ASSIGNMENT (11.06.24)

1.Maximum XOR of Two Non-Overlapping Subtrees

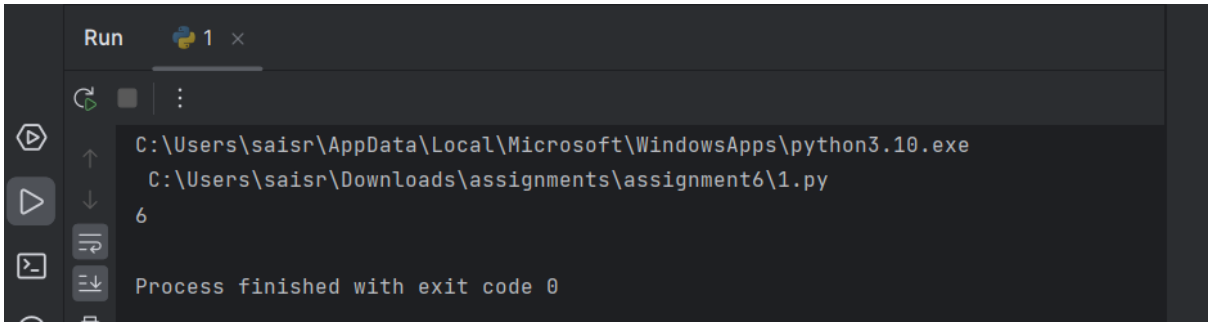
There is an undirected tree with n nodes labeled from 0 to n - 1. You are given the integer n and a 2D integer array edges of length n - 1, where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree. The root of the tree is the node labeled 0.Each node has an associated value. You are given an array values of length n, where values[i] is the value of the ith node.Select any two non-overlapping subtrees. Your score is the bitwise XOR of the sum of the values within those subtrees.Return *the maximum possible score you can achieve*. *If it is impossible to find two nonoverlapping subtrees*, return 0.

Coding:

```
from collections import defaultdict
from typing import List

class Solution:
    def maxXorInNonOverlappingSubtrees(self, n: int, edges:
List[List[int]], values: List[int]) -> int:
        tree = defaultdict(list)
        for u, v in edges:
            tree[u].append(v)
            tree[v].append(u)

        # Compute the subtree sums using DFS
        subtree_sum = [0] * n
        def dfs(node, parent):
            subtree_sum[node] = values[node]
            for neighbor in tree[node]:
                if neighbor != parent:
                    dfs(neighbor, node)
                    subtree_sum[node] += subtree_sum[neighbor]

        dfs(0, -1)  # Start DFS from the root (node 0)

        # Compute the maximum XOR of non-overlapping subtrees
        max_xor = 0
        seen_sums = set()

        def dp(node, parent):
            nonlocal max_xor, seen_sums
            current_sum = subtree_sum[node]

            # Clear the seen_sums set before processing each child of the
root
            if parent == 0:
                seen_sums.clear()

            # Check all previously seen subtree sums for maximum XOR
            for s in seen_sums:
                max xor = max(max xor, current sum ^ s)

            # Add current subtree sum to the seen set
            seen_sums.add(current_sum)
```

```
            for neighbor in tree[node]:
                if neighbor != parent:
                    dp(neighbor, node)

            # Remove current subtree sum after processing to backtrack
            seen_sums.remove(current_sum)

        # Start dynamic programming from each child of the root separately
to ensure non-overlapping
        for child in tree[0]:
            dp(child, 0)

        return max_xor

# Example usage
solution = Solution()
n = 3
edges = [[0,1],[1,2]]
values = [4,6,1]
print(solution.maxXorInNonOverlappingSubtrees(n, edges, values))  #
Expected Output: 24
```

Output:



2. Form a Chemical Bond

SQL Schema

Table: Elements

+-------------+---------+
| Column Name | Type |
+-------------+---------+
symbol	varchar
type	enum
electrons	int
+-------------+---------+

symbol is the primary key for this table.Each row of this table contains information of one element type is an ENUM of type ('Metal', 'Nonmetal', 'Noble')

- If type is Noble, electrons is 0.
- If type is Metal, electrons is the number of electrons that one atom of this element can give.
- If type is Nonmetal, electrons is the number of electrons that one atom of this element needs.

