

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

```
new.py
C:\Users\Chimer> cd Downloads > new.py > ...
1 import re
2 valid_emails = [
3     'user@example.com',
4     'user@domain.com',
5     'user_123@domain.co.in',
6     'test_email@domain.org',
7     'user12@sub.domain.com'
8 ]
9
10 invalid_emails = []
11 'user@example.com',
12 'user@.com',
13 'userexample.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 'user@'
21 None
22
23
24 def is_valid_email(email):
25     if not isinstance(email, str):
26         return False
27     pattern = "[A-Za-z0-9]+@[A-Za-z0-9.-]+\\.[A-Za-z0-9.-]+"
28
29     if not re.match(pattern, email):
30         return False
31     if email[-1] == '.':
32         return False
33     if '.' in email:
34         return True
35     if email[0] in "._.-":
36         return False
37     return True
38
39 print("---- Running Valid Email Tests ----")
40 for email in valid_emails:
41     assert is_valid_email(email) is True
42
43 print("---- Running Invalid Email Tests ----")
44 for email in invalid_emails:
45     assert is_valid_email(email) is False
46
47 user@example.com -> Valid
48 user@.com -> Invalid
49 user_123@domain.co.in -> Valid
50 test_email@domain.org -> Valid
51 user12@sub.domain.com -> Valid
52
53 ---- Running Invalid Email Tests ----
54 user@example.com -> Invalid
55 user@.com -> Invalid
56 user@domain.com -> Invalid
```

File Edit Selection View Go Run Terminal Help ↻ 🔍 Devops

RUN AND DEBUG Python File (Devops)

VARIABLES

WATCH

CALL STACK

BREAKPOINTS

Run Configuration: Python

OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Chimer\Downloads\Devops> cd C:\Users\Chimer\Downloads\Devops & "C:\Users\Chimer\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "C:\Users\Chimer\vscode\extensions\ms-python.python.debug-2625.18.0\python\Python\Python Debug\new.py"

---- Running Valid Email Tests ----

user@example.com -> Valid

user@.com -> Invalid

user_123@domain.co.in -> Valid

test_email@domain.org -> Valid

user12@sub.domain.com -> Valid

---- Running Invalid Email Tests ----

user@example.com -> Invalid

user@.com -> Invalid

user@domain.com -> Invalid

Ln 21, Col 30 Spaces: 4 UTR: 8 CRLF: [] Python 3.13.12 (Microsoft Store)

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

$$= 70 - 79 \rightarrow C$$

$$= 60-69 \Rightarrow D$$

– 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

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure for "Devops" containing "variables.py".
- Code Editor:** Displays the content of `variables.py`. The code defines a class `GradeCalculator` with methods for calculating average and assigning grades based on scores.
- Terminal:** Shows the output of running the script with various test cases. It includes valid test cases and invalid ones (e.g., non-integer scores, negative scores, and scores above 100).
- Output:** Shows the result of the test cases as "All valid test cases passed successfully!"
- Search:** A search bar at the top right with the text "Devops".
- Activity Bar:** Includes icons for File, Edit, Selection, View, Go, Run, Terminal, Help, and a GitHub icon.

