

# AI ASSISTED CODING

## ASSIGNMENT – 8.3

Name: Likhitha Pothunuri

Roll no: 2403A510D1

Batch: 05

Department: CSE

### **Task – 01**

#### **Prompt:**

Write a python code for valid email id where the it must contain @ and (.) character ,should not end with special characters , and should not contain multiple @.

Code generated:

```
python.py X
python.py > ...
1  def is_valid_email(email):
2      """
3      Validates an email address based on the following criteria:
4      - Must contain @ character
5      - Must contain . character
6      - Should not end with special characters
7      - Should not contain multiple @ characters
8      """
9
10     # Check if email is empty or None
11     if not email:
12         return False
13
14     # Check if email contains @ character
15     if '@' not in email:
16         return False
17
18     # Check if email contains . character
19     if '.' not in email:
20         return False
21
22     # Check if email contains multiple @ characters
23     if email.count('@') > 1:
24         return False
25
26     # Check if email ends with special characters
27     # Define special characters (excluding alphanumeric and dot)
28     special_chars = "!@#%&*()_+=[{}|;':\".,/<>?`~"
29     if email[-1] in special_chars:
30         return False
31
```

```
python.py X
python.py > test_email_validation
1  def is_valid_email(email):
2      return True
3
4  # Test the function with various email examples
5  def test_email_validation():
6      """Test function to demonstrate email validation"""
7
8      test_emails = [
9          "user@example.com",      # Valid
10         "test.email@domain.org", # Valid
11         "user123@test.co.uk",    # Valid
12         "invalid-email",         # Invalid - no @
13         "user@domain",          # Invalid - no .
14         "user@@domain.com",      # Invalid - multiple @
15         "user@domain.com!",      # Invalid - ends with special char
16         "user@domain.com@",      # Invalid - ends with @
17         "user@domain.com.",      # Invalid - ends with .
18         "",                     # Invalid - empty
19         "user@domain@com",       # Invalid - multiple @
20     ]
21
22     print("Email Validation Test Results:")
23     print("=" * 40)
24
25     for email in test_emails:
26         result = is_valid_email(email)
27         status = "✓ VALID" if result else "X INVALID"
28         print(f"{email:<25} -> {status}")
29
30 # Example usage
31 if __name__ == "__main__":
32     # Run the test
33     test_email_validation()
34
35     print("\n" + "=" * 40)
36     print("Interactive Email Validation")
37     print("=" * 40)
38
39     # Interactive testing
40     while True:

```

```
print("Email Validation Test Results:")
print("=" * 40)
# Ctrl+L to chat, Ctrl+K to generate
for email in test_emails:
    result = is_valid_email(email)
    status = "✓ VALID" if result else "X INVALID"
    print(f"{email:<25} -> {status}")

# Example usage
if __name__ == "__main__":
    # Run the test
    test_email_validation()

    print("\n" + "=" * 40)
    print("Interactive Email Validation")
    print("=" * 40)

    # Interactive testing
    while True:
        email = input("\nEnter an email to validate (or 'quit' to exit): ").strip()

        if email.lower() == 'quit':
            break

        if is_valid_email(email):
            print(f"✓ '{email}' is a valid email address!")
        else:
            print(f"X '{email}' is NOT a valid email address!")
```

## OUTPUT:

```
Problems Output Debug Console Terminal Ports
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
Email Validation Test Results:
=====
user@example.com      -> ✓ VALID
test.email@domain.org -> ✓ VALID
user123@test.co.uk   -> ✓ VALID
invalid-email         -> X INVALID
user@domain           -> X INVALID
user@domain.com       -> X INVALID
user@domain.com!      -> X INVALID
user@domain.com@      -> X INVALID
user@domain.com.       -> X INVALID
user@domain@com       -> X INVALID

=====
Interactive Email Validation
=====

Enter an email to validate (or 'quit' to exit): nadhiya
X 'nadhiya' is NOT a valid email address!

Enter an email to validate (or 'quit' to exit): nadhiya@gmail.com
✓ 'nadhiya@gmail.com' is a valid email address!

Enter an email to validate (or 'quit' to exit): exit
X 'exit' is NOT a valid email address!

Enter an email to validate (or 'quit' to exit):
```