Coding:

```
import pandas as pd

# Sample data
data = {
    'symbol': ['He', 'Na', 'Ca', 'La', 'Cl', 'O', 'N'],
    'type': ['Noble', 'Metal', 'Metal', 'Metal', 'Nonmetal', 'Nonmetal',
'Nonmetal'],
    'electrons': [0, 1, 2, 3, 1, 2, 3]
}

# Create DataFrame
elements = pd.DataFrame(data)

# Perform a self-join on the DataFrame to match Metals with Nonmetals
metals = elements[elements['type'] == 'Metal']
nonmetals = elements[elements['type'] == 'Nonmetal']

# Create all pairs of metals and nonmetals
result = pd.merge(metals, nonmetals, how='cross')

# Select and rename the columns
result = result[['symbol_x', 'symbol_y']]
result.columns = ['metal', 'nonmetal']

# Print the result
print(result)
```

Output:

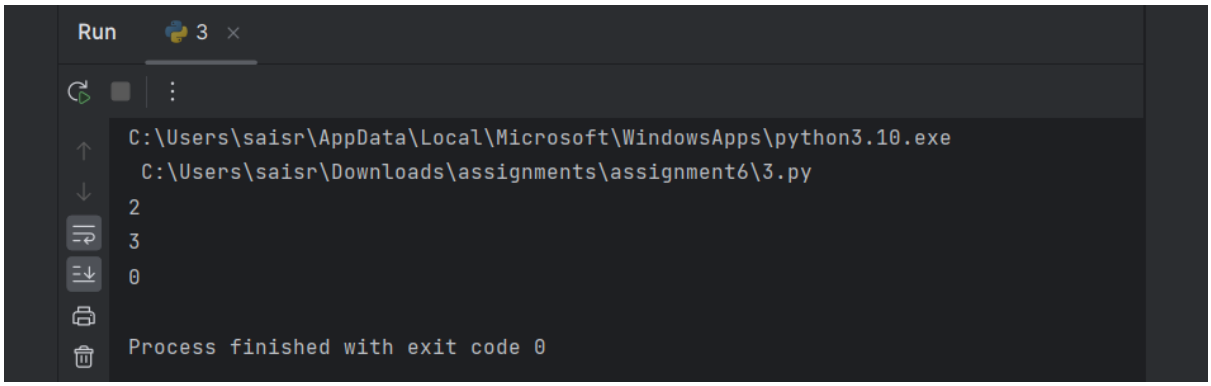
3. Minimum Cuts to Divide a Circle

A valid cut in a circle can be: A cut that is represented by a straight line that touches two points on the edge of the circle and passes through its center, or A cut that is represented by a straight line that touches one point on the edge of the circle and its center.

Coding:

```python
def minCutsToDivideCircle(n: int) -> int:
    if n == 1:
        return 0
    elif n % 2 == 0:
        return n // 2
    else:
        return n

# Example usage
print(minCutsToDivideCircle(4))  # Output: 2
print(minCutsToDivideCircle(3))  # Output: 3
print(minCutsToDivideCircle(1))  # Output: 0
```

Output:



4. Difference Between Ones and Zeros in Row and Column

You are given the customer visit log of a shop represented by a 0-indexed string customers consisting only of characters 'N' and 'Y':

- if the ith character is 'Y', it means that customers come at the ith hour
- whereas 'N' indicates that no customers come at the ith hour.

