

Assignment 3

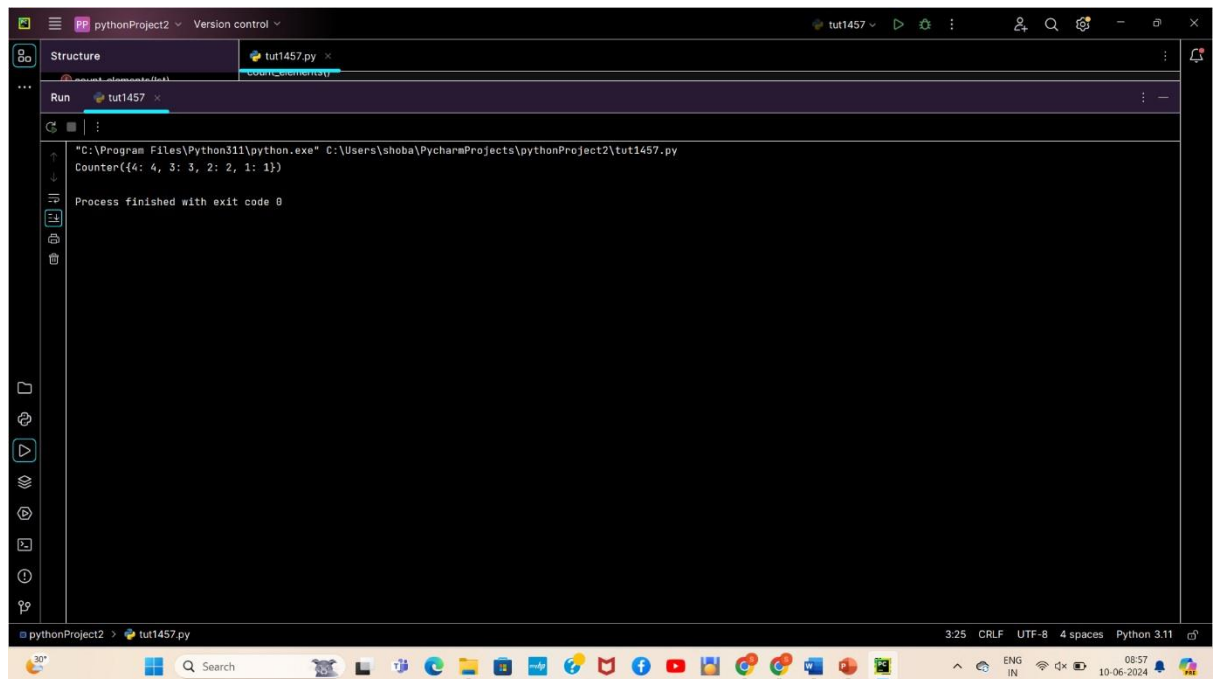
1. Counting Elements.

Program: -

```
from collections import Counter
def
count_elements(lst):
return Counter(lst)

# Example usage:
elements = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
element_counts = count_elements(elements)
print(element_counts)
```

Output:



```
pythonProject2 Version control tut1457
Structure tut1457.py x count_elements()
Run tut1457 x
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Counter({4: 4, 3: 3, 2: 2, 1: 1})
Process finished with exit code 0
pythonProject2 tut1457.py 3:25 CRLF UTF-8 4 spaces Python 3.11
```

2. Performing String Shifting.

Program:

```
def string_shift(s, shift):
    total_shift = 0
    for direction, amount in shift:
        total_shift += amount if direction == 1 else -amount

    total_shift %= len(s)

    s = s[-total_shift:] + s[:-total_shift]
    return s

s = "copilot"
shift_operations = [[1, 1], [0, 2], [1, 3]]
result = string_shift(s, shift_operations)
```

Output:

Copilot

[[1,1],[0,2],[1,3]]