### Observation:

1. **\*\*Simple Logic Flow\*\***: The function uses straightforward if-statements to check each requirement sequentially, making it easy to understand and debug.
2. **\*\*No External Dependencies\*\***: The code uses only built-in Python functions (`count()`, `in`, `not`) without any imports, keeping it lightweight and portable.

3. **\*\*Comprehensive Test Coverage\*\***: The test function includes both valid and invalid email examples covering all validation rules, making it easy to verify the function works correctly.

### **Task – 02**

#### **Prompt:**

write a python code to test cases for assigning grades ) where: 90-100: A, 80-89: B, 70-

79: C, 60-69: D, <60: F

- Include boundary values and invalid inputs (e.g., -5, 105, "eighty" ).

**Code generated:**

```
python.py > ...
1 def assign_grade(score):
2     """Assign grade based on score: 90-100: A, 80-89: B, 70-79: C, 60-69: D, <60: F"""
3     if not isinstance(score, (int, float)):
4         return "Invalid input: Score must be a number"
5
6     if score < 0 or score > 100:
7         return "Invalid input: Score must be between 0 and 100"
8
9     if score >= 90:
10        return "A"
11    elif score >= 80:
12        return "B"
13    elif score >= 70:
14        return "C"
15    elif score >= 60:
16        return "D"
17    else:
18        return "F"
19
20
21 def test_grade_assignment():
22     """Test cases for grade assignment function"""
23     test_cases = [
24         # Boundary values
25         (100, "A"), (90, "A"), (89, "B"), (80, "B"),
26         (79, "C"), (70, "C"), (69, "D"), (60, "D"), (59, "F"), (0, "F"),
27
28         # Invalid inputs
29         (-5, "Invalid input: Score must be between 0 and 100"),
30         (105, "Invalid input: Score must be between 0 and 100"),
31         ("eighty", "Invalid input: Score must be a number"),
32         ("90", "Invalid input: Score must be a number"),
33         (None, "Invalid input: Score must be a number"),
34
35         # Normal cases
36         (95, "A"), (85, "B"), (75, "C"), (65, "D"), (55, "F")
37     ]
38
39     print("Testing Grade Assignment Function")
40     print("=" * 60)
41
42     passed = 0
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
```

```
python.py
21 def test_grade_assignment():
22     """Test cases for grade assignment function"""
23     test_cases = [
24         # Boundary values
25         (100, "A"), (90, "A"), (89, "B"), (80, "B"),
26         (79, "C"), (70, "C"), (69, "D"), (60, "D"), (59, "F"), (0, "F"),
27
28         # Invalid inputs
29         (-5, "Invalid input: Score must be between 0 and 100"),
30         (105, "Invalid input: Score must be between 0 and 100"),
31         ("eighty", "Invalid input: Score must be a number"),
32         ("90", "Invalid input: Score must be a number"),
33         (None, "Invalid input: Score must be a number"),
34
35         # Normal cases
36         (95, "A"), (85, "B"), (75, "C"), (65, "D"), (55, "F")
37     ]
38
39     print("Testing Grade Assignment Function")
40     print("=" * 60)
41
42     passed = 0
43     total = len(test_cases)
44
45     for score, expected in test_cases:
46         result = assign_grade(score)
47         status = "PASS" if result == expected else "FAIL"
48         if status == "PASS":
49             passed += 1
50
51         print(f"Score: {score!r}>8} | Expected: {expected!r:>30} | Got: {result!r:>30} | {status}")
52
53     print("=" * 60)
54     print(f"Test Results: {passed}/{total} tests passed")
55     return passed == total
56
57
58 if __name__ == "__main__":
59     test_grade_assignment()
60
```

