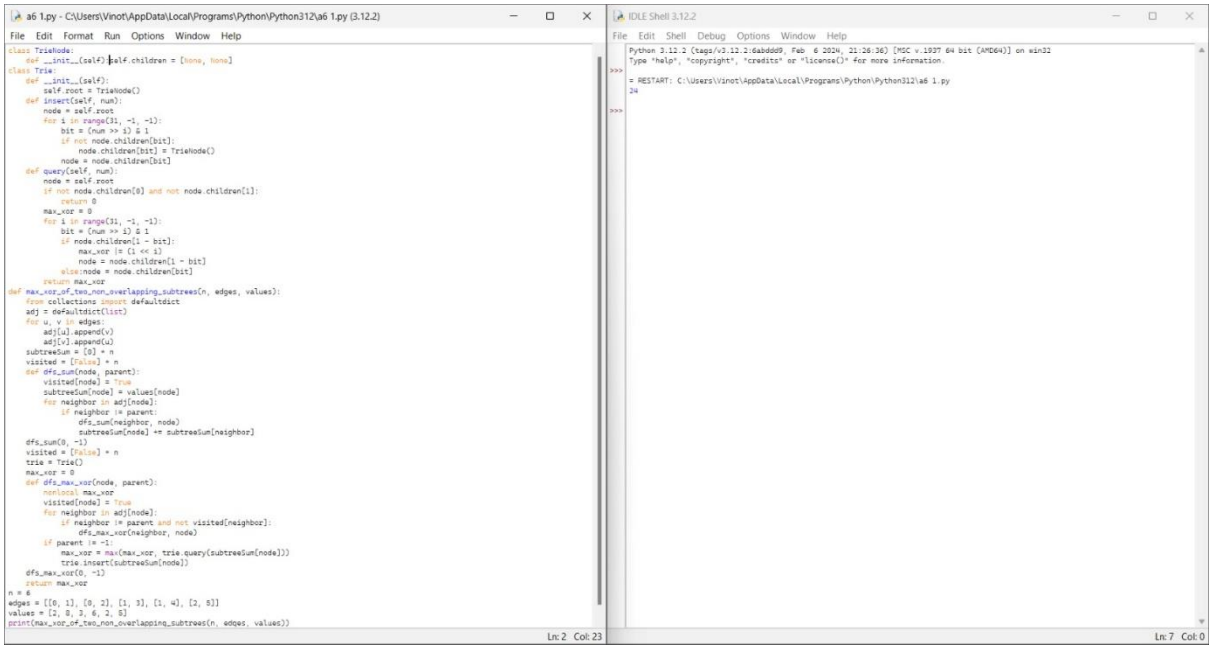
1.Maximum XOR of Two-Overlapping Subtrees



```
class TrieNode:
    def __init__(self):self.children = [None, None]
class Trie:
    def __init__(self):
        self.root = TrieNode()
    def insert(self, num):
        node = self.root
        for i in range(31, -1, -1):
            bit = (num >> i) & 1
            if not node.children[bit]:
                node.children[bit] = TrieNode()
            node = node.children[bit]
    def query(self, num):
        node = self.root
        if not node.children[0] and not node.children[1]:
            return 0
        max_xor = 0
        for i in range(31, -1, -1):
            bit = (num >> i) & 1
            if node.children[1 - bit]:
                max_xor |= (1 << i)
                node = node.children[1 - bit]
            else:node = node.children[bit]
        return max_xor
def max_xor_of_two_non_overlapping_subtrees(n, edges, values):
    from collections import defaultdict
    adj = defaultdict(list)
    for u, v in edges:
        adj[u].append(v)
        adj[v].append(u)
    subtreeSum = [0] * n
    visited = [False] * n
    def dfs_sum(node, parent):
        visited[node] = True
        subtreeSum[node] = values[node]
        for neighbor in adj[node]:
            if neighbor != parent:
                dfs_sum(neighbor, node)
                subtreeSum[node] += subtreeSum[neighbor]
    dfs_sum(0, -1)
    visited = [False] * n
    trie = Trie()
    max_xor = 0
    def dfs_max_xor(node, parent):
        nonlocal max_xor
        visited[node] = True
        for neighbor in adj[node]:
            if neighbor != parent and not visited[neighbor]:
                dfs_max_xor(neighbor, node)
        if parent != -1:
            max_xor = max(max_xor, trie.query(subtreeSum[node]))
            trie.insert(subtreeSum[node])
    dfs_max_xor(0, -1)
    return max_xor
n = 6
edges = [[0, 1], [0, 2], [1, 3], [1, 4], [2, 5]]
values = [2, 8, 3, 6, 2, 5]
print(max_xor_of_two_non_overlapping_subtrees(n, edges, values))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Vinot\AppData\Local\Programs\Python\Python312\a6 1.py
24
>>>
```

