

2303A51940

Batch: 25

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

File Edit Selection View Go Run Terminal Help ↺ 🔍 Devops

RUN AND DEBUG Python ... new.py

VARIABLES

```
1 #!/usr/bin/python
2
3 valid_emails = [
4     "user@example.com",
5     "user@sub.domain.com",
6     "value_123@domain.com",
7     "test_email@domain.org",
8     "user12@sub.domain.com"
9 ]
10
11 invalid_emails = [
12     "user@example.com",
13     "user@.com",
14     "user@.example.com",
15     "user@domain.com",
16     "user@domain.com",
17     "user@domain.com",
18     "user@.com",
19     "user@domain.com",
20     "."
21 ]
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
```

CALL STACK

BREAKPOINTS

Raised Exceptions
Unchecked Exceptions
User Caught Exceptions

POWER TOOLS Python Debugger Python File (Devops)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Chimaru\Downloads\Devops> cd "C:\Users\Chimaru\Downloads\Devops" & "C:\Users\Chimaru\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "C:\Users\Chimaru\vscode\extensions\ms-python.python\2625.18.0\Python\Python\Python Debug\new.py"
---- Running Valid Email Tests ----
user@example.com -> Valid
user@sub.domain.com -> Valid
value_123@domain.com -> Valid
test_email@domain.org -> Valid
user12@sub.domain.com -> Valid
---- Running Invalid Email Tests ----
user@.com -> Invalid
user@.example.com -> Invalid
user@domain.com -> Invalid
```

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

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

- File Explorer:** Shows a folder structure for "Devops" containing "new.py".
- Code Editor:** Displays the content of "new.py". The code defines a function `is_valid_email(email)` that checks if an email address is valid based on a regular expression pattern. It also includes a series of print statements and assertions to run tests on valid and invalid email addresses.
- Terminal:** At the bottom, the terminal window shows the command `py -m venv .` being run to create a virtual environment, followed by the output of running the test script `python -m unittest Devops/test_email.py`.
- Status Bar:** Shows the current file is "Python Debugger - Python File (Devops)" and the status "Ln 21 Col 30 Spaces: 4 UTR: 8 Python 3.11.12 [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 → 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

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

- File Explorer:** Shows a folder structure including 'myapp' and 'myapp.py'.
- Variables:** Shows variables like 'score' (80), 'grade' ('B'), and 'result' ('Grade: B').
- Search:** Shows search results for 'Score'.
- Call Stack:** Empty.
- Breakpoints:** Shows breakpoints for 'try' statements in the code.
- Terminal:** Displays the command 'cd "C:\Users\Chiranjeev\Downloads\myapp"' followed by the output of the Python script execution.
- Output:** Shows standard output from the script.
- Console:** Shows standard error from the script.
- TERMINAL:** Shows the command 'python myapp.py' being run.
- PROBLEMS:** Shows no problems.
- OUTPUT:** Shows no output.
- DEBUG CONSOLE:** Shows no output.
- TERMINAL PORTS:** Shows no output.

Code (myapp.py):

```
1 #!/usr/bin/python
2
3 valid_test_cases = [
4     100, "A",
5     90, "A",
6     80, "B",
7     70, "C",
8     50, "D",
9     30, "F",
10    0, "Boundary"
11 ]
12
13 def assign_grade(score):
14     if score < 0 or score > 100:
15         return "Score must be between 0 and 100."
16
17     if score == 100:
18         return "Grade: A"
19
20     elif 90 <= score <= 99:
21         return "Grade: A"
22
23     elif 80 <= score <= 89:
24         return "Grade: B"
25
26     elif 70 <= score <= 69:
27         return "Grade: C"
28
29     elif 60 <= score <= 59:
30         return "Grade: D"
31
32     else:
33         return "Grade: F"
34
35 print("---- Running Valid Test Cases ----")
36 for score, expected in valid_test_cases.items():
37     result = assign_grade(score)
38     assert result == expected
39     print("Score: {} | Grade: {} | Result: {}".format(score, grade, result))
40
41 print("---- Running Invalid Test Cases ----")
42 for score in invalid_test_cases:
43     try:
44         result = assign_grade(score)
45         print("Score: {} | Grade: {} | Result: {}".format(score, grade, result))
46     except Exception as e:
47         print("Score: {} | Grade: {} | Result: {} | Error: {}".format(score, grade, result, e))
48
49 print("---- All Test Cases Executed ----")
```

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 grades based on scores and for validating test cases.
- Terminal:** Shows the output of running the script with the command `python variables.py`. It prints "Running Valid Test Cases" and lists several test cases with their expected results (A+, A, B+, B, C+, C, D+, D, F) and actual results from the script.
- Output:** Shows the message "All valid test cases passed successfully!"
- Search:** Shows the search term "Devops".
- Call Stack:** Shows the call stack for the current file.
- Problems:** Shows no errors or warnings.
- Decorations:** Shows status icons for the file.
- Activity Bar:** Shows tabs for File, Edit, Selection, View, Go, Run, Terminal, Help, and a Python 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 various methods of a `ShoppingCart` class, including adding items, calculating total cost, removing items, and handling invalid prices.
- Terminal:** Shows command-line output from running the tests, indicating successful execution of all test cases.
- Output:** Shows logs for the Python debugger, including breakpoints and stack traces.
- Breakpoints:** A sidebar shows a list of breakpoints, including one at the start of the `__init__` method.

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 with the following details:

- File Explorer:** Shows a folder structure including 'BINARIES' and 'SEARCH'.
- Code Editor:** Displays the 'date_validator.py' file with the following content:

```
#!/usr/bin/env python3
# Date Validator Function
def validate_date(date_str):
    if not isinstance(date_str, str):
        raise TypeError("Input must be a string.")
    try:
        date_obj = datetime.strptime(date_str, "%Y-%m-%d")
    except ValueError:
        raise ValueError("Date must be in '%Y-%m-%d' format and valid.")
    return date_obj.strftime("%d-%m-%Y")

# Running Valid Test Cases
for date_input, expected_output in valid_test_cases.items():
    result = validate_date(date_input)
    assert result == expected_output
    print(f"(date_input) = {result}")

# Running Invalid Test Cases
for date_input in invalid_test_cases:
    try:
        validate_date(date_input)
    except ValueError as e:
        print(f"(date_input) = Error ({e})")

print("All test cases passed successfully!")

```

- Terminal:** Shows the command line output of running the script:

```
PS C:\Users\Chirantan\Downloads\Devops\4\4 ed c:\Users\Chirantan\Downloads\Devops\4\4 c:\Users\Chirantan\Downloads\Devops\4\4\date_validator.py
  File "c:\Users\Chirantan\Downloads\Devops\4\4\date_validator.py", line 13
    assert result == expected_output
    ^
AssertionError: None != None

Process finished with exit code 1
```

- PROBLEMS:** Shows no problems.
- OUTPUT:** Shows the terminal output above.
- TERMINAL:** Shows the command 'cd c:\Users\Chirantan\Downloads\Devops\4\4 ed c:\Users\Chirantan\Downloads\Devops\4\4\date_validator.py'.
- WORKS:** Shows the path 'C:\Users\Chirantan\Downloads\Devops\4\4'.
- Search Results:** Shows results for 'date_validator.py'.
- Call Stack:** Shows no entries.
- Measurements:** Shows 'Raised Exceptions' and 'Unc caught Exceptions'.
- Python Environment:** Shows 'Python 3.11.2 (Microsoft Store)'.