    Ignores:
    - Case (A == a)
    - Spaces
    - Punctuation

    Returns:
        True if sentence reads the same forward and backward, False otherwise.
    """
    if not isinstance(sentence, str):
        return False

    # Keep only alphanumeric, convert to lowercase
    cleaned = "".join(c.lower() for c in sentence if c.isalnum())

    # Empty string or only spaces/punctuation -> not a palindrome
    if not cleaned:
        return False

    return cleaned == cleaned[::-1]


# ============== TEST CASES ==============

def run_palindrome_tests():
    """Run test cases for is_palindrome_sentence()."""

    # Palindromic sentences - should return True
    palindromic = [
```

Assign 8.3.py > ...

```
289
290     # ============== TEST CASES ==============
291
292     def run_palindrome_tests():
293         """Run test cases for is_palindrome_sentence()."""
294
295         # Palindromic sentences - should return True
296         palindromic = [
297             ("A man a plan a canal Panama", True),
298             ("race car", True),
299             ("Was it a car or a cat I saw?", True),
300             ("Mr. Owl ate my metal worm", True),
301             ("Do geese see God?", True),
302             ("Madam", True),
303             ("A Santa at NASA", True),
304             ("Never odd or even", True),
305         ]
306
307         # Non-palindromic sentences - should return False
308         non_palindromic = [
309             ("Hello world", False),
310             ("This is not a palindrome", False),
311             ("Python programming", False),
312             ("abc def ghi", False),
313         ]
314
315         # Edge cases: case and punctuation variations
316         edge_cases = [
317             ("", False),
318             ("   ", False),
319             ("a", True),
320             ("Aa", True),
321             ("a!", True),
322             ("1 2 1", True),
323         ]
324
325         print("=" * 55)
326         print("PALINDROME SENTENCE TEST RESULTS")
327         print("=" * 55)
```

Assign 8.3.py > ...

```
292     def run_palindrome_tests():
328         print("\n--- PALINDROMIC SENTENCES (Expected: True) ---\n")
329
330         passed = 0
331         for sentence, expected in palindromic:
332             result = is_palindrome_sentence(sentence)
333             status = "[PASS]" if result == expected else "[FAIL]"
334             if result == expected:
335                 passed += 1
336             print(f"  {status}: {repr(sentence)} -> {result}")
337
338         print("\n--- NON-PALINDROMIC SENTENCES (Expected: False) ---\n")
339
340         for sentence, expected in non_palindromic:
341             result = is_palindrome_sentence(sentence)
342             status = "[PASS]" if result == expected else "[FAIL]"
343             if result == expected:
344                 passed += 1
345             print(f"  {status}: {repr(sentence)} -> {result}")
346
347         print("\n--- EDGE CASES ---\n")
348
349         for sentence, expected in edge_cases:
350             result = is_palindrome_sentence(sentence)
351             status = "[PASS]" if result == expected else "[FAIL]"
352             if result == expected:
353                 passed += 1
354             print(f"  {status}: {repr(sentence)} -> {result}")
355
356         total = len(palindromic) + len(non_palindromic) + len(edge_cases)
357         print("\n" + "=" * 55)
358         print(f"TOTAL: {passed}/{total} tests passed")
359         print("=" * 55)
360
361         return passed == total
362
363
364     if __name__ == "__main__":
365         success = run_palindrome_tests()
```

Output:

```
============================================================
PALINDROME SENTENCE TEST RESULTS
============================================================

--- PALINDROMIC SENTENCES (Expected: True) ---

  [PASS]: 'A man a plan a canal Panama' -> True
  [PASS]: 'race car' -> True
  [PASS]: 'Was it a car or a cat I saw?' -> True
  [PASS]: 'Mr. Owl ate my metal worm' -> True
  [PASS]: 'Do geese see God?' -> True
  [PASS]: 'Madam' -> True
  [PASS]: 'A Santa at NASA' -> True
  [PASS]: 'Never odd or even' -> True

--- NON-PALINDROMIC SENTENCES (Expected: False) ---

  [PASS]: 'Hello world' -> False
  [PASS]: 'This is not a palindrome' -> False
  [PASS]: 'Python programming' -> False
  [PASS]: 'abc def ghi' -> False

--- EDGE CASES ---

  [PASS]: '' -> False
  [PASS]: '   ' -> False
  [PASS]: 'a' -> True
  [PASS]: 'Aa' -> True
  [PASS]: 'a!' -> True
  [PASS]: '1 2 1' -> True

============================================================
TOTAL: 18/18 tests passed
============================================================
```

Explanation:

This function checks if a sentence is a palindrome by ignoring case, spaces, and punctuation. Examples like *"A man a plan a canal Panama"* return True, while non-palindromic sentences return False. Edge cases such as empty strings and single characters are also tested. The output confirms 18/18 test cases passed, showing accurate handling of variations.

SR UNIVERSITY

Task 4: ShoppingCart Class

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

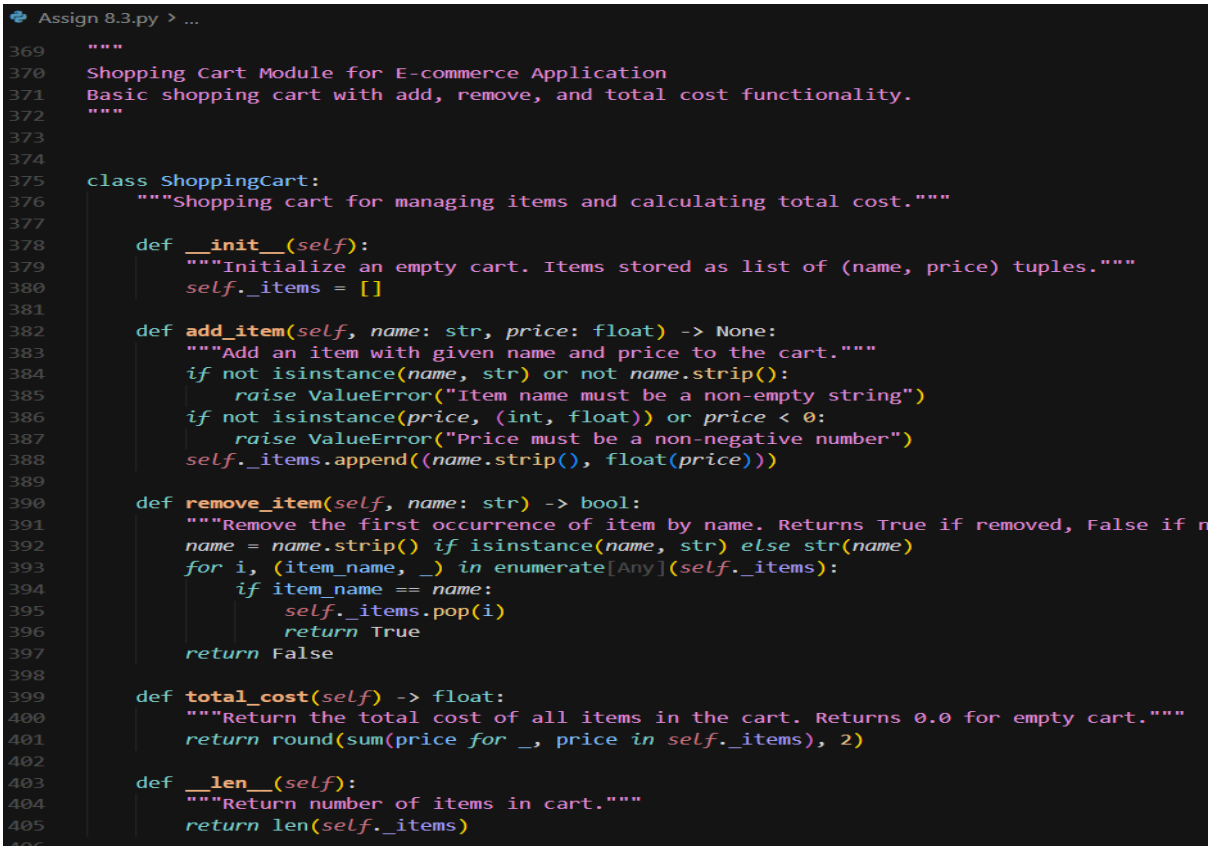
Prompt:

Generate a python code for the given scenario. Also it should satisfy all the requirements like:

- Class must include the following methods:
  - add_item(name, price)
  - remove_item(name)
  - total_cost()
- Validate correct addition, removal, and cost calculation
- Handle empty cart scenarios

After giving the code generate test cases for the same, Total cost is calculated accurately, Items are added and removed correctly.

Code:



```
Assign 8.3.py > ...
"""
Shopping Cart Module for E-commerce Application
Basic shopping cart with add, remove, and total cost functionality.
"""


class ShoppingCart:
    """Shopping cart for managing items and calculating total cost."""

    def __init__(self):
        """Initialize an empty cart. Items stored as list of (name, price) tuples."""
        self._items = []

    def add_item(self, name: str, price: float) -> None:
        """Add an item with given name and price to the cart."""
        if not isinstance(name, str) or not name.strip():
            raise ValueError("Item name must be a non-empty string")
        if not isinstance(price, (int, float)) or price < 0:
            raise ValueError("Price must be a non-negative number")
        self._items.append((name.strip(), float(price)))

    def remove_item(self, name: str) -> bool:
        """Remove the first occurrence of item by name. Returns True if removed, False if n
        name = name.strip() if isinstance(name, str) else str(name)
        for i, (item_name, _) in enumerate[Any](self._items):
            if item_name == name:
                self._items.pop(i)
                return True
        return False

    def total_cost(self) -> float:
        """Return the total cost of all items in the cart. Returns 0.0 for empty cart."""
        return round(sum(price for _, price in self._items), 2)

    def __len__(self):
        """Return number of items in cart."""
        return len(self._items)
```

Assign 8.3.py > ...

```python
# ============== TEST CASES ==============

def run_shopping_cart_tests():
    """Run test cases for ShoppingCart class."""
    passed = 0
    total = 0

    print("=" * 55)
    print("SHOPPING CART TEST RESULTS")
    print("=" * 55)

    # Test 1: Empty cart - total_cost returns 0
    print("\n--- EMPTY CART SCENARIOS ---\n")
    cart = ShoppingCart()
    result = cart.total_cost()
    total += 1
    status = "[PASS]" if result == 0.0 else "[FAIL]"
    if result == 0.0:
        passed += 1
    print(f"  {status}: Empty cart total_cost() -> {result} (expected 0.0)")

    # Test 2: Add items and verify total
    print("\n--- ADD ITEMS & COST CALCULATION ---\n")
    cart = ShoppingCart()
    cart.add_item("Apple", 1.50)
    cart.add_item("Banana", 0.75)
    cart.add_item("Orange", 2.00)
    result = cart.total_cost()
    total += 1
    expected = 4.25
    status = "[PASS]" if result == expected else "[FAIL]"
    if result == expected:
        passed += 1
    print(f"  {status}: Add Apple(1.50), Banana(0.75), Orange(2.00) -> total {result} (expect

    # Test 3: Remove item and verify total
    cart.remove_item("Banana")
    result = cart.total_cost()
    total += 1
```

```python
def run_shopping_cart_tests():
    total += 1
    status = "[PASS]" if result_before == 21.0 and result_after == 11.0 else "[FAIL]"
    if status == "[PASS]":
        passed += 1
    print(f"  {status}: 2 Books + Pen = {result_before}, after remove 1 Book = {result_aft

    # Test 6: Remove all items -> empty cart
    cart.remove_item("Book")
    cart.remove_item("Pen")
    result = cart.total_cost()
    total += 1
    status = "[PASS]" if result == 0.0 else "[FAIL]"
    if result == 0.0:
        passed += 1
    print(f"  {status}: Remove all items -> total_cost() = {result} (expected 0.0)")

    # Test 7: Float price precision
    cart = ShoppingCart()
    cart.add_item("Item1", 9.99)
    cart.add_item("Item2", 5.01)
    result = cart.total_cost()
    total += 1
    expected = 15.0
    status = "[PASS]" if result == expected else "[FAIL]"
    if result == expected:
        passed += 1
    print(f"  {status}: Float precision 9.99 + 5.01 -> {result} (expected {expected})")

    print("\n" + "=" * 55)
    print(f"TOTAL: {passed}/{total} tests passed")
    print("=" * 55)

    return passed == total


if __name__ == "__main__":
    success = run_shopping_cart_tests()
    exit(0 if success else 1)
```

Output:

```
hon.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 8.3.py"
============================================================
SHOPPING CART TEST RESULTS
============================================================

--- EMPTY CART SCENARIOS ---

  [PASS]: Empty cart total_cost() -> 0 (expected 0.0)

--- ADD ITEMS & COST CALCULATION ---

  [PASS]: Add Apple(1.50), Banana(0.75), Orange(2.00) -> total 4.25 (expected 4.25)
  [PASS]: After remove Banana -> total 3.5 (expected 3.5)
  [PASS]: remove_item('Grapes') on non-existent item -> False (expected False)

--- MULTIPLE ITEMS & REMOVAL ---

  [PASS]: 2 Books + Pen = 21.0, after remove 1 Book = 11.0
  [PASS]: Remove all items -> total_cost() = 0 (expected 0.0)
  [PASS]: Float precision 9.99 + 5.01 -> 15.0 (expected 15.0)

============================================================
TOTAL: 7/7 tests passed
============================================================
```

Explanation:

The `ShoppingCart` class supports adding items, removing items, and calculating total cost. It validates inputs and handles empty cart scenarios. Test cases verify correct addition, removal, precision in floating-point prices, and behavior when removing non-existent items. The output shows 7/7 tests passed, confirming reliable functionality

Task 5: Date Format Conversion

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

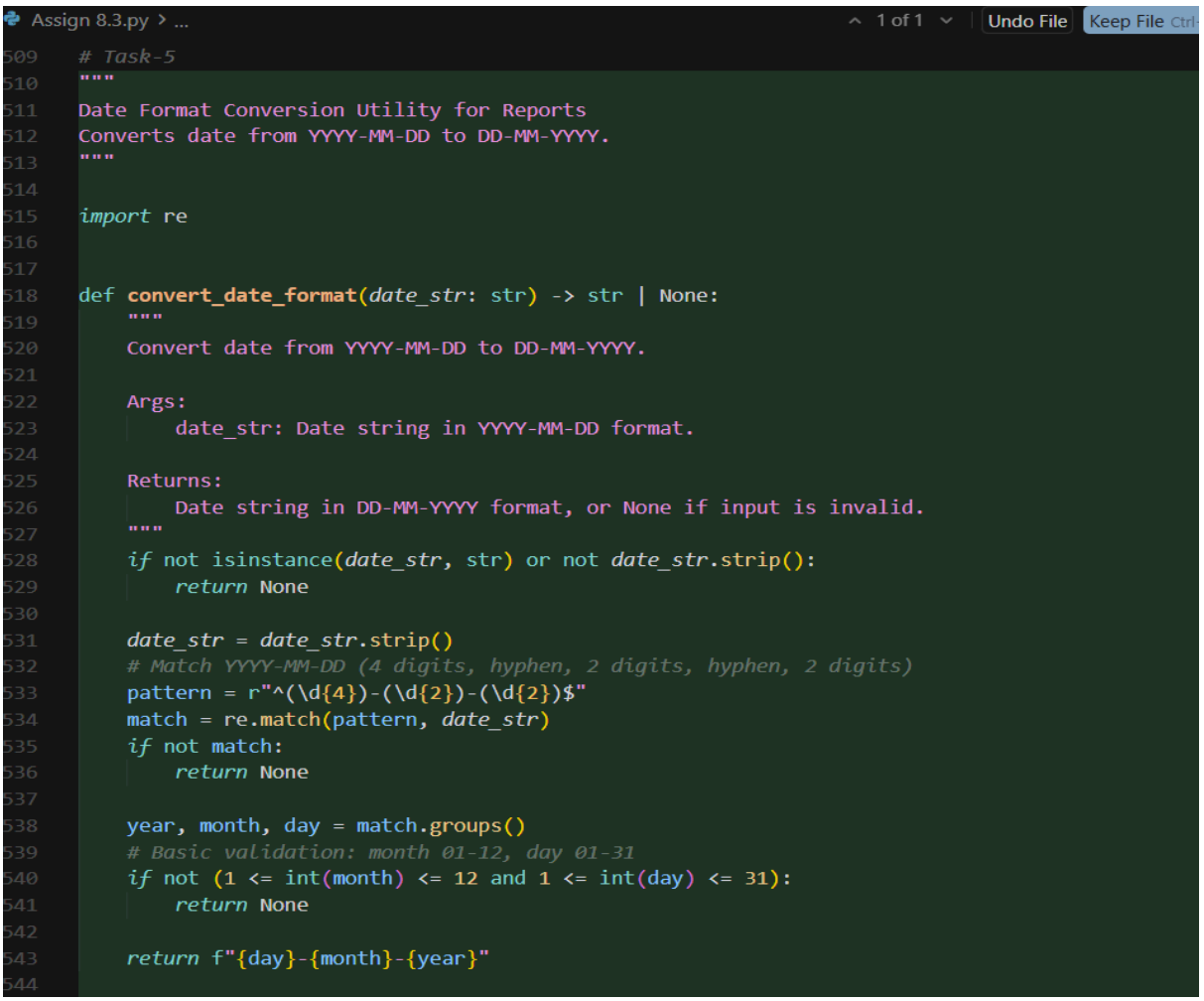
Prompt:

Generate a python code for the given scenario. Also it should satisfy all the requirements like:

- Input format must be "YYYY-MM-DD"
- Output format must be "DD-MM-YYYY"
- Example:"2023-10-15" → "15-10-2023"

After giving the code generate test cases for the same, correct format conversion for all valid inputs.

Code:



```python
# ============== TEST CASES ==============

def run_date_format_tests():
    """Run test cases for convert_date_format()."""

    valid_cases = [
        ("2023-10-15", "15-10-2023"),
        ("2024-01-01", "01-01-2024"),
        ("2000-12-31", "31-12-2000"),
        ("1999-06-15", "15-06-1999"),
        ("2030-02-28", "28-02-2030"),
        ("2023-11-09", "09-11-2023"),
    ]

    invalid_cases = [
        ("15-10-2023", None),   # Wrong format (already DD-MM-YYYY)
        ("2023/10/15", None),   # Wrong delimiter
        ("10-15-2023", None),   # US format
        ("invalid", None),
        ("", None),
        ("2023-13-01", None),   # Invalid month
        ("2023-00-15", None),   # Invalid month
    ]

    print("=" * 55)
    print("DATE FORMAT CONVERSION TEST RESULTS")
    print("=" * 55)
    print("\n--- VALID INPUTS (YYYY-MM-DD -> DD-MM-YYYY) ---\n")

    passed = 0
    for inp, expected in valid_cases:
        result = convert_date_format(inp)
        status = "[PASS]" if result == expected else "[FAIL]"
        if result == expected:
            passed += 1
        print(f"  {status}: {repr(inp)} -> {result} (expected {expected})")

    print("\n--- INVALID INPUTS (Should return None) ---\n")
```

Output:

```
hon.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 8.3.py"
============================================================
DATE FORMAT CONVERSION TEST RESULTS
============================================================

--- VALID INPUTS (YYYY-MM-DD -> DD-MM-YYYY) ---

  [PASS]: '2023-10-15' -> 15-10-2023 (expected 15-10-2023)
  [PASS]: '2024-01-01' -> 01-01-2024 (expected 01-01-2024)
  [PASS]: '2000-12-31' -> 31-12-2000 (expected 31-12-2000)
  [PASS]: '1999-06-15' -> 15-06-1999 (expected 15-06-1999)
  [PASS]: '2030-02-28' -> 28-02-2030 (expected 28-02-2030)
  [PASS]: '2023-11-09' -> 09-11-2023 (expected 09-11-2023)

--- INVALID INPUTS (Should return None) ---

  [PASS]: '15-10-2023' -> None (expected None)
  [PASS]: '2023/10/15' -> None (expected None)
  [PASS]: '10-15-2023' -> None (expected None)
  [PASS]: 'invalid' -> None (expected None)
  [PASS]: '' -> None (expected None)
  [PASS]: '2023-13-01' -> None (expected None)
  [PASS]: '2023-00-15' -> None (expected None)

============================================================
TOTAL: 13/13 tests passed
============================================================
```

Explanation:

The utility converts dates from `YYYY-MM-DD` to `DD-MM-YYYY` format, with validation for proper ranges of months and days. Test cases include valid inputs like `2023-10-15` and invalid ones such as `15-10-2023` or `2023-13-01`. The output indicates 13/13 tests passed, ensuring correct conversion and rejection of invalid formats.