# Assignment – 4

## 1. Odd String Difference.

Program:

```python
def odd_string_difference(s: str) -> str:
    differences = []

    for i in range(1, len(s), 2):
        diff = ord(s[i]) - ord(s[i - 1])
        differences.append(diff)

    result = ''.join(map(str, differences))
    return result


input_string = "abcdef"
output = odd_string_difference(input_string)
print(f"Odd String Difference of '{input_string}' is: {output}")
```

**Output:**

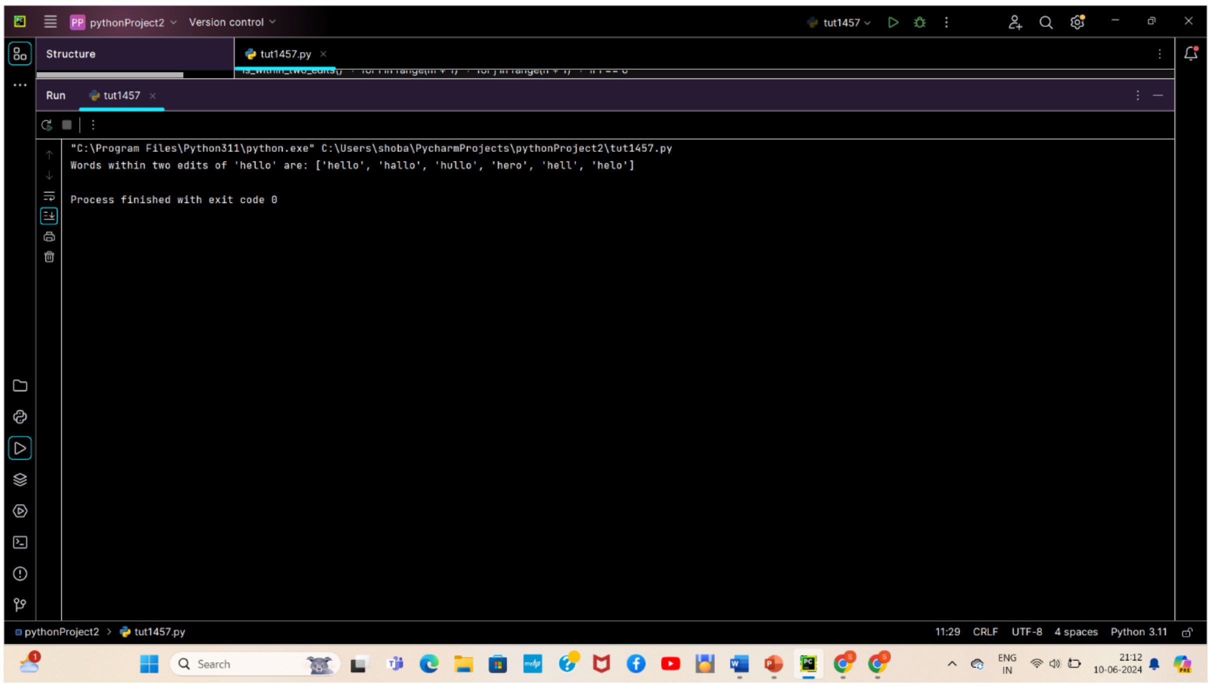
## 2. Words within Two Edits of Dictionary.

**Program:**

```python
def is_within_two_edits(word1: str, word2: str) -> bool:
    m, n = len(word1), len(word2)
    if abs(m - n) > 2:
        return False

    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(m + 1):
        for j in range(n + 1):
            if i == 0:
                dp[i][j] = j
            elif j == 0:
                dp[i][j] = i
            elif word1[i - 1] == word2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            else:
                dp[i][j] = 1 + min(dp[i - 1][j],  # Deletion
                                   dp[i][j - 1],  # Insertion
                                   dp[i - 1][j - 1])  # Substitution

    return dp[m][n] <= 2
```

```python
def words_within_two_edits(dictionary, word):
    result = []
    for dict_word in dictionary:
        if is_within_two_edits(word, dict_word):
            result.append(dict_word)
    return result


# Example usage:
dictionary = ["hello", "hallo", "hullo", "hero", "hell", "helo"]
word = "hello"
output = words_within_two_edits(dictionary, word)
print(f"Words within two edits of '{word}' are: {output}")
```

## Output:



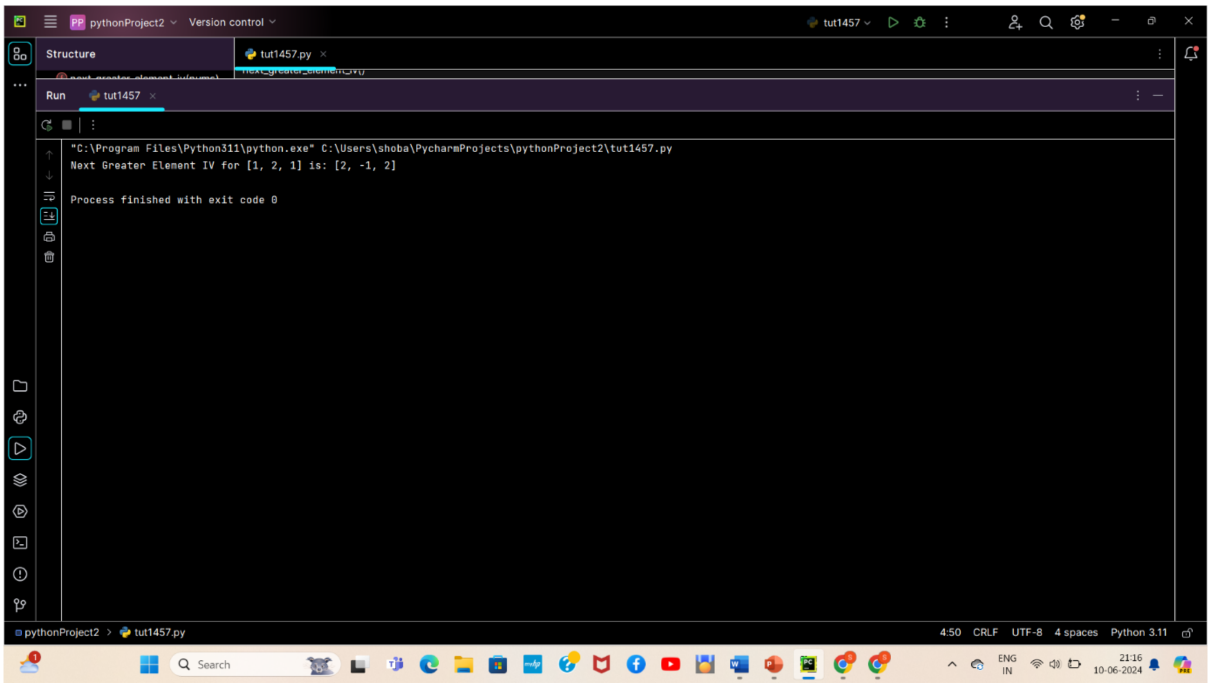
```
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Words within two edits of 'hello' are: ['hello', 'hallo', 'hullo', 'hero', 'hell', 'helo']

Process finished with exit code 0
```

## 3. Next Greater Element IV

## Program:

```python
def next_greater_element_iv(nums):
    n = len(nums)
    result = [-1] * n
    stack = []

    for i in range(2 * n):
        while stack and nums[stack[-1]] < nums[i % n]:
            result[stack.pop()] = nums[i % n]
        if i < n:
            stack.append(i)

    return result


nums = [1, 2, 1]
output = next_greater_element_iv(nums)
print(f"Next Greater Element IV for {nums} is: {output}")
```

## Output:



```
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Next Greater Element IV for [1, 2, 1] is: [2, -1, 2]

Process finished with exit code 0
```

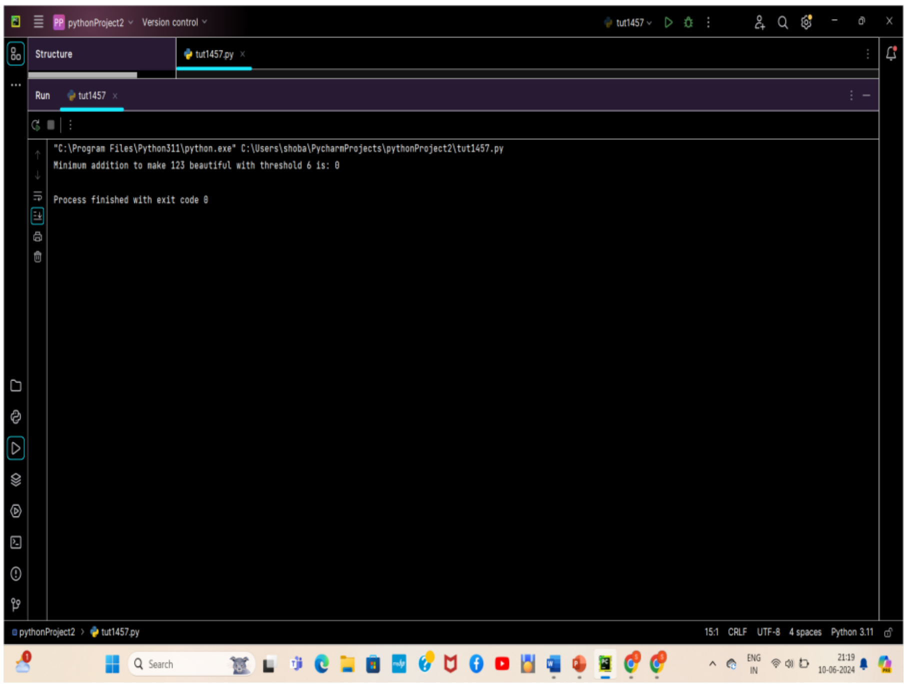
## 4. Minimum Addition to Make Integer Beautiful.

## Program:

```python
def sum_of_digits(x):
    return sum(int(digit) for digit in str(x))
```

```python
def minimum_addition_to_make_beautiful(n, k):
    addition = 0
    while sum_of_digits(n + addition) > k:
        addition += 1
    return addition

# Example usage:
n = 123
k = 6
output = minimum_addition_to_make_beautiful(n, k)
print(f"Minimum addition to make {n} beautiful with threshold {k} is: 
{output}")
```

## Output:



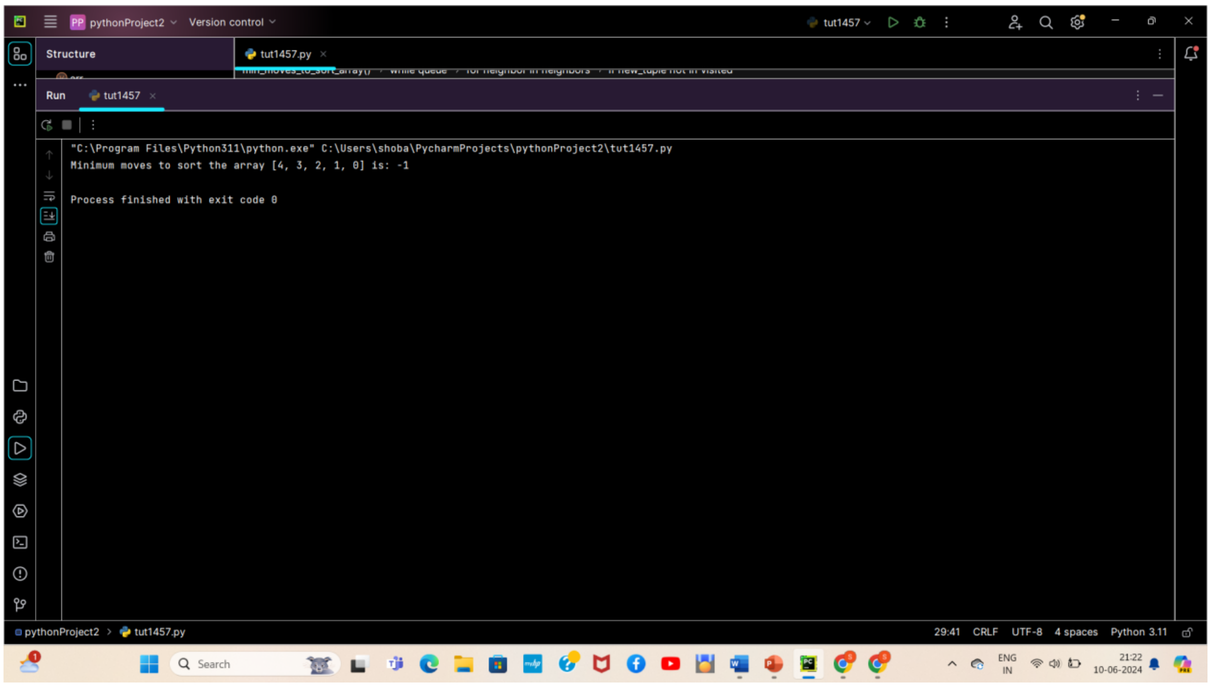
```
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Minimum addition to make 123 beautiful with threshold 6 is: 0

Process finished with exit code 0
```

## 5. Sort Array by Moving Items to Empty Space.

## Program:

```
from collections import deque


def min_moves_to_sort_array(arr):
    n = len(arr)
    target = sorted(arr)
    start = tuple(arr)

    queue = deque([(start, arr.index(0), 0)])
    visited = set()
    visited.add(start)

    while queue:
        current, empty_index, moves = queue.popleft()

        if list(current) == target:
            return moves

        neighbors = []
        if empty_index > 0:
            neighbors.append(empty_index - 1)
        if empty_index < n - 1:
            neighbors.append(empty_index + 1)

        for neighbor in neighbors:
            new_arr = list(current)
            new_arr[empty_index], new_arr[neighbor] = new_arr[neighbor],
new_arr[empty_index]
            new_tuple = tuple(new_arr)
            if new_tuple not in visited:
                visited.add(new_tuple)
                queue.append((new_tuple, neighbor, moves + 1))

    return -1

arr = [4, 3, 2, 1, 0]
output = min_moves_to_sort_array(arr)
print(f"Minimum moves to sort the array {arr} is: {output}")
```

## Output:



```
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Minimum moves to sort the array [4, 3, 2, 1, 0] is: -1

Process finished with exit code 0
```