

Assignment-8.3

Task1:

code for the function `is_valid_email(email)` to check valid and invalid email formats, then implement the function so that it contains exactly one @, includes ., does not start or end with special characters, and passes all test cases.user input and check correct or not

Code:

```
import re

def is_valid_email(email):

    # Check if email contains exactly one @
    if email.count('@') != 1 or email.count('@')>1:
        return False

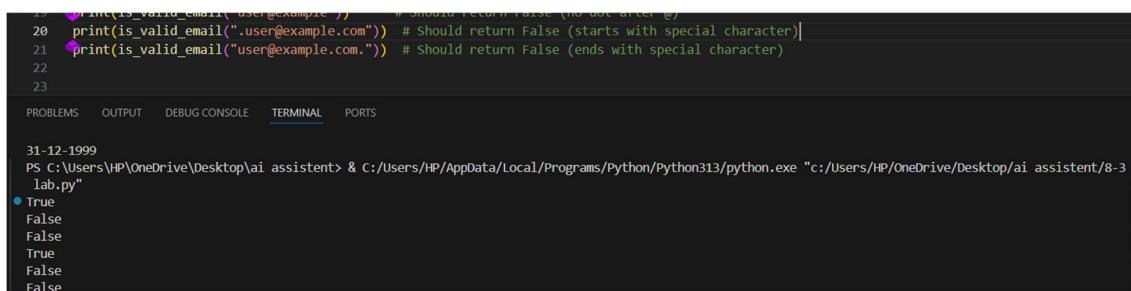
    # Check if email starts or ends with special characters
    if re.match(r'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\$', email):
        return False

    if "." == email[0] or "." == email[-1]:
        return False

    return True

# Test cases
print(is_valid_email("test@example.com")) # Should return True
print(is_valid_email("invalid.email")) # Should return False (no @)
print(is_valid_email("user@@example.com")) # Should return False (multiple @)
print(is_valid_email("user@example")) # Should return False (no dot after @)
print(is_valid_email(".user@example.com")) # Should return False (starts with special character)
print(is_valid_email("user@example.com.")) # Should return False (ends with special character)
```

Output:



```
1> print(is_valid_email("user@example")) # Should return False (no dot after @)
20 print(is_valid_email(".user@example.com")) # Should return False (starts with special character)
21 print(is_valid_email("user@example.com.")) # Should return False (ends with special character)
22
23
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
31-12-1999
PS C:\Users\HP\OneDrive\Desktop\ai assistent> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/HP/OneDrive/Desktop/ai assistent/8-3
lab.py"
● True
False
False
True
False
False
```

Analysis:

The function checks the user-entered email by ensuring it contains exactly one @, includes a dot, and does not start or end with special characters. Invalid email formats are rejected, and valid emails return True.

Task 2:

code for assign_grade(score) covering all grade ranges, boundary values, and invalid inputs, then implement the function using conditions so that all test cases pass and invalid inputs are handled properly.

Code:

```
import unittest

def assign_grade(score):
    if isinstance(score, (int, float)):
        if 90 <= score <= 100:
            return 'A'
        elif 80 <= score < 90:
            return 'B'
        elif 70 <= score < 80:
            return 'C'
        elif 60 <= score < 70:
            return 'D'
        elif score < 60:
            return 'F'
    return "Invalid input"
```

```

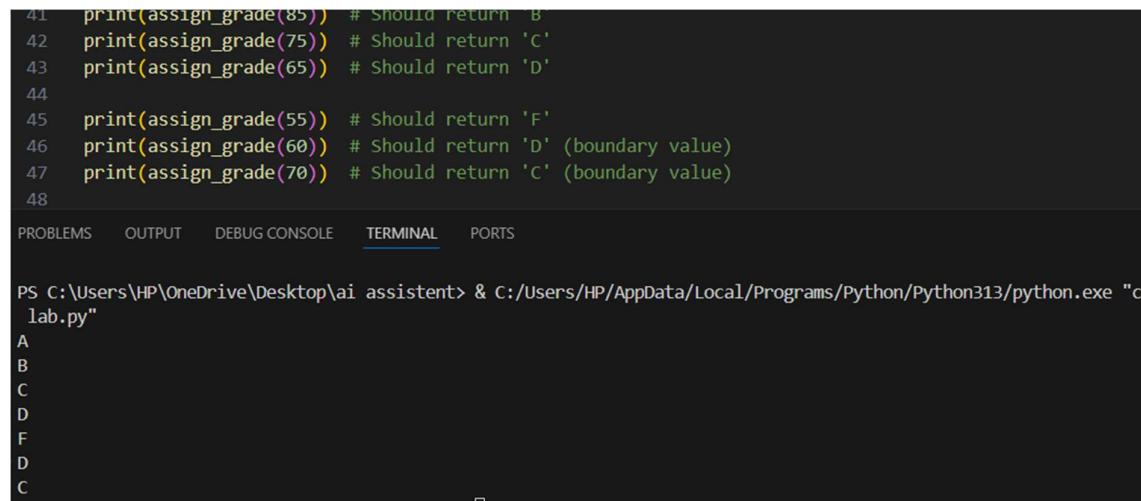
# user input and test cases

print(assign_grade(95)) # Should return 'A'
print(assign_grade(85)) # Should return 'B'
print(assign_grade(75)) # Should return 'C'
print(assign_grade(65)) # Should return 'D'

print(assign_grade(55)) # Should return 'F'
print(assign_grade(60)) # Should return 'D' (boundary value)
print(assign_grade(70)) # Should return 'C' (boundary value)

```

Output:



```

41 print(assign_grade(85)) # Should return 'B'
42 print(assign_grade(75)) # Should return 'C'
43 print(assign_grade(65)) # Should return 'D'
44
45 print(assign_grade(55)) # Should return 'F'
46 print(assign_grade(60)) # Should return 'D' (boundary value)
47 print(assign_grade(70)) # Should return 'C' (boundary value)
48

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\HP\OneDrive\Desktop\ai assistent> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c
lab.py"
A
B
C
D
F
D
C

```

Analysis:

The function assigns grades based on the user's score using conditional statements. Boundary values are handled correctly, and invalid inputs return an error message without crashing.

Task 3:

Code for `is_sentence_palindrome(sentence)` by ignoring case, spaces, and punctuation, then implement the function to correctly identify palindromic and non-palindromic sentences.

Code:

```

import unittest

import re

```

```

def is_sentence_palindrome(sentence):

    # Remove non-alphanumeric characters and convert to lowercase
    cleaned_sentence = re.sub(r'[^A-Za-z0-9]', " ", sentence).lower()

    # Check if the cleaned sentence is equal to its reverse
    return cleaned_sentence == cleaned_sentence[::-1]

# Test cases

print(is_sentence_palindrome("A man a plan a canal Panama")) # Should return True
print(is_sentence_palindrome("Was it a car or a cat I saw?")) # Should return True
print(is_sentence_palindrome("Hello, World!")) # Should return False
print(is_sentence_palindrome("No 'x' in Nixon")) # Should return True

```

Output:

```

59  print(is_sentence_palindrome("A man a plan a canal Panama")) # Should return True
60  print(is_sentence_palindrome("Was it a car or a cat I saw?")) # Should return True
61  print(is_sentence_palindrome("Hello, World!")) # Should return False
62  print(is_sentence_palindrome("No 'x' in Nixon")) # Should return True
63
64
65  # # Task 4: Code for the ShoppingCart class to test adding items, removing items, total cost calculati
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\HP\OneDrive\Desktop\ai_assistent> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe "c:/U
lab.py"
True
True
False
True
○ PS C:\Users\HP\OneDrive\Desktop\ai_assistent> []

```

Analysis:

The function removes spaces, punctuation, and ignores case before checking if the sentence is equal to its reverse. It correctly identifies palindromic and non-palindromic sentences.

Task 4:

Code for the ShoppingCart class to test adding items, removing items, total cost calculation, and empty cart cases, then implement the class so that all test cases pass successfully.

Code:

```

import unittest

class ShoppingCart:

    def __init__(self):

```

```

self.items = {}

def add_item(self, name, price):
    if name in self.items:
        self.items[name] += price
    else:
        self.items[name] = price

def remove_item(self, name):
    if name in self.items:
        del self.items[name]

def total_cost(self):
    return sum(self.items.values())

# Test cases
cart = ShoppingCart()
cart.add_item("Apple", 1.00)
cart.add_item("Banana", 0.50)
print(cart.total_cost()) # Should return 1.50
cart.add_item("Apple", 1.00)
print(cart.total_cost()) # Should return 2.50 (Apple added twice)
cart.remove_item("Banana")
print(cart.total_cost()) # Should return 2.00 (Banana removed)
cart.remove_item("Orange") # Removing an item that doesn't exist should not affect total cost
print(cart.total_cost()) # Should still return 2.00 (no change)

```

Output:

The screenshot shows a code editor interface with a dark theme. At the top, there is a code block containing Python test cases for a `ShoppingCart` class. The code includes adding items like "Apple" and "Banana" at different prices, calculating the total cost, removing items, and handling edge cases. Below the code, there is a terminal window showing the execution of the test script. The terminal output shows the expected results for each test case, such as 1.5, 2.5, and 2.0, indicating that the `total_cost` method works correctly for different scenarios.

```
83 # Test cases
84 cart = ShoppingCart()
85 cart.add_item("Apple", 1.00)
86 cart.add_item("Banana", 0.50)
87 print(cart.total_cost()) # Should return 1.50
88 cart.add_item("Apple", 1.00)
89 print(cart.total_cost()) # Should return 2.50 (Apple added twice)
90 cart.remove_item("Banana")
91 print(cart.total_cost()) # Should return 2.00 (Banana removed)
92 cart.remove_item("Orange") # Removing an item that doesn't exist should not affect total
93 print(cart.total_cost()) # Should still return 2.00 (no change)
94
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
True
PS C:\Users\HP\OneDrive\Desktop\ai_assistent> & C:/Users/HP/AppData/Local/Programs/Python/Python313/lab.py"
● 1.5
2.5
2.0
2.0
○ PS C:\Users\HP\OneDrive\Desktop\ai_assistent>
```

Analysis:

The `ShoppingCart` class allows adding and removing items and calculates the total cost accurately. It safely handles empty cart and invalid removal cases.

Task 5:

code for the function `convert_date_format(date_str)` to convert date from "YYYY-MM-DD" to "DD-MM-YYYY" format, then implement the function to handle valid and invalid date formats correctly.

Code:

```
import unittest

def convert_date_format(date_str):
    try:
        year, month, day = date_str.split('-')
        if len(year) == 4 and len(month) == 2 and len(day) == 2:
            return f"{day}-{month}-{year}"
        else:
            return "Invalid date format"
    except ValueError:
        return "Invalid date format"

# Test cases
```

```

print(convert_date_format("2023-10-15")) # Should return "15-10-2023"

print(convert_date_format("2023/10/15")) # Should return "Invalid date format" (wrong
separator)

print(convert_date_format("15-10-2023")) # Should return "Invalid date format" (wrong
order)

print(convert_date_format("2023-10-5")) # Should return "Invalid date format" (day not
two digits)

#valid date formats

print(convert_date_format("2024-01-01")) # Should return "01-01-2024"

print(convert_date_format("1999-12-31")) # Should return "31-12-1999"

```

Output:

```

108
109 print(convert_date_format("2023-10-15")) # Should return "15-10-2023"
110 print(convert_date_format("2023/10/15")) # Should return "Invalid date format" (wrong separator)
111 print(convert_date_format("15-10-2023")) # Should return "Invalid date format" (wrong order)
112 print(convert_date_format("2023-10-5")) # Should return "Invalid date format" (day not two digits)
113 #valid date formats
114 print(convert_date_format("2024-01-01")) # Should return "01-01-2024"
115 print(convert_date_format("1999-12-31")) # Should return "31-12-1999"
116
117
118
119
120
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
lab.py"
15-10-2023
Invalid date format
Invalid date format
Invalid date format
01-01-2024

```

Analysis:

The function converts dates from YYYY-MM-DD to DD-MM-YYYY format and returns an error message for invalid inputs. Exception handling ensures reliable execution.