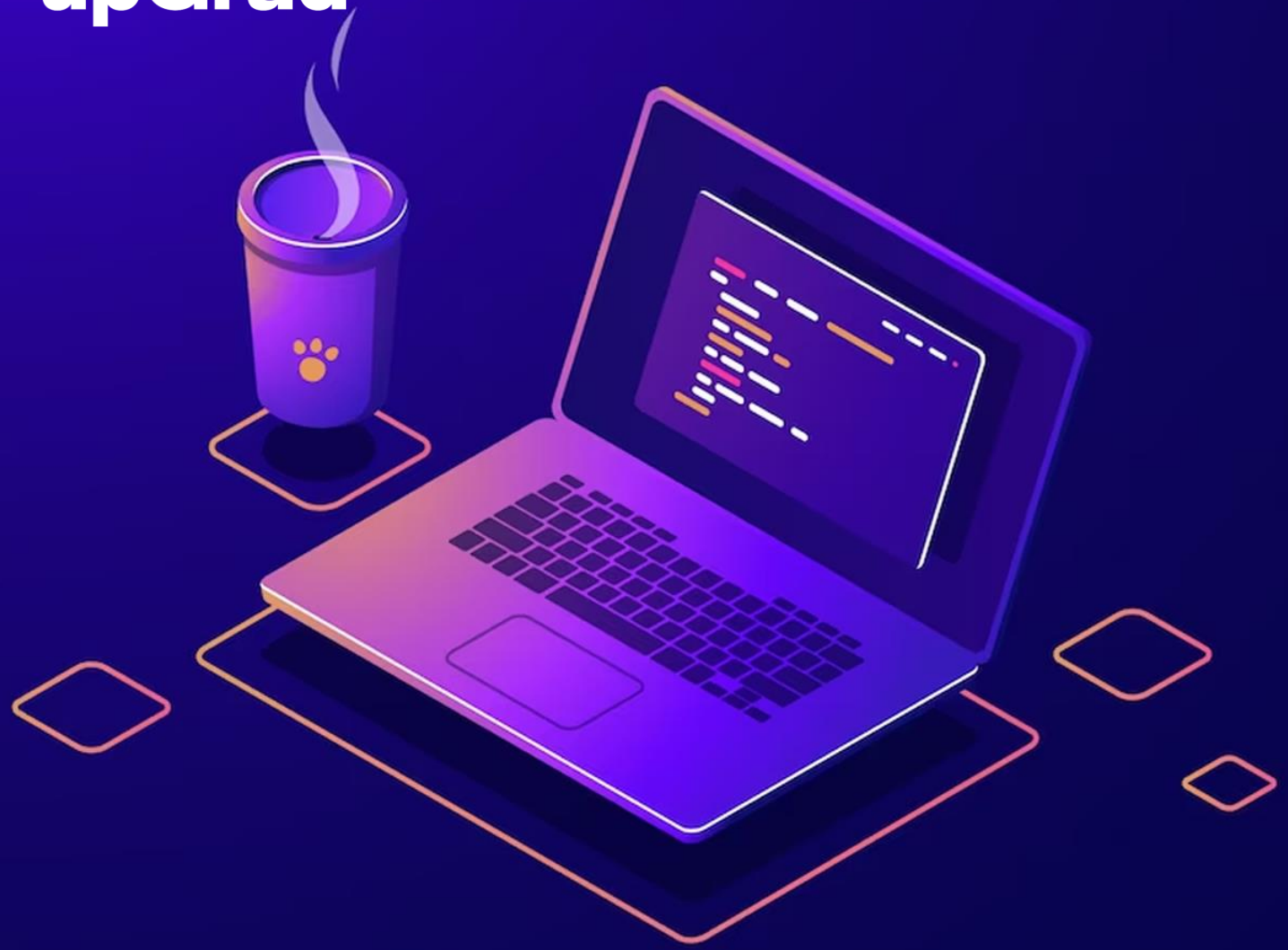
upGrad

Pioneering Excellence in Education, Research & Innovation



Python Lab Assignment

By:

Vaibhav Kolhale, Muna Sahu & Vandana NS

The video Link in Last Page

upGrad

❖ Problem statement

Given a list of integers, find the two elements whose sum is closest to zero. The list can contain both positive and negative integers.

