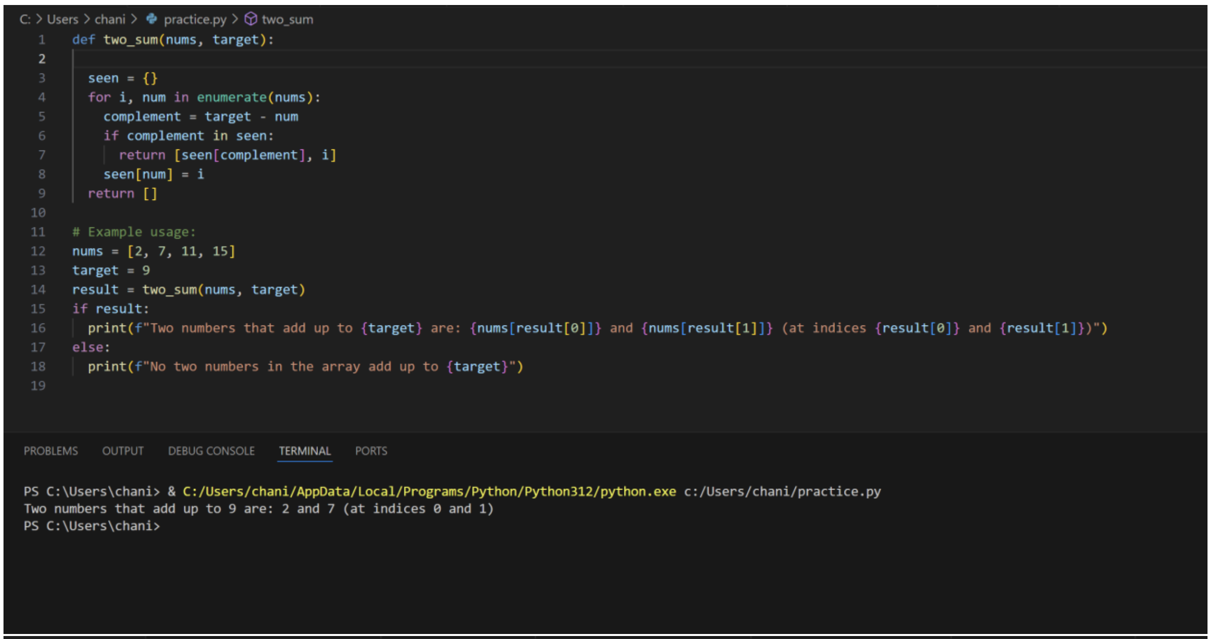
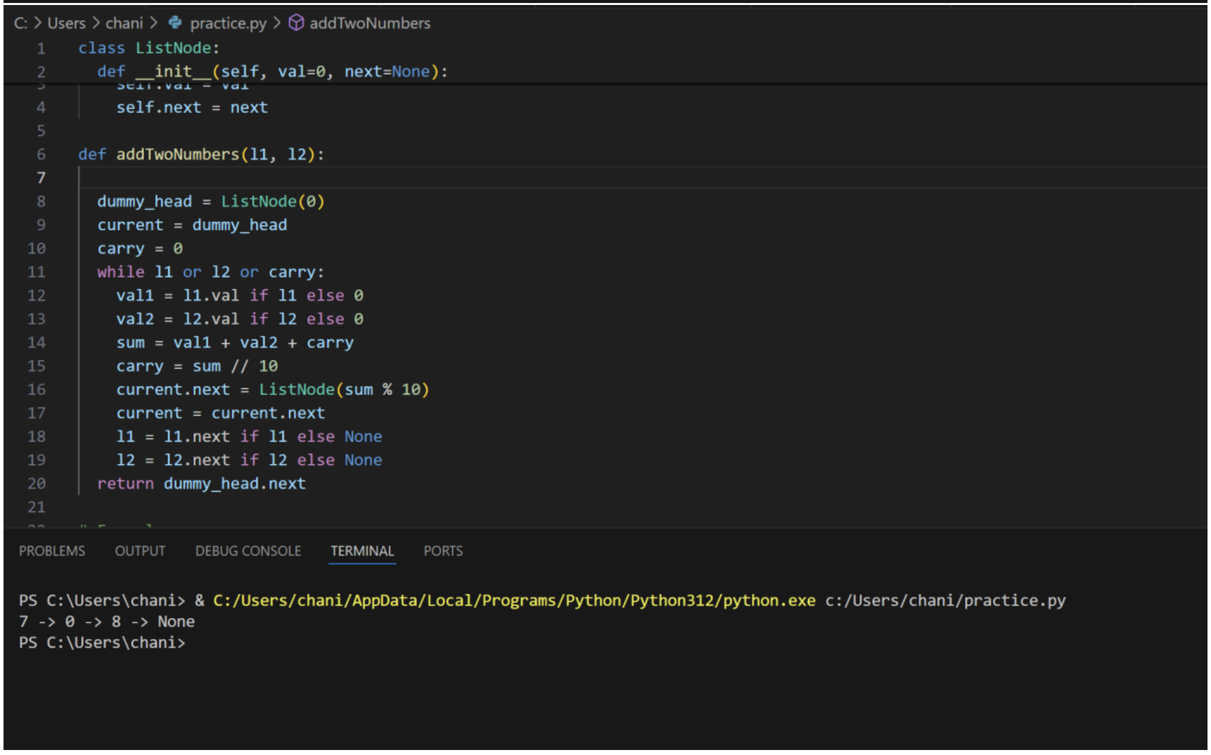