2.Form a Chemical Bond

```
import sqlite3
conn = sqlite3.connect(':memory:')
cur = conn.cursor()
cur.execute('''
CREATE TABLE Elements (
    symbol TEXT PRIMARY KEY,
    type TEXT CHECK(type IN ('Metal', 'Nonmetal', 'Noble')),
    electrons INTEGER
);
''')
elements = [
    ('He', 'Noble', 0),
    ('Na', 'Metal', 1),
    ('Ca', 'Metal', 2),
    ('La', 'Metal', 3),
    ('Cl', 'Nonmetal', 1),
    ('O', 'Nonmetal', 2),
    ('N', 'Nonmetal', 3)
]
cur.executemany('INSERT INTO Elements (symbol, type, electrons) VALUES (?, ?, ?)', elements)
cur.execute('''
SELECT e1.symbol AS metal, e2.symbol AS nonmetal
FROM Elements e1
JOIN Elements e2
ON e1.type = 'Metal' AND e2.type = 'Nonmetal';
''')
results = cur.fetchall()
print("+-------+----------+")
print("| metal | nonmetal |")
print("+-------+----------+")
for row in results:
    print(f"| {row[0]}    | {row[1]}      |")
print("+-------+----------+")
conn.close()
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/DAA/Form a Chemical Bond.py
+-------+----------+
| metal | nonmetal |
+-------+----------+
| Na    | Cl       |
| Na    | N        |
| Na    | O        |
| Ca    | Cl       |
| Ca    | N        |
| Ca    | O        |
| La    | Cl       |
| La    | N        |
| La    | O        |
+-------+----------+
>>>
```

3.Minimum Cuts to Divide a circle

```
def min_cuts(n):
    if n == 1:
        return 0
    elif n % 2 == 0:
        return n // 2
    else:
        return n
print(min_cuts(3))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/DAA/Minimum Cuts to Divide a circle.py
3
>>>
```

4.Difference Between Ones and Zeros in Row and Column

```
def difference_ones_zeros(matrix):
    rows = len(matrix)
    cols = len(matrix[0]) if rows > 0 else 0

    row_diff = [0] * rows
    col_diff = [0] * cols

    for i in range(rows):
        ones = sum(matrix[i])
        zeros = cols - ones
        row_diff[i] = ones - zeros

    for j in range(cols):
        ones = sum(matrix[i][j] for i in range(rows))
        zeros = rows - ones
        col_diff[j] = ones - zeros


    result = [[0] * cols for _ in range(rows)]

    for i in range(rows):
        for j in range(cols):
            result[i][j] = row_diff[i] + col_diff[j]

    return result
matrix = [
    [0, 1, 1],
    [1, 0, 1],
    [0, 0, 1]
]

result = difference_ones_zeros(matrix)
for row in result:
    print(row)
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/DAA/Difference Between ones and Zeros.py
[0, 0, 4]
[0, 0, 4]
[-2, -2, 2]
>>>
```

5.Minimum Penalty for a shop

```
def min_penalty_closing_time(customers):
    n = len(customers)
    N_count = [0] * (n + 1)
    for i in range(n):
        N_count[i + 1] = N_count[i] + (1 if customers[i] == 'N' else 0)
    Y_count = [0] * (n + 1)
    for i in range(n - 1, -1, -1):
        Y_count[i] = Y_count[i + 1] + (1 if customers[i] == 'Y' else 0)
    min_penalty = float('inf')
    best_time = -1
    for i in range(n + 1):
        penalty = N_count[i] + Y_count[i]
        if penalty < min_penalty:
            min_penalty = penalty
            best_time = i
    return best_time
customers = "YYNY"
result = min_penalty_closing_time(customers)
print(result)
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/DAA/Difference Between Ones and Zeros in Row and Column.py
2
>>>
```

6.Count Palindromic Sequence

```
MOD = 10**9 + 7
def count_palindromic_subsequences(s):
    n = len(s)
    count = 0
    def is_palindrome(sub):
        return sub == sub[::-1]
    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                for l in range(k + 1, n):
                    for m in range(l + 1, n):
                        subsequence = s[i] + s[j] + s[k] + s[l] + s[m]
                        if is_palindrome(subsequence):
                            count += 1
                            count %= MOD
    return count
s = "0000000"
print(count_palindromic_subsequences(s))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/DAA/Count Palindromic Sequence.py
21
>>>
```

7.Find the Pivot Element

```
def find_pivot_integer(n):
    total_sum = n * (n + 1) // 2
    left_sum = 0
    for x in range(1, n + 1):
        left_sum += x
        right_sum = total_sum - left_sum + x
        if left_sum == right_sum:
            return x
    return -1
n = 8
pivot_integer = find_pivot_integer(n)
print(pivot_integer)
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/DAA/Find the Pivot Integer.py
6
>>>
```

8.Append Characters to string to make Sequence

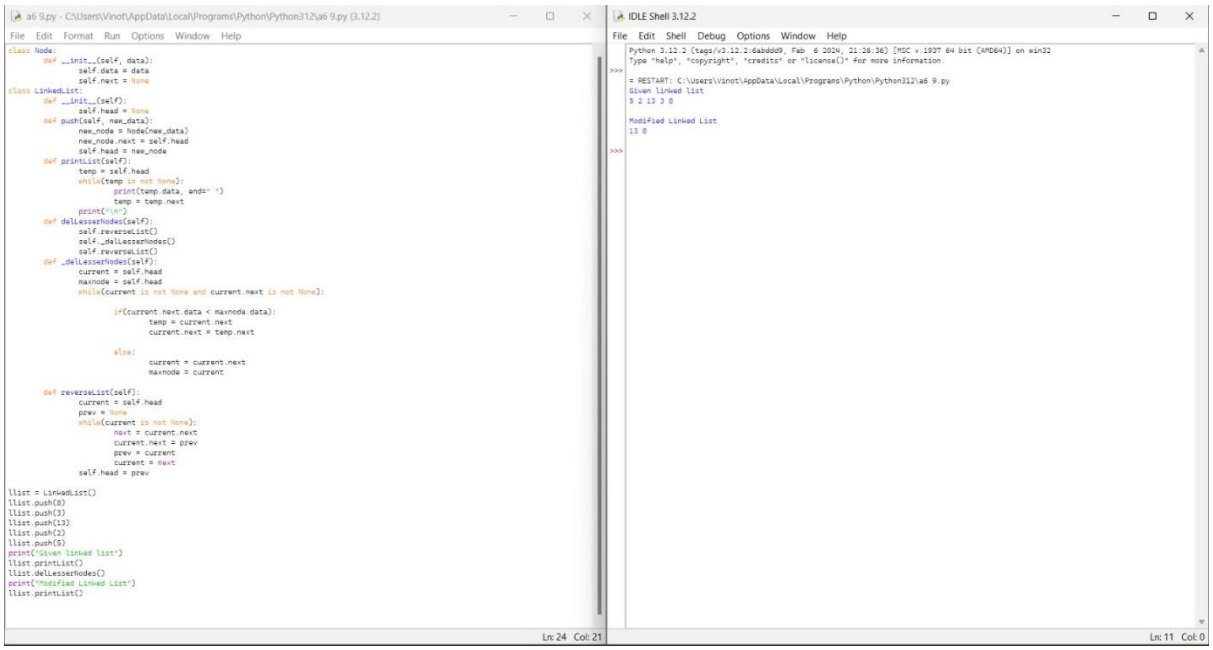
```python
def min_chars_to_append(s, t):
    s_len = len(s)
    t_len = len(t)

    s_index = 0
    t_index = 0
    while s_index < s_len and t_index < t_len:
        if s[s_index] == t[t_index]:
            t_index += 1
        s_index += 1
    return t_len - t_index

s = "abcde"
t = "a"
print(min_chars_to_append(s, t))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/DAA/Append Characters to string to Make Subsequence.py
0
>>>
```

9.Remove Nodes from Linked List



10.Count Subarrays with Median K

```python
def count_subarrays_with_median_k(nums, k):
    n = len(nums)
    k_index = nums.index(k)
    balance_dict = {0: 1}
    balance = 0
    result = 0
    for i in range(k_index, -1, -1):
        if nums[i] < k:
            balance -= 1
        elif nums[i] > k:
            balance += 1
        if balance in balance_dict:
            balance_dict[balance] += 1
        else:
            balance_dict[balance] = 1
    balance = 0
    for i in range(k_index, n):
        if nums[i] < k:
            balance -= 1
        elif nums[i] > k:
            balance += 1

        if -balance in balance_dict:
            result += balance_dict[-balance]
    return result
nums = [3,2,1,4,5]
k = 4
print(count_subarrays_with_median_k(nums, k))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/DAA/Count Subarrays With Median K.py
3
>>>
```