ASSIGNMENT - 08

Name : Aravind Reddy

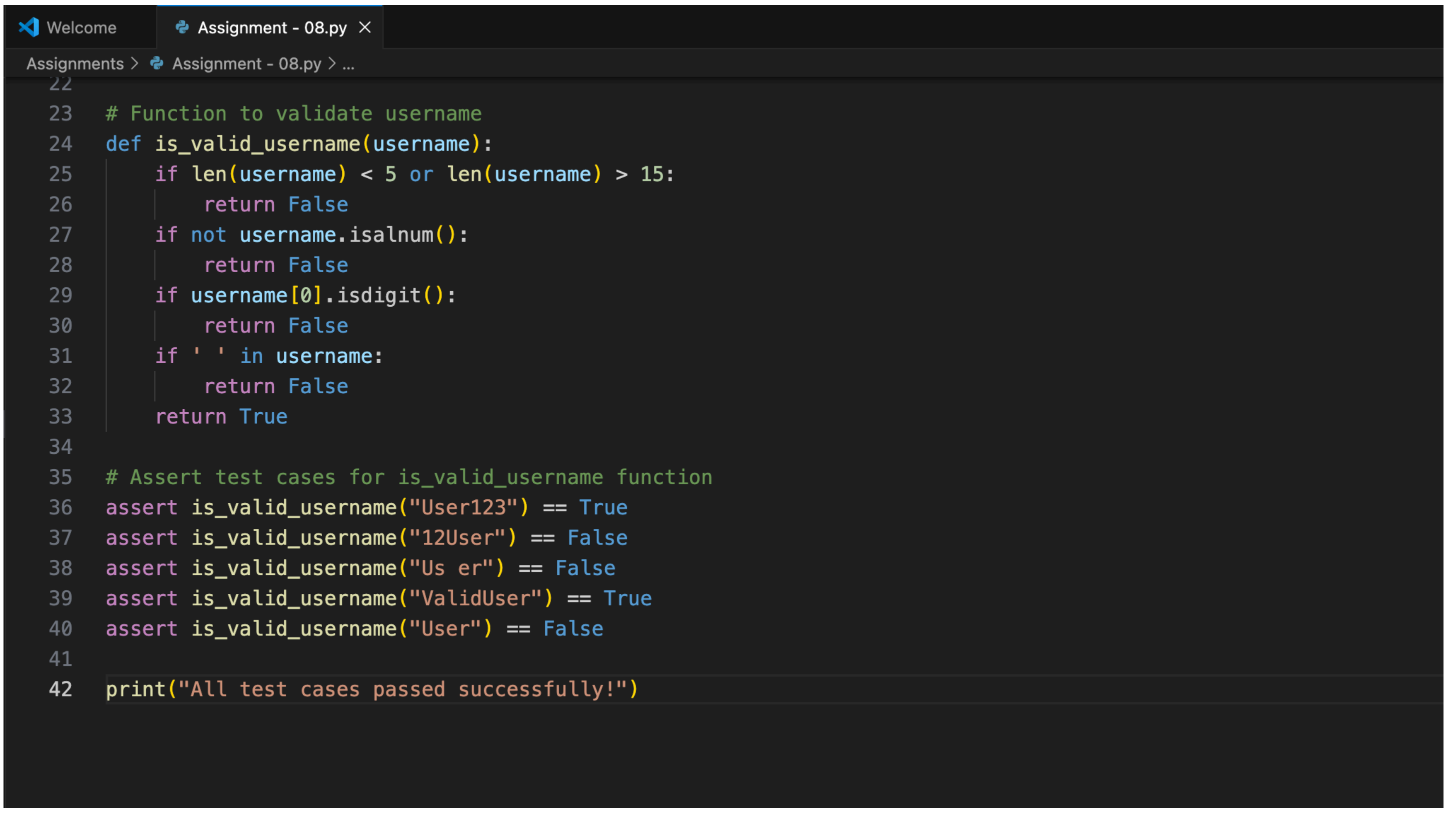
Hall Ticket No : 2303a51027

Batch No : 01

Course Title : AI Assisted Coding

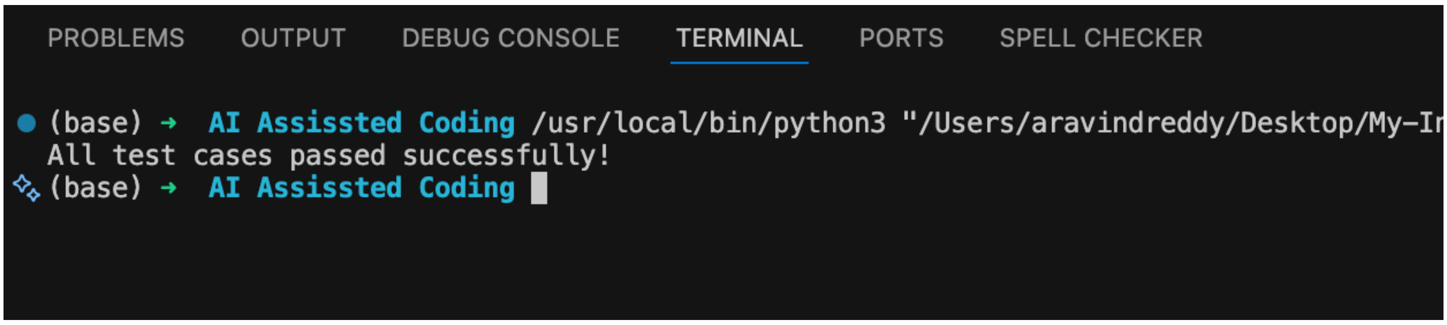
Instructor's Name : Mr. S Naresh Kumar

Task 1 : Username Validator – Apply AI in Authentication Context



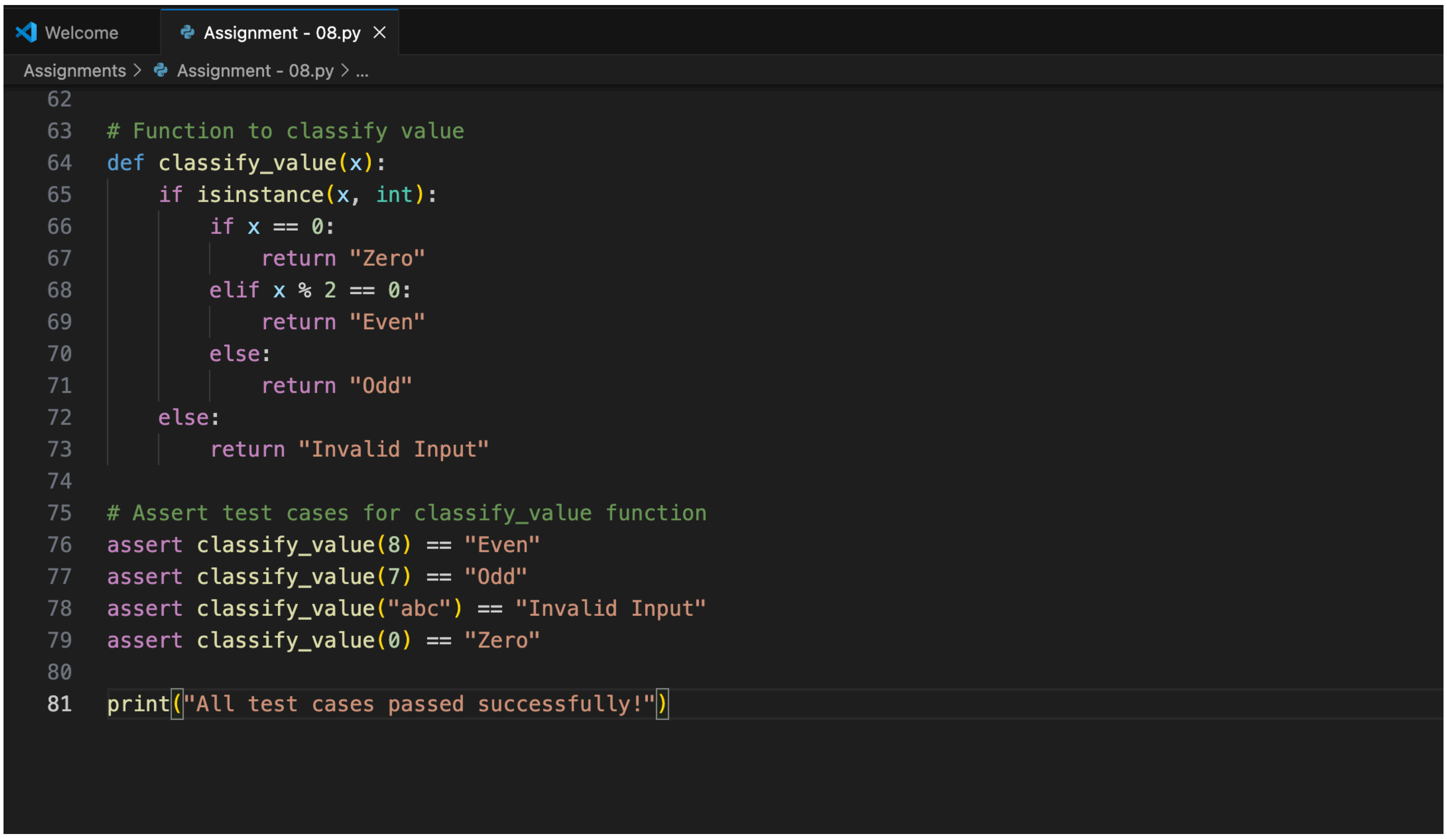
```python
# Function to validate username
def is_valid_username(username):
    if len(username) < 5 or len(username) > 15:
        return False
    if not username.isalnum():
        return False
    if username[0].isdigit():
        return False
    if ' ' in username:
        return False
    return True

# Assert test cases for is_valid_username function
assert is_valid_username("User123") == True
assert is_valid_username("12User") == False
assert is_valid_username("Us er") == False
assert is_valid_username("ValidUser") == True
assert is_valid_username("User") == False

print("All test cases passed successfully!")
```