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 #!/usr/bin/python
2
3 # Test cases for is_sentence_palindrome function
4
5 import string
6
7 palindrom_sentences = [
8     "A man a plan a canal Panama",
9     "Madam in Eden is Eden Madam",
10    "Was it a car or a cat I saw?",
11    "Dad I'm mad",
12    "Able was I ere I saw Elba",
13    "Never odd or even"
14]
15
16
17 # Non-palindrome sentences
18 nonPalindrome_sentences = [
19     "Hello world",
20     "Python programming",
21     "This is not a palindrome",
22     "Palindrome test case"
23]
24
25
26 # Invalid inputs
27 invalid_inputs = [
28     None,
29     12345,
30     ["not", "a", "string"]
31]
32
33 def is_sentencePalindrome(sentence):
34     if not isinstance(sentence, str):
35         raise TypeError("Input must be a string.")
36
37     # Remove punctuation and spaces, convert to lowercase
38     cleaned = ""
39     for char in sentence:
40         if char.isalpha():
41             cleaned += char
42
43     # Check palindrome
44     return cleaned == cleaned[::-1]
45
46 print("---- Testing Palindrome Sentences ----")
47 for sentence in palindrom_sentences:
48     result = is_sentencePalindrome(sentence)
49     assert result is True
50     print(f"'{sentence}' = {result}")
51
52 print("\n---- Testing Non-Palindrome Sentences ----")
53 for sentence in nonPalindrome_sentences:
54     result = is_sentencePalindrome(sentence)
55     assert result is False
56     print(f"'{sentence}' = {result}")
57
58 print("\n---- Testing Invalid Inputs ----")
59 for value in invalid_inputs:
60     try:
61         is_sentencePalindrome(value)
62     except TypeError as error:
63         print(f"'{value}' = {error}")
64
65 print("All test cases passed successfully!")
66

```

```

1 #!/usr/bin/python
2
3 # Test cases for is_sentence_palindrome function
4
5 import string
6
7 palindrom_sentences = [
8     "A man a plan a canal Panama",
9     "Madam in Eden is Eden Madam",
10    "Was it a car or a cat I saw?",
11    "Dad I'm mad",
12    "Able was I ere I saw Elba",
13    "Never odd or even"
14]
15
16
17 # Non-palindrome sentences
18 nonPalindrome_sentences = [
19     "Hello world",
20     "Python programming",
21     "This is not a palindrome",
22     "Palindrome test case"
23]
24
25
26 # Invalid inputs
27 invalid_inputs = [
28     None,
29     12345,
30     ["not", "a", "string"]
31]
32
33 def is_sentencePalindrome(sentence):
34     if not isinstance(sentence, str):
35         raise TypeError("Input must be a string.")
36
37     # Remove punctuation and spaces, convert to lowercase
38     cleaned = ""
39     for char in sentence:
40         if char.isalpha():
41             cleaned += char
42
43     # Check palindrome
44     return cleaned == cleaned[::-1]
45
46 print("---- Testing Palindrome Sentences ----")
47 for sentence in palindrom_sentences:
48     result = is_sentencePalindrome(sentence)
49     assert result is True
50     print(f"'{sentence}' = {result}")
51
52 print("\n---- Testing Non-Palindrome Sentences ----")
53 for sentence in nonPalindrome_sentences:
54     result = is_sentencePalindrome(sentence)
55     assert result is False
56     print(f"'{sentence}' = {result}")
57
58 print("\n---- Testing Invalid Inputs ----")
59 for value in invalid_inputs:
60     try:
61         is_sentencePalindrome(value)
62     except TypeError as error:
63         print(f"'{value}' = {error}")
64
65 print("All test cases passed successfully!")
66

```

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

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure: RUN AND DEBUG > Python > newapp > __init__.py.
- Code Editor:** Displays a Python test script named `test_shoppingcart.py`. The code uses the `unittest` framework to test a `ShoppingCart` class. It includes tests for adding items, removing items, and handling invalid prices.
- Terminal:** Shows command-line output from running the tests. It includes messages like "Testing ShoppingCart Tests", "All test cases passed successfully!", and a summary of passed tests: "Add item = Passed", "Remove non-existing item = Passed", "Multiple addition = Passed", and "Invalid price handling = Passed".
- Output:** Shows standard output and error streams.
- Debug:** Shows a list of breakpoints and the current stack trace.
- Search:** Shows search results for "Devops".

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

- Correct format conversion for all valid inputs

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

The screenshot shows the Microsoft Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar labeled 'DevOps'. The left sidebar has sections for RUN AND DEBUG, VARIABLES, WATCH, and CALL STACK. The main editor area contains Python code for date conversion. The bottom right corner shows the terminal window with the command 'python -m unittest discover' and its output, which lists several test cases and their results.

