

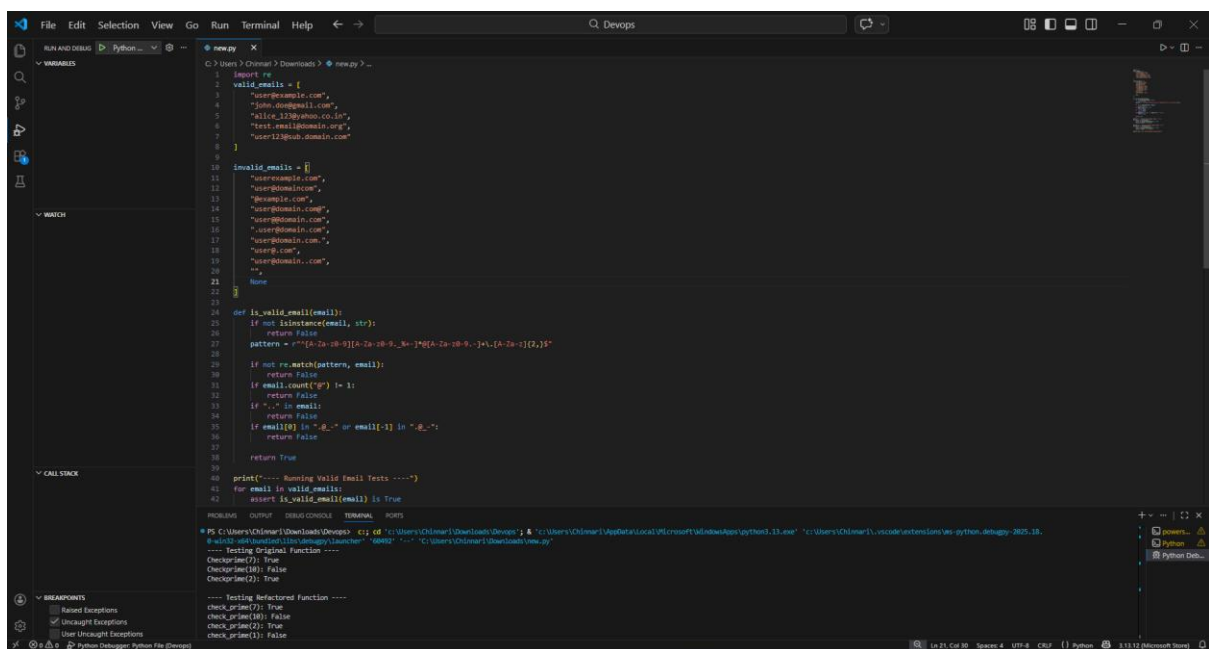
Batch: 25

Scenario

Requirements

- ### Expected Output

- Python function for email validation
- All AI-generated test cases pass successfully
- Invalid email formats are correctly rejected
- Valid email formats return True



```

1  import re
2  valid_email = [
3      "user@example.com",
4      "john.doe@wall.com",
5      "alice_123@ohm.co.in",
6      "test_email@domain.org",
7      "user123@sub.domain.com"
8  ]
9
10 invalid_email = [
11     "user@exampl.com",
12     "user@domain.com",
13     "@example.com",
14     "user@domain.com",
15     "user@domain.com",
16     "user@domain.com",
17     "user@domain.com",
18     "user@com",
19     "user@domain.com",
20     ""
21 ]
22 None
23
24 def is_valid_email(email):
25     if not isinstance(email, str):
26         return False
27     pattern = r"^[a-zA-Z0-9]{1,30}[a-zA-Z0-9._%+-]{0,30}@([a-zA-Z-]{1,3})$"
28     if not re.match(pattern, email):
29         return False
30     if email.count("@") != 1:
31         return False
32     if "." in email:
33         return False
34     if email[0] in ".@_" or email[-1] in ".@_":
35         return False
36     return True
37
38 print("---- Running Valid Email Tests ----")
39 for email in valid_email:
40     assert is_valid_email(email) is True
41
42 print("---- Running Invalid Email Tests ----")
43 for email in invalid_email:
44     assert is_valid_email(email) is False
45
46 print("All test cases passed successfully!")