Output Requirement: The function should return the two elements whose sum is closest to zero, and if there are multiple pairs with the same distance to zero, it should return any one of them. If the list is empty, the function should return None.

For example, given the list [-4, 7, 6, 2, -5], the function should return the pair (-5, 6), as the sum of these two elements is closest to zero.

Input Format: A list of integers
Output Format: The output contains the two elements with the sum closest to zero.

upGrad

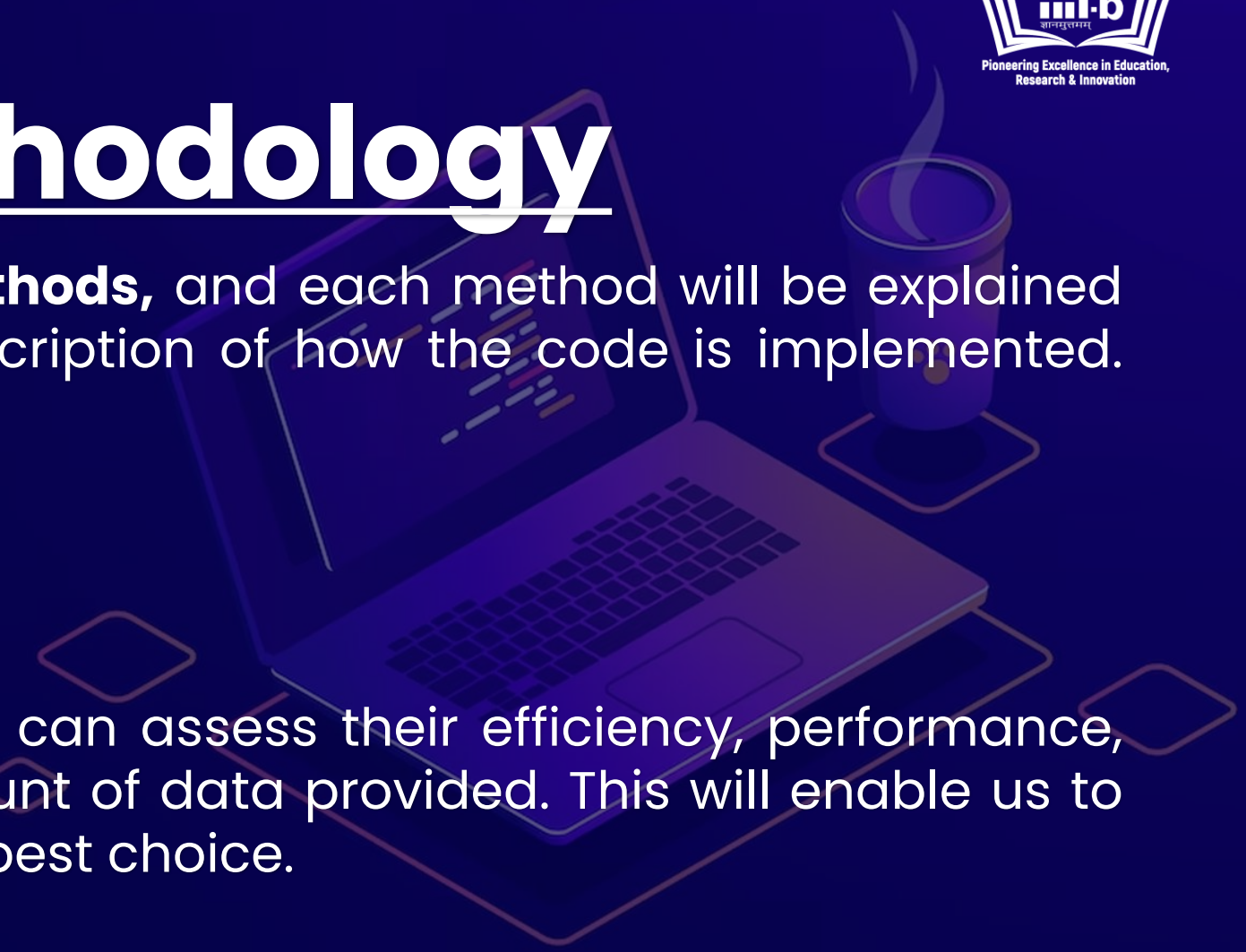
Pioneering Excellence in Education, Research & Innovation

