

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

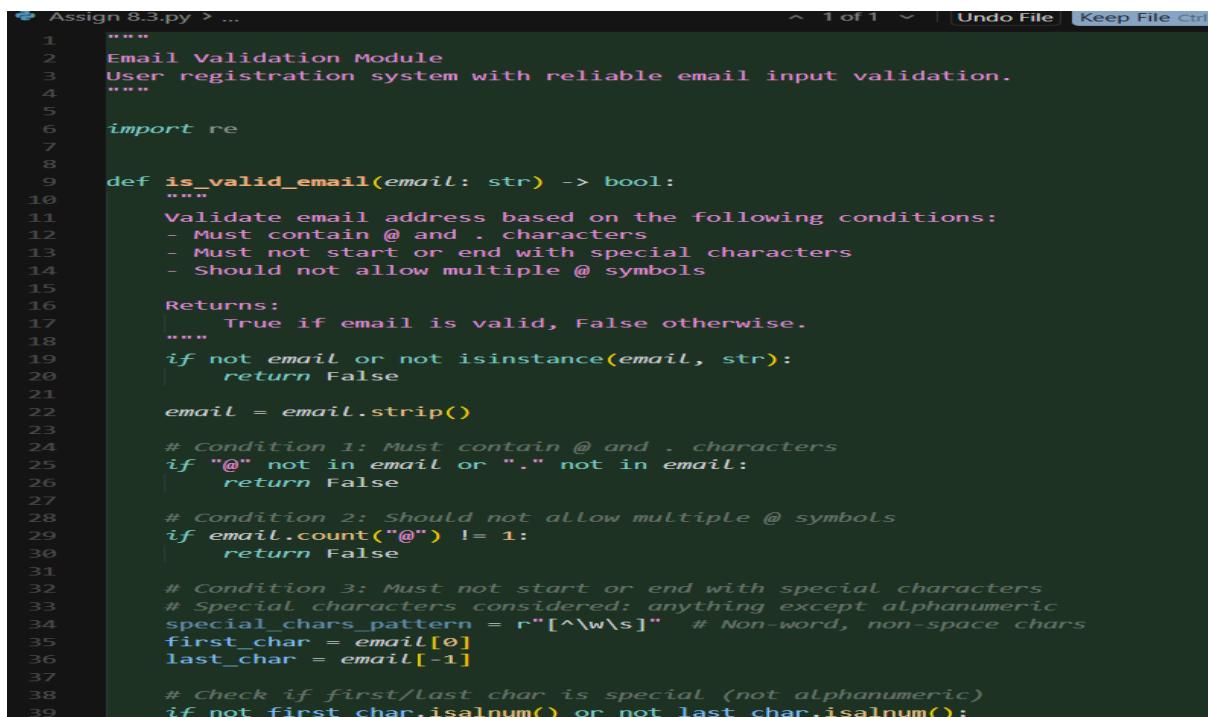
Prompt:

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:



The screenshot shows a code editor window titled "Assign 8.3.py > ...". The code is a Python script for validating emails. It starts with a multi-line docstring explaining the module's purpose and the conditions for validation. The `is_valid_email` function is defined, which first checks if the input is a string and strips whitespace. It then checks for the presence of both '@' and '.' characters. Finally, it checks if there is more than one '@' symbol. The code uses comments to explain each condition. The entire script is numbered from 1 to 39.

```
1  """
2  Email Validation Module
3  User registration system with reliable email input validation.
4  """
5
6  import re
7
8
9  def is_valid_email(email: str) -> bool:
10     """
11         Validate email address based on the following conditions:
12         - Must contain @ and . characters
13         - Must not start or end with special characters
14         - Should not allow multiple @ symbols
15
16     Returns:
17         True if email is valid, False otherwise.
18     """
19     if not email or not isinstance(email, str):
20         return False
21
22     email = email.strip()
23
24     # Condition 1: Must contain @ and . characters
25     if "@" not in email or "." not in email:
26         return False
27
28     # Condition 2: Should not allow multiple @ symbols
29     if email.count "@" != 1:
30         return False
31
32     # Condition 3: Must not start or end with special characters
33     # Special characters considered: anything except alphanumeric
34     special_chars_pattern = r"[\^w\$\s]" # Non-word, non-space chars
35     first_char = email[0]
36     last_char = email[-1]
37
38     # Check if first/last char is special (not alphanumeric)
39     if not first_char.isalnum() or not last_char.isalnum():
```

```

Assign 8.3.py > is_valid_email
51
52     # ===== TEST CASES =====
53
54 def run_tests():
55     """Run test cases for valid and invalid email formats."""
56
57     # Valid email formats - should return True
58     valid_emails = [
59         "user@example.com",
60         "john.doe@domain.org",
61         "test123@mail.co.uk",
62         "user_name@company.net",
63         "a@b.co",
64         "validemail@gmail.com",
65         "student123@university.edu",
66     ]
67
68     # Invalid email formats - should return False
69     invalid_emails = [
70         "missing-at-sign.com",                      # No @
71         "missingdot@domain",                        # No .
72         "double@@at.com",                           # Multiple @
73         "@startswith.com",                          # Starts with @
74         "endswith@.com",                            # Ends with .
75         ".startswith@dot.com",                     # Starts with .
76         "endswith@domain.",                         # Ends with .
77         "",                                      # Empty string
78         " ",                                       # Whitespace only
79         "noatsigndot",                            # No @ or . in proper format
80         "a@@b.com",                                # Multiple @
81         "@@domain.com",                            # Multiple @ at start
82         "user@domain@extra.com",                   # Multiple @
83     ]
84
85     print("=" * 50)
86     print("EMAIL VALIDATION TEST RESULTS")
87     print("=" * 50)
88     print("\n--- VALID EMAILS (Expected: True) ---\n")

```

```

Assign 8.3.py > is_valid_email
54 def run_tests():
55     print("=" * 50)
56     print("\n--- VALID EMAILS (Expected: True) ---\n")
57
58     passed_valid = 0
59     for email in valid_emails:
60         result = is_valid_email(email)
61         status = "[PASS]" if result else "[FAIL]"
62         if result:
63             passed_valid += 1
64         print(f" {status}: '{email}' -> {result}")
65
66     print(f"\n  Valid emails passed: {passed_valid}/{len(valid_emails)}")
67
68     print("\n--- INVALID EMAILS (Expected: False) ---\n")
69
70     passed_invalid = 0
71     for email in invalid_emails:
72         display_email = repr(email) if not email.strip() else f"'{email}'"
73         result = is_valid_email(email)
74         status = "[PASS]" if not result else "[FAIL]" # Invalid should return False
75         if not result:
76             passed_invalid += 1
77         print(f" {status}: {display_email} -> {result}")
78
79     print(f"\n  Invalid emails passed: {passed_invalid}/{len(invalid_emails)}")
80
81     total = len(valid_emails) + len(invalid_emails)
82     total_passed = passed_valid + passed_invalid
83     print("\n" + "=" * 50)
84     print(f"TOTAL: {total_passed}/{total} tests passed")
85     print("=" * 50)
86
87     return total_passed == total
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124 if __name__ == "__main__":
125     success = run_tests()
126     exit(0 if success else 1)

