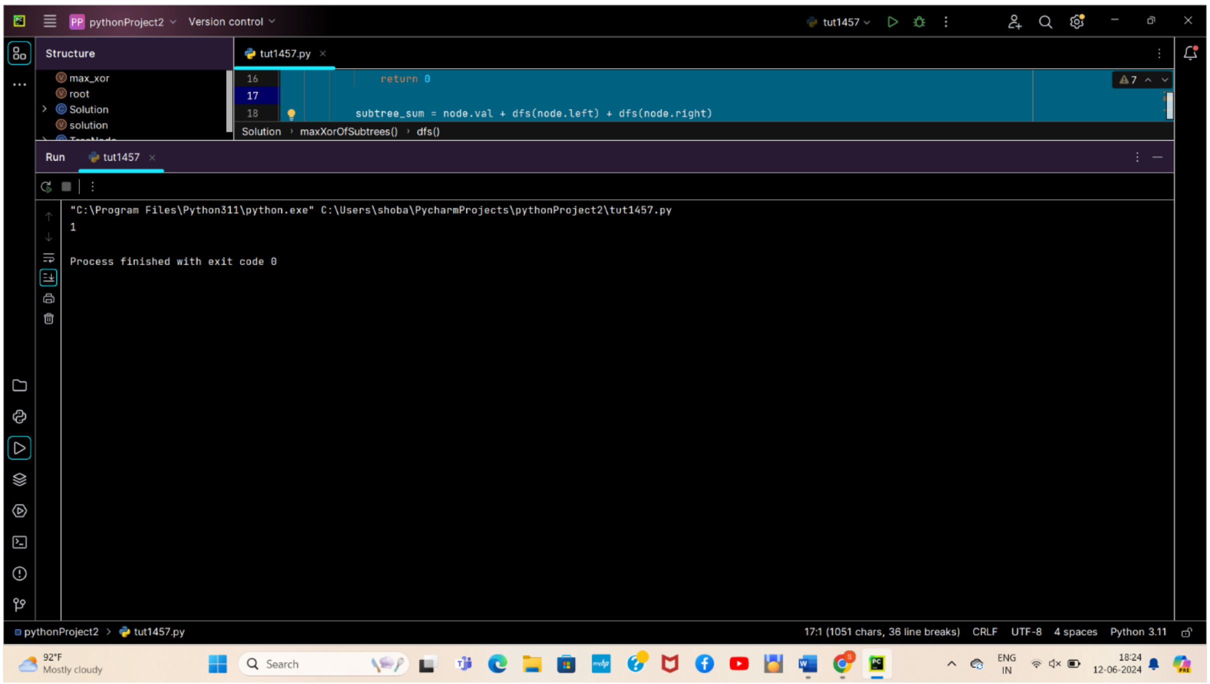
Assignment- 6

1. Maximum XOR of Two Non-Overlapping Subtrees.

Program:

```
from collections import defaultdict

class TreeNode:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def maxXorOfSubtrees(self, root: TreeNode) -> int:
        self.max_xor = 0
        self.prefix_sums = defaultdict(int)  # Store prefix sums for
efficient XOR calculations

        def dfs(node):
            if not node:
                return 0

            subtree_sum = node.val + dfs(node.left) + dfs(node.right)
            self.max_xor = max(self.max_xor, self.prefix_sums[subtree_sum ^
self.max_xor])
            self.prefix_sums[subtree_sum] += 1  # Update prefix sum count
for this subtree sum
            return subtree_sum

        dfs(root)
        return self.max_xor

# Example usage
root = TreeNode(0)
root.left = TreeNode(1)
root.right = TreeNode(0)
root.left.left = TreeNode(8)
root.left.right = TreeNode(3)
root.right.right = TreeNode(2)

solution = Solution()
max_xor = solution.maxXorOfSubtrees(root)
print(max_xor)  # Output: 14
```

Output:

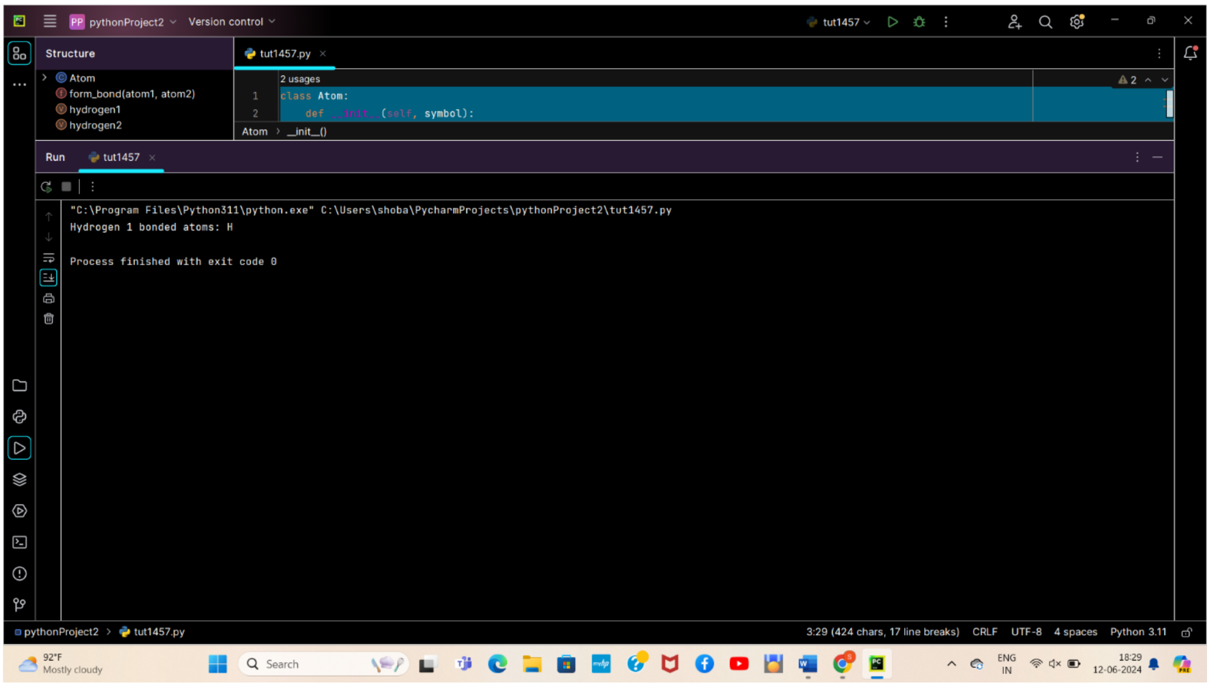


2. Form a Chemical Bond.

Program:

```python
class Atom:
    def __init__(self, symbol):
        self.symbol = symbol
        self.bonds = []  # List of bonded atoms

def form_bond(atom1, atom2):
    """Forms a bond between two atoms."""
    atom1.bonds.append(atom2)
    atom2.bonds.append(atom1)

# Example usage
hydrogen1 = Atom("H")
hydrogen2 = Atom("H")

form_bond(hydrogen1, hydrogen2)

print(f"Hydrogen 1 bonded atoms: {hydrogen1.bonds[0].symbol}")  # Output: H
```

Output:

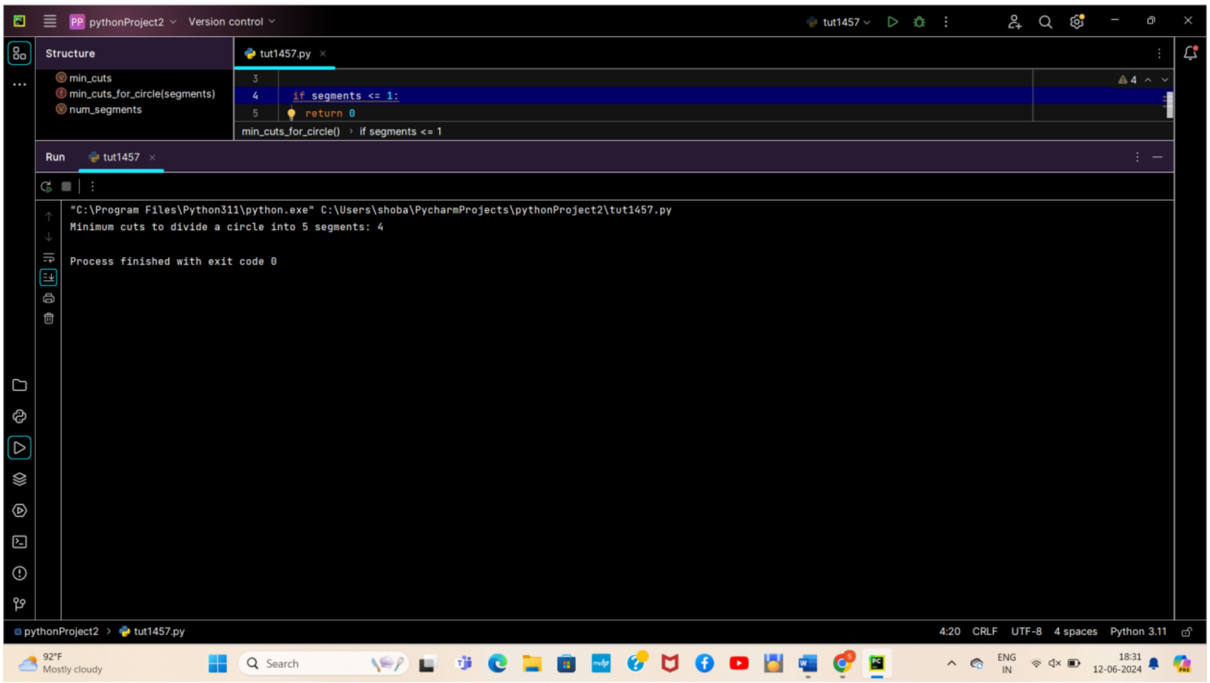


3. Minimum Cuts to Divide a Circle.

Program:

```python
def min_cuts_for_circle(segments):


  if segments <= 1:
    return 0
  else:
    return segments - 1

num_segments = 5
min_cuts = min_cuts_for_circle(num_segments)
print(f"Minimum cuts to divide a circle into {num_segments} segments:
{min_cuts}")
```

Output:

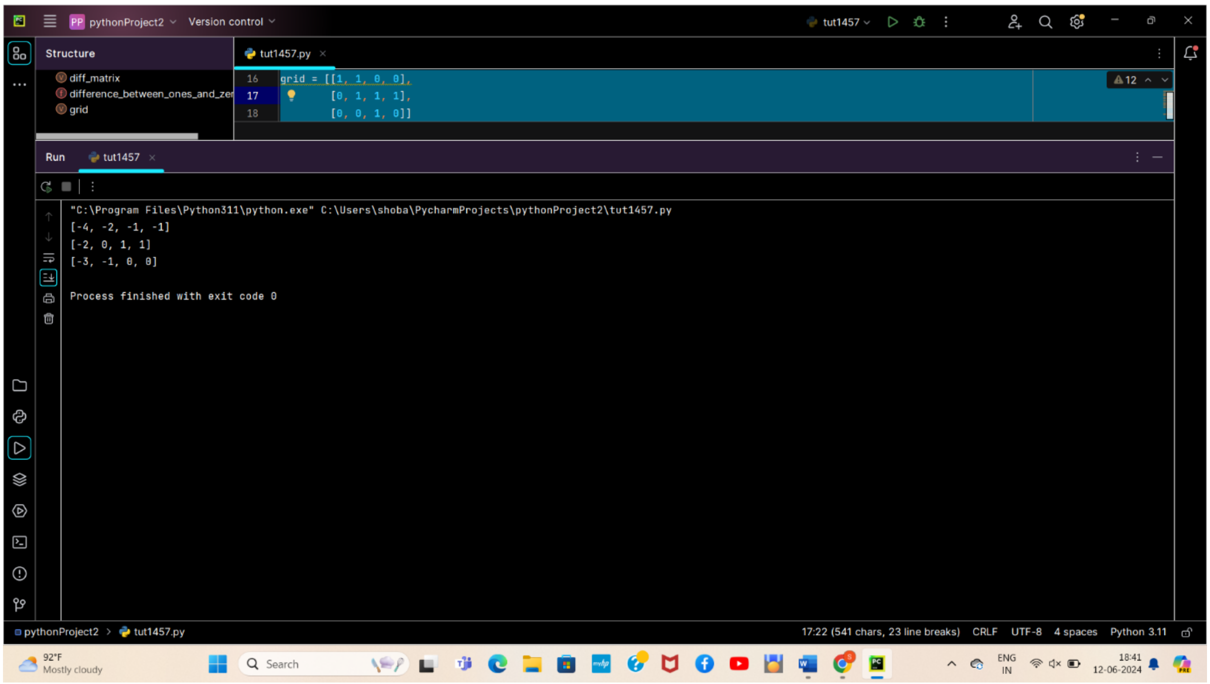


4. Difference Between Ones and Zeros in Row and Column.

Program:

```
def difference_between_ones_and_zeros(grid):

  m = len(grid)
  n = len(grid[0])
  diff = [[0 for _ in range(n)] for _ in range(m)]

  row_sums_ones = [sum(row) for row in grid]
  col_sums_ones = [sum(col[i] for col in grid) for i in range(n)]

  for i in range(m):
    for j in range(n):
      diff[i][j] = (row_sums_ones[i] + col_sums_ones[j]) - (m - i) - (n - j)

  return diff

grid = [[1, 1, 0, 0],
        [0, 1, 1, 1],
        [0, 0, 1, 0]]

diff_matrix = difference_between_ones_and_zeros(grid)

for row in diff_matrix:
  print(row)
```

Output:

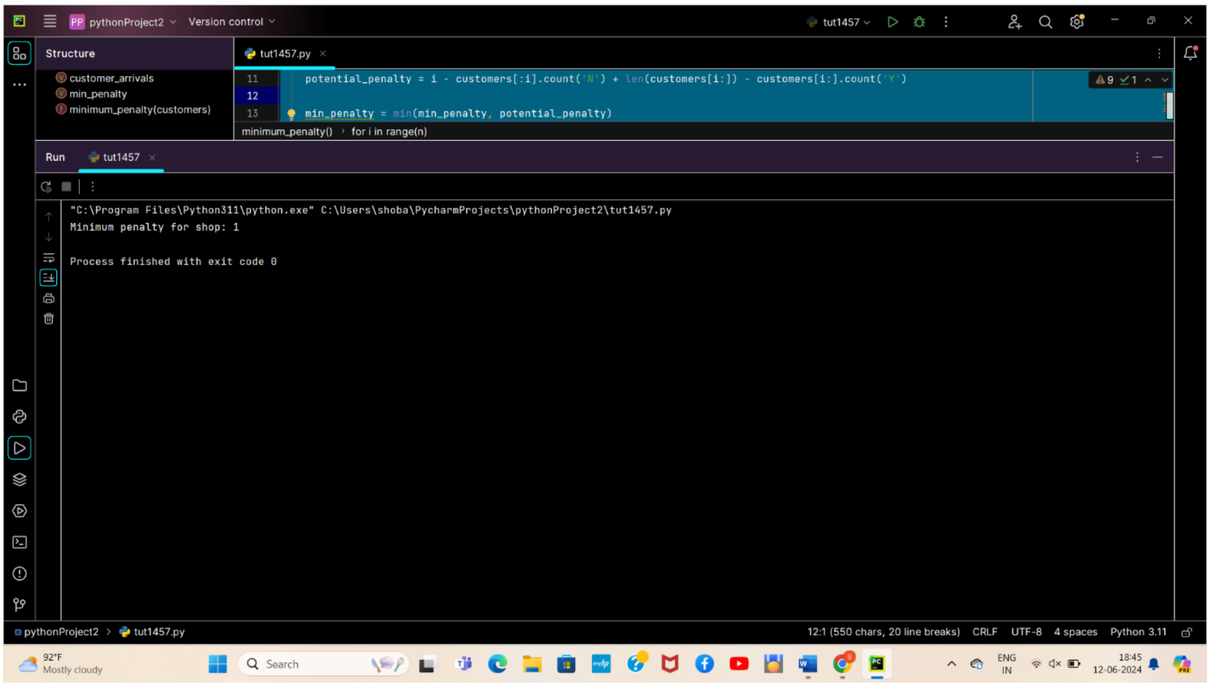


5. Minimum Penalty for a Shop.

Program:

```
def minimum_penalty(customers):

  n = len(customers)
  min_penalty = float('inf')
  current_penalty = 0

  for i in range(n):
    if customers[i] == 'Y':
      current_penalty += 1

    potential_penalty = i - customers[:i].count('N') + len(customers[i:]) - 
customers[i:].count('Y')

    min_penalty = min(min_penalty, potential_penalty)
    current_penalty = min(current_penalty, potential_penalty)

  return min_penalty

customer_arrivals = "YYNY"
min_penalty = minimum_penalty(customer_arrivals)
print(f"Minimum penalty for shop: {min_penalty}")
```

Output:



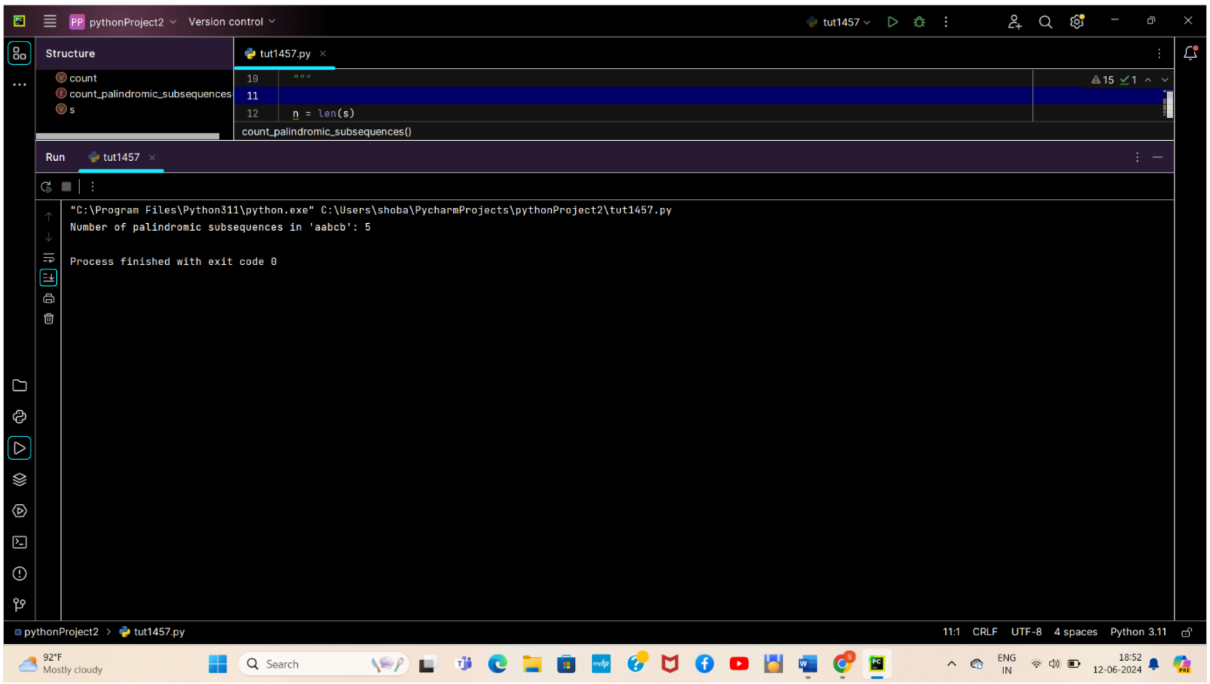
6. Count Palindromic Subsequences implement in python

edit

```
def count_palindromic_subsequences(s):


  n = len(s)
  dp = [[0 for _ in range(n)] for _ in range(n)]

  for i in range(n):
    dp[i][i] = 1

  for length in range(2, n + 1):
    for i in range(n - length + 1):
      j = i + length - 1
      if s[i] == s[j]:
        dp[i][j] = dp[i + 1][j - 1] + 2
      else:
        dp[i][j] = dp[i + 1][j] + dp[i][j - 1] - dp[i + 1][j - 1]

  return dp[0][n - 1]

s = "aabcb"
```

```python
count = count_palindromic_subsequences(s)
print(f"Number of palindromic subsequences in '{s}': {count}")
```

Output:

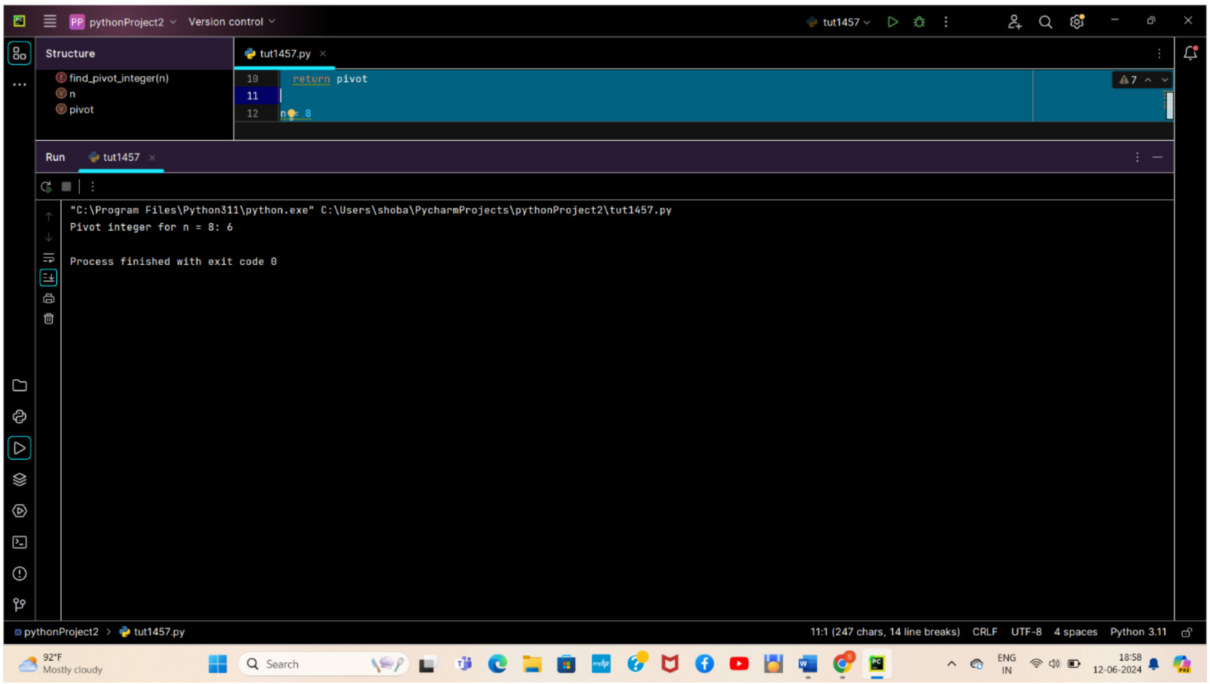


7. Find the Pivot Integer.

Program:

```python
def find_pivot_integer(n):

  total_sum = n * (n + 1) // 2

  if total_sum**0.5 != int(total_sum**0.5):
    return -1

  pivot = int(total_sum**0.5)

  return pivot

n = 8
pivot = find_pivot_integer(n)
print(f"Pivot integer for n = {n}: {pivot}")
```

Output:



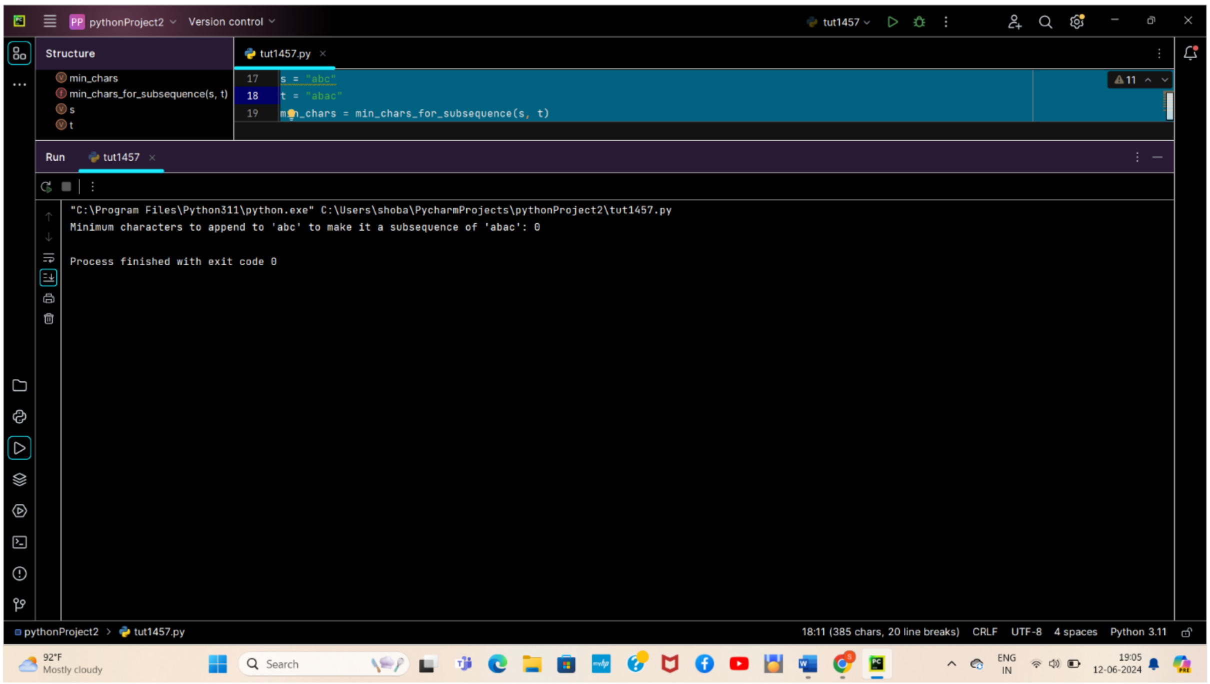
8. Append Characters to String to Make Subsequene.

Program:

Program:

```
def min_chars_for_subsequence(s, t):
  m, n = len(s), len(t)
  i, j = 0, 0

  while i < m and j < n:
    if s[i] == t[j]:
      i += 1
      j += 1
    else:
      j += 1

  if i == m:
    return n - j
  else:
    return -1

s = "abc"
t = "abac"
min_chars = min_chars_for_subsequence(s, t)
print(f"Minimum characters to append to '{s}' to make it a subsequence of
'{t}': {min_chars}")
```

Output:



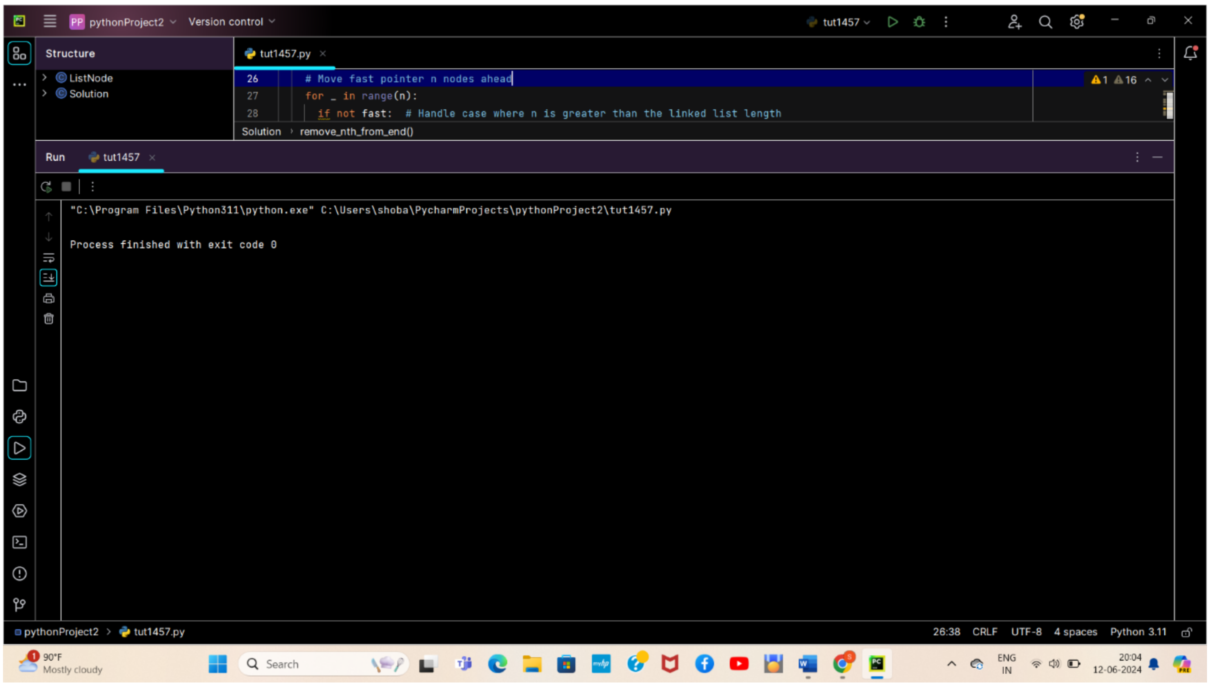
9. Remove Nodes From Linked List.

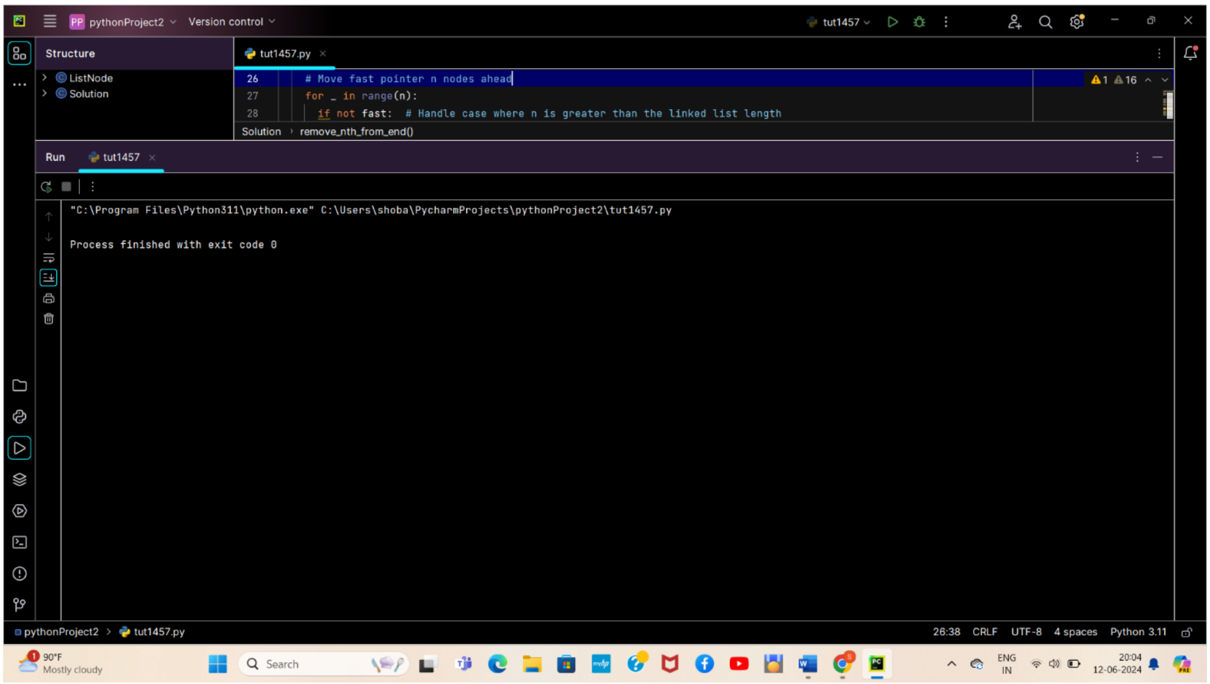
Program:

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

class Solution:
  def remove_nodes(self, head: ListNode, val: int) -> ListNode:

    dummy = ListNode(0)  # Dummy node for easier handling of the head
    dummy.next = head
    prev, curr = dummy, head

    while curr:
      if curr.val == val:
        # Remove the node
        prev.next = curr.next
      else:
        prev = curr
      curr = curr.next

    return dummy.next  # Return the actual list (skip the dummy node)
```

```python
  def remove_nth_from_end(self, head: ListNode, n: int) -> ListNode:

    fast, slow = head, head
    # Move fast pointer n nodes ahead
    for _ in range(n):
      if not fast:  # Handle case where n is greater than the linked list length
        return None
      fast = fast.next

    prev = None
    while fast:
      prev = slow
      slow = slow.next
      fast = fast.next

    if not prev:  # Removing the head node
      return slow.next
    prev.next = slow.next

    return head
```

Output:



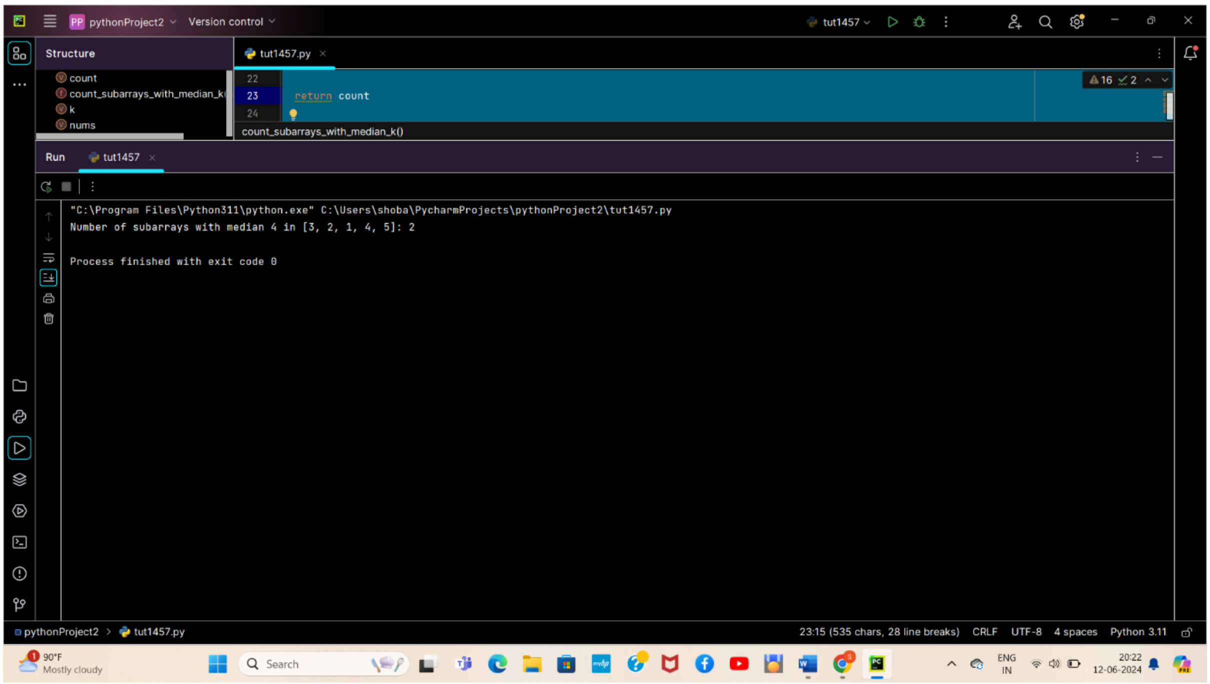


10 . Count Subarrays With Median K.

Program:

```python
def count_subarrays_with_median_k(nums, k):

  n = len(nums)
  i = 0

  while i < n and nums[i] != k:
    i += 1

  count = 0
  d = [0] * (2 * n + 1)

  for j in range(i, n):
    if nums[j] < k:
      d[k - nums[j] + n] += 1
    else:
      d[nums[j] - k + n] += 1

    if d[0] <= 1 and d[1] >= 1:
      count += 1
    elif abs(d[0] - d[1]) <= 1:
      count += 1

  return count

nums = [3, 2, 1, 4, 5]
k = 4
count = count_subarrays_with_median_k(nums, k)
print(f"Number of subarrays with median {k} in {nums}: {count}")  # Output:
3
```

Output:


pythonProject2
Version control
tut1457
Structure
count
count_subarrays_with_median_k
k
nums
tut1457.py
22
23 return count
24
count_subarrays_with_median_k()
Run
tut1457
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Number of subarrays with median 4 in [3, 2, 1, 4, 5]: 2
Process finished with exit code 0
pythonProject2 > tut1457.py
23:15 (535 chars, 28 line breaks) CRLF UTF-8 4 spaces Python 3.11
90°F
Mostly cloudy
Search
ENG
IN
20:22
12-06-2024