❖ Methodology

This assignment involves **two methods,** and each method will be explained in detail with a step-by-step description of how the code is implemented. The two methods are:

1) **Brute Force Approach**
2) **Optimized Approach**

By utilizing these approaches, we can assess their efficiency, performance, and suitability based on the amount of data provided. This will enable us to determine which approach is the best choice.



upGrad

1. Brute Force approach

- ✓ To find the pair of elements with the smallest absolute sum in a list, the brute force approach involves iterating through all possible pairs of elements and computing their sums.
- ✓ This method has a time complexity of O(n^2) and is inefficient for large lists. Therefore, for larger lists, alternative approaches such as sorting the list or using hash tables can be employed to achieve a lower time complexity.
- ✓ This brute-force method is applied in **Jupyter Notebook** a python platform for programming
- ✓ The **find_pair_closest_to_zero** function takes an array arr as input and returns a tuple containing the pair of elements whose sum is closest to zero.
- ✓ The function uses a brute force approach where it iterates through all possible pairs of elements in the array and calculates their sum. The minimum difference between the sum of any two pairs is continuously tracked, and the pair of elements associated with this minimum difference is returned.

upGrad

1. Brute Force approach

- ✓ To find the pair of elements with the smallest absolute sum in a list, the brute force approach involves iterating through all possible pairs of elements and computing their sums.
- ✓ This method has a time complexity of O(n^2) and is inefficient for large lists. Therefore, for larger lists, alternative approaches such as sorting the list or using hash tables can be employed to achieve a lower time complexity.
- ✓ This brute-force method is applied in **Jupyter Notebook** a python platform for programming.
- ✓ The **find_pair_closest_to_zero** function takes an array arr as input and returns a tuple containing the pair of elements whose sum is closest to zero.
- ✓ The function uses a brute force approach where it iterates through all possible pairs of elements in the array and calculates their sum. The minimum difference between the sum of any two pairs is continuously tracked, and the pair of elements associated with this minimum difference is returned.

upGrad

1. Brute Force approach

```
def find_pair_closest_to_zero(arr):
    min_diff = float('inf')
    pair = None
    for i in range(len(arr)):
        for j in range(i+1, len(arr)):
            curr_sum = arr[i] + arr[j]
            if abs(curr_sum) < min_diff:
                min_diff = abs(curr_sum)
                pair = (arr[j], arr[i])
    return pair

arr1 = [-4, 7, 6, 2, -5]
print("Brute force approach output values 1:",find_pair_closest_to_zero(arr1))

arr2 = [-50, 34, -19, 24, 33, 10, -46, -38]
print("Brute force approach output values 2:",find_pair_closest_to_zero(arr2))
```

```
Brute force approach output values 1: (-5, 6)
Brute force approach output values 2: (-38, 34)
```

upGrad

Let's break down the code:

```
def find_pair_closest_to_zero(arr):
```

This line defines the function and takes an array as input.

```
min_diff = float('inf')
pair = None
```

These lines initialize two variables - min_diff to infinity and pair to None. min_diff will keep track of the minimum difference between any two pairs of elements in the array. **pair** will store the pair of elements whose sum is closest to zero.

upGrad

Let's break down the code:

```
for i in range(len(arr)):
    for j in range(i+1, len(arr)):
        curr_sum = arr[i] + arr[j]
        if abs(curr_sum) < min_diff:
            min_diff = abs(curr_sum)
            pair = (arr[j], arr[i])
```

These lines implement the brute force approach. The outer loop iterates through each element in the array, and the inner loop iterates through all elements after the current element. This ensures that each pair of elements is considered only once.

