

ASSIGNMENT 5

1. Merge Two Sorted Lists

You are given the heads of two sorted linked lists `list1` and `list2`. Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists. Return *the head of the merged linked list*.

Coding:

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

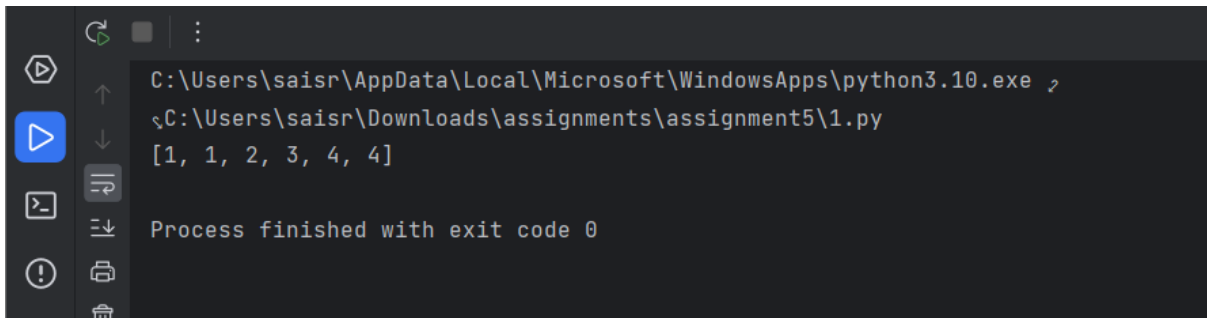
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
        return self.head

    def s(self):
        data = []
        current = self.head
        while current:
            data.append(current.data)
            current = current.next
        r = sorted(data)
        return r

l = LinkedList()
#l1
l.insert(1)
l.insert(2)
l.insert(4)
#l2
l.insert(1)
l.insert(3)
l.insert(4)

print(l.s())
```

Output:



```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe C:\Users\saisr\Downloads\assignments\assignment5\1.py
[1, 1, 2, 3, 4, 4]
Process finished with exit code 0
```

2. Merge k Sorted Lists

You are given an array of k linked-lists `lists`, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.

Coding:

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

class Linked list:
    def __init__(self):
        self.head = None

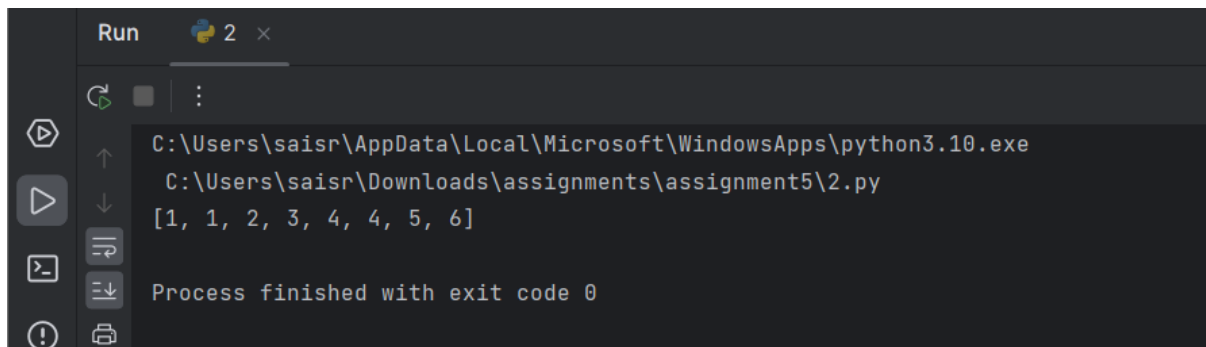
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
        return self.head

    def s(self):
        data = []
        current = self.head
        while current:
            data.append(current.data)
            current = current.next
        r = sorted(data)
        return r

ar = [[1,4,5],[1,3,4],[2,6]]
l = Linked list()
for i in range(len(ar)):
    l.insert(ar[i])

print(l.s())
```

Output:



```
Run 2 x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\2.py
[1, 1, 2, 3, 4, 4, 5, 6]
Process finished with exit code 0
```

3. Remove Duplicates from Sorted Array

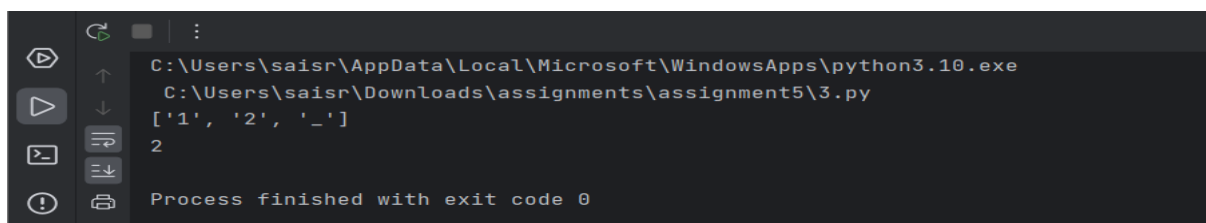
Given an integer array **nums** sorted in non-decreasing order, remove the duplicates **inplace** such that each unique element appears only once. The relative order of the elements should be kept the same. Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array **nums**. More formally, if there are **k** elements after removing the duplicates, then the first **k** elements of **nums** should hold the final result. It does not matter what you leave beyond the first **k** elements. Return **k** after placing the final result in the first *k* slots of **nums**.

Coding:

```
nums = [1,1,2]
ar = []
underscore = "_"
for i in range(len(nums)):
    if i+1 <= len(nums)-1:
        if nums[i] == nums[i+1]:
            if nums[i] is not ar:
                ar.append(str(nums[i]))
            ar.append(underscore)
        else:
            ar.append(str(nums[i]))

print(sorted(ar))
n=[]
for i in range(len(ar)):
    if ar[i] != "_":
        n.append(ar[i])
print(len(n))
```

Output:



```
Run 2 x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\3.py
['1', '2', '_']
2
Process finished with exit code 0
```

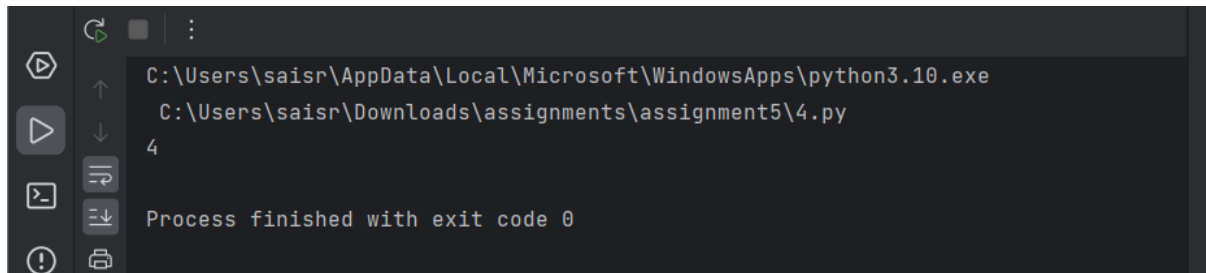
4. Search in Rotated Sorted Array

There is an integer array **nums** sorted in ascending order (with distinct values). Prior to being passed to your function, **nums** is possibly rotated at an unknown pivot index **k** ($1 \leq k < \text{nums.length}$) such that the resulting array is **[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]** (0-indexed). For example, **[0,1,2,4,5,6,7]** might be rotated at pivot index **3** and become **[4,5,6,7,0,1,2]**.

Coding:

```
nums = [4, 5, 6, 7, 0, 1, 2]
target = 0
c=0
ans=0
for i in range(len(nums)):
    if nums[i]==target:
        c=1
        ans=i
        break
if c==0:
    print("-1")
else:
    print(ans)
```

Output:



```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\4.py
4
Process finished with exit code 0
```

5. Find First and Last Position of Element in Sorted Array

Given an array of integers **nums** sorted in non-decreasing order, find the starting and ending position of a given target value. If target is not found in the array, return **[-1, -1]**.

Coding:

```
nums = [5, 7, 7, 8, 8, 10]
target = 8
ar=[]
for i in range(len(nums)):
    if nums[i]==target:
        ar.append(i)
print(ar)
```

Output:

```
Run 5 x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\5.py
[3, 4]
Process finished with exit code 0
```

6. Sort Colors

Given an array **nums** with **n** objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers **0**, **1**, and **2** to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function.

Coding:

```
nums = [2, 0, 2, 1, 1, 0]
n = len(nums)

for i in range(n):
    for j in range(0, n - i - 1):
        if nums[j] > nums[j + 1]:
            nums[j], nums[j + 1] = nums[j + 1], nums[j]
print(nums)
```

Output:

```
Run 6 x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\6.py
[0, 0, 1, 1, 2, 2]
Process finished with exit code 0
```

7. Remove Duplicates from Sorted List

Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

Coding:

```
class Node:
    def __init__(self, data=None):
        self.data = data
        self.next = None

class Linked_list:
    def __init__(self):
        self.head = None

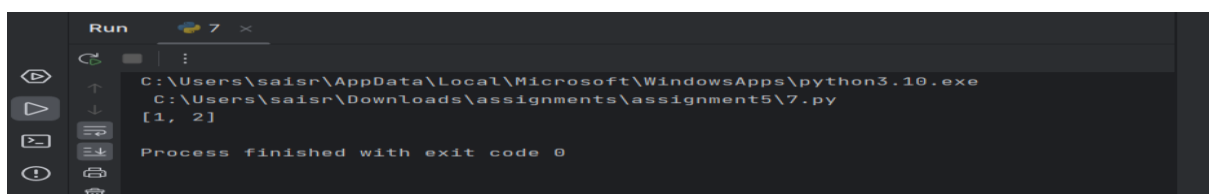
    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
        return self.head

    def dup(self):
        data = []
        current = self.head
        while current:
            data.append(current.data)
            current = current.next

        i = 0
        while i < len(data) - 1:
            if data[i] == data[i + 1]:
                data.pop(i)
            else:
                i += 1
        return data

ar = [1, 1, 2]
l = Linked_list()
for i in range(len(ar)):
    l.insert(ar[i])
print(l.dup())
```

Output:



```
Run 7 x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\7.py
[1, 2]
Process finished with exit code 0
```

8. Merge Sorted Array

You are given two integer arrays **nums1** and **nums2**, sorted in non-decreasing order, and two integers **m** and **n**, representing the number of elements in **nums1** and **nums2** respectively.

Coding:

```
def merge(nums1, m, nums2, n):
    p1, p2, p = m - 1, n - 1, m + n - 1
    while p2 >= 0:
        if p1 >= 0 and nums1[p1] > nums2[p2]:
            nums1[p] = nums1[p1]
            p1 -= 1
        else:
            nums1[p] = nums2[p2]
            p2 -= 1
        p -= 1

    return nums1

nums1 = [1, 2, 3, 0, 0, 0]
m = 3
nums2 = [2, 5, 6]
n = 3

result = merge(nums1, m, nums2, n)
print(result)
```

Output:

```
Run 8 x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\8.py
[1, 2, 2, 3, 5, 6]
Process finished with exit code 0
```

9. Convert Sorted Array to Binary Search Tree

Given an integer array **nums** where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.

Coding:

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
```

```

def sorte(nums):
    if not nums:
        return None

    mid = len(nums) // 2
    root = Node(nums[mid])
    root.left = sorte(nums[:mid])
    root.right = sorte(nums[mid + 1:])
    return root

def in_order(root):
    elements = []
    _in_order(root, elements)
    return elements

def _in_order(root, elements):
    if root:
        _in_order(root.left, elements)
        elements.append(root.key)
        _in_order(root.right, elements)

nums = [1, 2, 3, 4, 5, 6, 7]
bst_root = sorte(nums)
print("In-order Traversal of the BST:", in_order(bst_root))

```

Output:

The screenshot shows a Python IDE window titled 'Run' with a Python 3.10 icon. The output console displays the following text:

```

C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\9.py
In-order Traversal of the BST: [1, 2, 3, 4, 5, 6, 7]
Process finished with exit code 0

```

10. Insertion Sort List

Given the head of a singly linked list, sort the list using insertion sort, and return *the sorted list's head*.

Coding:

```

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

def insertionSortList(head):
    dummy = ListNode(0)
    curr = head
    while curr:
        prev = dummy
        while prev.next and prev.next.val < curr.val:
            prev = prev.next
        prev.next = curr
        curr = curr.next

```



```

        next_temp = curr.next
        curr.next = prev.next
        prev.next = curr
        curr = next_temp

    return dummy.next

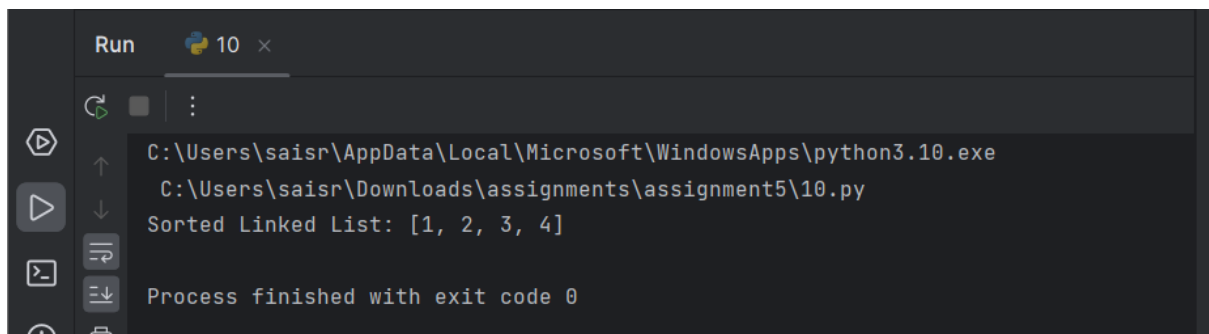
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

def linked_list_to_list(head):
    result = []
    current = head
    while current:
        result.append(current.val)
        current = current.next
    return result

values = [4, 2, 1, 3]
head = create_linked_list(values)
sorted_head = insertionSortList(head)
print("Sorted Linked List:", linked_list_to_list(sorted_head))

```

Output:



The screenshot shows a Python IDE's Run console. The console output is as follows:

```

C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\10.py
Sorted Linked List: [1, 2, 3, 4]
Process finished with exit code 0

```

11. Sort Characters By Frequency

Given a string *s*, sort it in decreasing order based on the frequency of the characters. The frequency of a character is the number of times it appears in the string. Return *the sorted string*. If there are multiple answers, return *any of them*.

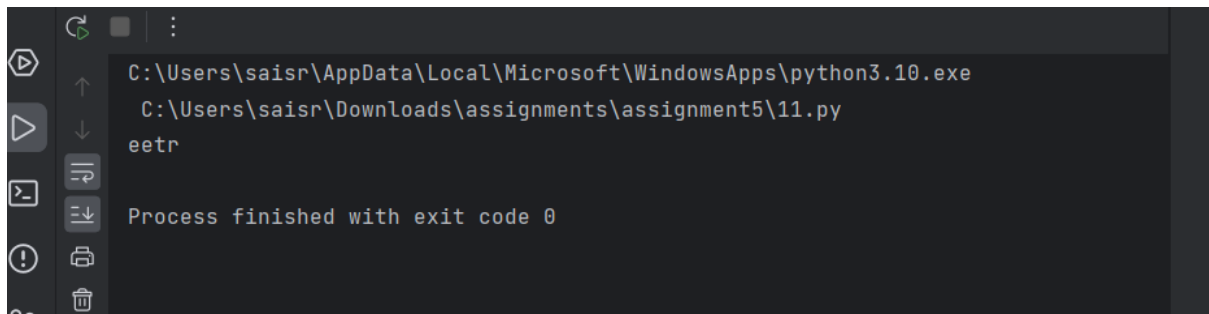
Coding:

```
from collections import Counter

def fsort(s):
    freq = Counter(s)
    chars = sorted(freq.items(), key=lambda x: x[1], reverse=True)
    result = ''.join([char * count for char, count in chars])
    return result

s = "tree"
print(fsort(s))
```

Output:



```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\11.py
eetr
Process finished with exit code 0
```

12. Example 1:

Input: head = [4,2,1,3]

Output: [1,2,3,4]

Coding:

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

def insertionSortList(head):
    dummy = ListNode()
    dummy.next = head
    prev = dummy
    curr = head

    while curr:
        if curr.next and curr.next.val < curr.val:
            while prev.next and prev.next.val < curr.next.val:
                prev = prev.next
            temp = prev.next
            prev.next = curr.next
```

```

        curr.next = curr.next.next
        prev.next.next = temp
        prev = dummy
    else:
        curr = curr.next

    return dummy.next

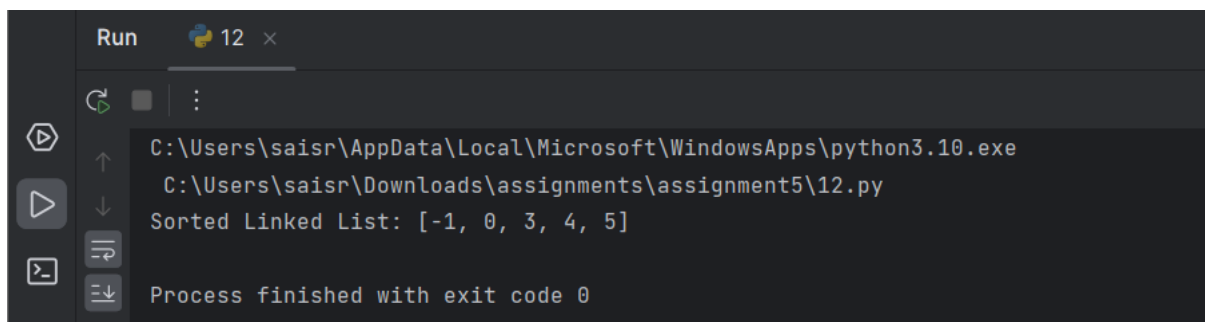
def create_linked_list(values):
    if not values:
        return None
    head = ListNode(values[0])
    current = head
    for value in values[1:]:
        current.next = ListNode(value)
        current = current.next
    return head

def linked_list_to_list(head):
    result = []
    current = head
    while current:
        result.append(current.val)
        current = current.next
    return result

values = [-1, 5, 3, 4, 0]
head = create_linked_list(values)
sorted_head = insertionSortList(head)
print("Sorted Linked List:", linked_list_to_list(sorted_head))

```

Output:



```

Run 12 x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\12.py
Sorted Linked List: [-1, 0, 3, 4, 5]
Process finished with exit code 0

```

13. Max Chunks To Make Sorted

You are given an integer array `arr` of length `n` that represents a permutation of the integers in the range `[0, n - 1]`. We split `arr` into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array. Return *the largest number of chunks we can make to sort the array*

Coding:

```
def max_chunks_to_sorted(arr):
    max_val = 0
    chunks = 0

    for i, num in enumerate(arr):
        max_val = max(max_val, num)

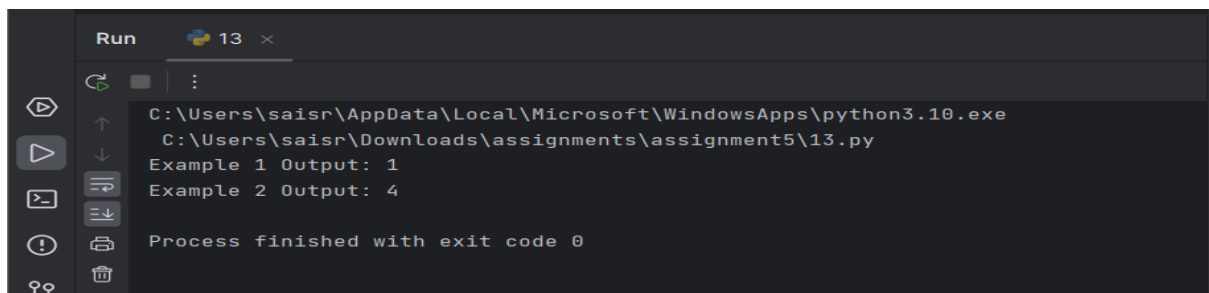
        if i == max_val:
            chunks += 1

    return chunks

# Example usage
arr1 = [4, 3, 2, 1, 0]
arr2 = [1, 0, 2, 3, 4]

print("Example 1 Output:", max_chunks_to_sorted(arr1)) # Output: 1
print("Example 2 Output:", max_chunks_to_sorted(arr2)) # Output: 4
```

Output:



```
Run 13 x
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\13.py
Example 1 Output: 1
Example 2 Output: 4
Process finished with exit code 0
```

14 Intersection of Three Sorted Arrays

Given three integer arrays arr1, arr2 and arr3 sorted in strictly increasing order, return a sorted array of only the integers that appeared in all three arrays.

Coding:

```
def intersection_of_three_arrays(arr1, arr2, arr3):
    result = []
    p1, p2, p3 = 0, 0, 0

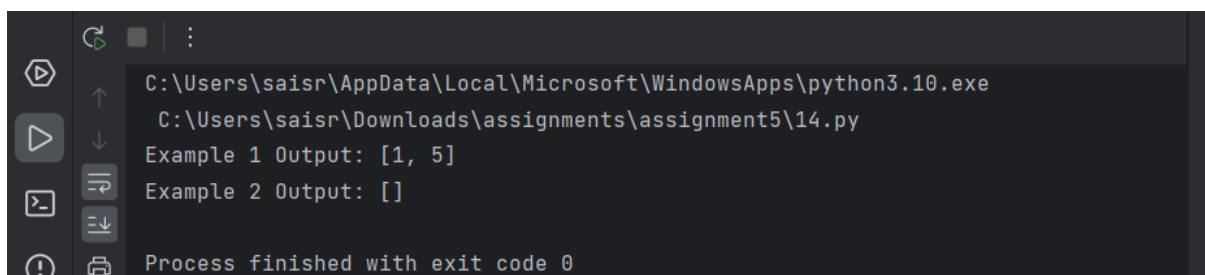
    while p1 < len(arr1) and p2 < len(arr2) and p3 < len(arr3):
        if arr1[p1] == arr2[p2] == arr3[p3]:
            result.append(arr1[p1])
            p1 += 1
            p2 += 1
            p3 += 1
        elif arr1[p1] < arr2[p2]:
            p1 += 1
        elif arr2[p2] < arr3[p3]:
            p2 += 1
        else:
            p3 += 1

    return result
```

```
# Example usage
arr1 = [1, 2, 3, 4, 5]
arr2 = [1, 2, 5, 7, 9]
arr3 = [1, 3, 4, 5, 8]
print("Example 1 Output:", intersection_of_three_arrays(arr1, arr2, arr3))
# Output: [1, 5]

arr1 = [197, 418, 523, 876, 1356]
arr2 = [501, 880, 1593, 1710, 1870]
arr3 = [521, 682, 1337, 1395, 1764]
print("Example 2 Output:", intersection_of_three_arrays(arr1, arr2, arr3))
# Output: []
```

Output:



```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe
C:\Users\saisr\Downloads\assignments\assignment5\14.py
Example 1 Output: [1, 5]
Example 2 Output: []
Process finished with exit code 0
```

15. Sort the Matrix Diagonally

A matrix diagonal is a diagonal line of cells starting from some cell in either the topmost row or leftmost column and going in the bottom-right direction until reaching the matrix's end. For example, the matrix diagonal starting from `mat[2][0]`, where `mat` is a `6 x 3` matrix, includes cells `mat[2][0]`, `mat[3][1]`, and `mat[4][2]`.

Coding:

```
def diagonalSort(mat):
    m, n = len(mat), len(mat[0])
    diagonals = {}

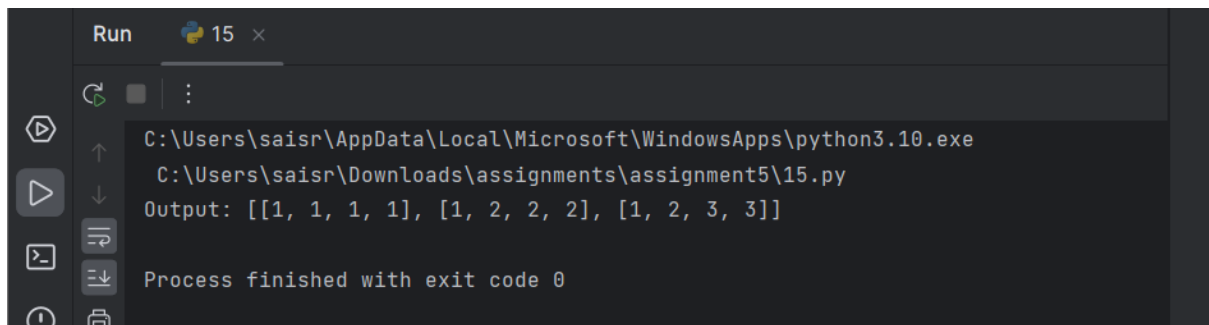
    for i in range(m):
        for j in range(n):
            if (i - j) not in diagonals:
                diagonals[i - j] = []
            diagonals[i - j].append(mat[i][j])

    for key in diagonals:
        diagonals[key].sort()

    for i in range(m):
        for j in range(n):
            mat[i][j] = diagonals[i - j].pop(0)
    return mat

mat = [[3, 3, 1, 1], [2, 2, 1, 2], [1, 1, 1, 2]]
print("Output:", diagonalSort(mat))
```

Output:



The screenshot shows a dark-themed IDE window with a tab labeled 'Run' and a Python icon followed by '15'. The console output is as follows:

```
C:\Users\saisr\AppData\Local\Microsoft\WindowsApps\python3.10.exe  
C:\Users\saisr\Downloads\assignments\assignment5\15.py  
Output: [[1, 1, 1, 1], [1, 2, 2, 2], [1, 2, 3, 3]]  
  
Process finished with exit code 0
```

On the left side of the console, there is a vertical toolbar with icons for running (a play button), stepping through code (up and down arrows), and other debugging actions.