OUTPUT:

```
45     for score, expected in test_cases:
46         result = assign_grade(score)

Problems Output Debug Console Terminal Ports
Testing Grade Assignment Function
=====
Score: 100 | Expected: 'A' | Got: 'A' | PASS
Score: 90 | Expected: 'A' | Got: 'A' | PASS
Score: 89 | Expected: 'B' | Got: 'B' | PASS
Score: 80 | Expected: 'B' | Got: 'B' | PASS
Score: 79 | Expected: 'C' | Got: 'C' | PASS
Score: 70 | Expected: 'C' | Got: 'C' | PASS
Score: 69 | Expected: 'D' | Got: 'D' | PASS
Score: 60 | Expected: 'D' | Got: 'D' | PASS
Score: 59 | Expected: 'F' | Got: 'F' | PASS
Score: 0 | Expected: 'F' | Got: 'F' | PASS
Score: -5 | Expected: 'Invalid input: Score must be between 0 and 100' | Got: 'Invalid input: Score must be between 0 and 100' | PASS
Score: 105 | Expected: 'Invalid input: Score must be between 0 and 100' | Got: 'Invalid input: Score must be between 0 and 100' | PASS
Score: 'eighty' | Expected: 'Invalid input: Score must be a number' | Got: 'Invalid input: Score must be a number' | PASS
Score: '90' | Expected: 'Invalid input: Score must be a number' | Got: 'Invalid input: Score must be a number' | PASS
Score: None | Expected: 'Invalid input: Score must be a number' | Got: 'Invalid input: Score must be a number' | PASS
Score: 95 | Expected: 'A' | Got: 'A' | PASS
Score: 85 | Expected: 'B' | Got: 'B' | PASS
Score: 75 | Expected: 'C' | Got: 'C' | PASS
Score: 65 | Expected: 'D' | Got: 'D' | PASS
Score: 55 | Expected: 'F' | Got: 'F' | PASS
=====
Test Results: 20/20 tests passed
PS C:\Users\nadhi\OneDrive\Desktop\AID>
```

## Observation:

The code effectively tests all boundary conditions (90, 89, 80, 79, etc.) and invalid inputs (negative numbers, over 100, non-numeric strings) with clear pass/fail reporting, making it a comprehensive test suite for grade assignment logic.

The formatting improvements you made (using `!r` for `repr()` and adjusting column widths) enhance readability by properly displaying string inputs with quotes and aligning the output columns better for visual clarity.

## Task -03

### Prompt:

write a python code for a sentence palindrome where test cases for `is_sentence_palindrome(sentence)` (ignores case, spaces, and punctuation).

- Example:

"A man a plan a canal Panama" → True.