The sum of the current pair is calculated and its absolute value is taken to ensure that the distance from zero is always positive. If this distance is smaller than the current minimum difference, the minimum difference is updated, and the current pair is stored in pair.

upGrad

Let's break down the code:

```
return pair
```

This line returns the pair of elements whose sum is closest to zero.

upGrad

2. Optimized approach

- ✓ The given Python program finds the pair of elements in a given array whose sum is closest to zero using a more efficient approach than the previous program.
- ✓ This approach involves sorting the array first and then finding the pair by iterating through the sorted array from both ends, moving inwards towards each other.
- ✓ Since the list is sorted, the pairs that are adjacent will be the closest in value.
- ✓ We can then return the pair with the smallest absolute sum. This method has a time complexity of O(n log n), where n is the length of the list.
- ✓ The **find_pair_closest_to_zero1** function takes an array **arr3** as input and returns a tuple containing the pair of elements whose sum is closest to zero.

upGrad

2. Optimized approach

```
def find_pair_closest_to_zero1(arr3):
    arr3.sort()
    left, right = 0, len(arr3)-1
    min_sum = float('inf')
    pair = None
    while left < right:
        curr_sum = arr3[left] + arr3[right]
        if abs(curr_sum) < min_sum:
            min_sum = abs(curr_sum)
            pair = (arr3[left], arr3[right])
        if curr_sum < 0:
            left += 1
        else:
            right -= 1
    return pair

arr1 = [-4, 7, 6, 2, -5]
print("Optimized approach output values 1:",find_pair_closest_to_zero1(arr1))

arr2 = [-50, 34, -19, 24, 33, 10, -46, -38]
print("Optimized approach output values 2:",find_pair_closest_to_zero1(arr2))
```

```
Optimized approach output values 1: (-5, 6)
Optimized approach output values 2: (-38, 34)
```

upGrad

Let's break down the code:

```
def find_pair_closest_to_zero1(arr3):
```

This line defines a function named **find_pair_closest_to_zero1** which takes a single argument **arr3.** This function will find and return a pair of numbers from the input array whose sum is closest to zero.

```
arr3.sort()
```

This line sorts the input array arr3 in ascending order.

```
left, right = 0, len(arr3)-1
```

This line initializes two variables left and right to represent the indices of the first and last elements in the sorted array, respectively.

upGrad

Let's break down the code:

```
min_sum = float('inf')
pair = None
```

These lines initialize two variables **min_sum** and **pair**. **min_sum** is initialized to infinity to ensure that any valid sum of two array elements will be smaller than it. pair is initialized to None because we don't yet know which pair of numbers will be closest to zero.

```
while left < right:
```

This line starts a while loop which continues as long as the index left is less than the index right.

```
curr_sum = arr3[left] + arr3[right]
```

This line calculates the sum of the current pair of elements, where arr3[left] and arr3[right] are the current elements at indices left and right, respectively.

upGrad

Let's break down the code:

```
if abs(curr_sum) < min_sum:
    min_sum = abs(curr_sum)
    pair = (arr3[left], arr3[right])
```

If statement checks if the absolute value of **curr_sum** is less than the current minimum sum **min_sum**. If yes, then **min_sum** is updated to the absolute value of **curr_sum**, and **pair** is set to the current pair of elements **(arr3[left], arr3[right]).**

```
if curr_sum < 0:
    left += 1
else:
    right -= 1
```

The above if-else statement updates the indices left and right depending on the sign of the current sum **curr_sum**. If **curr_sum** is negative, then left is incremented to move to the next element in the sorted array. Otherwise, right is decremented to move to the previous element in the sorted array.

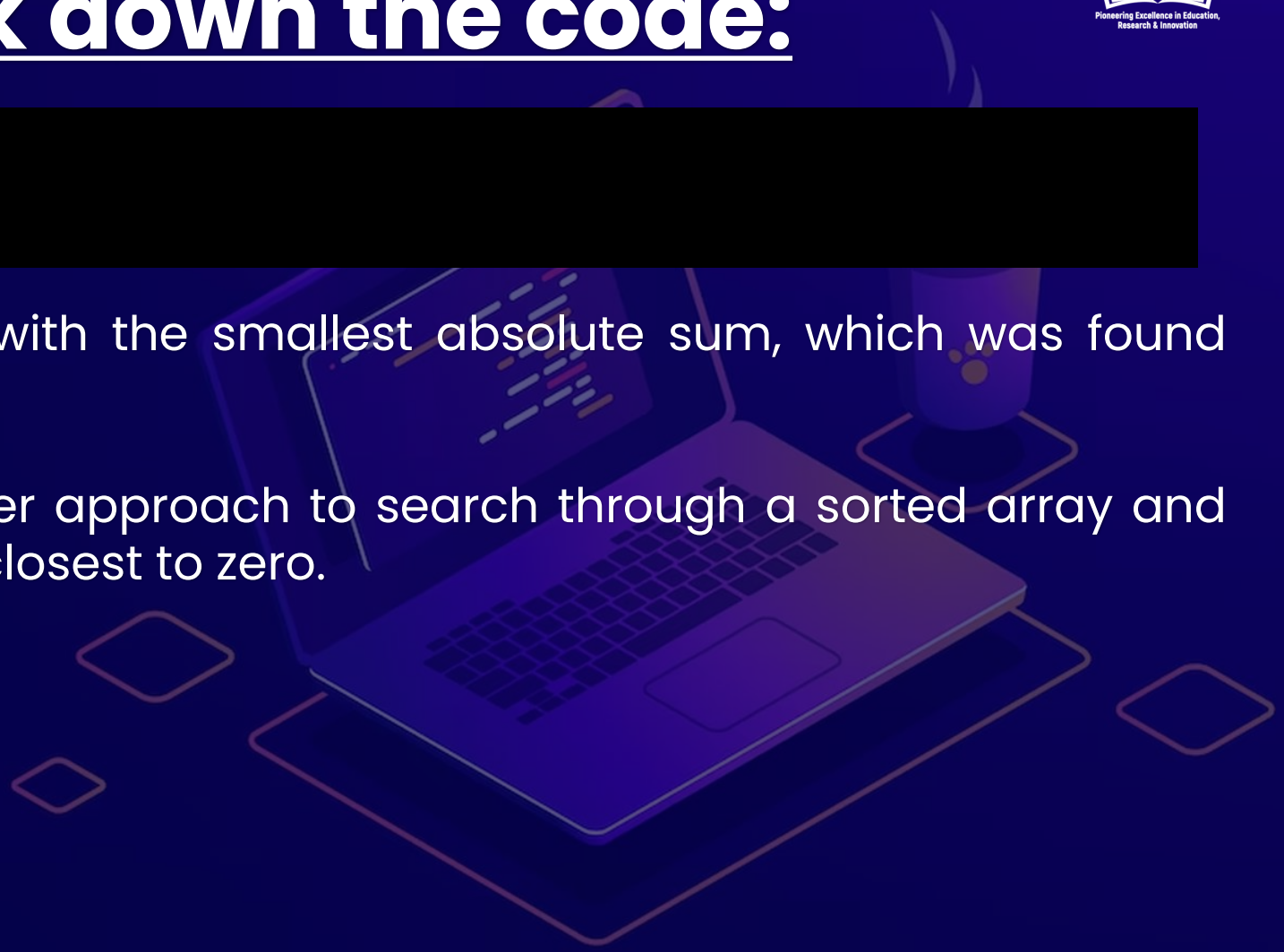
upGrad

Let's break down the code:

```
return pair
```

This line returns the pair of elements with the smallest absolute sum, which was found during the while loop.

Overall, this program uses a two-pointer approach to search through a sorted array and find the pair of elements whose sum is closest to zero.



upGrad

Conclusion

- Both methods produced the same output, which is a tuple of two integers representing the pair of elements whose sum is closest to zero. However, the two methods have different approaches and time complexities.

- The brute force method involves comparing every possible pair of elements in the input list, calculating their sum, and updating the minimum sum and corresponding pair of elements as necessary. This requires nested loops, with time complexity O(n^2) where n is the size of the input list. This method works reasonably well for small input sizes, but becomes inefficient for larger input sizes.

- On the other hand, the optimized method involves sorting the input list, and then comparing adjacent pairs of integers. This requires only a single loop, with time complexity O(n log n) for sorting the list and O(n) for comparing adjacent pairs. Overall, the time complexity of the optimized method is O(n log n). This method is more efficient than the brute force method for larger input sizes, as the number of comparisons needed is reduced.