Output :



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

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

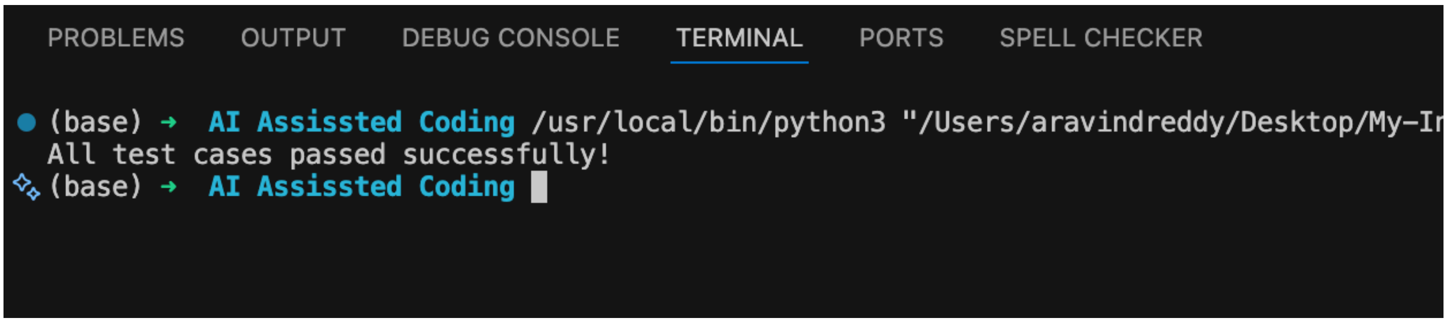


Welcome | Assignment - 08.py

Assignments > Assignment - 08.py > ...

```python

# Function to classify value
def classify_value(x):
    if isinstance(x, int):
        if x == 0:
            return "Zero"
        elif x % 2 == 0:
            return "Even"
        else:
            return "Odd"
    else:
        return "Invalid Input"

# Assert test cases for classify_value function
assert classify_value(8) == "Even"
assert classify_value(7) == "Odd"
assert classify_value("abc") == "Invalid Input"
assert classify_value(0) == "Zero"

print("All test cases passed successfully!")
```

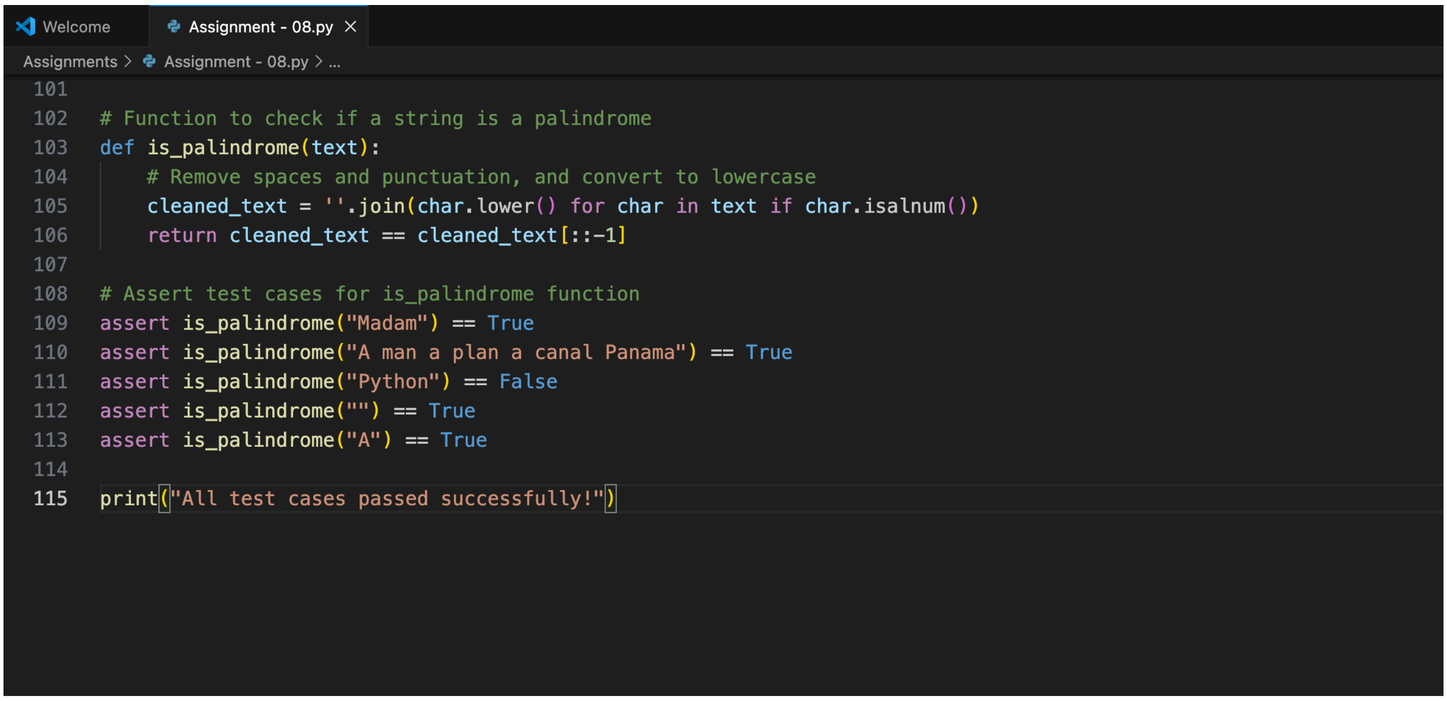
OutPut :



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 3 : Palindrome Checker – Apply AI for String Normalization

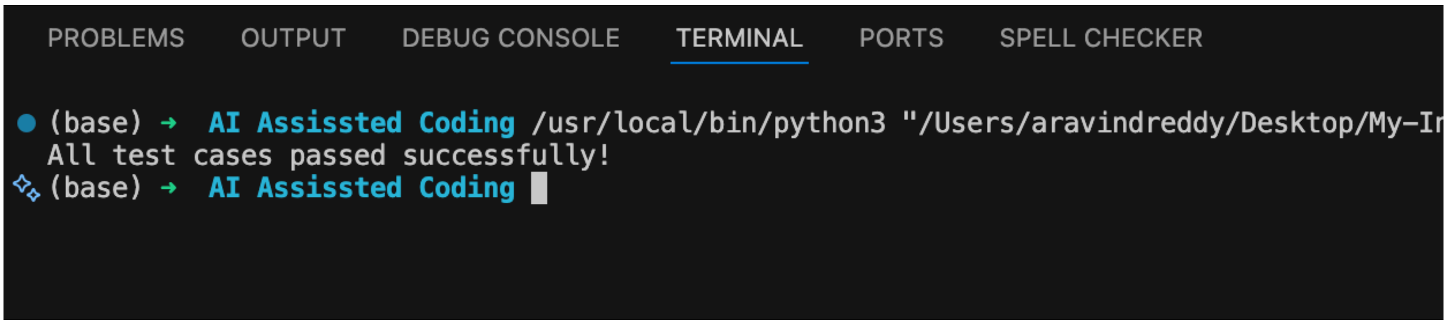


Welcome | Assignment - 08.py

Assignments > Assignment - 08.py > ...

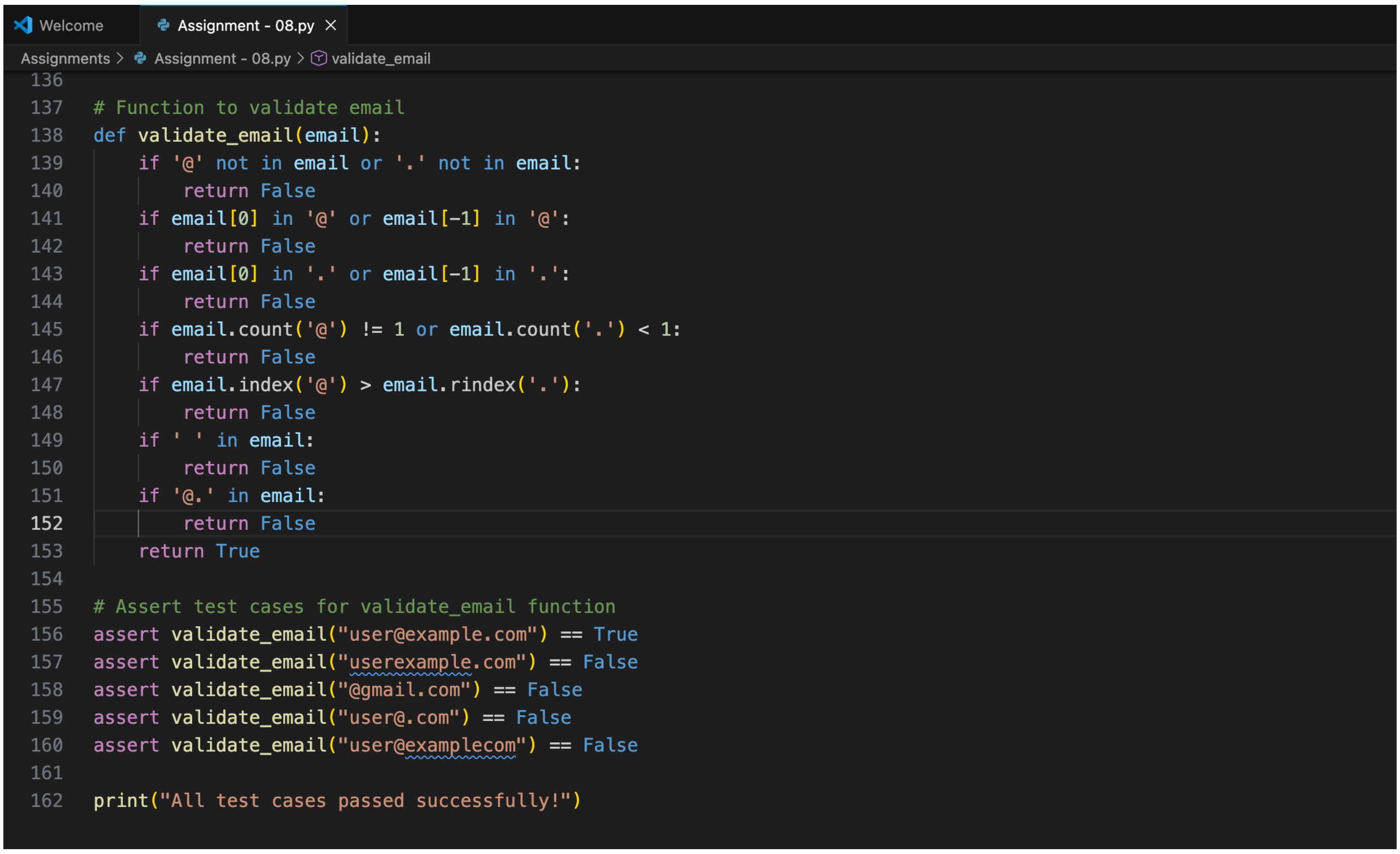
```python

# Function to check if a string is a palindrome
def is_palindrome(text):
    # Remove spaces and punctuation, and convert to lowercase
    cleaned_text = ''.join(char.lower() for char in text if char.isalnum())
    return cleaned_text == cleaned_text[::-1]

# Assert test cases for is_palindrome function
assert is_palindrome("Madam") == True
assert is_palindrome("A man a plan a canal Panama") == True
assert is_palindrome("Python") == False
assert is_palindrome("") == True
assert is_palindrome("A") == True

print("All test cases passed successfully!")
```

Output :



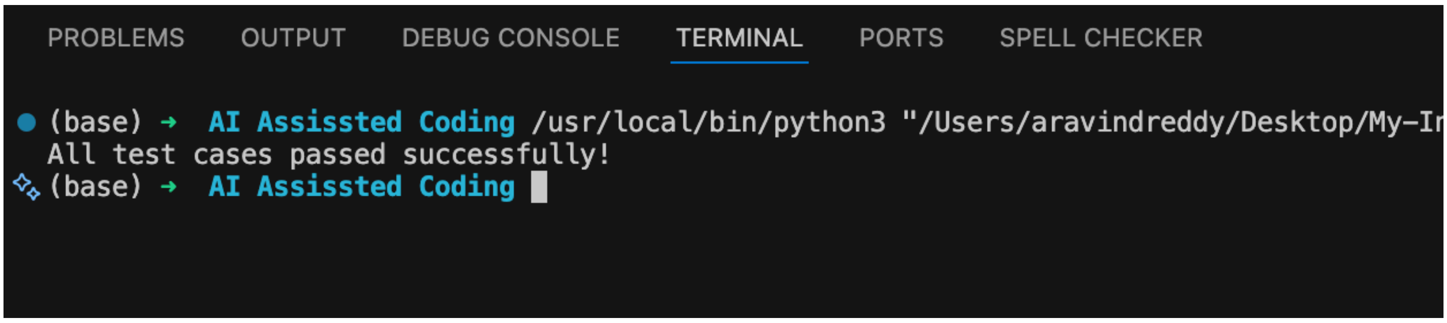
```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
  All test cases passed successfully!
  (base) → AI Assissted Coding █
```

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



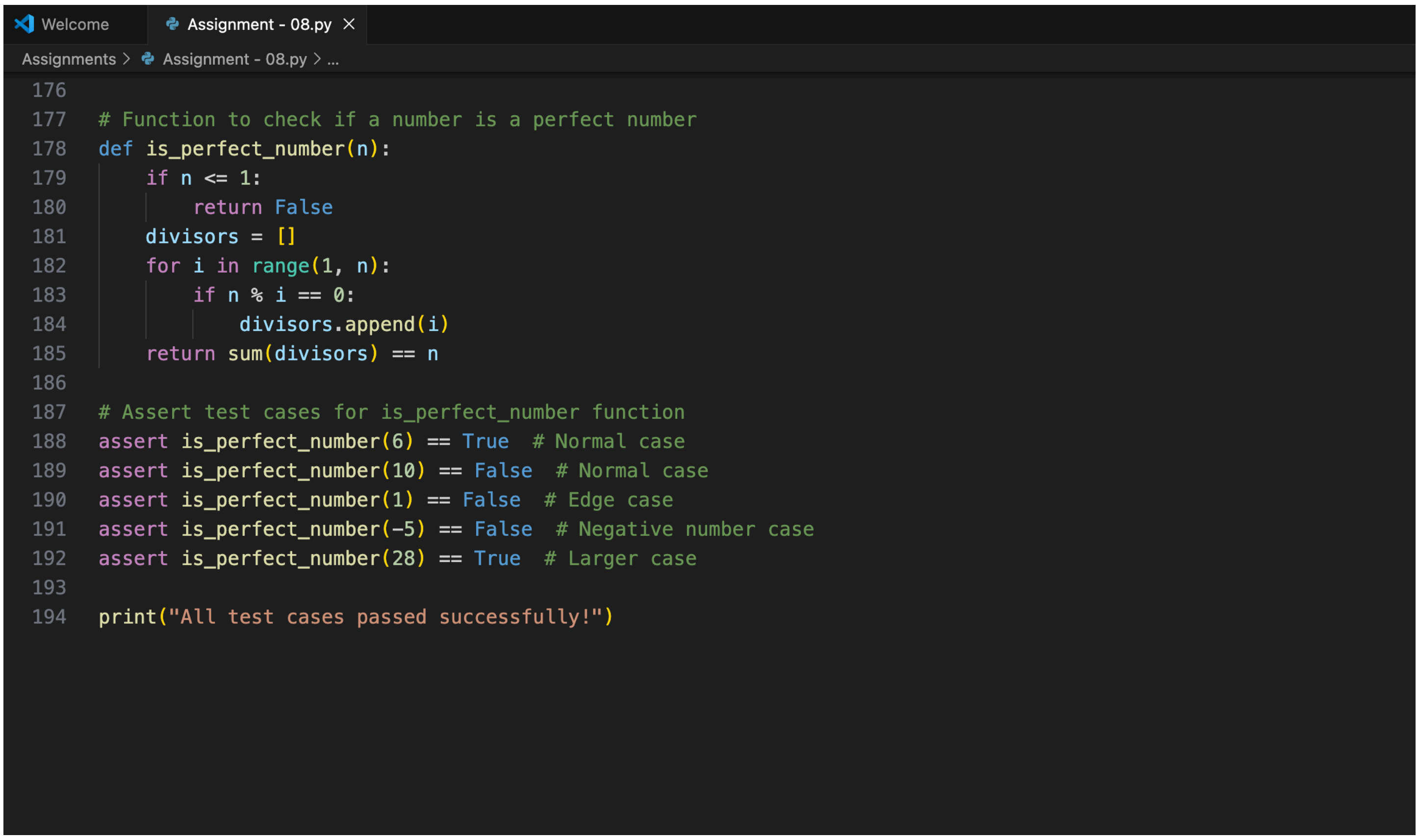
```python

# Function to validate email
def validate_email(email):
    if '@' not in email or '.' not in email:
        return False
    if email[0] in '@' or email[-1] in '@':
        return False
    if email[0] in '.' or email[-1] in '.':
        return False
    if email.count('@') != 1 or email.count('.') < 1:
        return False
    if email.index('@') > email.rindex('.'):
        return False
    if ' ' in email:
        return False
    if '@.' in email:
        return False
    return True

# Assert test cases for validate_email function
assert validate_email("user@example.com") == True
assert validate_email("userexample.com") == False
assert validate_email("@gmail.com") == False
assert validate_email("user@.com") == False
assert validate_email("user@examplecom") == False

print("All test cases passed successfully!")
```

Output :



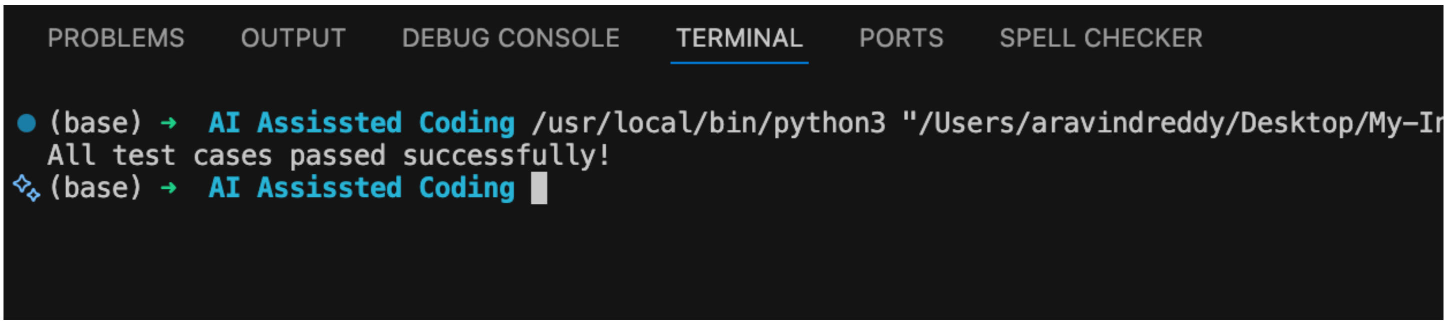
```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
  All test cases passed successfully!
  (base) → AI Assissted Coding █
```