3. Leftmost Column with least a One.

Program:

```
def leftmost_column_with_one(binary_matrix):
    # Start with the rightmost column
    leftmost_column = len(binary_matrix[0])
    # Iterate over each row
    for row in binary_matrix:
        # Use binary search to find the first '1' in the row
        low, high = 0, leftmost_column
        while low < high:
            mid = (low + high) // 2
            if row[mid] == 1:
                high = mid
            else:
                low = mid + 1
```

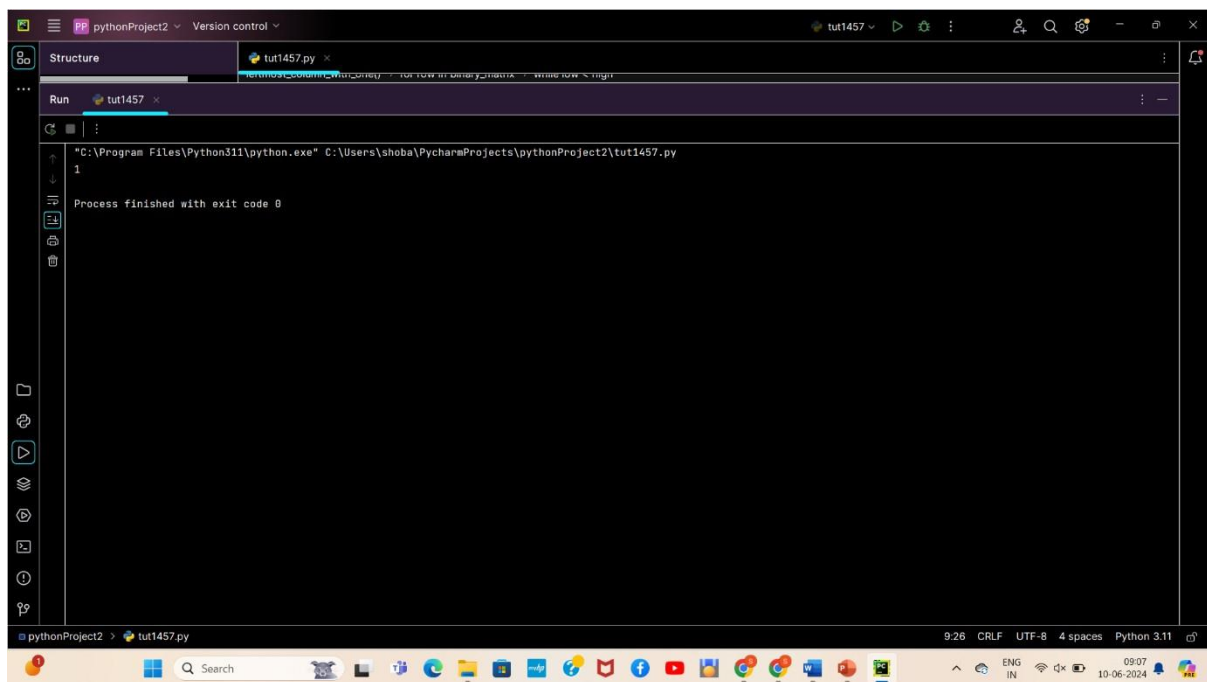
```

        # Update the index of the leftmost column with '1'
leftmost_column = min(leftmost_column, high)

        # If we have not found any '1', return -1
        return -1 if leftmost_column == len(binary_matrix[0]) else
leftmost_column
    binary_matrix
= [
    [0, 0,
0, 1],
    [0, 1, 1, 1],
    [0, 0, 1, 1],
    [0, 0, 0, 0]
]
print(leftmost_column_with_one(
    binary_matrix)) # Output will be 1, which is the index of the leftmost
column with at least a '1'

```

Output:



The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for file operations, search, and running code. The 'Run' button (a green play icon) is highlighted. Below the toolbar, the 'Run' console is open, showing the command prompt output: "C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py. The output of the script is "1". Below the command prompt, it says "Process finished with exit code 0". The bottom status bar shows the file path "pythonProject2 > tut1457.py", the time "9:26", the encoding "CRLF", the file encoding "UTF-8", the indentation "4 spaces", and the Python version "Python 3.11".