```

Output:

```
hon314/python.exe "c:/Users/Abhi/Documents/AI Assistant Cod
=====
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

Scenario: 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     """
128     Automated Grading System for Online Examination Platform
129     Maps numeric scores to letter grades with proper boundary and invalid input handling.
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 int)
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:
```

```

72     # ===== TEST CASES =====
73
74     def run_tests():
75         """Run test cases for assign_grade(score)."""
76
77         # Valid test cases: (score, expected_grade)
78         valid_cases = [
79             # Standard values
80             (95, "A"),
81             (90, "A"),    # Boundary: lowest A
82             (100, "A"),   # Boundary: highest score
83             (85, "B"),
84             (80, "B"),    # Boundary: lowest B
85             (89, "B"),    # Boundary: highest B
86             (75, "C"),
87             (70, "C"),    # Boundary: lowest C
88             (79, "C"),    # Boundary: highest C
89             (65, "D"),
90             (60, "D"),    # Boundary: lowest D
91             (69, "D"),    # Boundary: highest D
92             (50, "F"),
93             (0, "F"),     # Boundary: Lowest valid score
94             (59, "F"),    # Boundary: highest F
95             # Float scores
96             (90.5, "A"),
97             (79.9, "C"),
98             (60.1, "D"),
99         ]
100
101         # Invalid test cases: (score, expected_grade=None)
102         invalid_cases = [
103             (-5, None),      # Below 0
104             (-1, None),
105             (105, None),     # Above 100
106             (100.1, None),
107             ("eighty", None), # String instead of number
108             ("85", None),    # String number (could be converted, but type is wrong
109             (None, None),
110
111             def run_tests():
112                 PRINT STANDARD SYSTEM TEST RESULTS
113                 print("=" * 55)
114                 print("\n--- VALID INPUTS (Boundaries & Standard Values) ---\n")
115
116                 passed_valid = 0
117                 for score, expected in valid_cases:
118                     result = assign_grade(score)
119                     status = "[PASS]" if result == expected else "[FAIL]"
120                     if result == expected:
121                         passed_valid += 1
122                     print(f" {status}: assign_grade({repr(score)}) -> {result} (expected {expected})")
123
124                     print(f"\n  Valid cases passed: {passed_valid}/{len(valid_cases)}")
125
126                     print("\n--- INVALID INPUTS (Should return None) ---\n")
127
128                     passed_invalid = 0
129                     for score, expected in invalid_cases:
130                         result = assign_grade(score)
131                         status = "[PASS]" if result is None else "[FAIL]"
132                         if result is None:
133                             passed_invalid += 1
134                         print(f" {status}: assign_grade({repr(score)}) -> {result} (expected None)")
135
136                     print(f"\n  Invalid cases passed: {passed_invalid}/{len(invalid_cases)}")
137
138                     total = len(valid_cases) + len(invalid_cases)
139                     total_passed = passed_valid + passed_invalid
140                     print("\n" + "=" * 55)
141                     print(f"TOTAL: {total_passed}/{total} tests passed")
142                     print("=" * 55)
143
144                     return total_passed == total
145
146
147                     if __name__ == "__main__":
148                         success = run_tests()
149                         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(-9.1) -> 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 > ...
258     # Task-3
259     """
260     Text-Processing Utility: Palindrome Sentence Checker
261     Checks if a sentence is a palindrome, ignoring case, spaces, and punctuation.
262     """
263
264
265     def is_palindrome_sentence(sentence: str) -> bool:
266         """
267             Check if the given sentence is a palindrome.
268
269             Ignores:
270             - Case (A == a)
271             - Spaces
272             - Punctuation
273
274             Returns:
275             |   True if sentence reads the same forward and backward, False otherwise.
276             """
277         if not isinstance(sentence, str):
278             return False
279
280         # Keep only alphanumeric, convert to lowercase
281         cleaned = "".join(c.lower() for c in sentence if c.isalnum())
282
283         # Empty string or only spaces/punctuation -> not a palindrome
284         if not cleaned:
285             return False
286
287         return cleaned == cleaned[::-1]
288
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 = [
```

```

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():
293         print("\n--- PALINDROMIC SENTENCES (Expected: True) ---\n")
294
295         passed = 0
296         for sentence, expected in palindromic:
297             result = is_palindrome_sentence(sentence)
298             status = "[PASS]" if result == expected else "[FAIL]"
299             if result == expected:
300                 passed += 1
301             print(f" {status}: {repr(sentence)} -> {result}")
302