```

```

1  import re
2  valid_email = [
3      "user@example.com",
4      "john.doe@wall.com",
5      "alice_123@ohm.co.in",
6      "test_email@domain.org",
7      "user123@sub.domain.com"
8  ]
9
10 invalid_email = [
11     "user@exampl.com",
12     "user@domain.com",
13     "@example.com",
14     "user@domain.com",
15     "user@domain.com",
16     "user@domain.com",
17     "user@domain.com",
18     "user@com",
19     "user@domain.com",
20     ""
21 ]
22 None
23
24 def is_valid_email(email):
25     if not isinstance(email, str):
26         return False
27     pattern = r"^[a-zA-Z0-9]{1,30}[a-zA-Z0-9._%+-]{0,30}@([a-zA-Z-]{1,3})$"
28     if not re.match(pattern, email):
29         return False
30     if email.count("@") != 1:
31         return False
32     if "." in email:
33         return False
34     if email[0] in ".@_" or email[-1] in ".@_":
35         return False
36     return True
37
38 print("---- Running Valid Email Tests ----")
39 for email in valid_email:
40     assert is_valid_email(email) is True
41     print(f"{email} -> Valid")
42
43 print("---- Running Invalid Email Tests ----")
44 for email in invalid_email:
45     assert is_valid_email(email) is False
46     print(f"{email} -> Invalid")
47
48 print("All test cases passed successfully!")