### Code generated:

```
python.py X
python.py > ...
1 import re
2
3 def is_sentence_palindrome(sentence):
4     """
5     Check if a sentence is a palindrome, ignoring case, spaces, and punctuation.
6
7     Args:
8         sentence (str): The sentence to check
9
10    Returns:
11        bool: True if the sentence is a palindrome, False otherwise
12    """
13    # Remove all non-alphabetic characters and convert to lowercase
14    cleaned = re.sub(r'[^\a-zA-Z]', '', sentence.lower())
15
16    # Check if the cleaned string is equal to its reverse
17    return cleaned == cleaned[::-1]
18
19 def test_is_sentence_palindrome():
20     """Test cases for is_sentence_palindrome function"""
21
22     # Test case 1: Classic palindrome
23     assert is_sentence_palindrome("A man a plan a canal Panama") == True
24
25     # Test case 2: Simple palindrome
26     assert is_sentence_palindrome("racecar") == True
27
28     # Test case 3: Palindrome with punctuation
29     assert is_sentence_palindrome("Was it a car or a cat I saw?") == True
30
31     # Test case 4: Not a palindrome
32     assert is_sentence_palindrome("Hello world") == False
33
34     # Test case 5: Empty string
35     assert is_sentence_palindrome("") == True
36
37     # Test case 6: Single character
38     assert is_sentence_palindrome("a") == True
39
40     # Test case 7: Palindrome with numbers and special characters
41     assert is_sentence_palindrome("A1B2C3C2B1A") == True
42
43     # Test case 8: Case insensitive test
44     assert is_sentence_palindrome("Madam") == True
```

```
python.py > ...
19 def test_is_sentence_palindrome():
20     assert is_sentence_palindrome("A man a plan a canal Panama") == True
21
22     # Test case 2: Simple palindrome
23     assert is_sentence_palindrome("racecar") == True
24
25     # Test case 3: Palindrome with punctuation
26     assert is_sentence_palindrome("Was it a car or a cat I saw?") == True
27
28     # Test case 4: Not a palindrome
29     assert is_sentence_palindrome("Hello world") == False
30
31     # Test case 5: Empty string
32     assert is_sentence_palindrome("") == True
33
34     # Test case 6: Single character
35     assert is_sentence_palindrome("a") == True
36
37     # Test case 7: Palindrome with numbers and special characters
38     assert is_sentence_palindrome("A1B2C3C2B1A") == True
39
40     # Test case 8: Case insensitive test
41     assert is_sentence_palindrome("Madam") == True
42
43     # Test case 9: Complex palindrome with mixed punctuation
44     assert is_sentence_palindrome("A Santa at NASA!") == True
45
46     # Test case 10: Non-palindrome with similar characters
47     assert is_sentence_palindrome("Palindrome") == False
48
49     print("All test cases passed!")
50
51 if __name__ == "__main__":
52     # Run the test cases
53     test_is_sentence_palindrome()
54
55     # Example usage
56     test_sentence = "A man a plan a canal Panama"
57     result = is_sentence_palindrome(test_sentence)
58     print(f"'{test_sentence}' is a palindrome: {result}")
```

Output:

```
Score: 55 | Expected: 'F' | Got: 'F' | PASS
=====
Test Results: 20/20 tests passed
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
◆ All test cases passed!
  'A man a plan a canal Panama' is a palindrome: True
○ PS C:\Users\nadhi\OneDrive\Desktop\AI> []
```

### Observation:

1. **Efficient cleaning approach:** The regex `re.sub(r'^a-zA-Z', '', sentence.lower())` removes all non-alphabetic characters in one operation, making it more efficient than multiple string operations.
2. **Pythonic palindrome check:** Using `cleaned == cleaned[::-1]` leverages Python's slice notation to reverse the string, which is both readable and performant compared

### Task – 04

#### Prompt:

write a python code to nerate test cases for a ShoppingCart class (add\_item,remove\_item, total) where code should contain Add\_item(name,orice),Remove\_item(name),total\_cost().

#### Code generated:



```
python.py > ...
1 class ShoppingCart:
2     def __init__(self):
3         self.items = {}
4
5     def add_item(self, name, price):
6         if not isinstance(name, str) or not isinstance(price, (int, float)) or price < 0:
7             return "Invalid input"
8         self.items[name] = self.items.get(name, 0) + price
9         return f"Added {name} for {price}"
10
11     def remove_item(self, name):
12         if name in self.items:
13             del self.items[name]
14             return f"Removed {name}"
15         return "Item not found"
16
17     def total_cost(self):
18         return sum(self.items.values())
19
20
21 def test_shopping_cart():
22     cart = ShoppingCart()
23     test_cases = [
24         ("add", "Apple", 50, "Added Apple for 50"),
25         ("add", "Banana", 30, "Added Banana for 30"),
26         ("add", "Apple", 50, "Added Apple for 50"),
27         ("remove", "Banana", None, "Removed Banana"),
28         ("remove", "Mango", None, "Item not found"),
29         ("add", "Orange", -10, "Invalid input"),
30         ("add", 123, 40, "Invalid input"),
31     ]
32
33     print("Testing ShoppingCart")
```

```
python.py X
python.py > ...
21 def test_shopping_cart():
22     # Add test cases here, e.g., ("add", "Apple", 50, "Added Apple for 50"),
23     ("remove", "Banana", None, "Removed Banana"),
24     ("remove", "Mango", None, "Item not found"),
25     ("add", "Orange", -10, "Invalid input"),
26     ("add", 123, 40, "Invalid input"),
27 ]
28
29 print("Testing ShoppingCart")
30 print("=" * 50)
31 for action, name, price, expected in test_cases:
32     if action == "add":
33         result = cart.add_item(name, price)
34     elif action == "remove":
35         result = cart.remove_item(name)
36     print(f"{action.title()} {name}: Expected={expected}, Got={result},")
37     print("PASS" if result == expected else "FAIL")
38
39 # Final check for total cost
40 total = cart.total_cost()
41 print("=" * 50)
42 print(f"Total cost Expected=100, Got={total}, PASS" if total == 100 else "FAIL")
43
44
45 if __name__ == "__main__":
46     test_shopping_cart()
47
48
49
50
51 Ctrl+L to chat, Ctrl+K to generate
```

Output:

```
OK
PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
Testing ShoppingCart
=====
Add 'Apple': Expected='Added Apple for 50', Got='Added Apple for 50', PASS
Add 'Banana': Expected='Added Banana for 30', Got='Added Banana for 30', PASS
Add 'Apple': Expected='Added Apple for 50', Got='Added Apple for 50', PASS
Remove 'Banana': Expected='Removed Banana', Got='Removed Banana', PASS
Remove 'Mango': Expected='Item not found', Got='Item not found', PASS
Add 'Orange': Expected='Invalid input', Got='Invalid input', PASS
Add 123: Expected='Invalid input', Got='Invalid input', PASS
=====
Total cost Expected=100, Got=100 PASS
PS C:\Users\nadhi\OneDrive\Desktop\AI> 
```

### Observation:

The code correctly tests all functionalities of the **ShoppingCart** class, including valid/invalid additions, removals, and total cost calculation.

It handles edge cases well and produces clear PASS/FAIL outputs for each test case.

### Task – 05:

#### Prompt:

write a python code to test cases for `convert_date_format(date_str)` to switch from "YYYY-MM-DD" to "DD-MM-YYYY".

Example: "2023-10-15" → "15-10-2023" make sure user should give input.

### Code generated:

```

python.py > ...
1 def convert_date_format(date_str):
2     """
3     Convert date from YYYY-MM-DD format to DD-MM-YYYY format
4     """
5     try:
6         # Split the date string by '-'
7         parts = date_str.split('-')
8
9         # Check if we have exactly 3 parts
10        if len(parts) != 3:
11            raise ValueError("Invalid date format")
12
13        year, month, day = parts
14
15        # Validate that each part has correct length
16        if len(year) != 4 or len(month) != 2 or len(day) != 2:
17            raise ValueError("Invalid date format")
18
19        # Return in DD-MM-YYYY format
20        return f"{day}-{month}-{year}"
21
22    except Exception as e:
23        return f"Error: {str(e)}"
24
25 def test_convert_date_format():
26     """Test cases for convert_date_format function"""
27     test_cases = [
28         ("2023-10-15", "15-10-2023"),
29         ("2024-01-01", "01-01-2024"),
30         ("2023-12-31", "31-12-2023"),
31         ("2024-02-29", "29-02-2024"), # Leap year
32         ("2023-06-05", "05-06-2023"),
33         ("invalid-date", "Error: Invalid date format").

```

```

python.py > ...
25 def test_convert_date_format():
26     """Test cases for convert_date_format function"""
27     test_cases = [
28         ("2023-10-15", "15-10-2023"),
29         ("2024-01-01", "01-01-2024"),
30         ("2023-12-31", "31-12-2023"),
31         ("2024-02-29", "29-02-2024"), # Leap year
32         ("2023-06-05", "05-06-2023"),
33         ("invalid-date", "Error: Invalid date format"),
34         ("2023-13-01", "Error: Invalid date format"), # Invalid month
35         ("2023-01-32", "Error: Invalid date format"), # Invalid day
36     ]
37
38     print("Testing convert_date_format function:")
39     print("=" * 50)
40
41     for i, (input_date, expected) in enumerate(test_cases, 1):
42         result = convert_date_format(input_date)
43         status = "✓ PASS" if result == expected else "X FAIL"
44         print(f"Test {i}: {input_date} → {result} [{status}]")
45         if result != expected:
46             print(f"    Expected: {expected}")
47
48     def main():
49         """Main function with user input"""
50         print("Date Format Converter")
51         print("Convert from YYYY-MM-DD to DD-MM-YYYY")
52         print("=" * 40)
53
54         # Run test cases
55         test_convert_date_format()
56
57         print("\n" + "=" * 10)

```

Problems Output Debug Console Terminal Ports

```
python.py X
python.py > ...
46 |         print(f"Expected: {expected}")
47 |
48 | def main():
49 |     """Main Function with user input"""
50 |     print("Date Format Converter")
51 |     print("Convert from YYYY-MM-DD to DD-MM-YYYY")
52 |     print("=" * 40)
53 |
54 |     # Run test cases
55 |     test_convert_date_format()
56 |
57 |     print("\n" + "=" * 40)
58 |     print("User Input Testing:")
59 |
60 |     while True:
61 |         user_input = input("\nEnter a date (YYYY-MM-DD) or 'quit' to exit: ")
62 |
63 |         if user_input.lower() == 'quit':
64 |             print("Goodbye!")
65 |             break
66 |
67 |         result = convert_date_format(user_input)
68 |         print(f"Converted: {result}")
69 |
70 | if __name__ == "__main__":
71 |     main()
72 |
```

## Output:

```
4.3...
docx Total cost Expected=100, Got=100 PASS
pdf PS C:\Users\nadhi\OneDrive\Desktop\AI> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Desktop/AI/python.py
Date Format Converter
Convert from YYYY-MM-DD to DD-MM-YYYY
=====
Testing convert_date_format function:
=====
Test 1: 2023-10-15 → 15-10-2023 [✓ PASS]
Test 2: 2024-01-01 → 01-01-2024 [✓ PASS]
Test 3: 2023-12-31 → 31-12-2023 [✓ PASS]
Test 4: 2024-02-29 → 29-02-2024 [✓ PASS]
Test 5: 2023-06-05 → 05-06-2023 [✓ PASS]
Test 6: invalid-date → Error: Invalid date format [✓ PASS]
Test 7: 2023-13-01 → 01-13-2023 [X FAIL]
Expected: Error: Invalid date format
Test 8: 2023-01-32 → 32-01-2023 [X FAIL]
Expected: Error: Invalid date format
=====
User Input Testing:

Enter a date (YYYY-MM-DD) or 'quit' to exit: 2015-01-22
Converted: 22-01-2015

Enter a date (YYYY-MM-DD) or 'quit' to exit: 2023-03-12
Converted: 12-03-2023

Enter a date (YYYY-MM-DD) or 'quit' to exit: exit
Converted: Error: Invalid date format

Enter a date (YYYY-MM-DD) or 'quit' to exit: 
```

## Observation:

The code successfully converts dates from **YYYY-MM-DD** to **DD-MM-YYYY** format and includes both automated test cases and interactive user input.

However, it only checks string structure, not actual calendar validity (e.g., invalid months/days may pass if formatted correctly).