```

File Edit Selection View Go Run Terminal Help < > DevOps
RUN AND DEBUG Python... newpy
VARIABLES
WATCH
CALL STACK
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
P C:\Users\Chinmay\Downloads\DevOps: 11.py; m "C:\Users\Chinmay\Downloads\DevOps\11.py" "C:\Users\Chinmay\Downloads\DevOps\11.exe" "C:\Users\Chinmay\Downloads\DevOps\11.log" -- "C:\Users\Chinmay\Downloads\DevOps\11.py"
11.py:1: in <module>
    from datetime import datetime
  ^
SyntaxError: invalid syntax
11.py:2: in <module>
    valid_test_cases = [
  ^
IndentationError: expected an indented block
11.py:3: in <module>
        "2023-10-10T10:10:10Z",
  ^
SyntaxError: invalid syntax
11.py:4: in <module>
        "1999-12-31T23:59:59Z",
  ^
SyntaxError: invalid syntax
11.py:5: in <module>
        "2024-01-29T10:10:10Z"
  ^
SyntaxError: invalid syntax
11.py:6: in <module>
    invalid_test_cases = [
  ^
IndentationError: expected an indented block
11.py:7: in <module>
        "15-10-2023",
  ^
SyntaxError: invalid syntax
11.py:8: in <module>
        "2023/01/01",
  ^
SyntaxError: invalid syntax
11.py:9: in <module>
        "2023-02-28",
  ^
SyntaxError: invalid syntax
11.py:10: in <module>
        "20231015",
  ^
SyntaxError: invalid syntax
11.py:11: in <module>
    None,
  ^
NameError: name 'None' is not defined
11.py:12: in <module>
    20231015
  ^
SyntaxError: invalid syntax
11.py:13: in <module>
    # Date Conversion Function
  ^
IndentationError: expected an indented block
11.py:14: in <module>
    def convert_date_format(date_str):
  ^
IndentationError: expected an indented block
11.py:15: in <module>
        if not isinstance(date_str, str):
  ^
IndentationError: expected an indented block
11.py:16: in <module>
            raise TypeError("Input must be a string.")
  ^
IndentationError: expected an indented block
11.py:17: in <module>
        try:
  ^
IndentationError: expected an indented block
11.py:18: in <module>
            date_obj = datetime.strptime(date_str, "%Y-%m-%d")
  ^
IndentationError: expected an indented block
11.py:19: in <module>
        except ValueError:
  ^
IndentationError: expected an indented block
11.py:20: in <module>
            raise ValueError("Date must be in 'YYYY-MM-DD' format and valid.")
  ^
IndentationError: expected an indented block
11.py:21: in <module>
        return date_obj.strftime("%G-%m-%d")
  ^
IndentationError: expected an indented block
11.py:22: in <module>
    # Running Valid Test Cases
  ^
IndentationError: expected an indented block
11.py:23: in <module>
    for date_input, expected_output in valid_test_cases.items():
  ^
IndentationError: expected an indented block
11.py:24: in <module>
        result = convert_date_format(date_input)
  ^
IndentationError: expected an indented block
11.py:25: in <module>
        assert result == expected_output, f"Expected {expected_output}, got {result}"
  ^
IndentationError: expected an indented block
11.py:26: in <module>
        print(f"(date_input={date_input}) = {result}")
  ^
IndentationError: expected an indented block
11.py:27: in <module>
    # Running Invalid Test Cases
  ^
IndentationError: expected an indented block
11.py:28: in <module>
    for date_input in invalid_test_cases:
  ^
IndentationError: expected an indented block
11.py:29: in <module>
        try:
  ^
IndentationError: expected an indented block
11.py:30: in <module>
            convert_date_format(date_input)
  ^
IndentationError: expected an indented block
11.py:31: in <module>
        except ValueError as e:
  ^
IndentationError: expected an indented block
11.py:32: in <module>
            print(f"(date_input={date_input}) = {e}")
  ^
IndentationError: expected an indented block
11.py:33: in <module>
        print("All test cases passed successfully!")
  ^
IndentationError: expected an indented block
  
```

TERMINAL

