

# ASSIGNMENT - 08

**Name :** Aravind Reddy

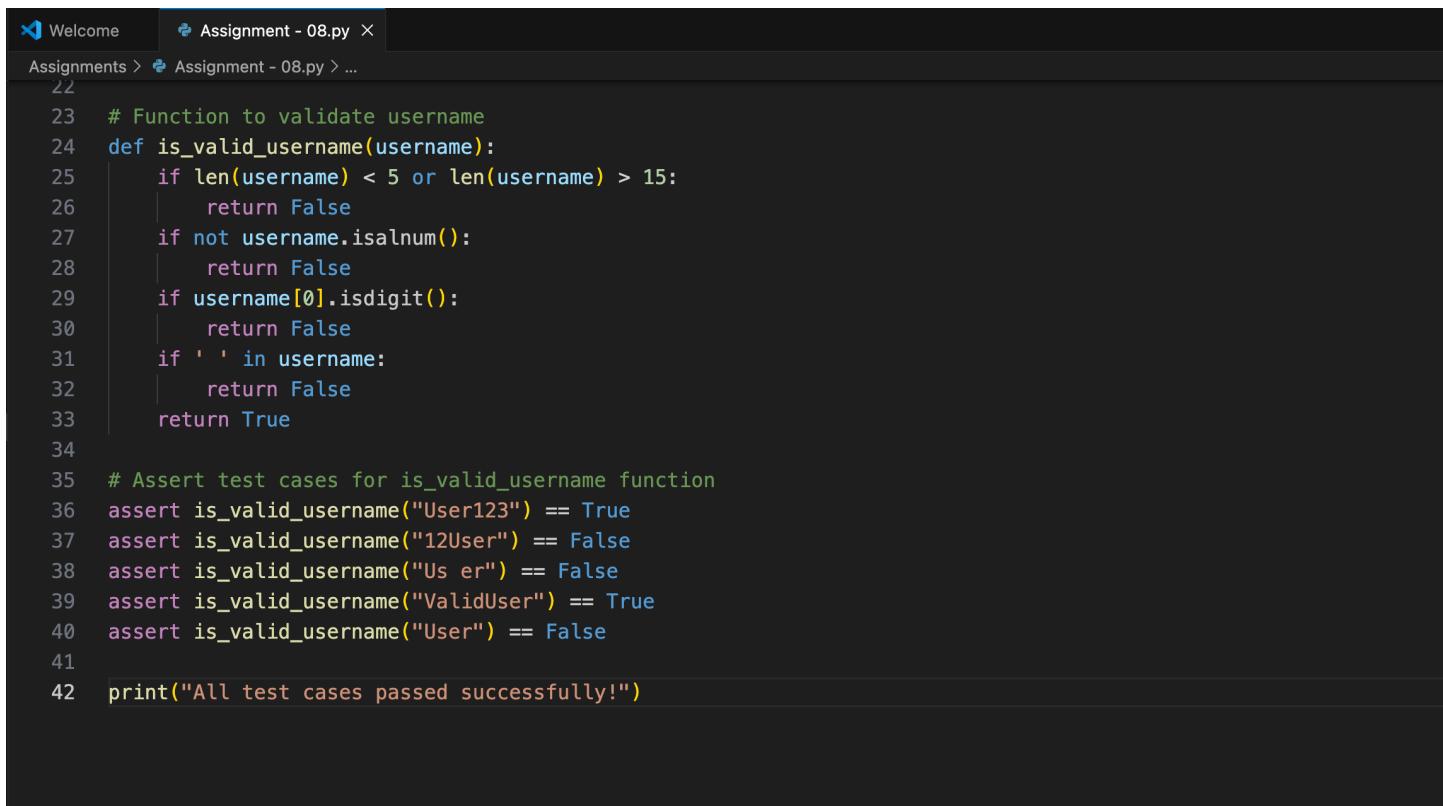
**Hall Ticket No :** 2303a51027

**Batch No :** 01

**Course Title :** AI Assisted Coding

**Instructor's Name :** Mr. S Naresh Kumar

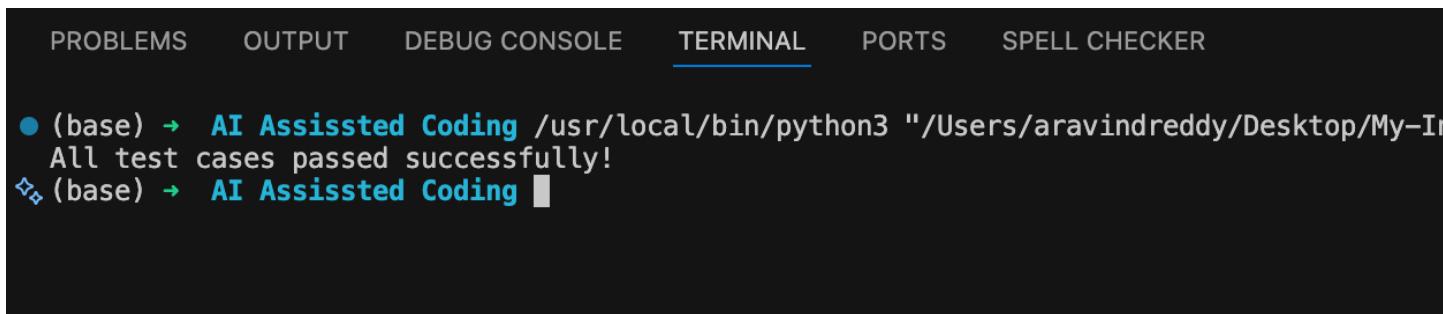
## Task 1 : Username Validator – Apply AI in Authentication Context



The screenshot shows a code editor window with a dark theme. At the top, there are tabs for "Welcome" and "Assignment - 08.py". Below the tabs, the file path "Assignments > Assignment - 08.py > ..." is visible. The code itself is a Python script named "Assignment - 08.py". It contains a function "is\_valid\_username" that checks if a given username is valid based on several criteria: length (between 5 and 15 characters), alphanumeric characters, starting with a digit, and no consecutive spaces. It also includes a series of assertions to test the function with various inputs, all of which pass successfully.

```
22
23 # Function to validate username
24 def is_valid_username(username):
25     if len(username) < 5 or len(username) > 15:
26         return False
27     if not username.isalnum():
28         return False
29     if username[0].isdigit():
30         return False
31     if ' ' in username:
32         return False
33     return True
34
35 # Assert test cases for is_valid_username function
36 assert is_valid_username("User123") == True
37 assert is_valid_username("12User") == False
38 assert is_valid_username("Us er") == False
39 assert is_valid_username("ValidUser") == True
40 assert is_valid_username("User") == False
41
42 print("All test cases passed successfully!")
```

## Output :



The screenshot shows a terminal window with a dark theme. At the top, there are tabs for "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "TERMINAL", "PORTS", and "SPELL CHECKER". The "TERMINAL" tab is currently selected. The terminal output shows the command "(base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR" and the response "All test cases passed successfully!". The prompt "(base) → AI Assissted Coding" is shown again at the bottom.