```

Task 2: Grade Assignment using Loops

Scenario

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

Requirements

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

– 90–100 → A

– 80–89 → B

– 70–79 → C

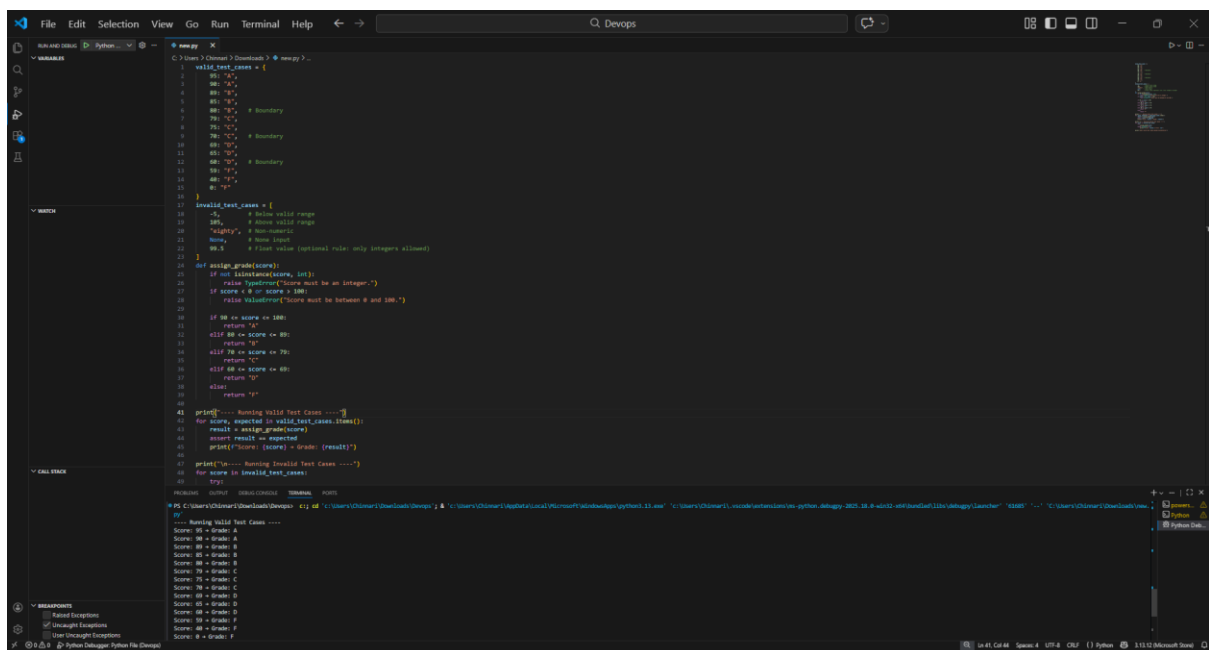
– 60–69 → D

– Below 60 → F

- Include boundary values (60, 70, 80, 90)
- Include invalid inputs such as -5, 105, "eighty"
- Implement the function using a test-driven approach

Expected Output

- Grade assignment function implemented in Python
- Boundary values handled correctly
- Invalid inputs handled gracefully
- All AI-generated test cases pass



```
File Edit Selection View Go Run Terminal Help
C:\Users\Chenar\Downloads\Devs - empty > ...
1 valid_test_cases = [
2     60, "A",
3     60, "A",
4     60, "A",
5     60, "A", # Boundary
6     70, "A",
7     70, "A",
8     70, "A",
9     70, "A", # Boundary
10    80, "A",
11    80, "A",
12    80, "A", # Boundary
13    90, "A",
14    90, "A",
15    90, "A",
16    90, "A",
17 ]
18 invalid_test_cases = [
19     -5, # Below value range
20     105, # Above value range
21     "eighty", # Non-numeric
22     None, # None input
23     60.5 # Float value (optional rule: only integers allowed)
24 ]
25
26 def assign_grade(score):
27     if not isinstance(score, int):
28         raise TypeError("Score must be an integer.")
29     if score < 0 or score > 100:
30         raise ValueError("Score must be between 0 and 100.")
31
32     if 90 <= score <= 100:
33         return "A"
34     elif 80 <= score <= 89:
35         return "B"
36     elif 70 <= score <= 79:
37         return "C"
38     elif 60 <= score <= 69:
39         return "D"
40     else:
41         return "F"
42
43 print("... Running Valid Test Cases ...")
44 for score, expected in valid_test_cases.items():
45     result = assign_grade(score)
46     assert result == expected
47     print(f"Score: {score} > Grade: {result}")
48
49 print("... Running Invalid Test Cases ...")
50 for score in invalid_test_cases:
51     try:
52         assign_grade(score)
53     except (TypeError, ValueError):
54         print(f"Invalid input: {score} - {type(score)}")
55
56 # Expected Output:
57 # ... Running Valid Test Cases ...
58 # Score: 60 > Grade: A
59 # Score: 60 > Grade: A
60 # Score: 60 > Grade: A
61 # Score: 60 > Grade: A
62 # Score: 70 > Grade: A
63 # Score: 70 > Grade: A
64 # Score: 70 > Grade: A
65 # Score: 70 > Grade: A
66 # Score: 80 > Grade: B
67 # Score: 80 > Grade: B
68 # Score: 80 > Grade: B
69 # Score: 80 > Grade: B
70 # Score: 90 > Grade: C
71 # Score: 90 > Grade: C
72 # Score: 90 > Grade: C
73 # Score: 90 > Grade: C
74 # Score: 100 > Grade: A
75 # Score: 100 > Grade: A
76 # Score: 100 > Grade: A
77 # Score: 100 > Grade: A
78 # ... Running Invalid Test Cases ...
79 # Invalid input: -5 - <int>
80 # Invalid input: 105 - <int>
81 # Invalid input: eighty - <str>
82 # Invalid input: None - <NoneType>
83 # Invalid input: 60.5 - <float>
```

```
11 def assign_grade(score):
12     if not isinstance(score, int):
13         raise TypeError("Score must be an integer.")
14     if score < 0 or score > 100:
15         raise ValueError("Score must be between 0 and 100.")
16
17     if 90 <= score <= 100:
18         return "A"
19     elif 80 <= score <= 89:
20         return "B"
21     elif 70 <= score <= 79:
22         return "C"
23     elif 60 <= score <= 69:
24         return "D"
25     else:
26         return "F"
27
28 # Test cases
29 print("Running valid test cases ----")
30 for score, expected in valid_test_cases.items():
31     result = assign_grade(score)
32     assert result == expected
33     print(f"Score: {score} = Grade: {result}")
34
35 print("Running invalid test cases ----")
36 for score in invalid_test_cases:
37     try:
38         assign_grade(score)
39     except Exception as e:
40         print(f"Score: {score} = Error: {e}")
41
42 print("All valid test cases passed successfully!")
```

Running valid test cases ----
Score: 95 = Grade: A
Score: 85 = Grade: B
Score: 75 = Grade: C
Score: 65 = Grade: D
Score: 55 = Grade: F

Running invalid test cases ----
Score: 105 = Error: Score must be between 0 and 100.
Score: -5 = Error: Score must be between 0 and 100.
Score: 100.5 = Error: Score must be an integer.
Score: None = Error: Score must be an integer.

All valid test cases passed successfully!

Task 3: Sentence Palindrome Checker

Scenario

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

Requirements

- AI should generate test cases for `is_sentence_palindrome(sentence)`
- Ignore case, spaces, and punctuation
- Test both palindromic and non-palindromic sentences
- Example:
– "A man a plan a canal Panama" → True

Expected Output

- Function correctly identifies sentence palindromes
- Case and punctuation are ignored
- Returns True or False accurately
- All AI-generated test cases pass

```

1 # Import the unittest module
2 import unittest
3
4 # Test sentences
5 test_sentences = [
6     "A man a plan a canal Panama",
7     "Was it a car or a cat I saw?",
8     "No lemon, no melon",
9     "While was I saw I saw Ella",
10    "Wasawar",
11]
12
13 # Non-palindrome sentences
14 non_palindrome_sentences = [
15     "Hello world",
16     "Python programming",
17     "Spam! is missing",
18     "This is not a palindrome",
19     "Palindrome test case!"
20]
21
22 # Invalid inputs
23 invalid_inputs = [
24     None,
25     12345,
26     ["not", "a", "string"]
27]
28
29 def is_sentence_palindrome(sentence):
30     if not isinstance(sentence, str):
31         raise TypeError("Input must be a string.")
32
33     # Remove punctuation and spaces, convert to lowercase
34     cleaned = ""
35     for char in sentence:
36         if char.isalnum():
37             cleaned += char.lower()
38
39     # Check palindrome
40     return cleaned == cleaned[::-1]
41
42 print("---- Testing Palindrome Sentences ----")
43 for sentence in test_sentences:
44     result = is_sentence_palindrome(sentence)
45     print(f"Test: {sentence} -> {result}")
46
47 print("---- Testing Non-Palindrome Sentences ----")
48 for sentence in non_palindrome_sentences:
49     result = is_sentence_palindrome(sentence)
50     print(f"Test: {sentence} -> {result}")
51
52 # Run the tests
53 if __name__ == "__main__":
54     unittest.main()
55
56 # Expected output:
57 # ---- Testing Palindrome Sentences ----
58 # 'A man a plan a canal Panama' = True
59 # 'Was it a car or a cat I saw?' = True
60 # 'No lemon, no melon' = True
61 # 'While was I saw I saw Ella' = True
62 # 'Wasawar' = True
63 # ----
64 # ---- Testing Non-Palindrome Sentences ----
65 # 'Hello world' = False
66 # 'Python programming' = False
67 # 'Spam! is missing' = False
68 # 'This is not a palindrome' = False
69 # 'Palindrome test case!' = False
70 # ----

```

```

1 # Import the unittest module
2 import unittest
3
4 # Test sentences
5 test_sentences = [
6     "A man a plan a canal Panama",
7     "Was it a car or a cat I saw?",
8     "No lemon, no melon",
9     "While was I saw I saw Ella",
10    "Wasawar",
11]
12
13 # Non-palindrome sentences
14 non_palindrome_sentences = [
15     "Hello world",
16     "Python programming",
17     "Spam! is missing",
18     "This is not a palindrome",
19     "Palindrome test case!"
20]
21
22 # Invalid inputs
23 invalid_inputs = [
24     None,
25     12345,
26     ["not", "a", "string"]
27]
28
29 def is_sentence_palindrome(sentence):
30     if not isinstance(sentence, str):
31         raise TypeError("Input must be a string.")
32
33     # Remove punctuation and spaces, convert to lowercase
34     cleaned = ""
35     for char in sentence:
36         if char.isalnum():
37             cleaned += char.lower()
38
39     # Check palindrome
40     return cleaned == cleaned[::-1]
41
42 print("---- Testing Palindrome Sentences ----")
43 for sentence in test_sentences:
44     result = is_sentence_palindrome(sentence)
45     print(f"Test: {sentence} -> {result}")
46
47 print("---- Testing Non-Palindrome Sentences ----")
48 for sentence in non_palindrome_sentences:
49     result = is_sentence_palindrome(sentence)
50     print(f"Test: {sentence} -> {result}")
51
52 # Run the tests
53 if __name__ == "__main__":
54     unittest.main()
55
56 # Expected output:
57 # ---- Testing Palindrome Sentences ----
58 # 'A man a plan a canal Panama' = True
59 # 'Was it a car or a cat I saw?' = True
60 # 'No lemon, no melon' = True
61 # 'While was I saw I saw Ella' = True
62 # 'Wasawar' = True
63 # ----
64 # ---- Testing Non-Palindrome Sentences ----
65 # 'Hello world' = False
66 # 'Python programming' = False
67 # 'Spam! is missing' = False
68 # 'This is not a palindrome' = False
69 # 'Palindrome test case!' = False
70 # ----
71
72 # ---- Testing Invalid Inputs ----
73 # None = Error: Input must be a string.
74 # 12345 = Error: Input must be a string.
75 # ['not', 'a', 'string'] = Error: Input must be a string.
76 # ----
77
78 # All test cases passed successfully!
79 # Python Debugger: Python File Debugger

```

Task 4: ShoppingCart Class

Scenario

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

Requirements

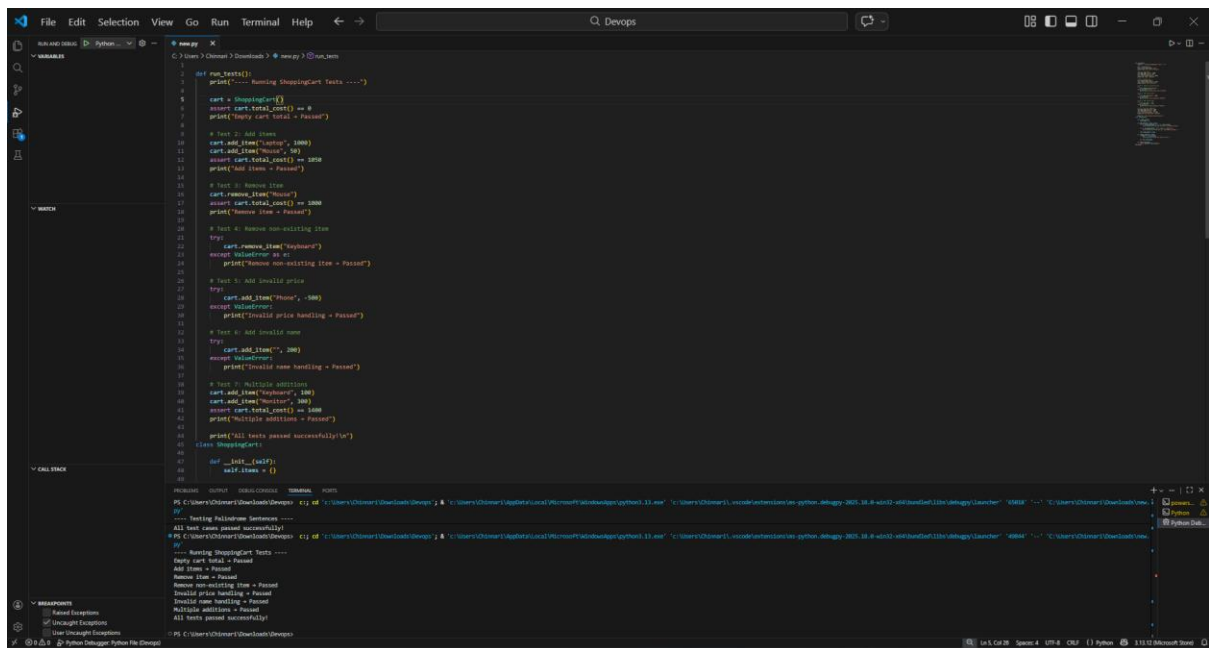
- AI should generate test cases for the ShoppingCart class
- Class must include the following methods:
 - add_item(name, price)
 - remove_item(name)

– total_cost()

- Validate correct addition, removal, and cost calculation
- Handle empty cart scenarios

Expected Output

- Fully implemented ShoppingCart class
- All methods pass AI-generated test cases
- Total cost is calculated accurately
- Items are added and removed correctly give code for this



```
def run_tests():
    print("Running ShoppingCart Tests ....")

    cart = ShoppingCart()
    assert cart.total_cost() == 0
    print("Empty cart total = Passed")

    # Test 1: Add item
    cart.add_item("apple", 1000)
    cart.add_item("banana", 50)
    assert cart.total_cost() == 1050
    print("Add item = Passed")

    # Test 2: Remove item
    cart.remove_item("banana")
    assert cart.total_cost() == 1000
    print("Remove item = Passed")

    # Test 3: Remove non-existing item
    try:
        cart.remove_item("watermelon")
    except ValueError as e:
        print("Remove non-existing item = Passed")

    # Test 4: Add invalid price
    try:
        cart.add_item("orange", -500)
    except ValueError:
        print("Invalid price handling = Passed")

    # Test 5: Add invalid name
    try:
        cart.add_item("", 100)
    except ValueError:
        print("Invalid name handling = Passed")

    # Test 6: Multiple additions
    cart.add_item("watermelon", 1500)
    cart.add_item("mango", 300)
    assert cart.total_cost() == 1400
    print("Multiple additions = Passed")

    print("All tests passed successfully!")

class ShoppingCart:
    def __init__(self):
        self.items = {}

# Run the tests
run_tests()
```