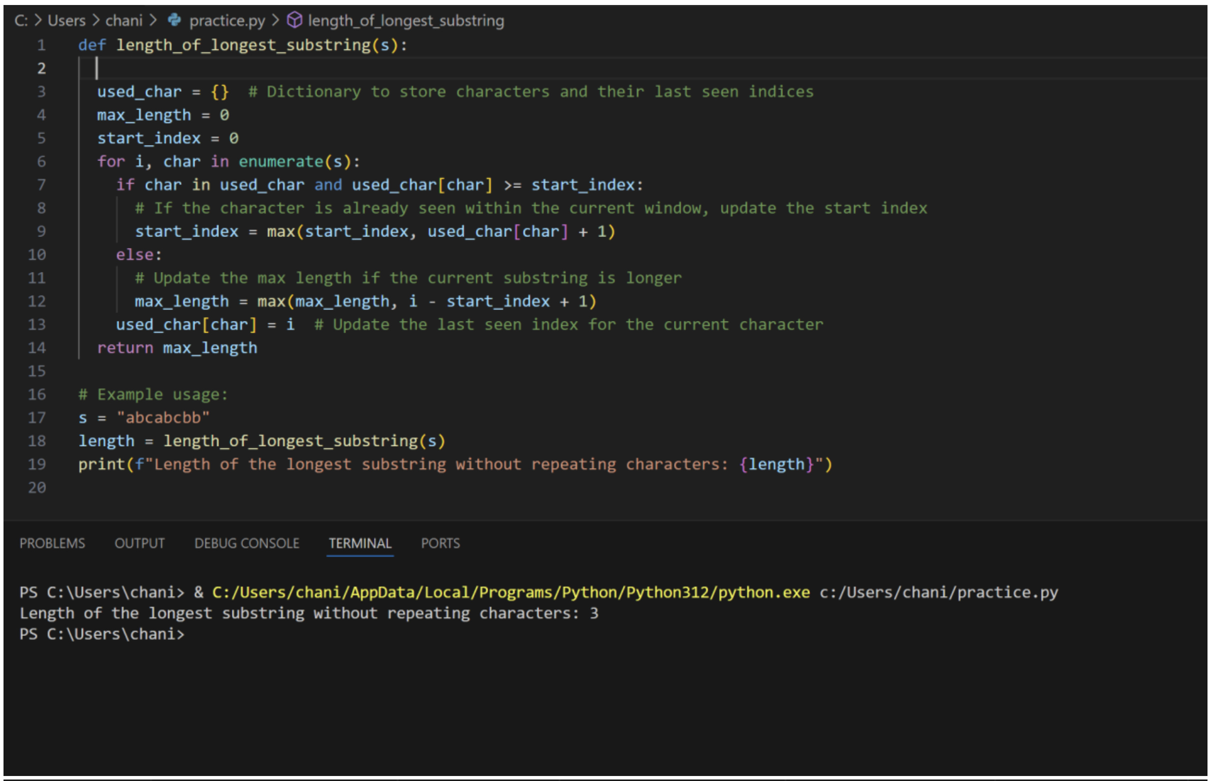
```python
def two_sum(nums, target):

    seen = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in seen:
            return [seen[complement], i]
        seen[num] = i
    return []

# Example usage:
nums = [2, 7, 11, 15]
target = 9
result = two_sum(nums, target)
if result:
    print(f"Two numbers that add up to {target} are: {nums[result[0]]} and {nums[result[1]]} (at indices {result[0]} and {result[1]})")
else:
    print(f"No two numbers in the array add up to {target}")
```