```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR"
All test cases passed successfully!
❖ (base) → AI Assissted Coding
```

## Task 2 : Even–Odd & Type Classification – Apply AI for Robust Input Handling

A screenshot of a code editor window titled "Assignment - 08.py". The code defines a function to classify integers as Zero, Even, Odd, or Invalid Input. It includes several assert statements to verify the correctness of the classification logic. The code ends with a print statement confirming all test cases passed.

```
62
63 # Function to classify value
64 def classify_value(x):
65     if isinstance(x, int):
66         if x == 0:
67             return "Zero"
68         elif x % 2 == 0:
69             return "Even"
70         else:
71             return "Odd"
72     else:
73         return "Invalid Input"
74
75 # Assert test cases for classify_value function
76 assert classify_value(8) == "Even"
77 assert classify_value(7) == "Odd"
78 assert classify_value("abc") == "Invalid Input"
79 assert classify_value(0) == "Zero"
80
81 print("All test cases passed successfully!")
```

**OutPut :**

A screenshot of a terminal window showing the output of running the Python script. The terminal shows the command being run, the output message indicating all test cases passed successfully, and the prompt again.

```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
All test cases passed successfully!
❖ (base) → AI Assissted Coding █
```

### Task 3 : Palindrome Checker – Apply AI for String Normalization

A screenshot of a code editor window titled "Assignment - 08.py". The code defines a function to check if a string is a palindrome by removing spaces and punctuation, and converting it to lowercase. It includes several assert statements to verify the correctness of the palindrome detection logic. The code ends with a print statement confirming all test cases passed.

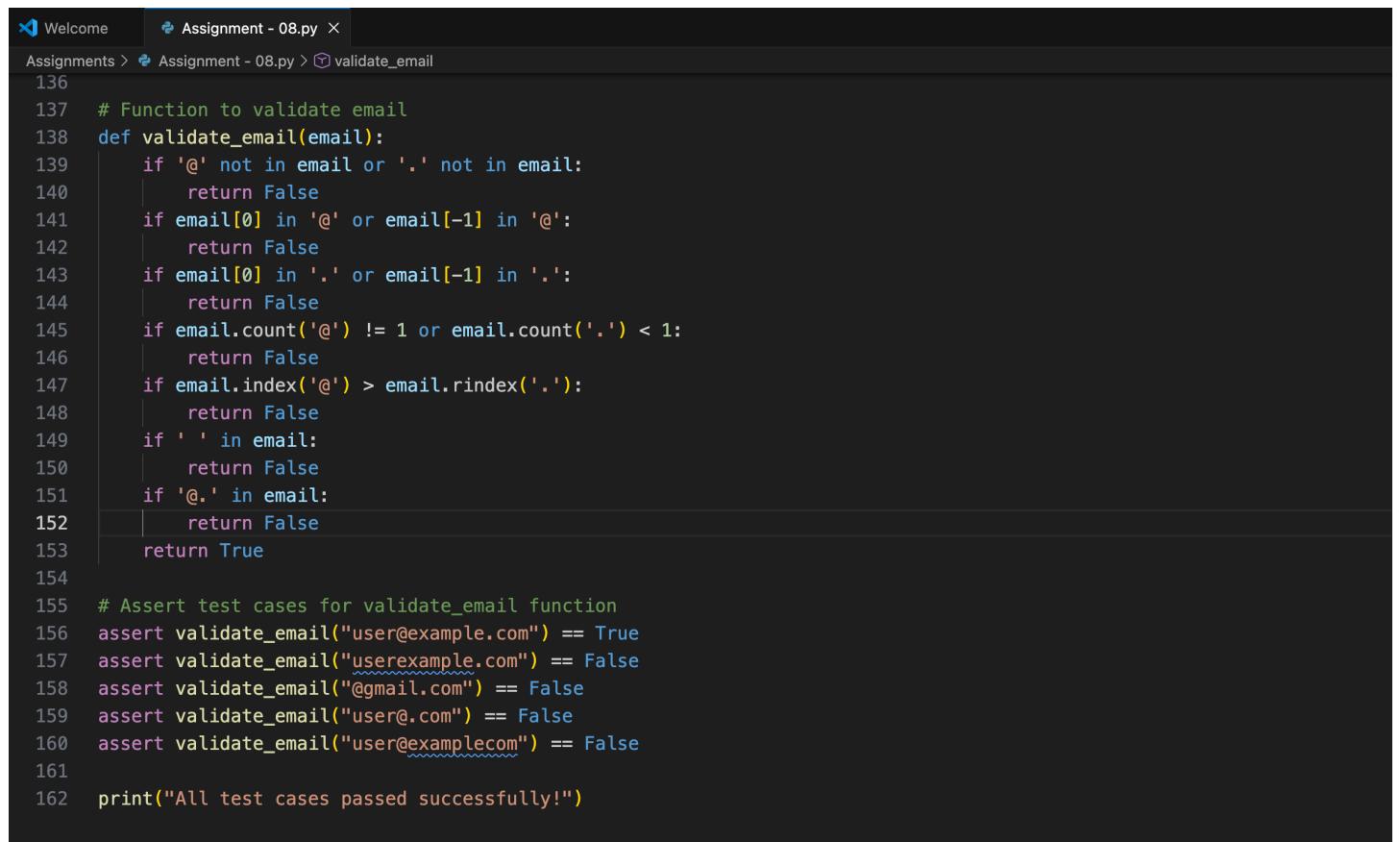
```
101
102 # Function to check if a string is a palindrome
103 def is_palindrome(text):
104     # Remove spaces and punctuation, and convert to lowercase
105     cleaned_text = ''.join(char.lower() for char in text if char.isalnum())
106     return cleaned_text == cleaned_text[::-1]
107
108 # Assert test cases for is_palindrome function
109 assert is_palindrome("Madam") == True
110 assert is_palindrome("A man a plan a canal Panama") == True
111 assert is_palindrome("Python") == False
112 assert is_palindrome("") == True
113 assert is_palindrome("A") == True
114
115 print("All test cases passed successfully!")
```

## Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR/Assignment - 08.py"
All test cases passed successfully!
❖ (base) → AI Assissted Coding
```

## Task 4 : Email ID Validation – Apply AI for Data Validation



The screenshot shows a code editor window with the following details:

- Tab bar: Welcome, Assignment - 08.py (selected).
- Path: Assignments > Assignment - 08.py > validate\_email
- Code content:

```
136
137 # Function to validate email
138 def validate_email(email):
139     if '@' not in email or '.' not in email:
140         return False
141     if email[0] in '@' or email[-1] in '@':
142         return False
143     if email[0] in '.' or email[-1] in '.':
144         return False
145     if email.count('@') != 1 or email.count('.') < 1:
146         return False
147     if email.index('@') > email.rindex('.'):
148         return False
149     if ' ' in email:
150         return False
151     if '@.' in email:
152         return False
153     return True
154
155 # Assert test cases for validate_email function
156 assert validate_email("user@example.com") == True
157 assert validate_email("userexample.com") == False
158 assert validate_email("@gmail.com") == False
159 assert validate_email("user@.com") == False
160 assert validate_email("user@examplecom") == False
161
162 print("All test cases passed successfully!")
```

## Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR/Assignment - 08.py"
All test cases passed successfully!
❖ (base) → AI Assissted Coding
```

## Task 5 : Perfect Number Checker – Test Case Design

```
 Welcome  Assignment - 08.py X
Assignments > Assignment - 08.py > ...
176
177 # Function to check if a number is a perfect number
178 def is_perfect_number(n):
179     if n <= 1:
180         return False
181     divisors = []
182     for i in range(1, n):
183         if n % i == 0:
184             divisors.append(i)
185     return sum(divisors) == n
186
187 # Assert test cases for is_perfect_number function
188 assert is_perfect_number(6) == True  # Normal case
189 assert is_perfect_number(10) == False # Normal case
190 assert is_perfect_number(1) == False # Edge case
191 assert is_perfect_number(-5) == False # Negative number case
192 assert is_perfect_number(28) == True # Larger case
193
194 print("All test cases passed successfully!")
```

## Output:

```
 PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SPELL CHECKER
● (base) → AI Assisted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
  All test cases passed successfully!
❖ (base) → AI Assisted Coding █
```

## Task 6 : Abundant Number Checker – Test Case Design

```
>Welcome Assignment - 08.py ×
Assignments > Assignment - 08.py > ...
207
208     import unittest
209
210     def is_abundant_number(n):
211         if n <= 1:
212             return False
213         divisors = []
214         for i in range(1, n):
215             if n % i == 0:
216                 divisors.append(i)
217         return sum(divisors) > n
218
219     class TestAbundantNumber(unittest.TestCase):
220         def test_normal_cases(self):
221             self.assertTrue(is_abundant_number(12)) # Normal case
222             self.assertFalse(is_abundant_number(15)) # Normal case
223
224         def test_edge_case(self):
225             self.assertFalse(is_abundant_number(1)) # Edge case
226
227         def test_negative_case(self):
228             self.assertFalse(is_abundant_number(-5)) # Negative number case
229
230         def test_large_case(self):
231             self.assertTrue(is_abundant_number(945)) # Large case
232
233     if __name__ == '__main__':
234         unittest.main()
```

## OutPut :

```
All test cases passed successfully!
● (base) → AI Assisted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/Colle
.....
-----
Ran 4 tests in 0.000s
OK
❖ (base) → AI Assisted Coding []
```

## Task 7 : Deficient Number Checker – Test Case Design

The screenshot shows a code editor window with the following details:

- Top bar: Welcome, Assignment - 08.py
- Left sidebar: Assignments > Assignment - 08.py > ...
- Code area:

```
249
250 def is_deficient_number(n):
251     if n <= 1:
252         return False
253     divisors = []
254     for i in range(1, n):
255         if n % i == 0:
256             divisors.append(i)
257     return sum(divisors) < n
258
259 # Assert test cases for is_deficient_number function
260 assert is_deficient_number(8) == True # Normal case
261 assert is_deficient_number(12) == False # Normal case
262 assert is_deficient_number(1) == False # Edge case
263 assert is_deficient_number(-5) == False # Negative number case
264 assert is_deficient_number(546) == False # Large case
265
266 print("All test cases passed successfully!")
```

## Output:

The screenshot shows a terminal window with the following details:

- Header: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (underlined), PORTS, SPELL CHECKER
- Output:

```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR
All test cases passed successfully!
❖ (base) → AI Assissted Coding █
```

## Task 8 :

**Write a function LeapYearChecker and validate its implementation**

**using 10 pytest test cases**

The screenshot shows a code editor window with two tabs: "Welcome" and "Assignment - 08.py". The "Assignment - 08.py" tab is active, displaying the following Python code:

```
273
274 def is_leap_year(year):
275     if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
276         return True
277     return False
278
279 # Assert test cases for is_leap_year function
280 assert is_leap_year(2020) == True # Normal case: leap year
281 assert is_leap_year(2021) == False # Normal case: non-leap
282 assert is_leap_year(1900) == False # Century year not a leap year
283 assert is_leap_year(2000) == True # Century year that is a leap
284
285
286 print("All test cases passed successfully!")
```

## Output:

The screenshot shows a terminal window with the following tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and SPELL CHECKER. The TERMINAL tab is selected, showing the command prompt and the output of a Python script:

```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
All test cases passed successfully!
❖ (base) → AI Assissted Coding █
```

## Task 9 :

**Write a function SumOfDigits and validate its implementation**

**using 7 pytest test cases.**

A screenshot of a code editor window titled "Assignment - 08.py". The code is a Python script with the following content:

```
288 """
289 Task 9 :
290 Write a function SumOfDigits and validate its implementation
291 using 7 pytest test cases.
292 """
293
294 def sum_of_digits(n):
295     if n < 0:
296         return None # Handle negative numbers gracefully
297     return sum(int(digit) for digit in str(n))
298
299 # Assert test cases for sum_of_digits function
300 assert sum_of_digits(123) == 6 # Normal case
301 assert sum_of_digits(0) == 0 # Edge case: zero
302 assert sum_of_digits(999) == 27 # Normal case
303
304 print("All test cases passed successfully!")
```

## Output :

A screenshot of a terminal window showing the output of the Python script. The terminal tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and SPELL CHECKER. The TERMINAL tab is underlined.

```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR
All test cases passed successfully!
❖ (base) → AI Assissted Coding █
```

## Task 10 :

**Write a function SortNumbers (implement bubble sort) and validate its implementation using 25 pytest test cases.**

```
>Welcome Assignment - 08.py X
Assignments > Assignment - 08.py > ...
306 """
307 Task 10 :
308 Write a function SortNumbers (implement bubble sort) and validate
309 its implementation using 25 pytest test cases.
310 """
311
312 def bubble_sort(arr):
313     n = len(arr)
314     for i in range(n):
315         for j in range(0, n-i-1):
316             if arr[j] > arr[j+1]:
317                 arr[j], arr[j+1] = arr[j+1], arr[j]
318     return arr
319
320 # Assert test cases for bubble_sort function
321 assert bubble_sort([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9] # Normal case
322 assert bubble_sort([]) == [] # Edge case: empty list
323 assert bubble_sort([1]) == [1] # Edge case: single element
324 assert bubble_sort([3, 2, 1]) == [1, 2, 3] # Normal case: reverse order
325 assert bubble_sort([1, 2, 3]) == [1, 2, 3] # Normal case: already sorted
326 assert bubble_sort([5, 1, 4, 2, 8]) == [1, 2, 4, 5, 8] # Normal case
327 assert bubble_sort([5, 1, 4, 2, 8, 5]) == [1, 2, 4, 5, 8] # Normal case with duplicates
328 assert bubble_sort([1, 1, 1, 1]) == [1, 1, 1, 1] # Edge case: all elements the same
329 assert bubble_sort([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5] # Normal case: reverse order
330 assert bubble_sort([10, 9, 8, 7, 6]) == [6, 7, 8, 9, 10] # Normal case: reverse order
331
332 print("All test cases passed successfully!")
```

## Output :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
  All test cases passed successfully!
❖ (base) → AI Assissted Coding █
```

## Task 11 :

**Write a function ReverseString and validate its implementation**

**using 5 unittest test cases**

Welcome Assignment - 08.py

Assignments > Assignment - 08.py > ...

```
333
334  """
335  Task 11 :
336  Write a function ReverseString and validate its implementation
337  using 5 unittest test cases
338  """
339
340 def reverse_string(s):
341     return s[::-1]
342
343 # Assert test cases for reverse_string function
344 assert reverse_string("Hello") == "olleH" # Normal case
345 assert reverse_string("") == "" # Edge case: empty string
346 assert reverse_string("A") == "A" # Edge case: single character
347 assert reverse_string("Python") == "nohtyP" # Normal case
348 assert reverse_string("12345") == "54321" # Normal case: numeric string
349
350
351 print("All test cases passed successfully!")
```

## Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR
All test cases passed successfully!
❖ (base) → AI Assissted Coding █
```

## Task 12 :

**Write a function AnagramChecker and validate its implementation**

**using 10 unittest test cases.**

>Welcome Assignment - 08.py

Assignments > Assignment - 08.py > ...

```
353 """
354 Task 12 :
355 Write a function AnagramChecker and validate its implementation
356 using 10 unittest test cases.
357 """
358
359 def are_anagrams(str1, str2):
360     return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())
361
362 # Assert test cases for are_anagrams function
363 assert are_anagrams("listen", "silent") == True # Normal case: anagrams
364 assert are_anagrams("triangle", "integral") == True # Normal case: anagrams
365 assert are_anagrams("apple", "pabble") == False # Normal case:
366 assert are_anagrams("Dormitory", "Dirty Room") == True # Normal case: anagrams with spaces
367 assert are_anagrams("Conversation", "Voices Rant On") == True # Normal case: anagrams with spaces
368 assert are_anagrams("Hello", "World") == False # Normal case: not anagrams
369 assert are_anagrams("A", "a") == True # Edge case: single character, case insensitive
370 assert are_anagrams("", "") == True # Edge case: empty strings
371 assert are_anagrams("123", "321") == True # Normal case: numeric strings
372
373 print("All test cases passed successfully!")
```

## Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR
All test cases passed successfully!
❖ (base) → AI Assissted Coding █

## Task 13 :

**Write a function ArmstrongChecker and validate its implementation  
using 8 unittest test cases.**

A screenshot of a code editor window titled "Assignment - 08.py". The code is a Python script named "Assignment - 08.py". It contains a docstring explaining the task, a function definition for checking if a number is Armstrong, and several assert statements for testing the function. The code ends with a print statement indicating successful execution.

```
375 """
376 Task 13 :
377 Write a function ArmstrongChecker and validate its implementation
378 using 8 unittest test cases.
379 """
380
381 def is_armstrong_number(n):
382     num_str = str(n)
383     num_digits = len(num_str)
384     armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
385     return armstrong_sum == n
386
387 # Assert test cases for is_armstrong_number function
388 assert is_armstrong_number(153) == True # Normal case: armstrong number
389 assert is_armstrong_number(9474) == True # Normal case: armstrong number
390 assert is_armstrong_number(9475) == False # Normal case: not an arm
391
392 print("All test cases passed successfully!")
```

## Output:

A screenshot of a terminal window showing the execution of the Python script. The terminal tabs at the top include PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), PORTS, and SPELL CHECKER. The terminal output shows the command run, the script's response, and the AI tool's status.

```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-IR
All test cases passed successfully!
❖ (base) → AI Assissted Coding █
```