303         print("\n--- NON-PALINDROMIC SENTENCES (Expected: False) ---\n")
304
305         for sentence, expected in non_palindromic:
306             result = is_palindrome_sentence(sentence)
307             status = "[PASS]" if result == expected else "[FAIL]"
308             if result == expected:
309                 passed += 1
310             print(f" {status}: {repr(sentence)} -> {result}")
311
312         print("\n--- EDGE CASES ---\n")
313
314         for sentence, expected in edge_cases:
315             result = is_palindrome_sentence(sentence)
316             status = "[PASS]" if result == expected else "[FAIL]"
317             if result == expected:
318                 passed += 1
319             print(f" {status}: {repr(sentence)} -> {result}")
320
321         total = len(palindromic) + len(non_palindromic) + len(edge_cases)
322         print("\n" + "=" * 55)
323         print(f"TOTAL: {passed}/{total} tests passed")
324         print("=" * 55)
325
326         return passed == total
327
328
329     if __name__ == "__main__":
330         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.

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 > ...
369 """
370     Shopping Cart Module for E-commerce Application
371     Basic shopping cart with add, remove, and total cost functionality.
372 """
373
374
375 class ShoppingCart:
376     """Shopping cart for managing items and calculating total cost."""
377
378     def __init__(self):
379         """Initialize an empty cart. Items stored as list of (name, price) tuples."""
380         self._items = []
381
382     def add_item(self, name: str, price: float) -> None:
383         """Add an item with given name and price to the cart."""
384         if not isinstance(name, str) or not name.strip():
385             raise ValueError("Item name must be a non-empty string")
386         if not isinstance(price, (int, float)) or price < 0:
387             raise ValueError("Price must be a non-negative number")
388         self._items.append((name.strip(), float(price)))
389
390     def remove_item(self, name: str) -> bool:
391         """Remove the first occurrence of item by name. Returns True if removed, False if not found."""
392         name = name.strip() if isinstance(name, str) else str(name)
393         for i, (item_name, _) in enumerate(self._items):
394             if item_name == name:
395                 self._items.pop(i)
396                 return True
397         return False
398
399     def total_cost(self) -> float:
400         """Return the total cost of all items in the cart. Returns 0.0 for empty cart."""
401         return round(sum(price for _, price in self._items), 2)
402
403     def __len__(self):
404         """Return number of items in cart."""
405         return len(self._items)
```

```

Assign 8.3.py > ...

408     # ===== TEST CASES =====
409
410 def run_shopping_cart_tests():
411     """Run test cases for ShoppingCart class."""
412     passed = 0
413     total = 0
414
415     print("=" * 55)
416     print("SHOPPING CART TEST RESULTS")
417     print("=" * 55)
418
419     # Test 1: Empty cart - total_cost returns 0
420     print("\n--- EMPTY CART SCENARIOS ---\n")
421     cart = ShoppingCart()
422     result = cart.total_cost()
423     total += 1
424     status = "[PASS]" if result == 0.0 else "[FAIL]"
425     if result == 0.0:
426         passed += 1
427     print(f" {status}: Empty cart total_cost() -> {result} (expected 0.0)")
428
429     # Test 2: Add items and verify total
430     print("\n--- ADD ITEMS & COST CALCULATION ---\n")
431     cart = ShoppingCart()
432     cart.add_item("Apple", 1.50)
433     cart.add_item("Banana", 0.75)
434     cart.add_item("Orange", 2.00)
435     result = cart.total_cost()
436     total += 1
437     expected = 4.25
438     status = "[PASS]" if result == expected else "[FAIL]"
439     if result == expected:
440         passed += 1
441     print(f" {status}: Add Apple(1.50), Banana(0.75), Orange(2.00) -> total {result} (expected {expected})")
442
443     # Test 3: Remove item and verify total
444     cart.remove_item("Banana")
445     result = cart.total_cost()
446     total += 1

410 def run_shopping_cart_tests():
411     total += 1
412     status = "[PASS]" if result_before == 21.0 and result_after == 11.0 else "[FAIL]"
413     if status == "[PASS]":
414         passed += 1
415     print(f" {status}: 2 Books + Pen = {result_before}, after remove 1 Book = {result_after}")
416
417     # Test 6: Remove all items -> empty cart
418     cart.remove_item("Book")
419     cart.remove_item("Pen")
420     result = cart.total_cost()
421     total += 1
422     status = "[PASS]" if result == 0.0 else "[FAIL]"
423     if result == 0.0:
424         passed += 1
425     print(f" {status}: Remove all items -> total_cost() = {result} (expected 0.0)")

426     # Test 7: Float price precision
427     cart = ShoppingCart()
428     cart.add_item("Item1", 9.99)
429     cart.add_item("Item2", 5.01)
430     result = cart.total_cost()
431     total += 1
432     expected = 15.0
433     status = "[PASS]" if result == expected else "[FAIL]"
434     if result == expected:
435         passed += 1
436     print(f" {status}: Float precision 9.99 + 5.01 -> {result} (expected {expected})")

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

440     return passed == total

441
442
443 if __name__ == "__main__":
444     success = run_shopping_cart_tests()
445     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:

```
# Assign 8.3.py > ...
# Task-5
"""
Date Format Conversion Utility for Reports
Converts date from YYYY-MM-DD to DD-MM-YYYY.
"""

import re

def convert_date_format(date_str: str) -> str | None:
    """
    Convert date from YYYY-MM-DD to DD-MM-YYYY.

    Args:
        date_str: Date string in YYYY-MM-DD format.

    Returns:
        Date string in DD-MM-YYYY format, or None if input is invalid.
    """
    if not isinstance(date_str, str) or not date_str.strip():
        return None

    date_str = date_str.strip()
    # Match YYYY-MM-DD (4 digits, hyphen, 2 digits, hyphen, 2 digits)
    pattern = r"^\d{4}-\d{2}-\d{2}$"
    match = re.match(pattern, date_str)
    if not match:
        return None

    year, month, day = match.groups()
    # Basic validation: month 01-12, day 01-31
    if not (1 <= int(month) <= 12 and 1 <= int(day) <= 31):
        return None

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

```
Assignment.py  ...
546     # ===== TEST CASES =====
547
548 def run_date_format_tests():
549     """Run test cases for convert_date_format()."""
550
551     valid_cases = [
552         ("2023-10-15", "15-10-2023"),
553         ("2024-01-01", "01-01-2024"),
554         ("2000-12-31", "31-12-2000"),
555         ("1999-06-15", "15-06-1999"),
556         ("2030-02-28", "28-02-2030"),
557         ("2023-11-09", "09-11-2023"),
558     ]
559
560     invalid_cases = [
561         ("15-10-2023", None),    # Wrong format (already DD-MM-YYYY)
562         ("2023/10/15", None),   # Wrong delimiter
563         ("10-15-2023", None),  # US format
564         ("invalid", None),
565         ("", None),
566         ("2023-13-01", None),  # Invalid month
567         ("2023-00-15", None),  # Invalid month
568     ]
569
570     print("=" * 55)
571     print("DATE FORMAT CONVERSION TEST RESULTS")
572     print("=" * 55)
573     print("\n--- VALID INPUTS (YYYY-MM-DD -> DD-MM-YYYY) ---\n")
574
575     passed = 0
576     for inp, expected in valid_cases:
577         result = convert_date_format(inp)
578         status = "[PASS]" if result == expected else "[FAIL]"
579         if result == expected:
580             passed += 1
581         print(f" {status}: {repr(inp)} -> {result} (expected {expected})")
582
583     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.