Task 5 : Perfect Number Checker – Test Case Design



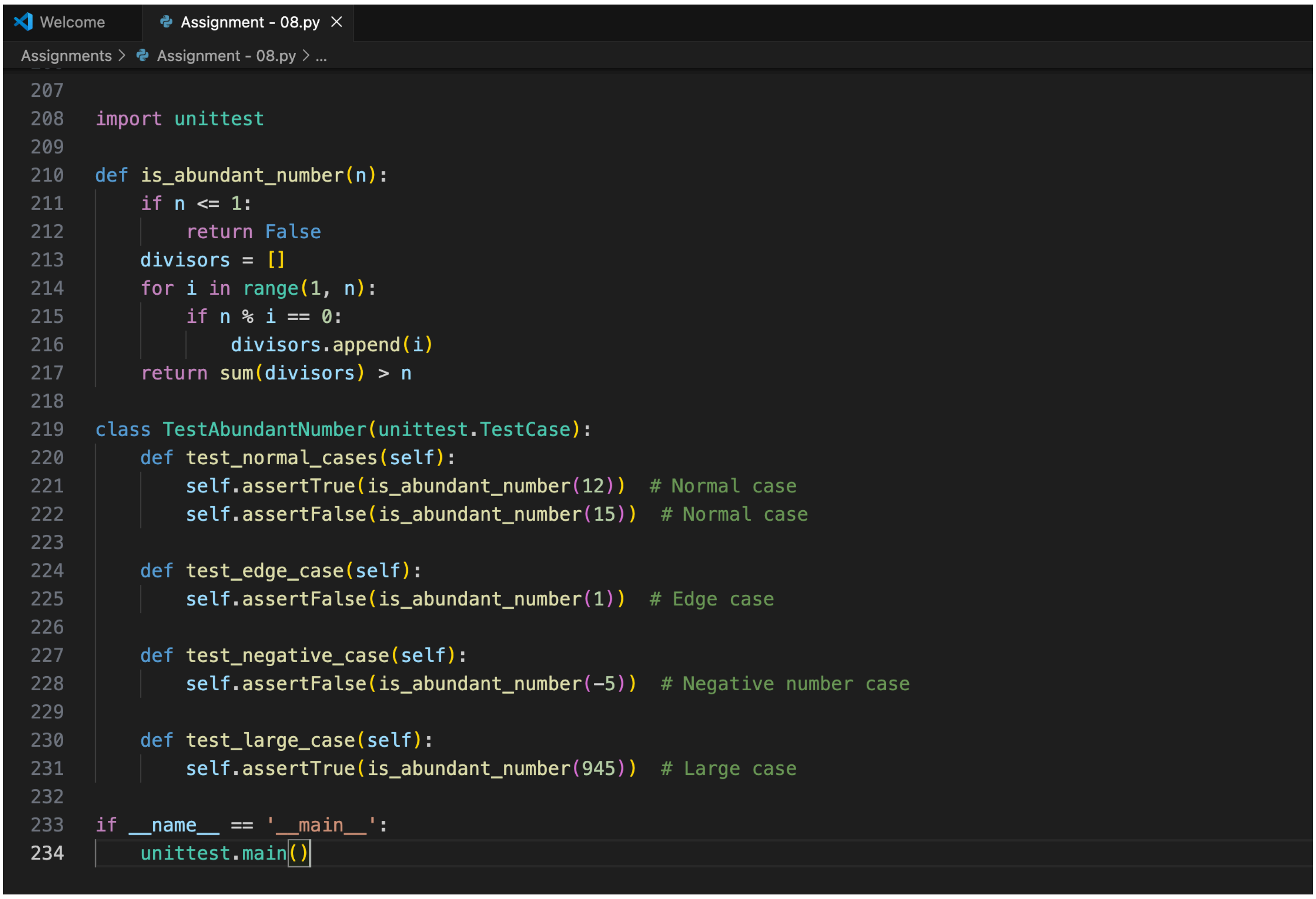
```python

# Function to check if a number is a perfect number
def is_perfect_number(n):
    if n <= 1:
        return False
    divisors = []
    for i in range(1, n):
        if n % i == 0:
            divisors.append(i)
    return sum(divisors) == n

# Assert test cases for is_perfect_number function
assert is_perfect_number(6) == True  # Normal case
assert is_perfect_number(10) == False  # Normal case
assert is_perfect_number(1) == False  # Edge case
assert is_perfect_number(-5) == False  # Negative number case
assert is_perfect_number(28) == True  # Larger case

print("All test cases passed successfully!")
```

Output :



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

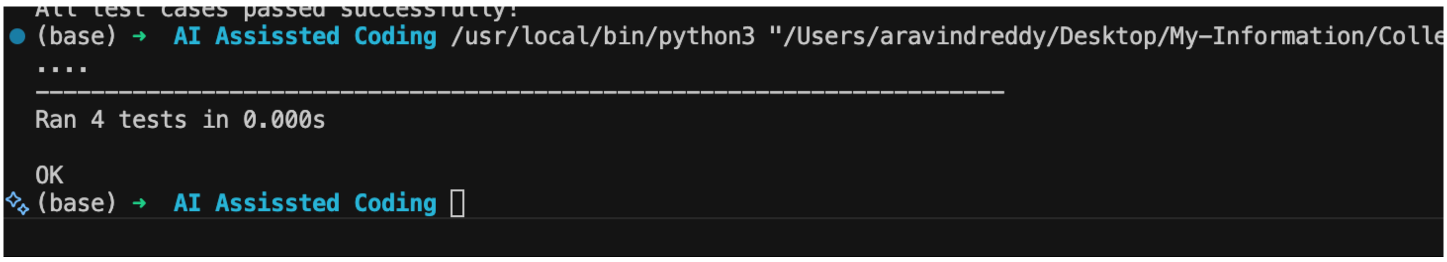
Task 6 : Abundant Number Checker – Test Case Design


Welcome | Assignment - 08.py

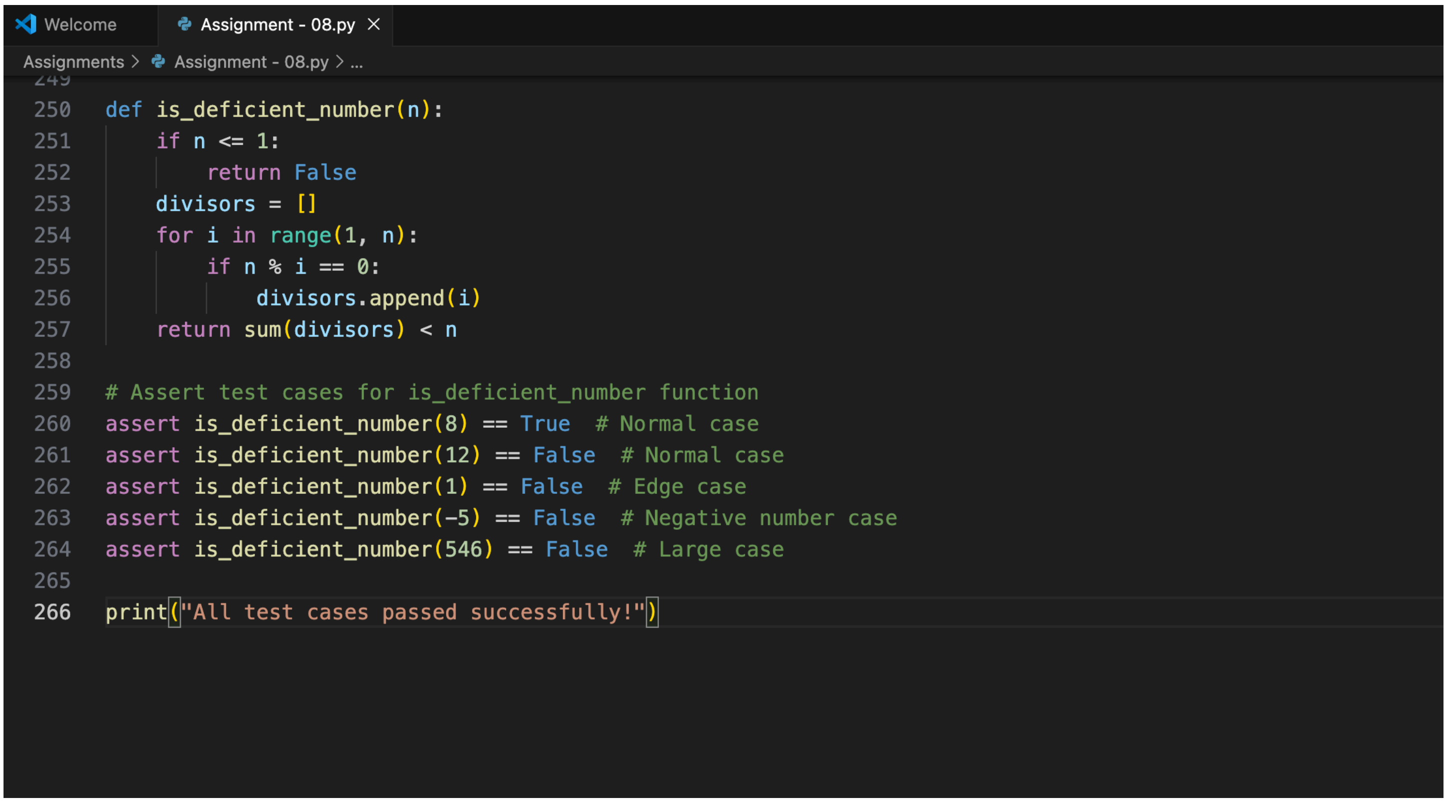
Assignments > Assignment - 08.py > ...

```python

import unittest

def is_abundant_number(n):
    if n <= 1:
        return False
    divisors = []
    for i in range(1, n):
        if n % i == 0:
            divisors.append(i)
    return sum(divisors) > n

class TestAbundantNumber(unittest.TestCase):
    def test_normal_cases(self):
        self.assertTrue(is_abundant_number(12))  # Normal case
        self.assertFalse(is_abundant_number(15))  # Normal case

    def test_edge_case(self):
        self.assertFalse(is_abundant_number(1))  # Edge case

    def test_negative_case(self):
        self.assertFalse(is_abundant_number(-5))  # Negative number case

    def test_large_case(self):
        self.assertTrue(is_abundant_number(945))  # Large case

if __name__ == '__main__':
    unittest.main()
```

OutPut :

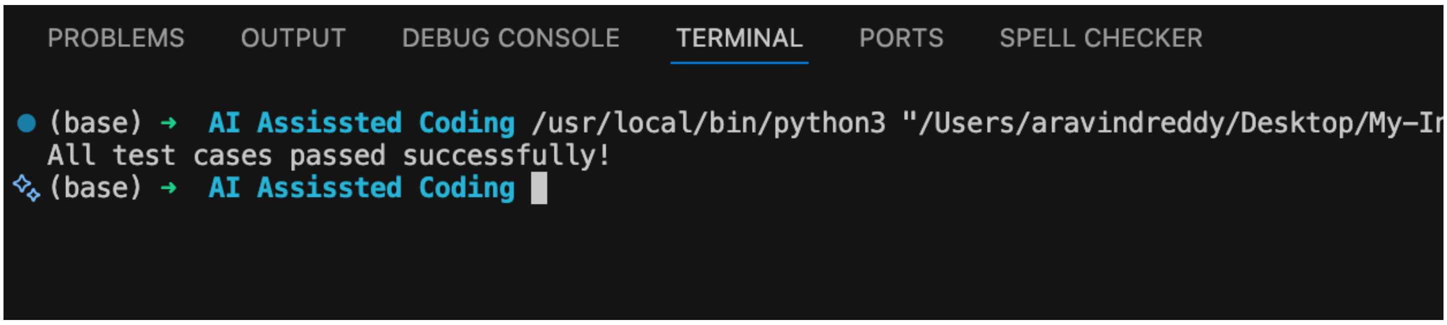

```
All test cases passed successfully!
● (base) ➜  AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/Colle
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
(base) ➜  AI Assissted Coding ▯
```

Task 7 : Deficient Number Checker – Test Case Design



```python
def is_deficient_number(n):
    if n <= 1:
        return False
    divisors = []
    for i in range(1, n):
        if n % i == 0:
            divisors.append(i)
    return sum(divisors) < n

# Assert test cases for is_deficient_number function
assert is_deficient_number(8) == True  # Normal case
assert is_deficient_number(12) == False  # Normal case
assert is_deficient_number(1) == False  # Edge case
assert is_deficient_number(-5) == False  # Negative number case
assert is_deficient_number(546) == False  # Large case

print("All test cases passed successfully!")
```

Output :

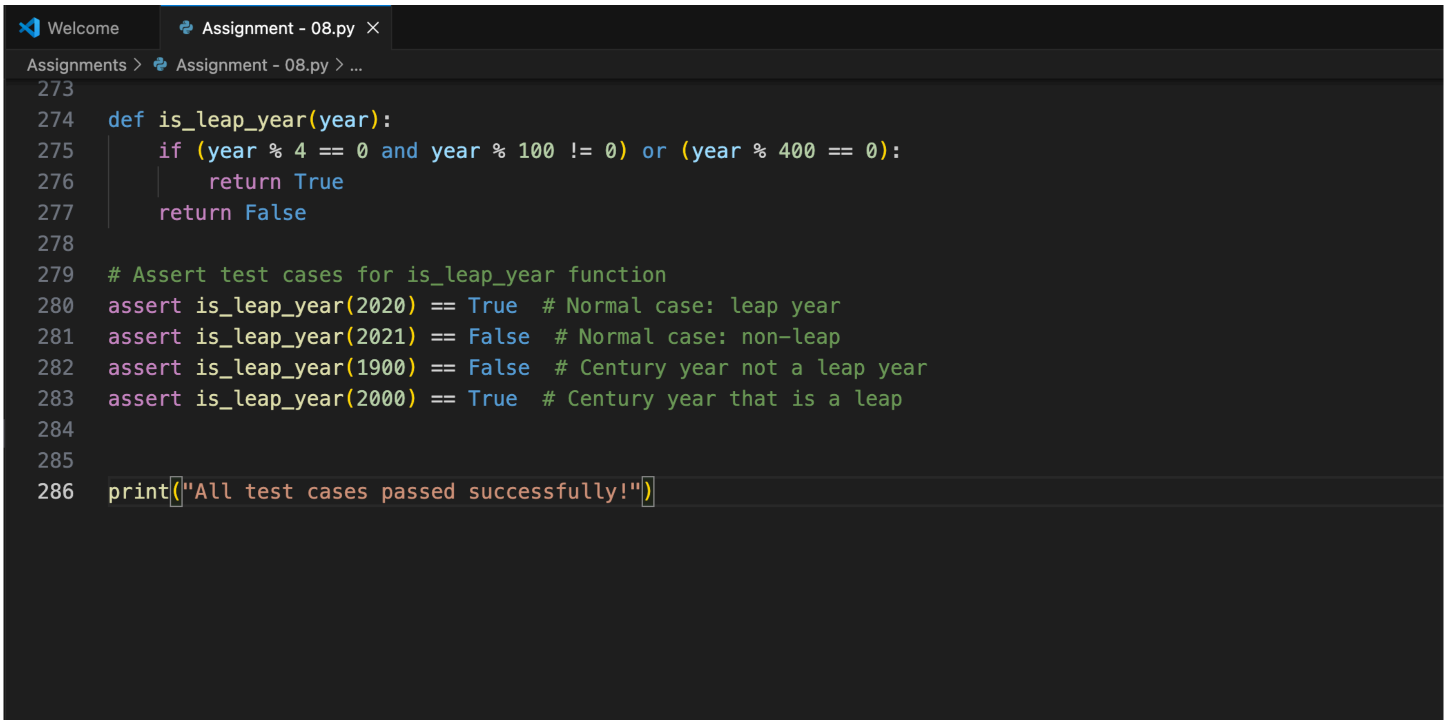


```
(base) ➜  AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-In
All test cases passed successfully!
(base) ➜  AI Assissted Coding
```

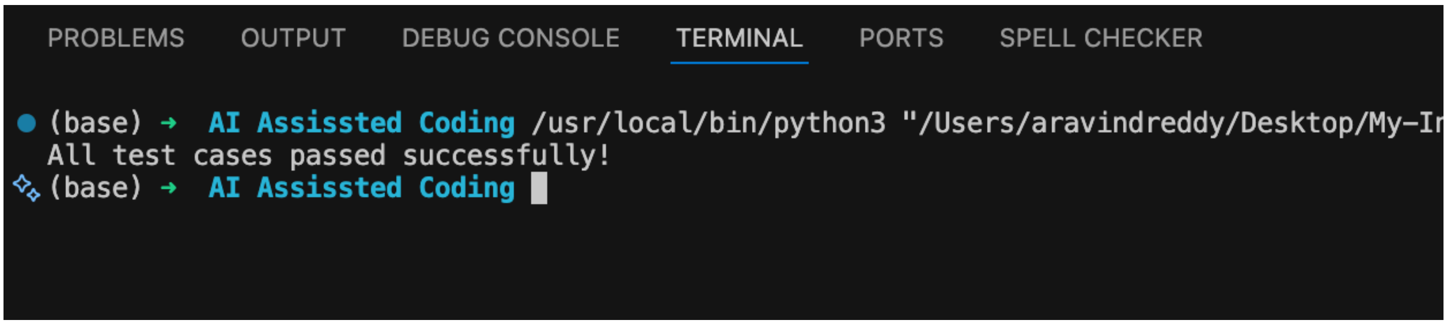
Task 8 :

Write a function LeapYearChecker and validate its implementation

using 10 pytest test cases


```
Welcome    Assignment - 08.py ×
Assignments > Assignment - 08.py > ...

def is_leap_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return True
    return False

# Assert test cases for is_leap_year function
assert is_leap_year(2020) == True  # Normal case: leap year
assert is_leap_year(2021) == False  # Normal case: non-leap
assert is_leap_year(1900) == False  # Century year not a leap year
assert is_leap_year(2000) == True  # Century year that is a leap


print("All test cases passed successfully!")
```

Output :

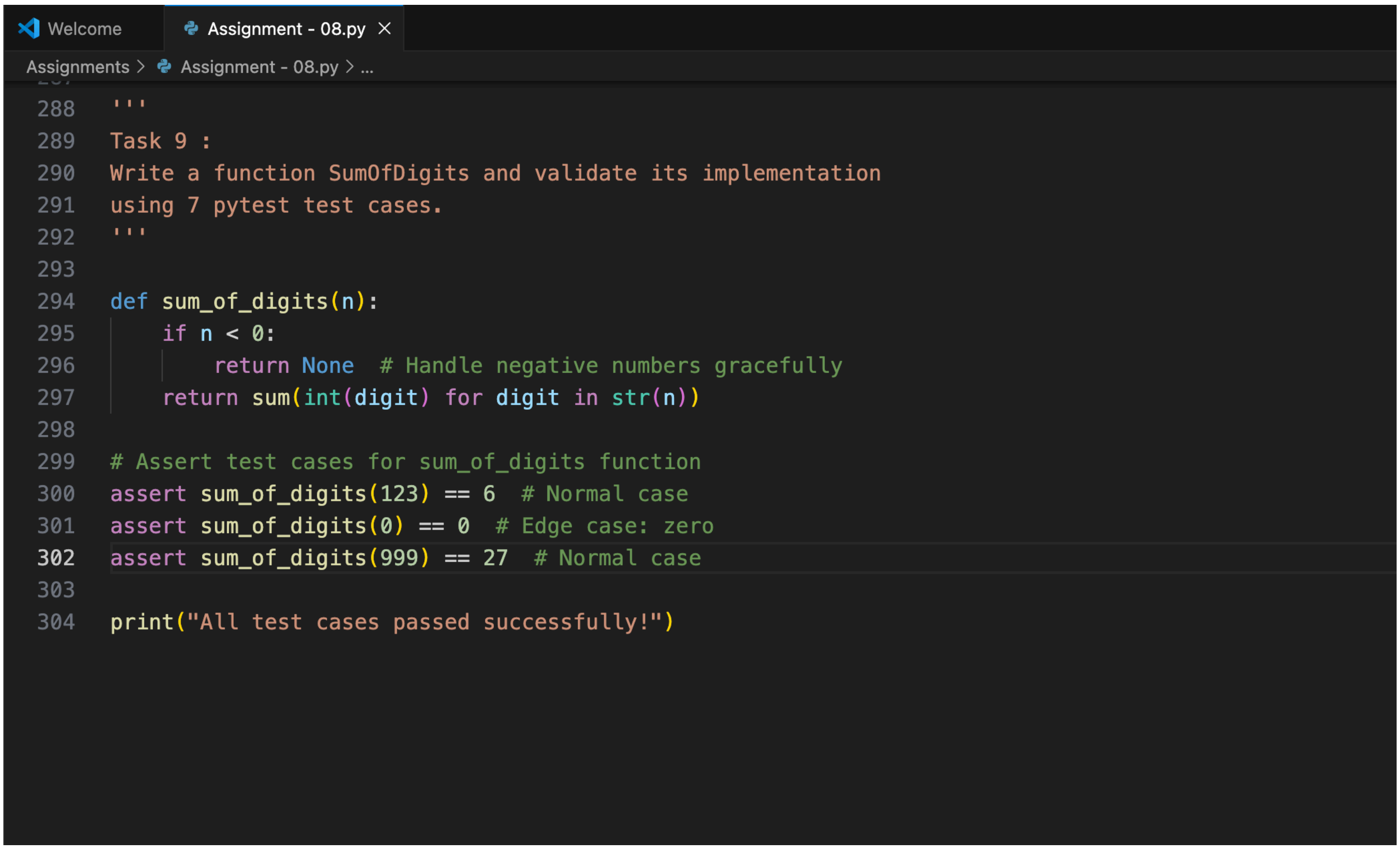


```
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
  All test cases passed successfully!
  (base) → AI Assissted Coding
```

Task 9 :

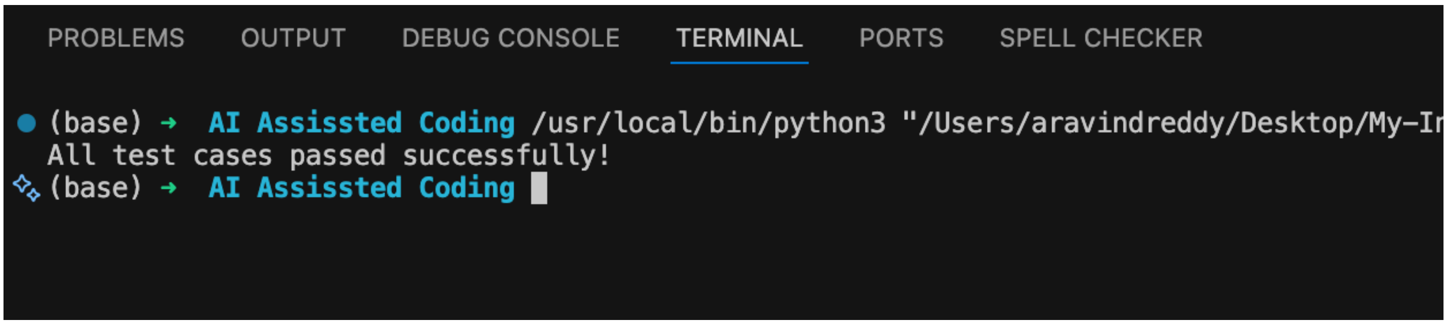
Write a function SumOfDigits and validate its implementation

using 7 pytest test cases.



```python
'''
Task 9 :
Write a function SumOfDigits and validate its implementation
using 7 pytest test cases.
'''

def sum_of_digits(n):
    if n < 0:
        return None  # Handle negative numbers gracefully
    return sum(int(digit) for digit in str(n))

# Assert test cases for sum_of_digits function
assert sum_of_digits(123) == 6  # Normal case
assert sum_of_digits(0) == 0  # Edge case: zero
assert sum_of_digits(999) == 27  # Normal case

print("All test cases passed successfully!")
```

Output :

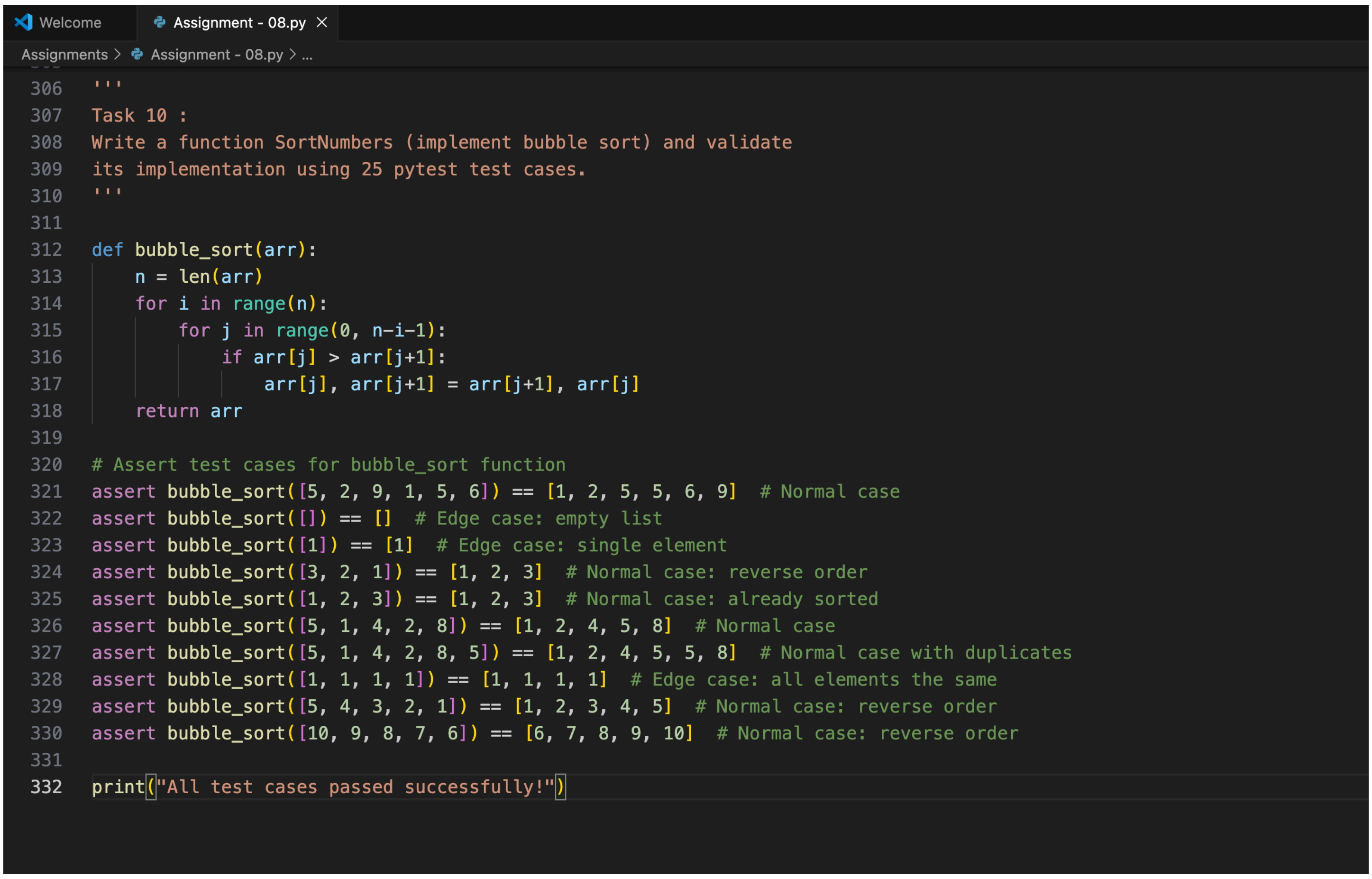


```
(base) ➜  AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
All test cases passed successfully!
(base) ➜  AI Assissted Coding
```

Task 10 :

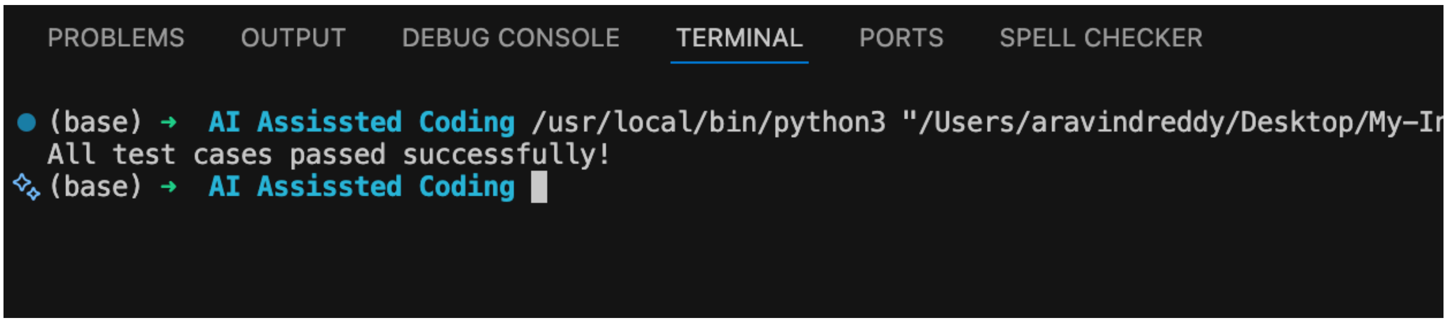
Write a function SortNumbers (implement bubble sort) and validate

its implementation using 25 pytest test cases.



```python
'''
Task 10 :
Write a function SortNumbers (implement bubble sort) and validate
its implementation using 25 pytest test cases.
'''

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Assert test cases for bubble_sort function
assert bubble_sort([5, 2, 9, 1, 5, 6]) == [1, 2, 5, 5, 6, 9]  # Normal case
assert bubble_sort([]) == []  # Edge case: empty list
assert bubble_sort([1]) == [1]  # Edge case: single element
assert bubble_sort([3, 2, 1]) == [1, 2, 3]  # Normal case: reverse order
assert bubble_sort([1, 2, 3]) == [1, 2, 3]  # Normal case: already sorted
assert bubble_sort([5, 1, 4, 2, 8]) == [1, 2, 4, 5, 8]  # Normal case
assert bubble_sort([5, 1, 4, 2, 8, 5]) == [1, 2, 4, 5, 5, 8]  # Normal case with duplicates
assert bubble_sort([1, 1, 1, 1]) == [1, 1, 1, 1]  # Edge case: all elements the same
assert bubble_sort([5, 4, 3, 2, 1]) == [1, 2, 3, 4, 5]  # Normal case: reverse order
assert bubble_sort([10, 9, 8, 7, 6]) == [6, 7, 8, 9, 10]  # Normal case: reverse order

print("All test cases passed successfully!")
```