Task 5: Date Format Conversion

Scenario

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

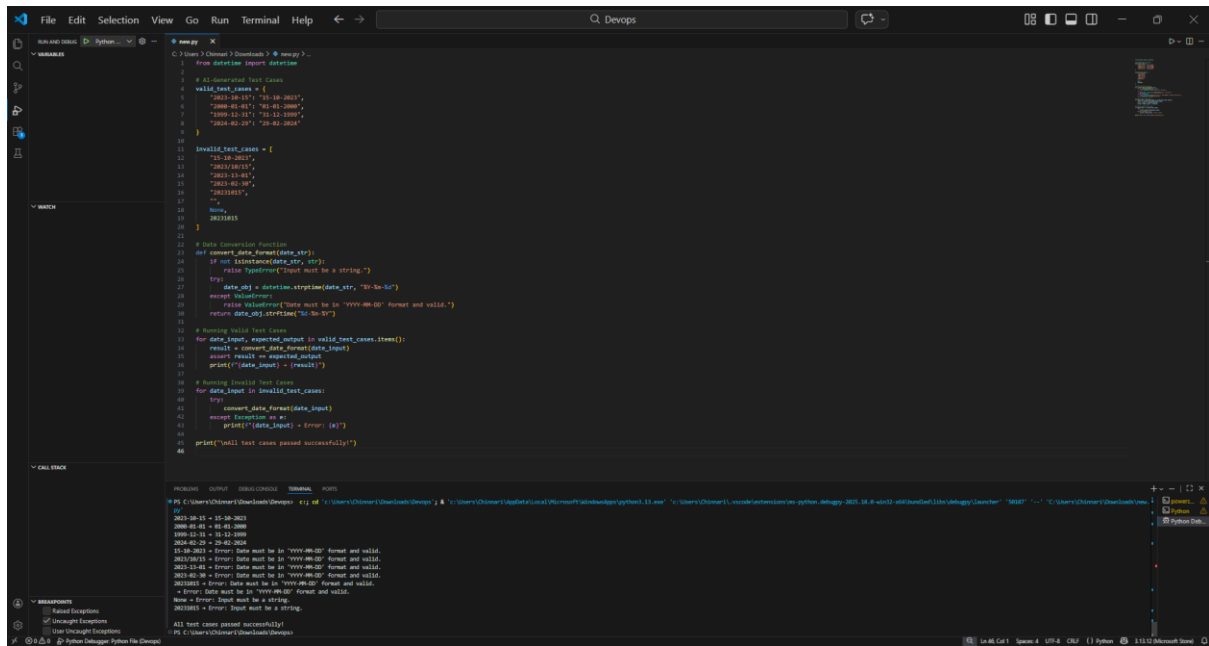
Requirements

- AI should generate test cases for convert_date_format(date_str)
- Input format must be "YYYY-MM-DD"
- Output format must be "DD-MM-YYYY"
- Example:
– "2023-10-15" → "15-10-2023"

Expected Output

- Date conversion function implemented in Python
- Correct format conversion for all valid inputs

- All AI-generated test cases pass successfully give code for this



```
File Edit Selection View Go Run Terminal Help
C:\Users\Chinmay> Download > energy > ..
1 from datetime import datetime
2
3 # All generated test cases
4 valid_test_cases = [
5     "2023-10-20", "15-10-2023",
6     "2000-01-01", "31-12-2000",
7     "1999-12-31", "11-11-1999",
8     "2024-01-20", "20-01-2024"
9 ]
10
11 invalid_test_cases = [
12     "15-10-2022",
13     "2023/10/20",
14     "2023-10-01",
15     "2023-02-30",
16     "20231015",
17     ""
18 ]
19 Name:
20 20231015
21
22 # Date Conversion Function
23 def convert_date_format(date_str):
24     if not isinstance(date_str, str):
25         raise TypeError("Input must be a string.")
26     try:
27         date_obj = datetime.strptime(date_str, "%Y-%m-%d")
28         return date_obj.strftime("%Y-%m-%d")
29     except ValueError:
30         raise ValueError("Date must be in 'YYYY-MM-DD' format and valid.")
31
32 # Running Valid Test Cases
33 for data_input, expected_output in valid_test_cases.items():
34     result = convert_date_format(data_input)
35     assert result == expected_output
36     print(f"[data_input] = {result}")
37
38 # Running Invalid Test Cases
39 for data_input in invalid_test_cases:
40     try:
41         convert_date_format(data_input)
42     except Exception as e:
43         print(f"[data_input] = Error: {e}")
44
45 print("All test cases passed successfully!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Chinmay\Downloads> python energy.py
2023-10-20 = 15-10-2023
2000-01-01 = 01-01-2000
1999-12-31 = 31-12-1999
2024-01-20 = 20-01-2024
15-10-2023 = Error: Date must be in 'YYYY-MM-DD' format and valid.
2023/10/20 = Error: Date must be in 'YYYY-MM-DD' format and valid.
2023-10-01 = Error: Date must be in 'YYYY-MM-DD' format and valid.
2023-02-30 = Error: Date must be in 'YYYY-MM-DD' format and valid.
20231015 = Error: Date must be in 'YYYY-MM-DD' format and valid.
= Error: Date must be in 'YYYY-MM-DD' format and valid.
= Error: Input must be a string.
20231015 = Error: Input must be a string.

All test cases passed successfully!
PS C:\Users\Chinmay\Downloads>
```