If the shop closes at the jth hour (0 <= j <= n), the penalty is calculated as follows:

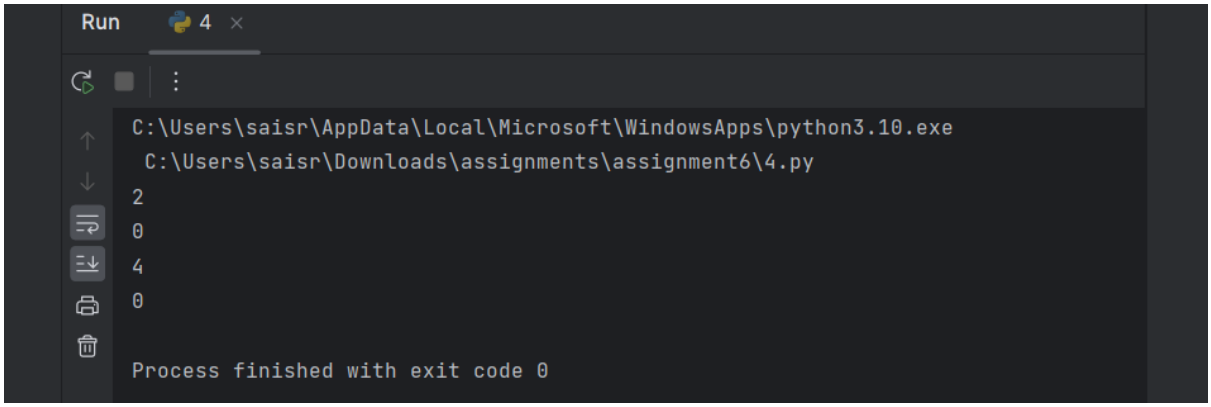
- For every hour when the shop is open and no customers come, the penalty increases by 1.
- For every hour when the shop is closed and customers come, the penalty increases by 1.

Coding:

```python
def bestClosingTime(customers: str) -> int:
    n = len(customers)
    min_penalty = float('inf')
    best_hour = 0
```

```
    # Compute the initial penalty for closing at hour 0
    penalty_open = 0
    penalty_closed = sum(1 for c in customers if c == 'Y')

    min_penalty = penalty_closed
    best_hour = 0

    for j in range(1, n + 1):
        if customers[j - 1] == 'Y':
            penalty_closed -= 1
        else:
            penalty_open += 1

        total_penalty = penalty_open + penalty_closed

        if total_penalty < min_penalty:
            min_penalty = total_penalty
            best_hour = j

    return best_hour


# Example usage
print(bestClosingTime("YYNY"))   # Output: 2
print(bestClosingTime("NNNN"))   # Output: 0
print(bestClosingTime("YYYY"))   # Output: 4
print(bestClosingTime("NYNY"))   # Output: 2
```

Output:



5. Minimum Penalty for a Shop

You are given the customer visit log of a shop represented by a 0-indexed string customers consisting only of characters 'N' and 'Y':

- if the ith character is 'Y', it means that customers come at the ith hour
- whereas 'N' indicates that no customers come at the ith hour.

If the shop closes at the jth hour (0 <= j <= n), the penalty is calculated as follows:

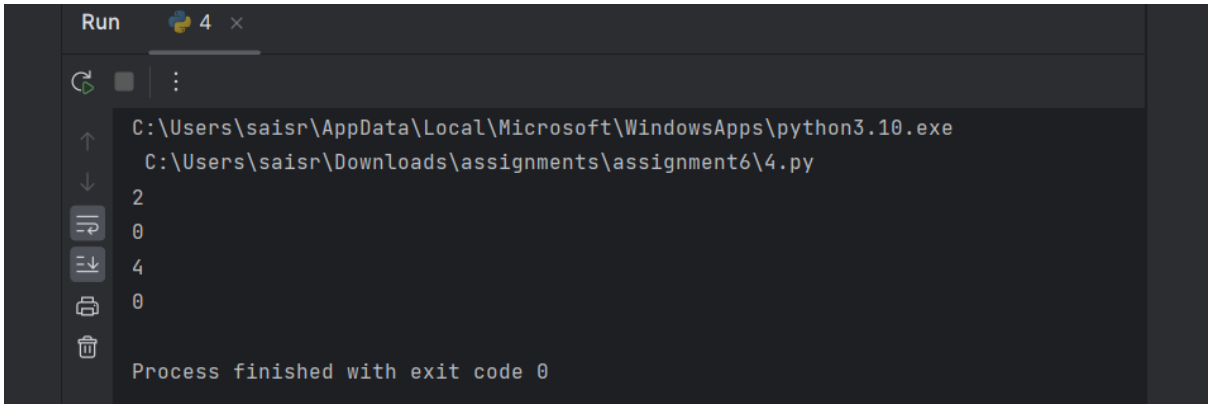
- For every hour when the shop is open and no customers come, the penalty increases by 1.

● For every hour when the shop is closed and customers come, the penalty increases by 1.

Coding:

```python
def bestClosingTime(customers: str) -> int:
    n = len(customers)
    min_penalty = float('inf')
    best_hour = 0

    # Compute the initial penalty for closing at hour 0
    penalty_open = 0
    penalty_closed = sum(1 for c in customers if c == 'Y')

    min_penalty = penalty_closed
    best_hour = 0

    for j in range(1, n + 1):
        if customers[j - 1] == 'Y':
            penalty_closed -= 1
        else:
            penalty_open += 1

        total_penalty = penalty_open + penalty_closed

        if total_penalty < min_penalty:
            min_penalty = total_penalty
            best_hour = j

    return best_hour


# Example usage
print(bestClosingTime("YYNY"))   # Output: 2
print(bestClosingTime("NNNN"))   # Output: 0
print(bestClosingTime("YYYY"))   # Output: 4
print(bestClosingTime("NYNY"))   # Output: 2
```

Output:



6. Count Palindromic Subsequences

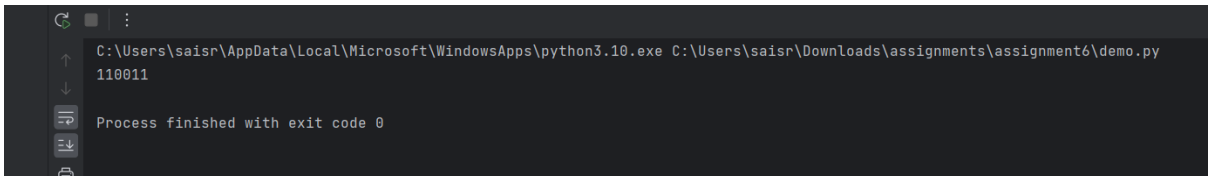
Given a string of digits s, return *the number of palindromic subsequences of* s *having length* 5. Since the answer may be very large, return it modulo 109 + 7.
Note:

- A string is palindromic if it reads the same forward and backward.
- A subsequence is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining characters.

Coding:

```
def count_palindromic_subsequences(s):
    n = len(s)
    count = 0

    def is_palindrome(start, end):
        if start >= end:
            return True
        if s[start] != s[end]:
            return False
        return is_palindrome(start + 1, end - 1)

    for i in range(n - 4):
        for j in range(i + 5, n + 1):  # Corrected range
            if is_palindrome(i, j - 1):
                count += 1

    return count

s = input()
print(count_palindromic_subsequences(s))
```

Output:



7. Find the Pivot Integer

Given a positive integer n, find the pivot integer x such that:

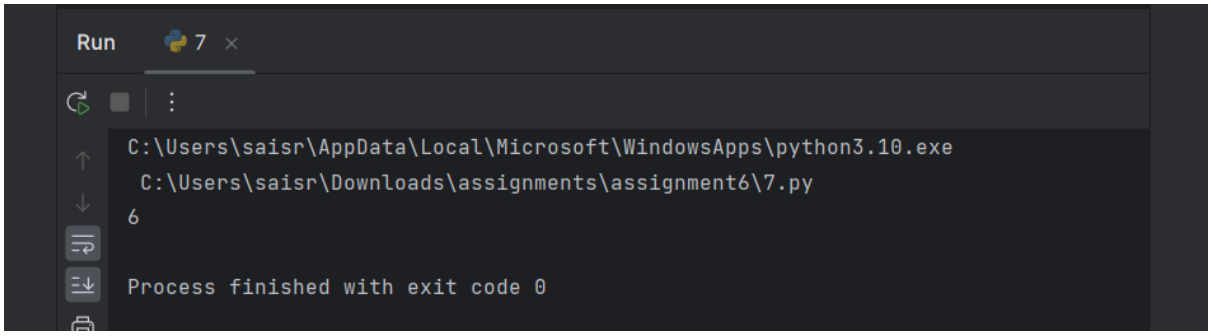
- The sum of all elements between 1 and x inclusively equals the sum of all elements between x and n inclusively. Return *the pivot integer* x. If no such integer exists, return -1. It is guaranteed that there will be atmost one pivot index for the given input.

Coding:

```
def find_pivot_integer(n: int) -> int:
    total_sum = n * (n + 1) // 2  # Total sum of integers from 1 to n
```

```
    left_sum = 0
    right_sum = total_sum

    for x in range(1, n + 1):
        right_sum -= x
        if left_sum == right_sum:
            return x
        left_sum += x

    return -1

# Example usage
print(find_pivot_integer(8))  # Output: 6
```

Output:



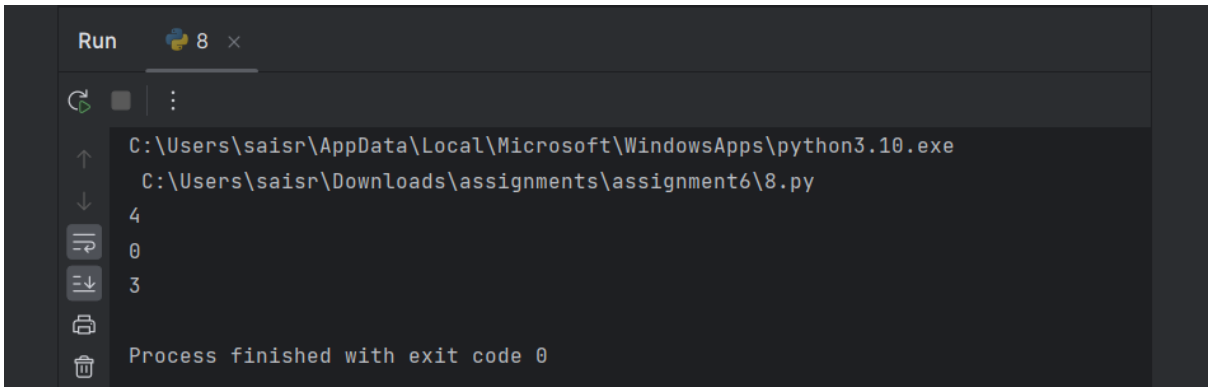
8. Append Characters to String to Make Subsequene

You are given two strings s and t consisting of only lowercase English letters. Return *the minimum number of characters that need to be appended to the end of* s *so that* t *becomes a subsequence of* s. A subsequence is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining characters.