Output :

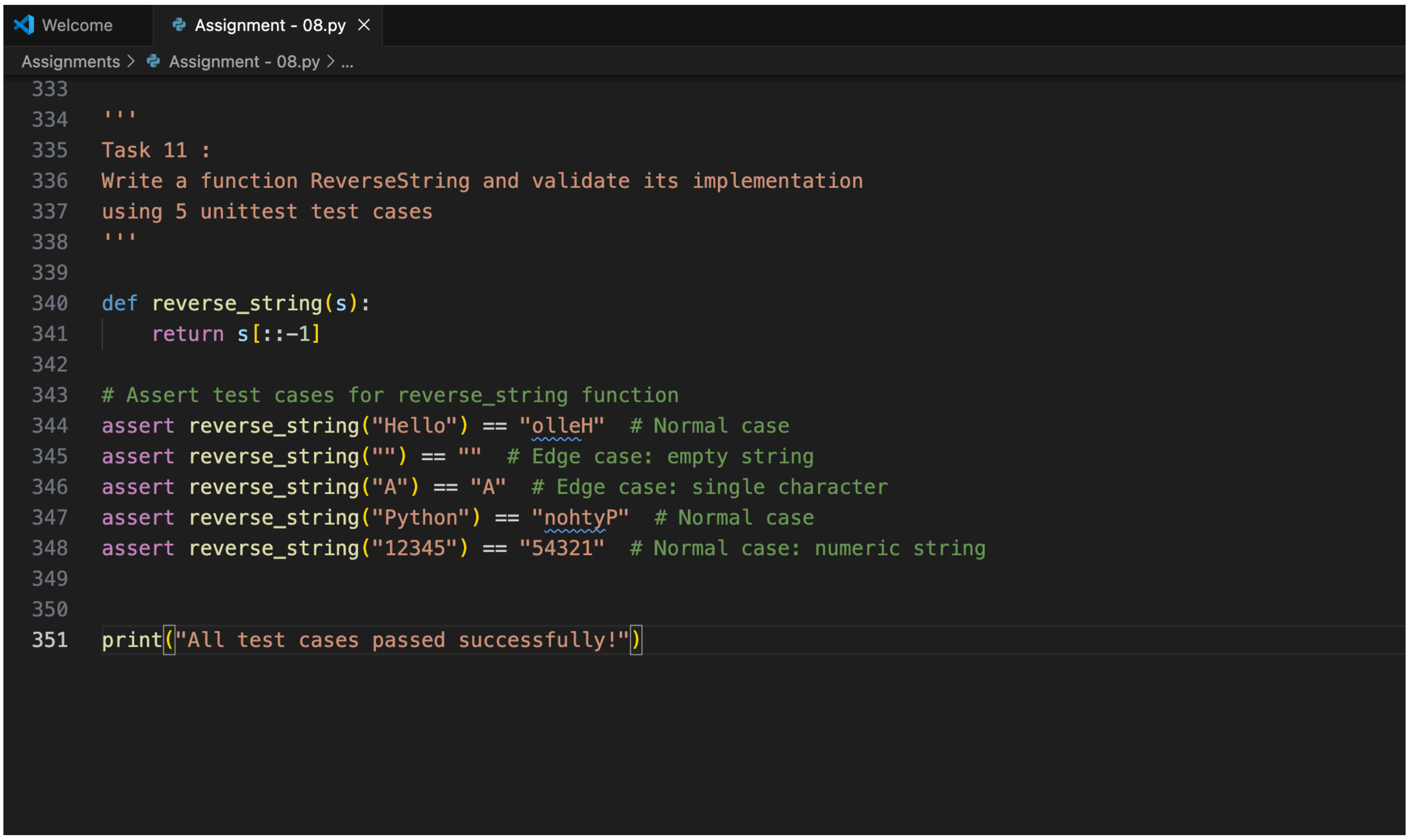


```
(base) ➜  AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
All test cases passed successfully!
(base) ➜  AI Assissted Coding 
```

Task 11 :

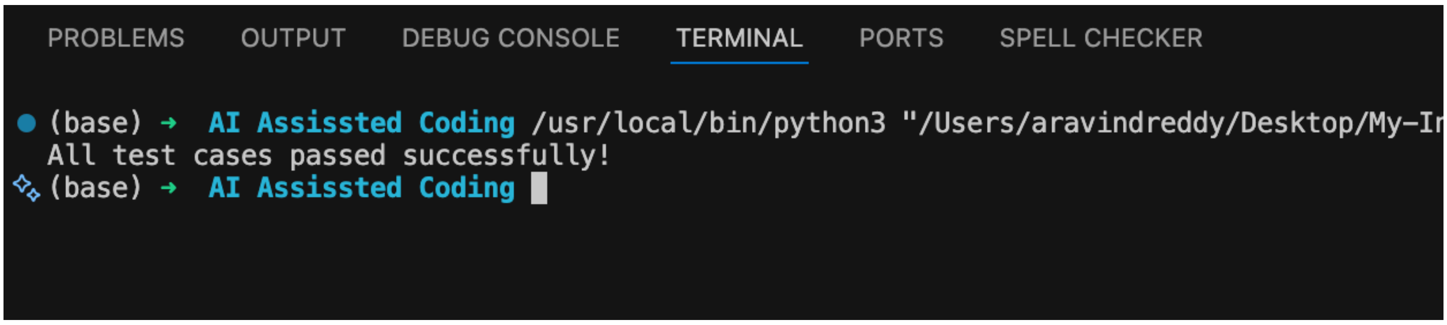
Write a function ReverseString and validate its implementation

using 5 unittest test cases



```python

'''
Task 11 :
Write a function ReverseString and validate its implementation
using 5 unittest test cases
'''

def reverse_string(s):
    return s[::-1]

# Assert test cases for reverse_string function
assert reverse_string("Hello") == "olleH"  # Normal case
assert reverse_string("") == ""  # Edge case: empty string
assert reverse_string("A") == "A"  # Edge case: single character
assert reverse_string("Python") == "nohtyP"  # Normal case
assert reverse_string("12345") == "54321"  # Normal case: numeric string


print("All test cases passed successfully!")
```

Output :

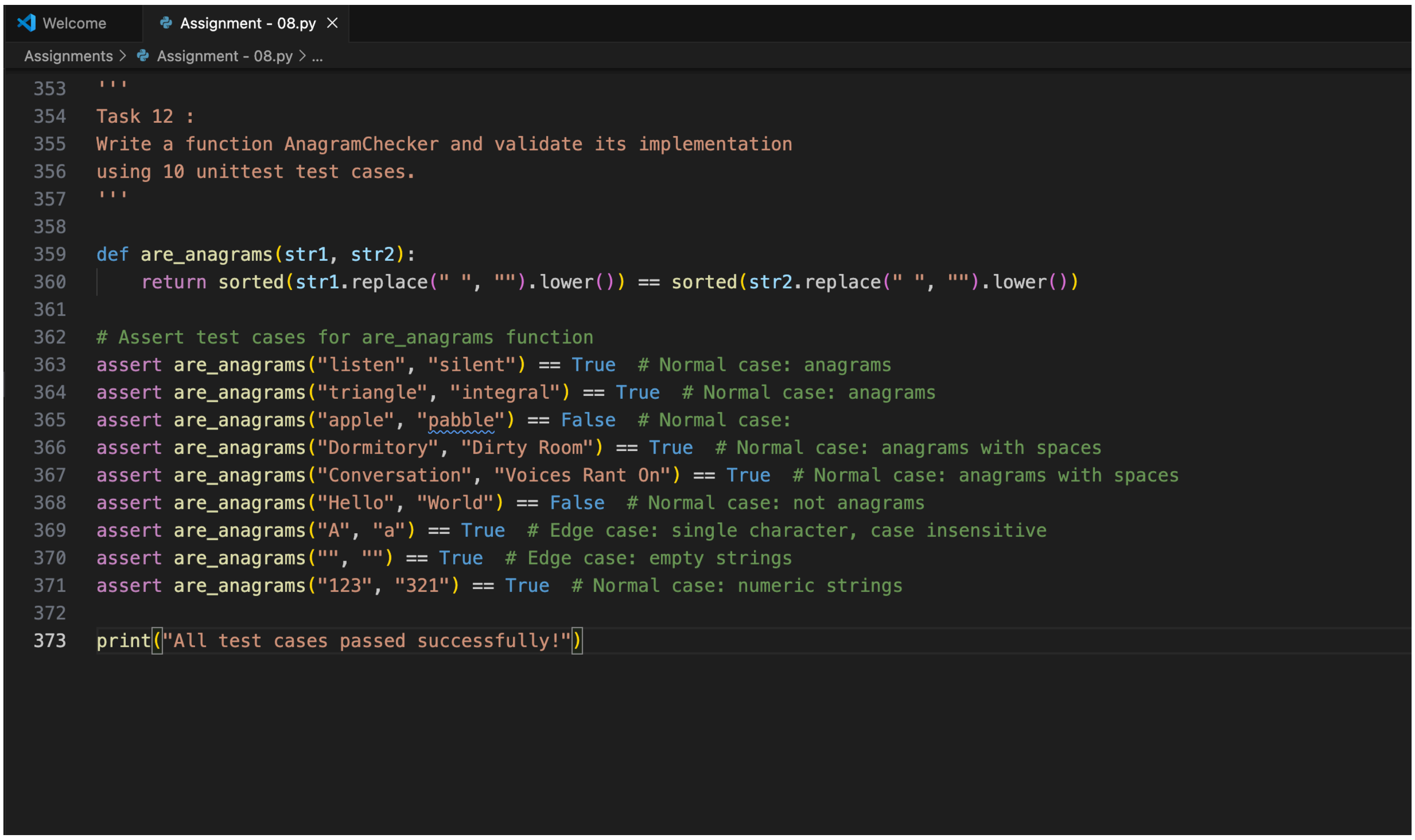


```
(base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
All test cases passed successfully!
(base) → AI Assissted Coding 
```

Task 12 :

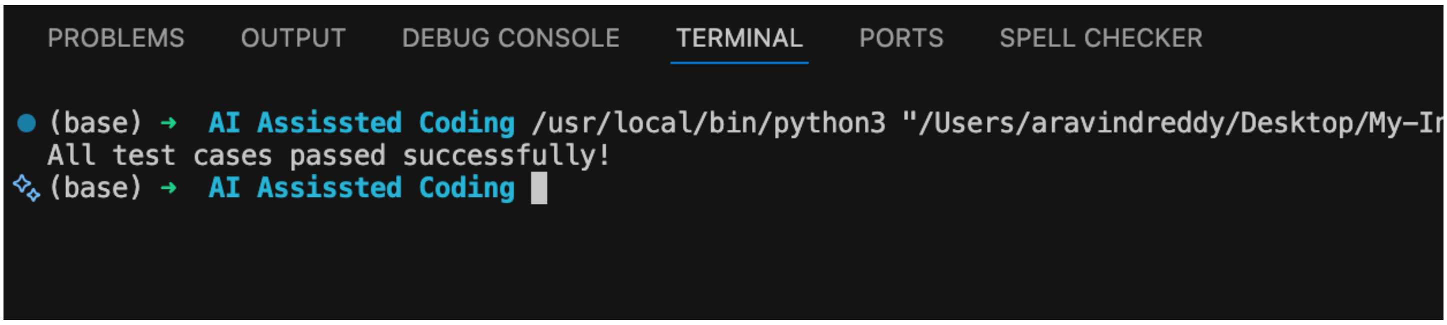
Write a function AnagramChecker and validate its implementation

using 10 unittest test cases.



```python
'''
Task 12 :
Write a function AnagramChecker and validate its implementation
using 10 unittest test cases.
'''

def are_anagrams(str1, str2):
    return sorted(str1.replace(" ", "").lower()) == sorted(str2.replace(" ", "").lower())

# Assert test cases for are_anagrams function
assert are_anagrams("listen", "silent") == True  # Normal case: anagrams
assert are_anagrams("triangle", "integral") == True  # Normal case: anagrams
assert are_anagrams("apple", "pabble") == False  # Normal case:
assert are_anagrams("Dormitory", "Dirty Room") == True  # Normal case: anagrams with spaces
assert are_anagrams("Conversation", "Voices Rant On") == True  # Normal case: anagrams with spaces
assert are_anagrams("Hello", "World") == False  # Normal case: not anagrams
assert are_anagrams("A", "a") == True  # Edge case: single character, case insensitive
assert are_anagrams("", "") == True  # Edge case: empty strings
assert are_anagrams("123", "321") == True  # Normal case: numeric strings

print("All test cases passed successfully!")
```

Output :

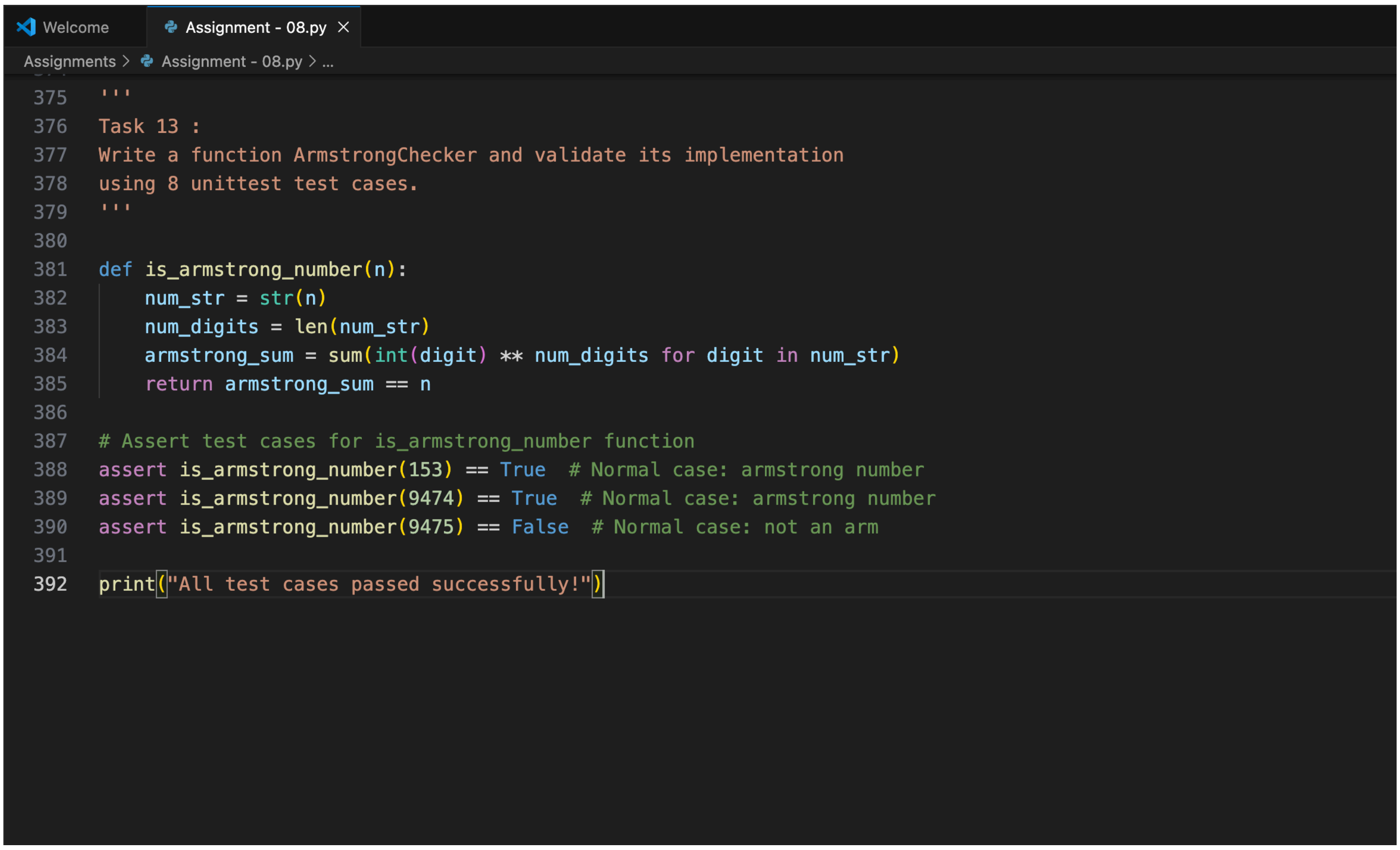


```
(base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Ir
All test cases passed successfully!
(base) → AI Assissted Coding
```

Task 13 :

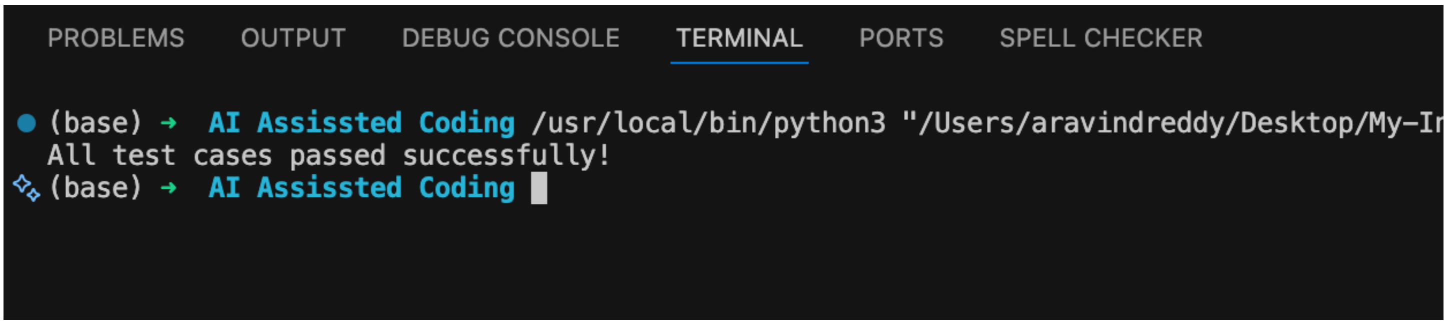
Write a function ArmstrongChecker and validate its implementation

using 8 unittest test cases.



```python
'''
Task 13 :
Write a function ArmstrongChecker and validate its implementation
using 8 unittest test cases.
'''

def is_armstrong_number(n):
    num_str = str(n)
    num_digits = len(num_str)
    armstrong_sum = sum(int(digit) ** num_digits for digit in num_str)
    return armstrong_sum == n

# Assert test cases for is_armstrong_number function
assert is_armstrong_number(153) == True  # Normal case: armstrong number
assert is_armstrong_number(9474) == True  # Normal case: armstrong number
assert is_armstrong_number(9475) == False  # Normal case: not an arm

print("All test cases passed successfully!")
```

Output :



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