- Therefore, for practical purposes, the optimized method is generally a better choice as it has a better time complexity for larger input sizes. However, for small input sizes, the brute force method may still be a reasonable approach.

upGrad

Pioneering Excellence in Education, Research & Innovation

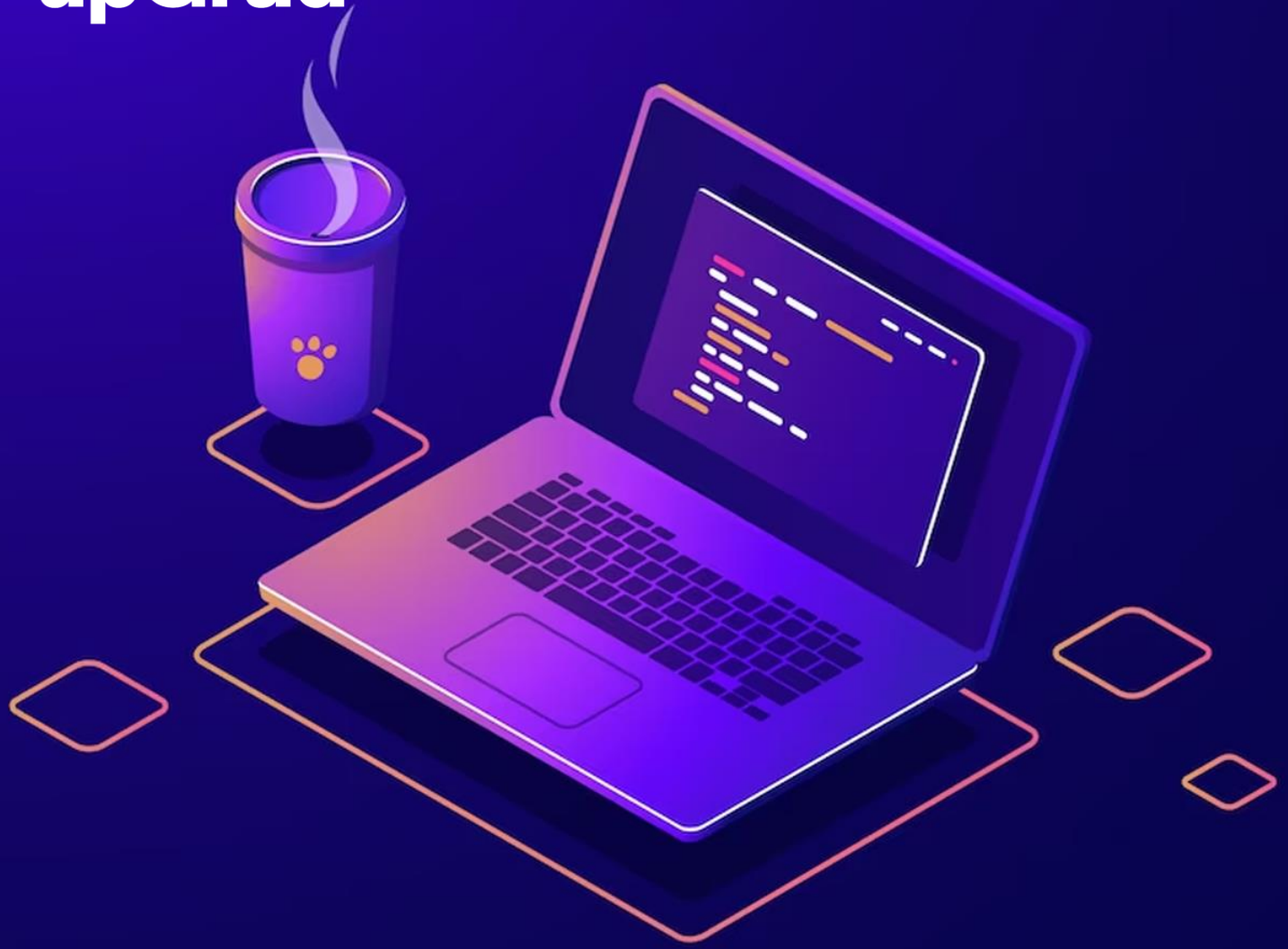
Lab Assignment

Thank You

Presented By:

Vaibhav Kolhale, Muna Sahu & Vandana NS

upGrad



Pioneering Excellence in Education, Research & Innovation

Video Link
https://youtu.be/E-kjyaioc2c