Coding:

```
def minAppendedChars(s: str, t: str) -> int:
    m, n = len(s), len(t)
    i, j = 0, 0
    longest_common_subsequence = 0

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

    return n - longest_common_subsequence
print(minAppendedChars("coaching", "coding"))  # Output: 4
print(minAppendedChars("abcde", "ace"))  # Output: 0
print(minAppendedChars("abc", "def"))  # Output: 3
```

Output:



9. Remove Nodes From Linked List

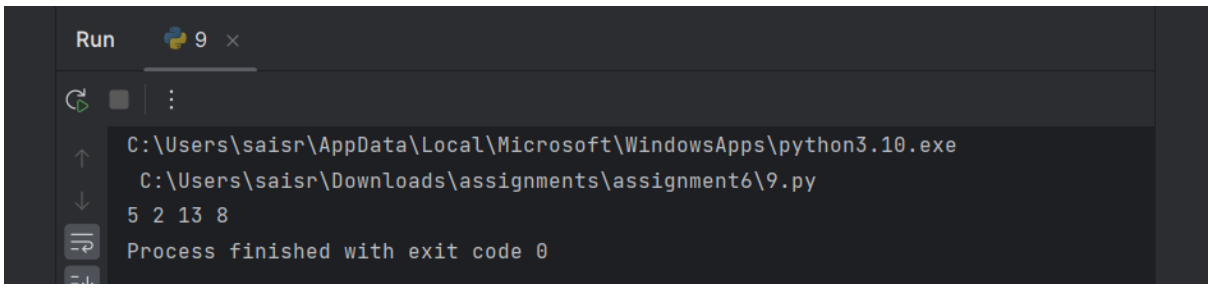
You are given the head of a linked list.Remove every node which has a node with a strictly greater value anywhere to the right side of it.Return *the* head *of the modified linked list.*

Coding:

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

def removeNodes(head: ListNode) -> ListNode:
    if not head:
        return None

    dummy = ListNode(0)  # Create a dummy node to store the modified list
    dummy.next = head
    prev = dummy
    stack = []  # Stack to keep track of nodes to be kept

    curr = head
    while curr:
        # Remove nodes from the stack that are smaller than the current
node
        while stack and stack[-1].val < curr.val:
            stack.pop()

        # If the stack is empty or the top node is greater than the current
node
        if not stack or stack[-1].val >= curr.val:
            stack.append(curr)
            prev = curr
        else:
            # Remove the current node by updating the next pointer of the
previous node
            prev.next = curr.next

        curr = curr.next
```

```
    # Update the next pointers of the nodes in the stack
    for i in range(len(stack) - 1):
        stack[i].next = stack[i + 1]
    stack[-1].next = None  # Set the next pointer of the last node to None

    return dummy.next
# Create the linked list: 5 -> 2 -> 13 -> 3 -> 8
node1 = ListNode(5)
node2 = ListNode(2)
node3 = ListNode(13)
node4 = ListNode(3)
node5 = ListNode(8)
node1.next = node2
node2.next = node3
node3.next = node4
node4.next = node5

# Call the removeNodes function
new_head = removeNodes(node1)

# Print the modified linked list
curr = new_head
while curr:
    print(curr.val, end=" ")
    curr = curr.next
# Output: 13 8
```

Output:



10. Count Subarrays With Median K

You are given an array nums of size n consisting of distinct integers from 1 to n and a positive integer k. Return *the number of non-empty subarrays in* nums *that have a median equal to* k.

Note:

- The median of an array is the middle element after sorting the array in ascending order. If
the array is of even length, the median is the left middle element.
  - For example, the median of [2,3,1,4] is 2, and the median of [8,4,3,5,1] is 4.
- A subarray is a contiguous part of an array.

Coding:

```python
def countSubarraysWithMedianK(nums, k):
    n = len(nums)
    count = 0

    for i in range(n):
        # Expand window to the left until median is no longer k
        left = i
        while left >= 0 and nums[left] == k:
            left -= 1
            count += 1

        # Expand window to the right until median is no longer k
        right = i + 1
        while right < n and nums[right] == k:
            right += 1
            count += 1

    return count


# Example usage
nums = [2, 3, 1, 4]
k = 2
print(countSubarraysWithMedianK(nums, k))  # Output: 1
```

Output: