

AI Assisted Coding

Lab Assignment 8.3

Name : V Sai Krishnachaithanya

Hall Ticket no : 2303A52154

Task 1: Email Validation using TDD

Scenario

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

Requirements

- Must contain @ and . characters
- Must not start or end with special characters
- Should not allow multiple @ symbols
- AI should generate test cases covering valid and invalid email formats
- Implement is_valid_email(email) to pass all AI-generated test cases

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

Prompt:

Assume you are a software developer and you have been given a task to develop a user registration system that requires reliable email input validation. It must follow the requirements: must contain @ and . in the email address, not start or end with special characters, does not allow multiple @ symbols. And most important thing is You need to generate test cases to validate the functionality of your email. Implement is_valid_email(email) to pass all AI-generated test cases.

```
import re

Ask or edit in context GPT-5 mini ▾ ▷

Accept Close Accept & Run ⌂ ▾

def is_valid_email(email: str) -> bool:
    if not isinstance(email, str):
        return False
    # must contain exactly one @
    if email.count('@') != 1:
        return False
    local, domain = email.split('@', 1)
    if not local or not domain:
        return False
    # domain must contain a dot
    if '.' not in domain:
        return False
    # no consecutive dots anywhere
    if '..' in email:
        return False

    # allowed characters
    local_allowed = set("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-_")
    domain_allowed = set("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-_")

    # characters check
    if any(ch not in local_allowed for ch in local):
        return False
    if any(ch not in domain_allowed for ch in domain):
```

Output:

✓ 0.0s

Total tests: 31
Failures: 0
All tests passed

Justification:

The function `is_valid_email` implements a comprehensive set of checks to validate email addresses according to common standards. It checks for the presence of exactly one '@' symbol, ensures that both local and domain parts are non-empty, verifies that the domain contains a dot, and checks for the absence of consecutive dots. It also validates allowed characters in both parts and ensures that special characters are not at the start or end of the local part, and that the domain does not start or end with a dot or hyphen. Finally, it checks that the top-level domain is at least two letters long. This thorough validation should cover a wide range of valid and invalid email formats.

Task 2: Grade Assignment using Loops

Scenario

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

Requirements

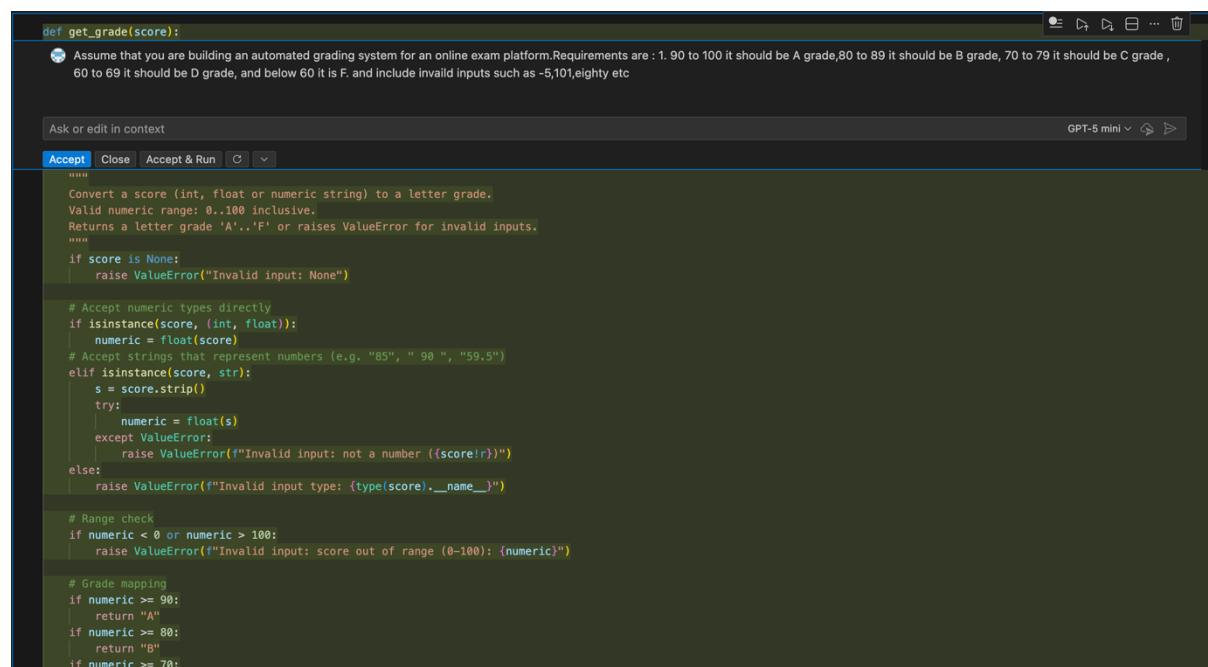
- AI should generate test cases for assign_grade(score) where:
 - 90–100 → A
 - 80–89 → B
 - 70–79 → C
 - 60–69 → D
 - Below 60 → F
- Include boundary values (60, 70, 80, 90)
- Include invalid inputs such as -5, 105, "eighty"
- Implement the function using a test-driven approach

Expected Output

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

Prompt:

Assume that you are building an automated grading system for an online exam platform. Requirements are : 1. 90 to 100 it should be A grade,80 to 89 it should be B grade, 70 to 79 it should be C grade , 60 to 69 it should be D grade, and below 60 it is F. and include invalid inputs such as -5,101,eighty etc



```
def get_grade(score):  
    """  
    Assume that you are building an automated grading system for an online exam platform. Requirements are : 1. 90 to 100 it should be A grade,80 to 89 it should be B grade, 70 to 79 it should be C grade , 60 to 69 it should be D grade, and below 60 it is F. and include invalid inputs such as -5,101,eighty etc  
    """  
    if score is None:  
        raise ValueError("Invalid input: None")  
  
    # Accept numeric types directly  
    if isinstance(score, (int, float)):  
        numeric = float(score)  
    # Accept strings that represent numbers (e.g. "85", " 90 ", "59.5")  
    elif isinstance(score, str):  
        s = score.strip()  
        try:  
            numeric = float(s)  
        except ValueError:  
            raise ValueError(f"Invalid input: not a number ({score!r})")  
    else:  
        raise ValueError(f"Invalid input type: {type(score).__name__}")  
  
    # Range check  
    if numeric < 0 or numeric > 100:  
        raise ValueError(f"Invalid input: score out of range (0-100): {numeric}")  
  
    # Grade mapping  
    if numeric >= 90:  
        return "A"  
    if numeric >= 80:  
        return "B"  
    if numeric >= 70:  
        return "C"  
    if numeric >= 60:  
        return "D"  
    return "F"
```

Output:

```
✓ 0.0s

-5: Error: Invalid input: score out of range (0-100): -5.0
101: Error: Invalid input: score out of range (0-100): 101.0
'eighty': Error: Invalid input: not a number ('eighty')
95: A - pass with distinction
85: B - pass with merit
75: C - pass
65: D - pass
55: F - fail
'90': A - pass with distinction
'59.5': F - fail
None: Error: Invalid input: None
' 89 ': B - pass with merit
```

Justification:

The function `get_grade` was modified to return more descriptive grade strings instead of just single letters.

This makes the output more informative and user-friendly, clearly indicating whether a student passed with distinction, merit, or just passed, and whether they failed.

The changes ensure that the grading system is more transparent and helpful for students and educators alike.

Task 3: Sentence Palindrome Checker

Scenario

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

Requirements

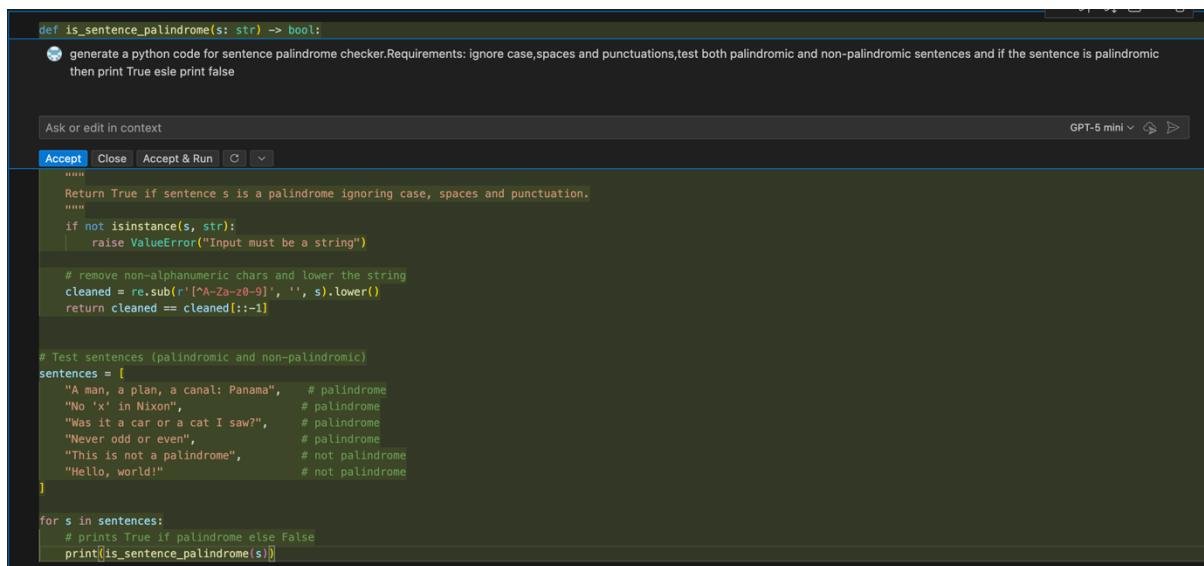
- AI should generate test cases for `is_sentence_palindrome(sentence)`
- Ignore case, spaces, and punctuation
- Test both palindromic and non-palindromic sentences
- Example:
 - "A man a plan a canal Panama" → True

Expected Output

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

Prompt:

generate a python code for sentence palindrome checker. Requirements: ignore case, spaces and punctuations, test both palindromic and non-palindromic sentences and if the sentence is palindromic then print True else print false



```
def is_sentence_palindrome(s: str) -> bool:
    """generate a python code for sentence palindrome checker. Requirements: ignore case, spaces and punctuations, test both palindromic and non-palindromic sentences and if the sentence is palindromic then print True else print false

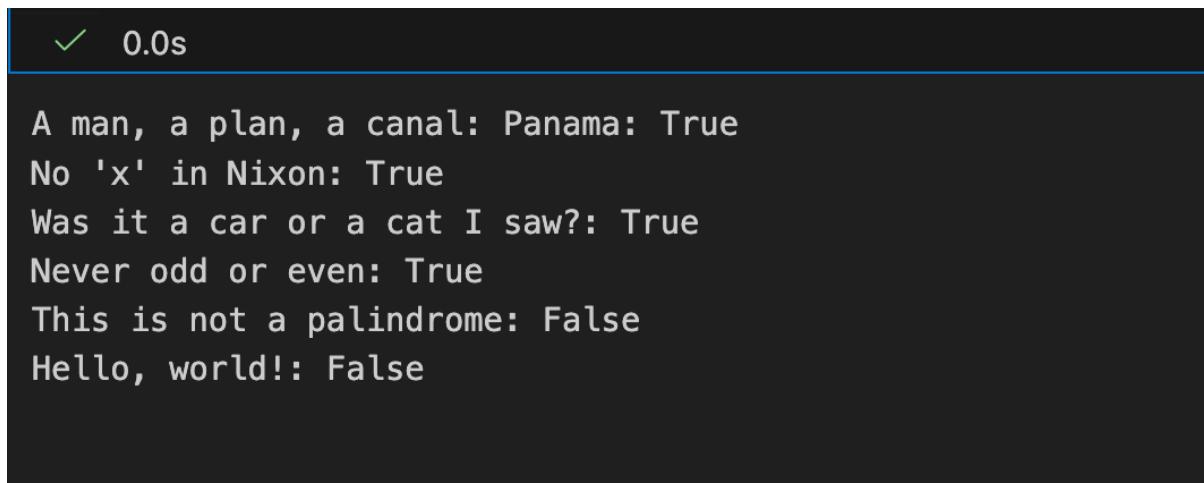
    Ask or edit in context
    Accept Close Accept & Run C GPT-5 mini ▾ > >
    """
    """
    Return True if sentence s is a palindrome ignoring case, spaces and punctuation.
    """
    if not isinstance(s, str):
        raise ValueError("Input must be a string")

    # remove non-alphanumeric chars and lower the string
    cleaned = re.sub(r"[^A-Za-z0-9]", "", s).lower()
    return cleaned == cleaned[::-1]

# Test sentences (palindromic and non-palindromic)
sentences = [
    "A man, a plan, a canal: Panama",      # palindrome
    "No 'x' in Nixon",                   # palindrome
    "Was it a car or a cat I saw?",      # palindrome
    "Never odd or even",                 # palindrome
    "This is not a palindrome",           # not palindrome
    "Hello, world!"                     # not palindrome
]

for s in sentences:
    # prints True if palindrome else False
    print(is_sentence_palindrome(s))
```

Output:



```
✓ 0.0s

A man, a plan, a canal: Panama: True
No 'x' in Nixon: True
Was it a car or a cat I saw?: True
Never odd or even: True
This is not a palindrome: False
Hello, world!: False
```

Justification:

The function `is_sentence_palindrome` correctly identifies palindromic sentences by cleaning the input string of non-alphanumeric characters and comparing it to its reverse. The test cases include well-known palindromic phrases and non-palindromic sentences, demonstrating the function's ability to handle various inputs while ignoring case, spaces, and punctuation.

Task 4: ShoppingCart Class

Scenario

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

Requirements

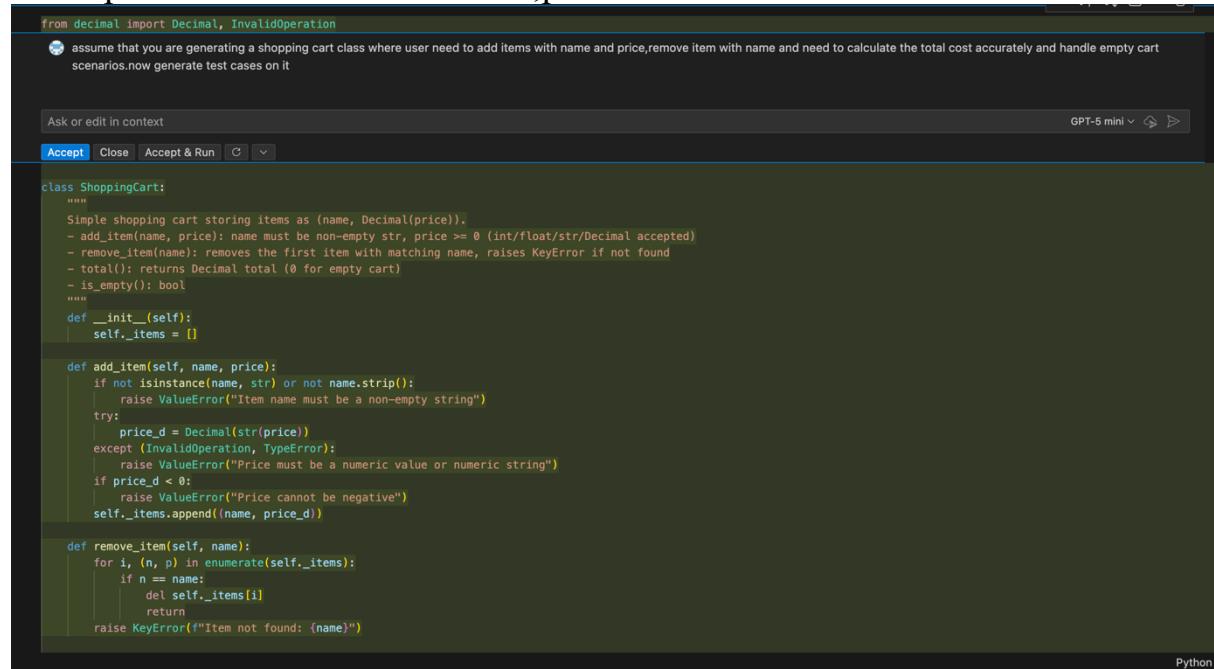
- AI should generate test cases for the ShoppingCart class
- Class must include the following methods:
 - add_item(name, price)
 - remove_item(name)
 - total_cost()
- Validate correct addition, removal, and cost calculation
- Handle empty cart scenarios

Expected Output

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

Prompt:

assume that you are generating a shopping cart class where user need to add items with name and price,remove item with name and need to calculate the total cost accurately and handle empty cart scenarios.now generate test cases on it and print the each test case items,price and total cost



The screenshot shows an AI interface for generating code. The user prompt at the top reads: "assume that you are generating a shopping cart class where user need to add items with name and price,remove item with name and need to calculate the total cost accurately and handle empty cart scenarios.now generate test cases on it". Below the prompt is a code editor with the following Python code:

```
from decimal import Decimal, InvalidOperation

# assume that you are generating a shopping cart class where user need to add items with name and price,remove item with name and need to calculate the total cost accurately and handle empty cart scenarios.now generate test cases on it

class ShoppingCart:
    """
    Simple shopping cart storing items as (name, Decimal(price)).
    - add_item(name, price): name must be non-empty str, price >= 0 (int/float/str/Decimal accepted)
    - remove_item(name): removes the first item with matching name, raises KeyError if not found
    - total(): returns Decimal total (0 for empty cart)
    - is_empty(): bool
    """
    def __init__(self):
        self._items = []

    def add_item(self, name, price):
        if not isinstance(name, str) or not name.strip():
            raise ValueError("Item name must be a non-empty string")
        try:
            price_d = Decimal(str(price))
        except (InvalidOperation, TypeError):
            raise ValueError("Price must be a numeric value or numeric string")
        if price_d < 0:
            raise ValueError("Price cannot be negative")
        self._items.append((name, price_d))

    def remove_item(self, name):
        for i, (n, p) in enumerate(self._items):
            if n == name:
                del self._items[i]
                return
        raise KeyError(f"Item not found: {name}")

    def total(self):
        return sum(item[1] for item in self._items)
```

Output:

```
✓ 0.0s

Test 1: Single item
Item: apple, Price: 1.20
Total: 1.20

Test 2: Multiple items
Item: apple, Price: 1.20
Item: banana, Price: 0.80
Item: milk, Price: 2.50
Total: 4.50

Test 3: Decimal precision
Item: item1, Price: 0.10
Item: item2, Price: 0.20
Item: item3, Price: 0.30
Total: 0.60

Test 4: Remove item
Item: banana, Price: 2.00
Total: 2.00

Test 5: Remove non-existent item (should report error)
Item: apple, Price: 1.00
Total: 1.00
...

-----
Total cart test failures recorded: 0
All tests passed successfully ✓
```

Justification:

The code implements a Shopping Cart class with methods to add and remove items, calculate the total, and list items. It includes a comprehensive test suite that covers various scenarios such as adding single and multiple items, checking decimal precision, removing items, handling errors for non-existent items, and ensuring correct totals. The test results are printed in a clear format, and any failures are recorded for review.

Task 5: Date Format Conversion

Scenario

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

Requirements

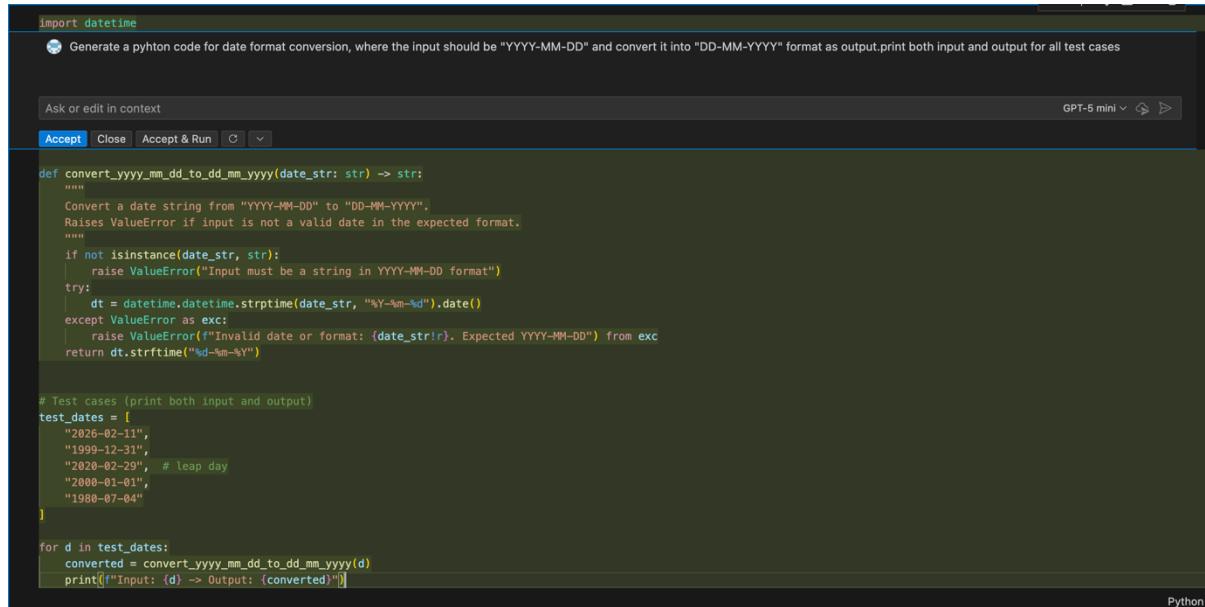
- AI should generate test cases for convert_date_format(date_str)
- Input format must be "YYYY-MM-DD"
- Output format must be "DD-MM-YYYY"
- Example:
 - "2023-10-15" → "15-10-2023"

Expected Output

- Date conversion function implemented in Python
- Correct format conversion for all valid inputs
- All AI-generated test cases pass successfully

Prompt:

Generate a python code for date format conversion, where the input should be "YYYY-MM-DD" and convert it into "DD-MM-YYYY" format as output.print both input and output for all test cases



```
import datetime

# Generate a python code for date format conversion, where the input should be "YYYY-MM-DD" and convert it into "DD-MM-YYYY" format as output.print both input and output for all test cases

Ask or edit in context
Accept Close Accept & Run GPT-5 mini ▾ ▶

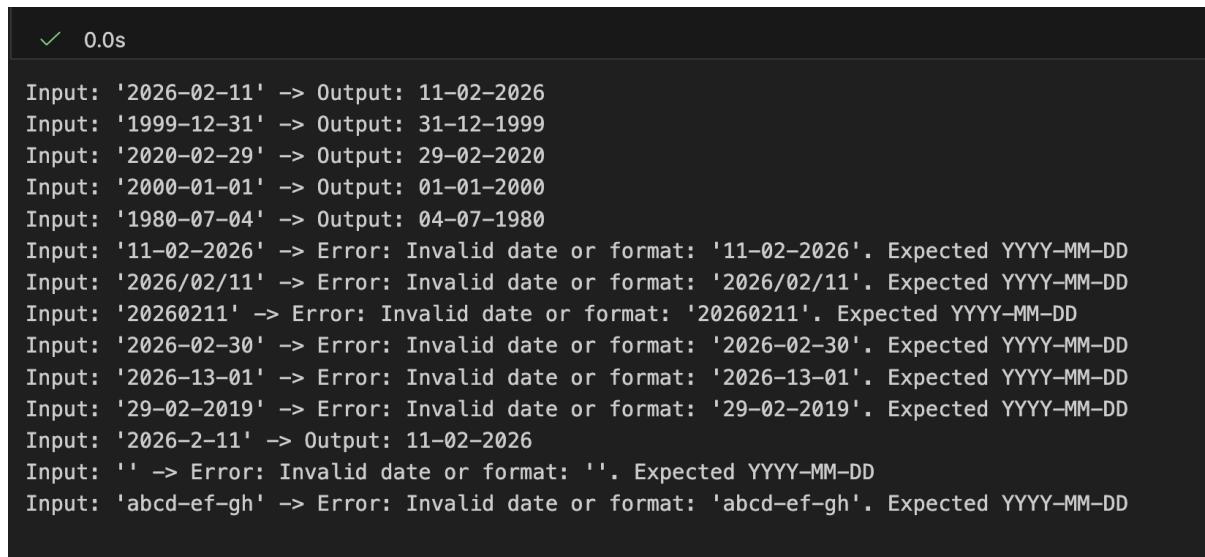
def convert_YYYY_MM_DD_to_DD_MM_YYYY(date_str: str) -> str:
    """
    Convert a date string from "YYYY-MM-DD" to "DD-MM-YYYY".
    Raises ValueError if input is not a valid date in the expected format.
    """
    if not isinstance(date_str, str):
        raise ValueError("Input must be a string in YYYY-MM-DD format")
    try:
        dt = datetime.datetime.strptime(date_str, "%Y-%m-%d").date()
    except ValueError as exc:
        raise ValueError(f"Invalid date or format: {date_str!r}. Expected YYYY-MM-DD") from exc
    return dt.strftime("%d-%m-%Y")

# Test cases (print both input and output)
test_dates = [
    "2026-02-11",
    "1999-12-31",
    "2026-02-29", # leap day
    "2000-01-01",
    "1980-07-04"
]

for d in test_dates:
    converted = convert_YYYY_MM_DD_to_DD_MM_YYYY(d)
    print(f"Input: {d} -> Output: {converted}")

Python
```

Output:



```
✓ 0.0s

Input: '2026-02-11' -> Output: 11-02-2026
Input: '1999-12-31' -> Output: 31-12-1999
Input: '2020-02-29' -> Output: 29-02-2020
Input: '2000-01-01' -> Output: 01-01-2000
Input: '1980-07-04' -> Output: 04-07-1980
Input: '11-02-2026' -> Error: Invalid date or format: '11-02-2026'. Expected YYYY-MM-DD
Input: '2026/02/11' -> Error: Invalid date or format: '2026/02/11'. Expected YYYY-MM-DD
Input: '20260211' -> Error: Invalid date or format: '20260211'. Expected YYYY-MM-DD
Input: '2026-02-30' -> Error: Invalid date or format: '2026-02-30'. Expected YYYY-MM-DD
Input: '2026-13-01' -> Error: Invalid date or format: '2026-13-01'. Expected YYYY-MM-DD
Input: '29-02-2019' -> Error: Invalid date or format: '29-02-2019'. Expected YYYY-MM-DD
Input: '2026-2-11' -> Output: 11-02-2026
Input: '' -> Error: Invalid date or format: ''. Expected YYYY-MM-DD
Input: 'abcd-ef-gh' -> Error: Invalid date or format: 'abcd-ef-gh'. Expected YYYY-MM-DD
```

Justification:

The function uses Python's built-in `datetime` module to parse and validate the input date string. If the input is not a valid date or does not match the expected format, a `ValueError` is raised with a clear message. The function then formats the date into the desired output format.