4. First Unique Number. Program:

```

from collections import OrderedDict
class FirstUnique:
    def
__init__(self, nums):
self.queue = OrderedDict()
self.is_unique = {}
for
num in nums:
self.add(num)
    def showFirstUnique(self):
for num in self.queue:
if self.is_unique[num]:
        return num
return -1
    def add(self, value):
        if
value not in self.is_unique:
self.is_unique[value] = True
self.queue[value] = None
        elif
self.is_unique[value]:
self.is_unique[value] = False
self.queue.pop(value)

# Example usage:
firstUnique = FirstUnique([2, 3, 5])
print(firstUnique.showFirstUnique())
firstUnique.add(2)
print(firstUnique.showFirstUnique())
firstUnique.add(3)
print(firstUnique.showFirstUnique())

```

Output:

The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for file operations, running, and debugging. The 'Run' tab is active, showing the execution of 'tut1457.py'. The console output displays the numbers 2, 3, and 5 on separate lines, followed by the message 'Process finished with exit code 0'. The bottom status bar indicates the file is 'tut1457.py' with 16 lines, 807 characters, and 32 line breaks, using CRLF line endings, UTF-8 encoding, and 4 spaces for indentation, running on Python 3.11.

5. Check If a String Is a Valid Sequence from Root to Leaves Path in a Binary Tree.

Program:

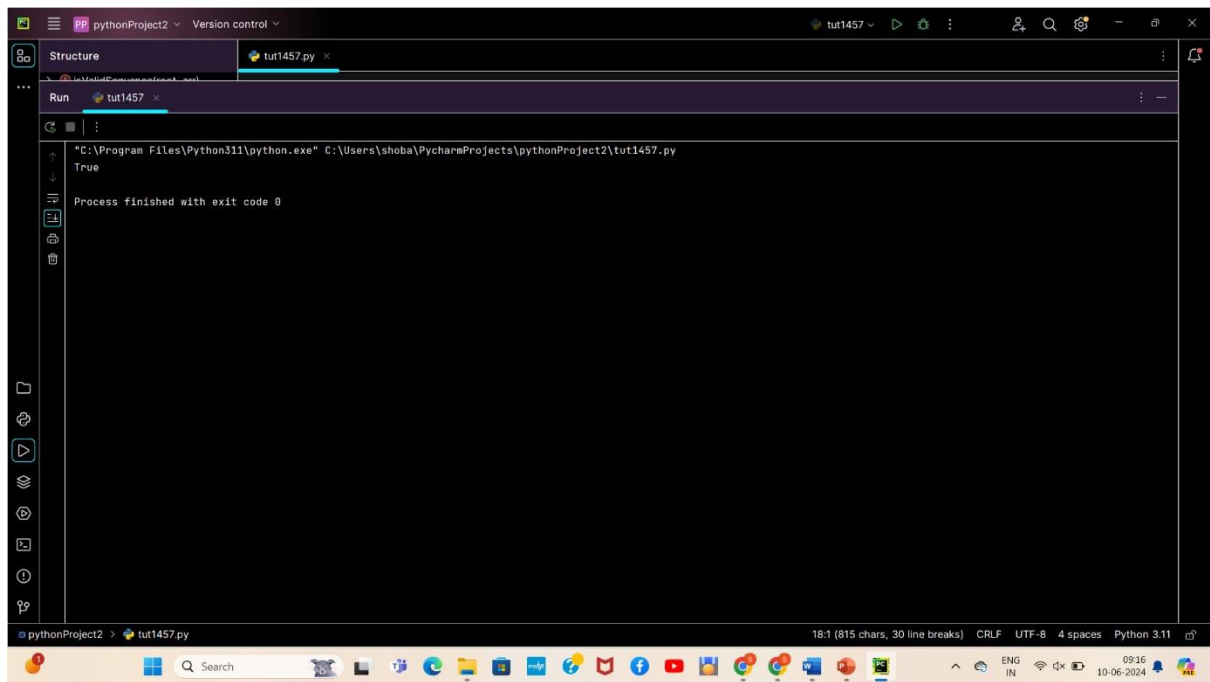
```
class TreeNode:    def __init__(self, value=0,
left=None, right=None):
    self.val = value
self.left = left
self.right = right
    def isValidSequence(root,
arr):
    def dfs(node, index):        if node is None or index == len(arr)
or node.val != arr[index]:            return False
        if index == len(arr) - 1 and node.left is None and node.right is
None:
return True
        return dfs(node.left, index + 1) or dfs(node.right, index + 1)

    return dfs(root, 0)

root = TreeNode(0) root.left =
TreeNode(1) root.right =
TreeNode(0) root.left.left =
TreeNode(0) root.left.left.left =
TreeNode(1) root.left.left.right =
TreeNode(0) root.right.left =
TreeNode(1) root.right.left.right
= TreeNode(0) root.right.right =
TreeNode(0)

print(isValidSequence(root, [0, 1, 0, 1]))
```