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
Two numbers that add up to 9 are: 2 and 7 (at indices 0 and 1)
PS C:\Users\chani>
```



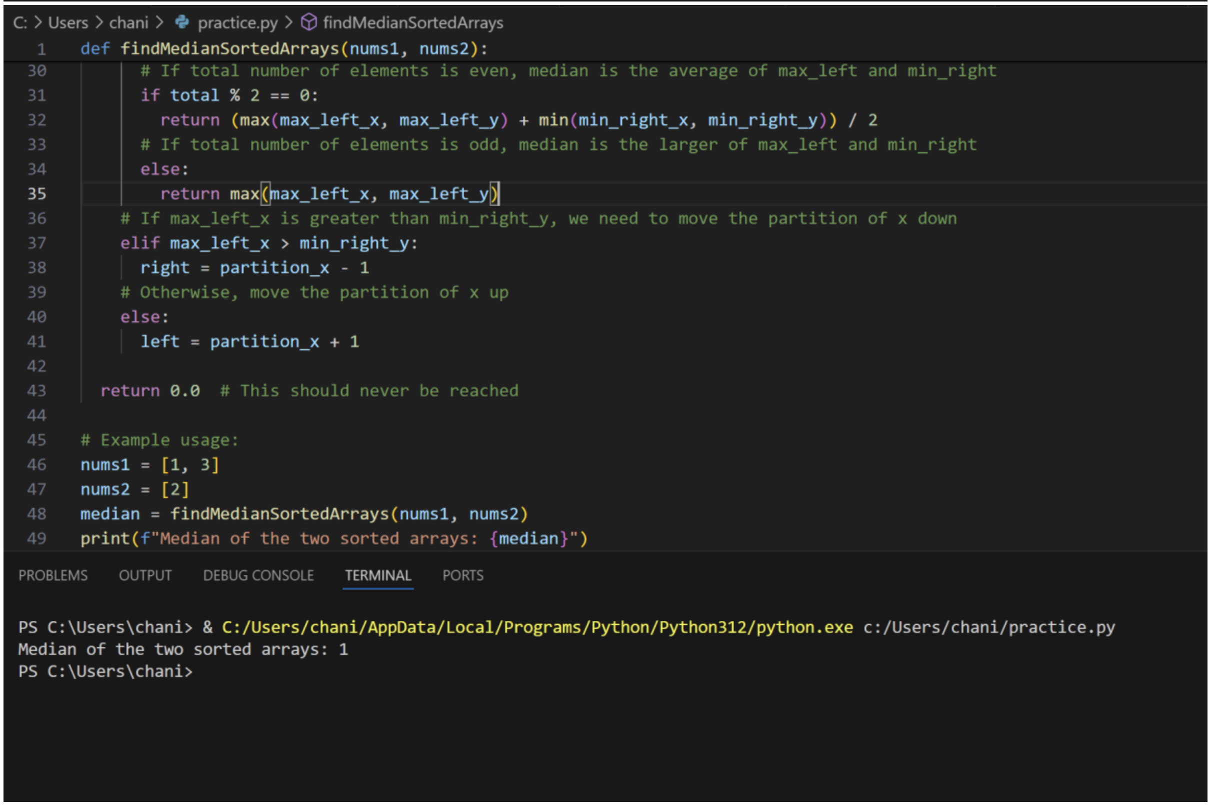
```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def addTwoNumbers(l1, l2):

    dummy_head = ListNode(0)
    current = dummy_head
    carry = 0
    while l1 or l2 or carry:
        val1 = l1.val if l1 else 0
        val2 = l2.val if l2 else 0
        sum = val1 + val2 + carry
        carry = sum // 10
        current.next = ListNode(sum % 10)
        current = current.next
        l1 = l1.next if l1 else None
        l2 = l2.next if l2 else None
    return dummy_head.next

```

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
7 -> 0 -> 8 -> None
PS C:\Users\chani>
```


C: > Users > chani > practice.py > length_of_longest_substring

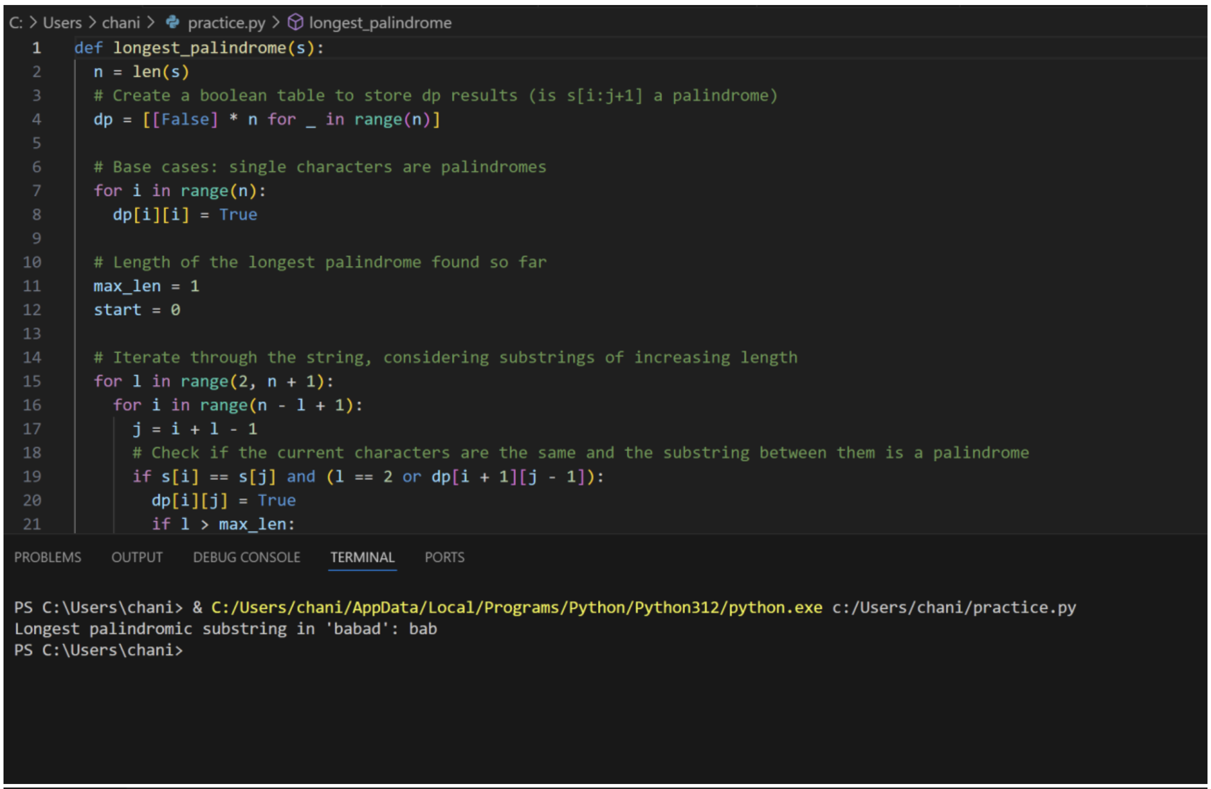
```python
def length_of_longest_substring(s):

    used_char = {}  # Dictionary to store characters and their last seen indices
    max_length = 0
    start_index = 0
    for i, char in enumerate(s):
        if char in used_char and used_char[char] >= start_index:
            # If the character is already seen within the current window, update the start index
            start_index = max(start_index, used_char[char] + 1)
        else:
            # Update the max length if the current substring is longer
            max_length = max(max_length, i - start_index + 1)
        used_char[char] = i  # Update the last seen index for the current character
    return max_length

# Example usage:
s = "abcabcbb"
length = length_of_longest_substring(s)
print(f"Length of the longest substring without repeating characters: {length}")
```

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
Length of the longest substring without repeating characters: 3
PS C:\Users\chani>
```



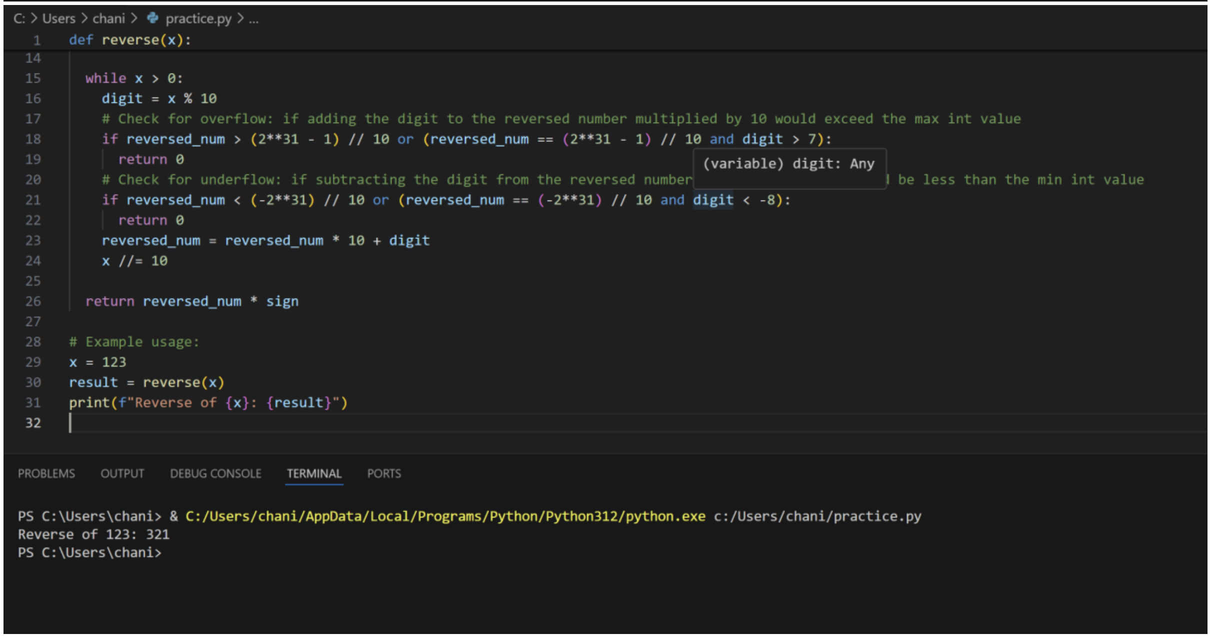
```python
def findMedianSortedArrays(nums1, nums2):
            # If total number of elements is even, median is the average of max_left and min_right
            if total % 2 == 0:
                return (max(max_left_x, max_left_y) + min(min_right_x, min_right_y)) / 2
            # If total number of elements is odd, median is the larger of max_left and min_right
            else:
                return max(max_left_x, max_left_y)
        # If max_left_x is greater than min_right_y, we need to move the partition of x down
        elif max_left_x > min_right_y:
            right = partition_x - 1
        # Otherwise, move the partition of x up
        else:
            left = partition_x + 1

    return 0.0  # This should never be reached

# Example usage:
nums1 = [1, 3]
nums2 = [2]
median = findMedianSortedArrays(nums1, nums2)
print(f"Median of the two sorted arrays: {median}")
```

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
Median of the two sorted arrays: 1
PS C:\Users\chani>
```



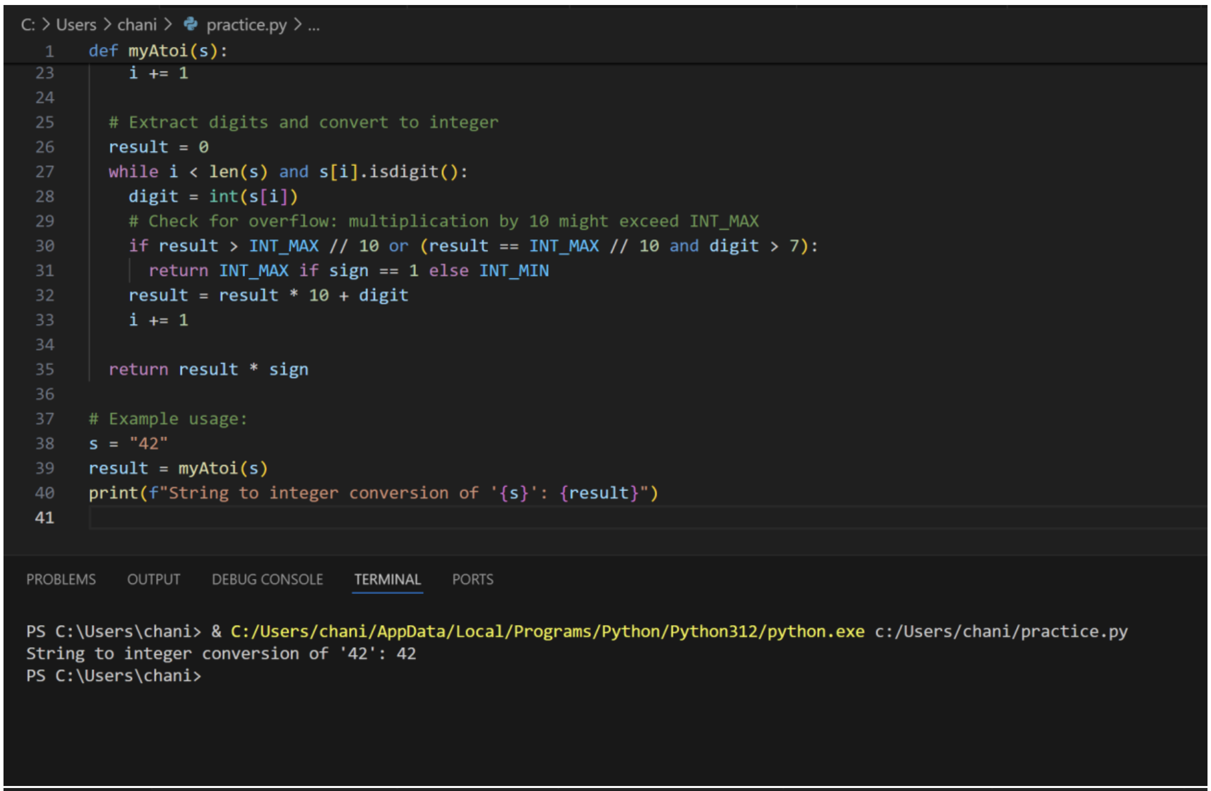
```python
def longest_palindrome(s):
  n = len(s)
  # Create a boolean table to store dp results (is s[i:j+1] a palindrome)
  dp = [[False] * n for _ in range(n)]

  # Base cases: single characters are palindromes
  for i in range(n):
    dp[i][i] = True

  # Length of the longest palindrome found so far
  max_len = 1
  start = 0

  # Iterate through the string, considering substrings of increasing length
  for l in range(2, n + 1):
    for i in range(n - l + 1):
      j = i + l - 1
      # Check if the current characters are the same and the substring between them is a palindrome
      if s[i] == s[j] and (l == 2 or dp[i + 1][j - 1]):
        dp[i][j] = True
        if l > max_len:
```

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
Longest palindromic substring in 'babad': bab
PS C:\Users\chani>
```



```python
def reverse(x):

  while x > 0:
    digit = x % 10
    # Check for overflow: if adding the digit to the reversed number multiplied by 10 would exceed the max int value
    if reversed_num > (2**31 - 1) // 10 or (reversed_num == (2**31 - 1) // 10 and digit > 7):
      return 0
    # Check for underflow: if subtracting the digit from the reversed number (variable) digit: Any l be less than the min int value
    if reversed_num < (-2**31) // 10 or (reversed_num == (-2**31) // 10 and digit < -8):
      return 0
    reversed_num = reversed_num * 10 + digit
    x //= 10

  return reversed_num * sign

# Example usage:
x = 123
result = reverse(x)
print(f"Reverse of {x}: {result}")

```

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
Reverse of 123: 321
PS C:\Users\chani>
```

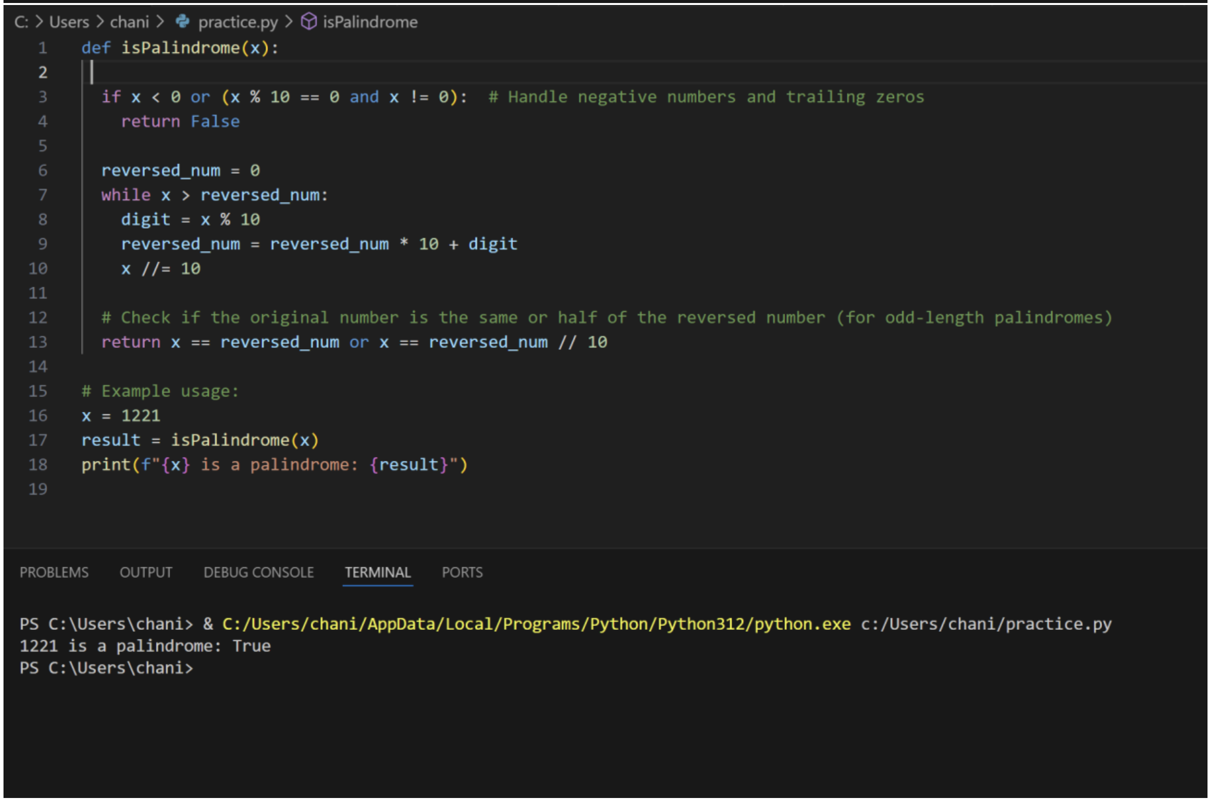


C: > Users > chani > practice.py > ...

```python
def myAtoi(s):
        i += 1

    # Extract digits and convert to integer
    result = 0
    while i < len(s) and s[i].isdigit():
        digit = int(s[i])
        # Check for overflow: multiplication by 10 might exceed INT_MAX
        if result > INT_MAX // 10 or (result == INT_MAX // 10 and digit > 7):
            return INT_MAX if sign == 1 else INT_MIN
        result = result * 10 + digit
        i += 1

    return result * sign

# Example usage:
s = "42"
result = myAtoi(s)
print(f"String to integer conversion of '{s}': {result}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
String to integer conversion of '42': 42
PS C:\Users\chani>
```

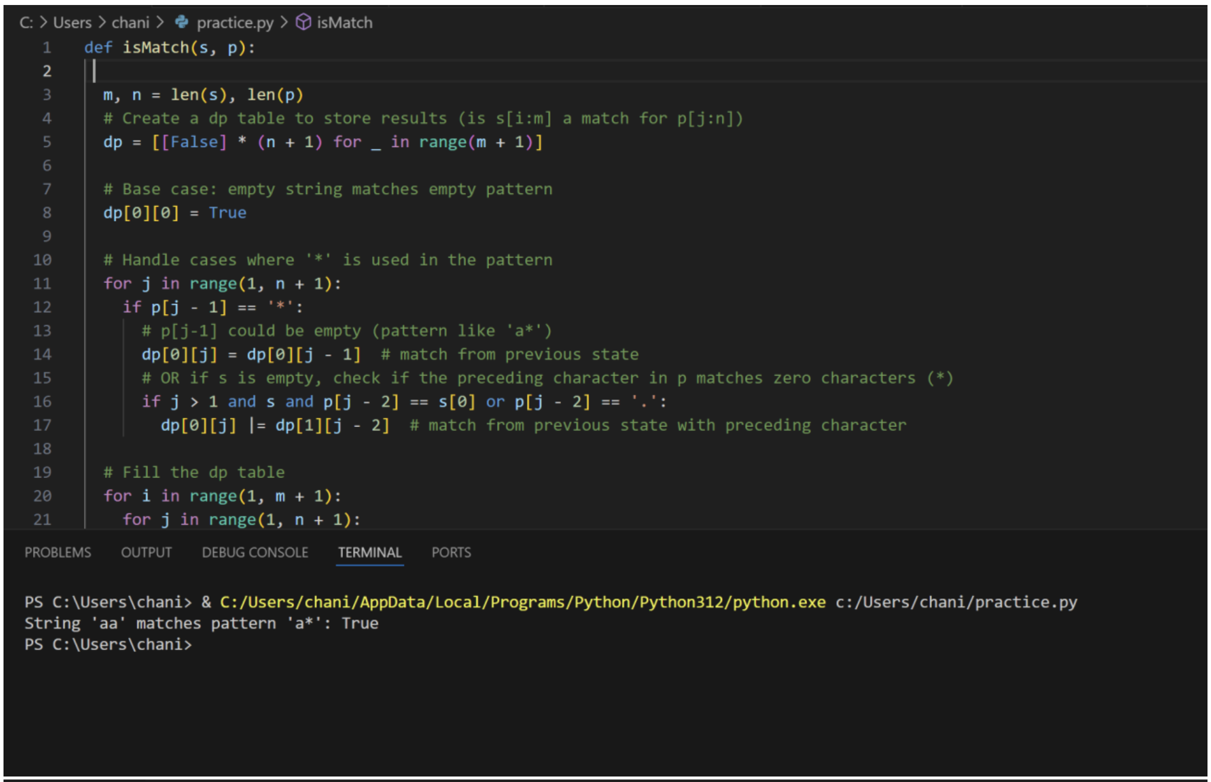


C: > Users > chani > practice.py > isPalindrome

```python
def isPalindrome(x):

    if x < 0 or (x % 10 == 0 and x != 0):  # Handle negative numbers and trailing zeros
        return False

    reversed_num = 0
    while x > reversed_num:
        digit = x % 10
        reversed_num = reversed_num * 10 + digit
        x //= 10

    # Check if the original number is the same or half of the reversed number (for odd-length palindromes)
    return x == reversed_num or x == reversed_num // 10

# Example usage:
x = 1221
result = isPalindrome(x)
print(f"{x} is a palindrome: {result}")

```

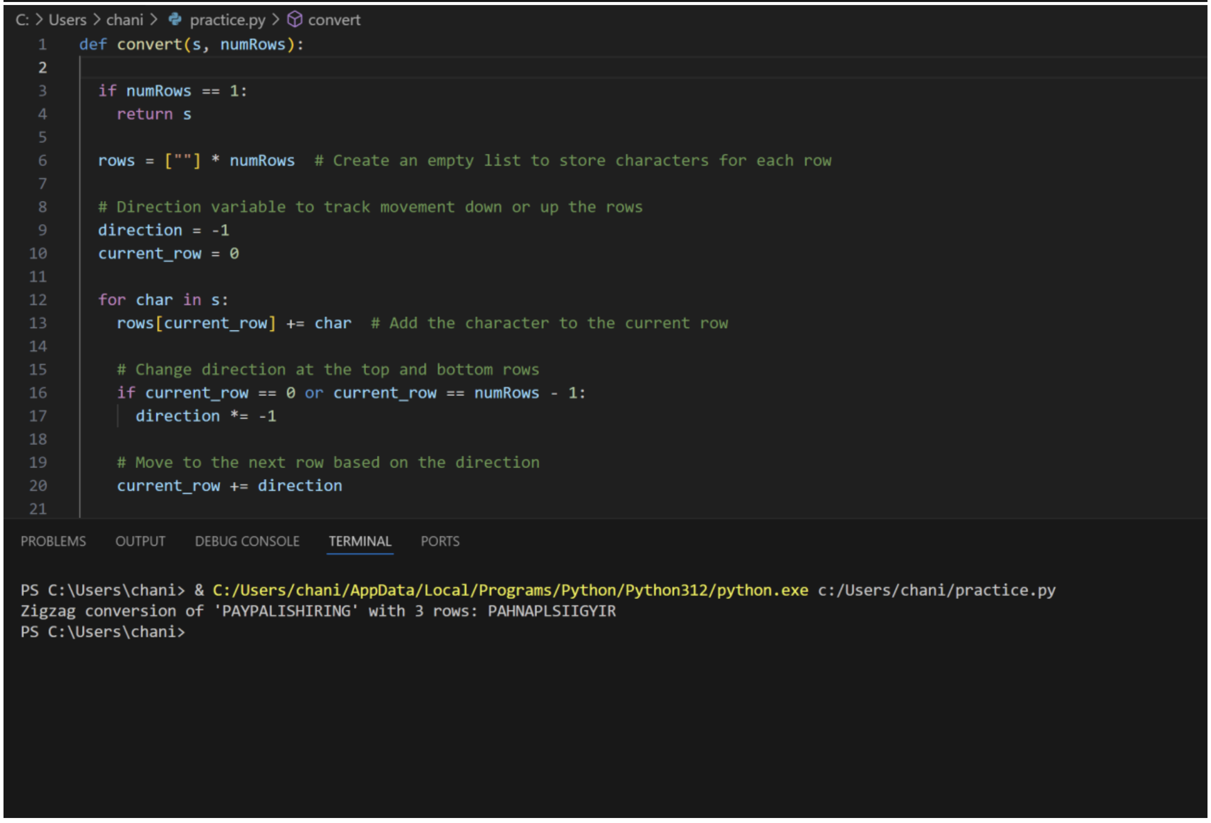
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
1221 is a palindrome: True
PS C:\Users\chani>
```



```python
def isMatch(s, p):

  m, n = len(s), len(p)
  # Create a dp table to store results (is s[i:m] a match for p[j:n])
  dp = [[False] * (n + 1) for _ in range(m + 1)]

  # Base case: empty string matches empty pattern
  dp[0][0] = True

  # Handle cases where '*' is used in the pattern
  for j in range(1, n + 1):
    if p[j - 1] == '*':
      # p[j-1] could be empty (pattern like 'a*')
      dp[0][j] = dp[0][j - 1]  # match from previous state
      # OR if s is empty, check if the preceding character in p matches zero characters (*)
      if j > 1 and s and p[j - 2] == s[0] or p[j - 2] == '.':
        dp[0][j] |= dp[1][j - 2]  # match from previous state with preceding character

  # Fill the dp table
  for i in range(1, m + 1):
    for j in range(1, n + 1):
```

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
String 'aa' matches pattern 'a*': True
PS C:\Users\chani>
```



```python
def convert(s, numRows):

  if numRows == 1:
    return s

  rows = [""] * numRows  # Create an empty list to store characters for each row

  # Direction variable to track movement down or up the rows
  direction = -1
  current_row = 0

  for char in s:
    rows[current_row] += char  # Add the character to the current row

    # Change direction at the top and bottom rows
    if current_row == 0 or current_row == numRows - 1:
      direction *= -1

    # Move to the next row based on the direction
    current_row += direction

```

```
PS C:\Users\chani> & C:/Users/chani/AppData/Local/Programs/Python/Python312/python.exe c:/Users/chani/practice.py
Zigzag conversion of 'PAYPALISHIRING' with 3 rows: PAHNAPLSIIGYIR
PS C:\Users\chani>
```