```

C:\Users\Chinmay\Downloads\DevOps: 11.py; m "C:\Users\Chinmay\Downloads\DevOps\11.py" "C:\Users\Chinmay\Downloads\DevOps\11.exe" "C:\Users\Chinmay\Downloads\DevOps\11.log" -- "C:\Users\Chinmay\Downloads\DevOps\11.py"
11.py:1: in <module>
    from datetime import datetime
  ^
SyntaxError: invalid syntax
11.py:2: in <module>
    valid_test_cases = [
  ^
IndentationError: expected an indented block
11.py:3: in <module>
        "2023-10-10T10:10:10Z",
  ^
SyntaxError: invalid syntax
11.py:4: in <module>
        "1999-12-31T23:59:59Z",
  ^
SyntaxError: invalid syntax
11.py:5: in <module>
        "2024-01-29T10:10:10Z"
  ^
SyntaxError: invalid syntax
11.py:6: in <module>
    invalid_test_cases = [
  ^
IndentationError: expected an indented block
11.py:7: in <module>
        "15-10-2023",
  ^
SyntaxError: invalid syntax
11.py:8: in <module>
        "2023/01/01",
  ^
SyntaxError: invalid syntax
11.py:9: in <module>
        "2023-02-28",
  ^
SyntaxError: invalid syntax
11.py:10: in <module>
        "20231015",
  ^
SyntaxError: invalid syntax
11.py:11: in <module>
    None,
  ^
NameError: name 'None' is not defined
11.py:12: in <module>
    20231015
  ^
SyntaxError: invalid syntax
11.py:13: in <module>
    # Date Conversion Function
  ^
IndentationError: expected an indented block
11.py:14: in <module>
    def convert_date_format(date_str):
  ^
IndentationError: expected an indented block
11.py:15: in <module>
        if not isinstance(date_str, str):
  ^
IndentationError: expected an indented block
11.py:16: in <module>
            raise TypeError("Input must be a string.")
  ^
IndentationError: expected an indented block
11.py:17: in <module>
        try:
  ^
IndentationError: expected an indented block
11.py:18: in <module>
            date_obj = datetime.strptime(date_str, "%Y-%m-%d")
  ^
IndentationError: expected an indented block
11.py:19: in <module>
        except ValueError:
  ^
IndentationError: expected an indented block
11.py:20: in <module>
            raise ValueError("Date must be in 'YYYY-MM-DD' format and valid.")
  ^
IndentationError: expected an indented block
11.py:21: in <module>
        return date_obj.strftime("%G-%m-%d")
  ^
IndentationError: expected an indented block
11.py:22: in <module>
    # Running Valid Test Cases
  ^
IndentationError: expected an indented block
11.py:23: in <module>
    for date_input, expected_output in valid_test_cases.items():
  ^
IndentationError: expected an indented block
11.py:24: in <module>
        result = convert_date_format(date_input)
  ^
IndentationError: expected an indented block
11.py:25: in <module>
        assert result == expected_output, f"Expected {expected_output}, got {result}"
  ^
IndentationError: expected an indented block
11.py:26: in <module>
        print(f"(date_input={date_input}) = {result}")
  ^
IndentationError: expected an indented block
11.py:27: in <module>
    # Running Invalid Test Cases
  ^
IndentationError: expected an indented block
11.py:28: in <module>
    for date_input in invalid_test_cases:
  ^
IndentationError: expected an indented block
11.py:29: in <module>
        try:
  ^
IndentationError: expected an indented block
11.py:30: in <module>
            convert_date_format(date_input)
  ^
IndentationError: expected an indented block
11.py:31: in <module>
        except ValueError as e:
  ^
IndentationError: expected an indented block
11.py:32: in <module>
            print(f"(date_input={date_input}) = {e}")
  ^
IndentationError: expected an indented block
11.py:33: in <module>
        print("All test cases passed successfully!")
  ^
IndentationError: expected an indented block
  
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

TERMINAL

```

C:\Users\Chinmay\Downloads\DevOps: 11.py; m "C:\Users\Chinmay\Downloads\DevOps\11.py" "C:\Users\Chinmay\Downloads\DevOps\11.exe" "C:\Users\Chinmay\Downloads\DevOps\11.log" -- "C:\Users\Chinmay\Downloads\DevOps\11.py"
11.py:1: in <module>
    from datetime import datetime
  ^
SyntaxError: invalid syntax
11.py:2: in <module>
    valid_test_cases = [
  ^
IndentationError: expected an indented block
11.py:3: in <module>
        "2023-10-10T10:10:10Z",
  ^
SyntaxError: invalid syntax
11.py:4: in <module>
        "1999-12-31T23:59:59Z",
  ^
SyntaxError: invalid syntax
11.py:5: in <module>
        "2024-01-29T10:10:10Z"
  ^
SyntaxError: invalid syntax
11.py:6: in <module>
    invalid_test_cases = [
  ^
IndentationError: expected an indented block
11.py:7: in <module>
        "15-10-2023",
  ^
SyntaxError: invalid syntax
11.py:8: in <module>
        "2023/01/01",
  ^
SyntaxError: invalid syntax
11.py:9: in <module>
        "2023-02-28",
  ^
SyntaxError: invalid syntax
11.py:10: in <module>
        "20231015",
  ^
SyntaxError: invalid syntax
11.py:11: in <module>
    None,
  ^
NameError: name 'None' is not defined
11.py:12: in <module>
    20231015
  ^
SyntaxError: invalid syntax
11.py:13: in <module>
    # Date Conversion Function
  ^
IndentationError: expected an indented block
11.py:14: in <module>
    def convert_date_format(date_str):
  ^
IndentationError: expected an indented block
11.py:15: in <module>
        if not isinstance(date_str, str):
  ^
IndentationError: expected an indented block
11.py:16: in <module>
            raise TypeError("Input must be a string.")
  ^
IndentationError: expected an indented block
11.py:17: in <module>
        try:
  ^
IndentationError: expected an indented block
11.py:18: in <module>
            date_obj = datetime.strptime(date_str, "%Y-%m-%d")
  ^
IndentationError: expected an indented block
11.py:19: in <module>
        except ValueError:
  ^
IndentationError: expected an indented block
11.py:20: in <module>
            raise ValueError("Date must be in 'YYYY-MM-DD' format and valid.")
  ^
IndentationError: expected an indented block
11.py:21: in <module>
        return date_obj.strftime("%G-%m-%d")
  ^
IndentationError: expected an indented block
11.py:22: in <module>
    # Running Valid Test Cases
  ^
IndentationError: expected an indented block
11.py:23: in <module>
    for date_input, expected_output in valid_test_cases.items():
  ^
IndentationError: expected an indented block
11.py:24: in <module>
        result = convert_date_format(date_input)
  ^
IndentationError: expected an indented block
11.py:25: in <module>
        assert result == expected_output, f"Expected {expected_output}, got {result}"
  ^
IndentationError: expected an indented block
11.py:26: in <module>
        print(f"(date_input={date_input}) = {result}")
  ^
IndentationError: expected an indented block
11.py:27: in <module>
    # Running Invalid Test Cases
  ^
IndentationError: expected an indented block
11.py:28: in <module>
    for date_input in invalid_test_cases:
  ^
IndentationError: expected an indented block
11.py:29: in <module>
        try:
  ^
IndentationError: expected an indented block
11.py:30: in <module>
            convert_date_format(date_input)
  ^
IndentationError: expected an indented block
11.py:31: in <module>
        except ValueError as e:
  ^
IndentationError: expected an indented block
11.py:32: in <module>
            print(f"(date_input={date_input}) = {e}")
  ^
IndentationError: expected an indented block
11.py:33: in <module>
        print("All test cases passed successfully!")
  ^
IndentationError: expected an indented block
  
```