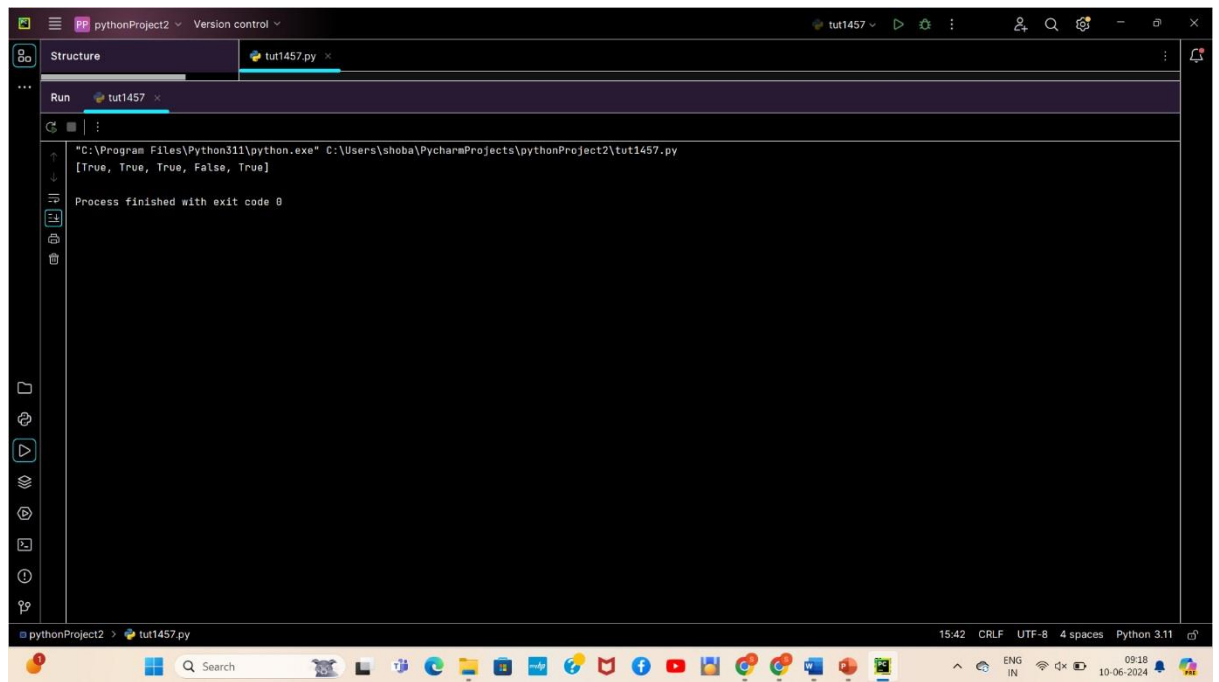
Output:



6. Kids With the Greatest Number of Candies. Program:

```
def kids_with_candies(candies, extra_candies):  
    max_candies = max(candies)  
  
    result = []  
    for candy in  
candies:  
        result.append(candy + extra_candies >= max_candies)  
    return  
result  
  
candies = [2, 3, 5, 1, 3] extra_candies = 3  
print(kids_with_candies(candies,  
extra_candies))  
# Output: [True, True, True, False, True]
```

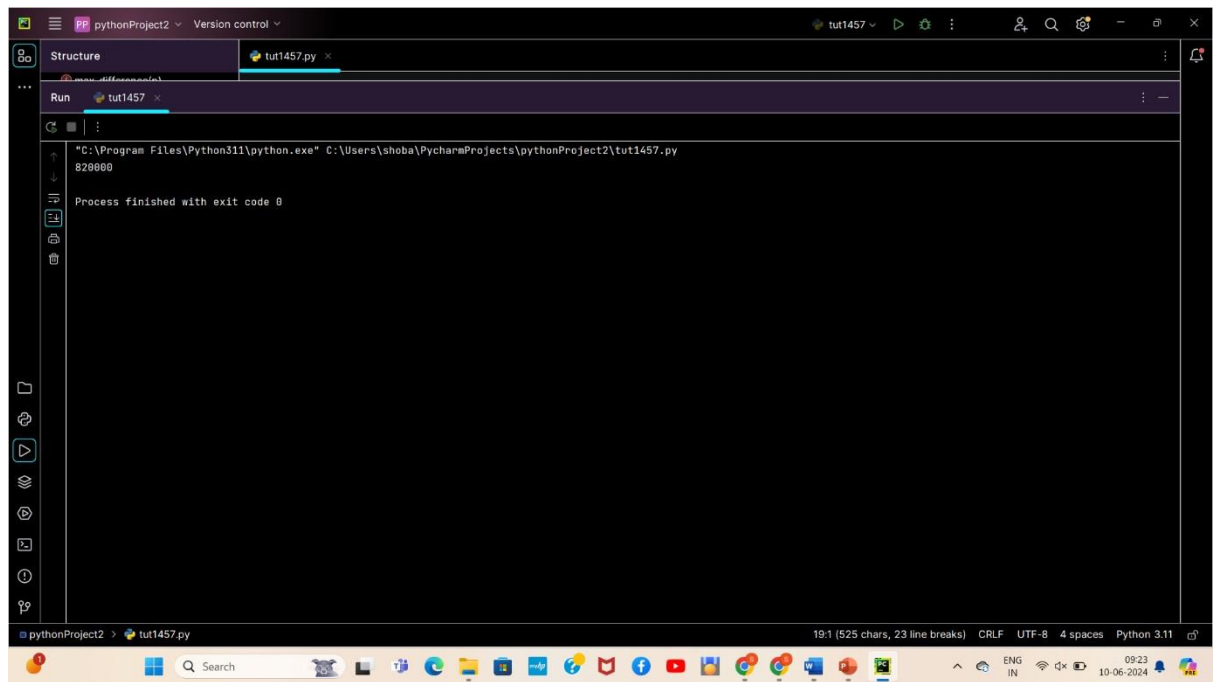
Output:



7. Max Difference You Can Get From Changing an Integer. Program:

```
def max_difference(n):
    str_n = str(n)
    max_val, min_val = str_n, str_n
    for i, digit in enumerate(str_n):
        if digit != '9':
            max_val = str_n[:i] + '9' + str_n[i + 1:]
            break
        if str_n[0] != '1':
            min_val = '1' + str_n[1:]
        else:
            for i, digit in enumerate(str_n[1:], start=1):
                if digit > '0':
                    min_val = str_n[:i] + '0' + str_n[i + 1:]
                    break
    return int(max_val) - int(min_val)
n = 123456
print(max_difference(n))
```

Output:



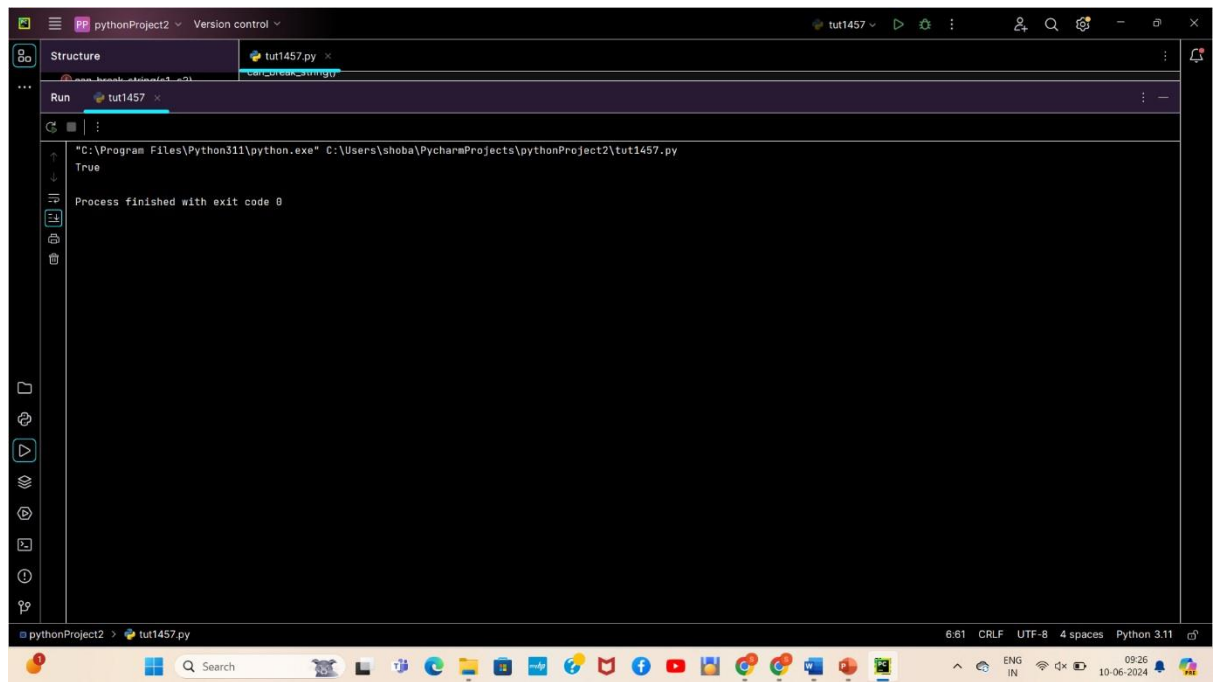
8. Check If a String Can Break Another String.

Program:

```
def can_break_string(s1, s2):
    sorted_s1 = sorted(s1)
    sorted_s2 = sorted(s2)
    can_s1_break_s2 = all(c1 >= c2 for c1, c2 in zip(sorted_s1, sorted_s2))
    can_s2_break_s1 = all(c2 >= c1 for c1, c2 in zip(sorted_s1, sorted_s2))
    return can_s1_break_s2 or can_s2_break_s1

s1 = "abc"
s2 = "xya"
print(can_break_string(s1, s2))
```

output:



9. Number of Ways to Wear Different Hats to Each Other.

Program:

```
def number_ways(hats):
    # Number of people
    n = len(hats)
    # All hat numbers available
    all_hats = set(range(1, 41))
    # Map each hat to the list of people who can wear it
    hat_to_people = {i: [] for i in range(1, 41)}
    for i, person_hats in enumerate(hats):
        for hat in person_hats:
            hat_to_people[hat].append(i)

    def backtrack(assigned, available_hats):
        if len(assigned) == n:
            return 1
        ways = 0
        next_person = len(assigned)
        for hat in available_hats:
            if next_person in hat_to_people[hat]:
                ways += backtrack(assigned + [hat], available_hats - {hat})
        return ways

    return backtrack([], all_hats)

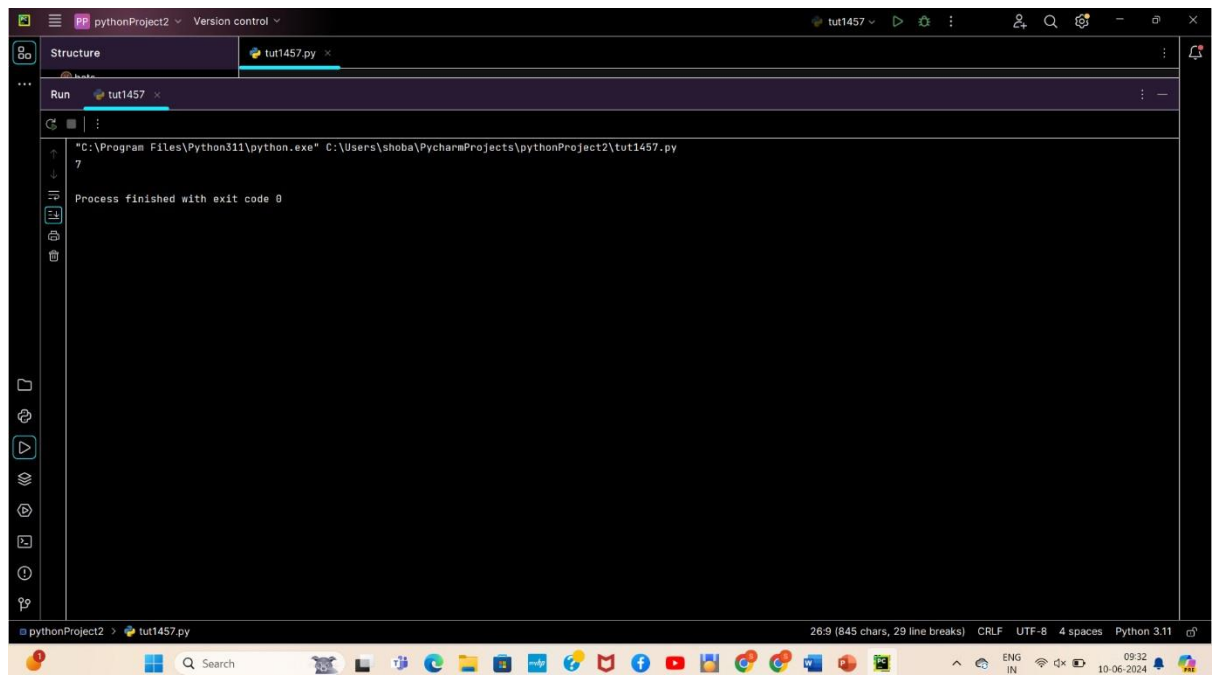
hats = [
```

```

    [1, 2, 3], [2, 3, 4],
]
print(number_ways(hats)) # Output will be the number of ways to wear different
hats

```

Output:



The screenshot shows the PyCharm IDE interface. The 'Run' tab is active, displaying the output of the script 'tut1457.py'. The output is the number '7', which is the result of the 'number_ways' function. The console also shows the command prompt path and the message 'Process finished with exit code 0'.

10. Next Permutations.

Program:

```

def number_ways(hats):
# Number of people      n
= len(hats)
    # All hat numbers available
all_hats = set(range(1, 41))
    # Map each hat to the list of people who can wear it
hat_to_people = {i: [] for i in range(1, 41)}
    for i,
person_hats in enumerate(hats):
        for hat in
person_hats:
            hat_to_people[hat].append(i)

```

```

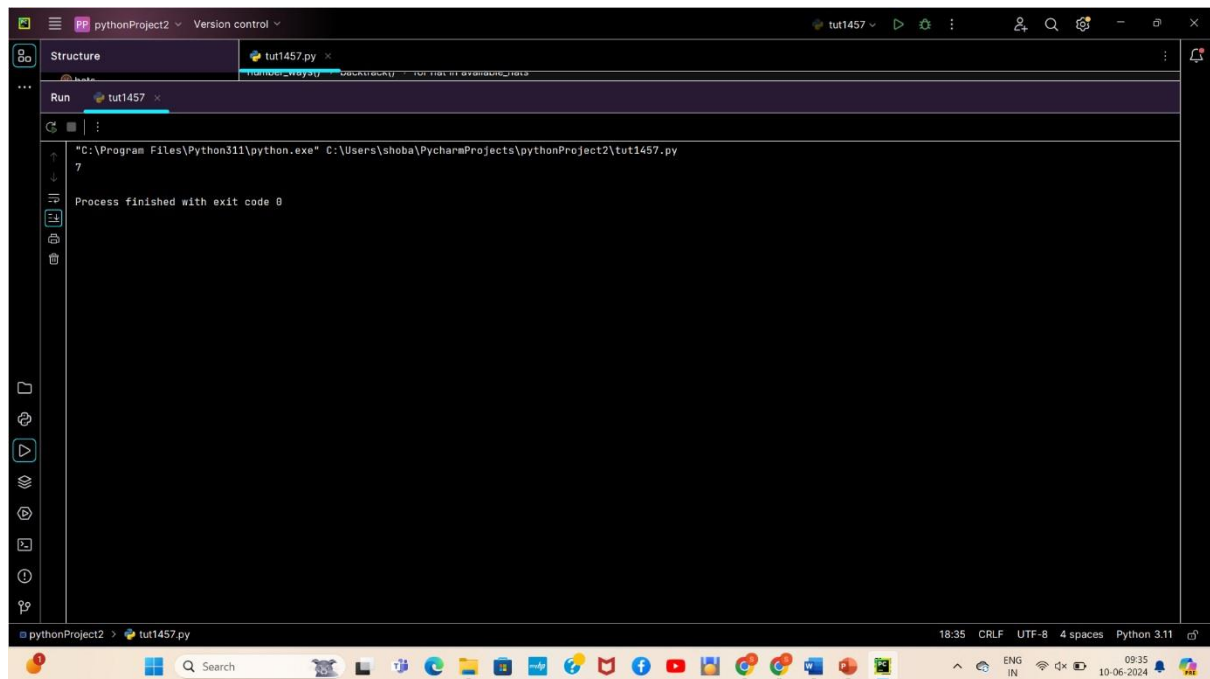
def backtrack(assigned, available_hats):
    if len(assigned) == n:
        return 1
ways = 0
    next_person = len(assigned)
    for hat in available_hats:
        if next_person in hat_to_people[hat]:
            ways += backtrack(assigned + [hat], available_hats - {hat})
    return ways

return backtrack([], all_hats)

hats = [
    [1, 2, 3], [2, 3, 4],
]
print(number_ways(hats)) # Output will be the number of ways to wear
different hats

```

Output:



The screenshot shows the PyCharm IDE interface. The top toolbar includes icons for running and debugging. The 'Run' tab is active, showing the command: `"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py`. The output of the script is displayed as `7`. Below the output, it states `Process finished with exit code 0`. The bottom status bar indicates the file is `tut1457.py`, the encoding is `UTF-8`, and the Python version is `3.11`. The system taskbar at the bottom shows the date as `10-06-2024` and the time as `18:35`.