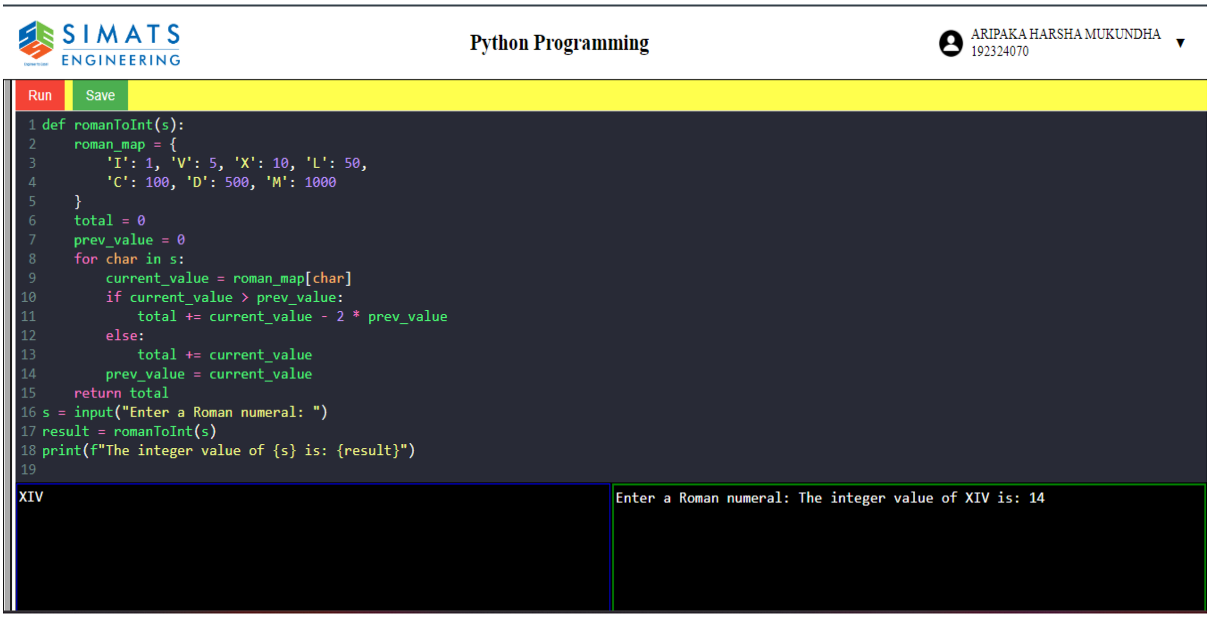
ASSIGMNENT→2

A. Harsha Mukundha

192324070

1) Converting Roman Numbers to integers



```
def romanToInt(s):
    roman_map = {
        'I': 1, 'V': 5, 'X': 10, 'L': 50,
        'C': 100, 'D': 500, 'M': 1000
    }
    total = 0
    prev_value = 0
    for char in s:
        current_value = roman_map[char]
        if current_value > prev_value:
            total += current_value - 2 * prev_value
        else:
            total += current_value
        prev_value = current_value
    return total
s = input("Enter a Roman numeral: ")
result = romanToInt(s)
print(f"The integer value of {s} is: {result}")
```

```
XIV
```

```
Enter a Roman numeral: The integer value of XIV is: 14
```

2)

Longest Common Prefix

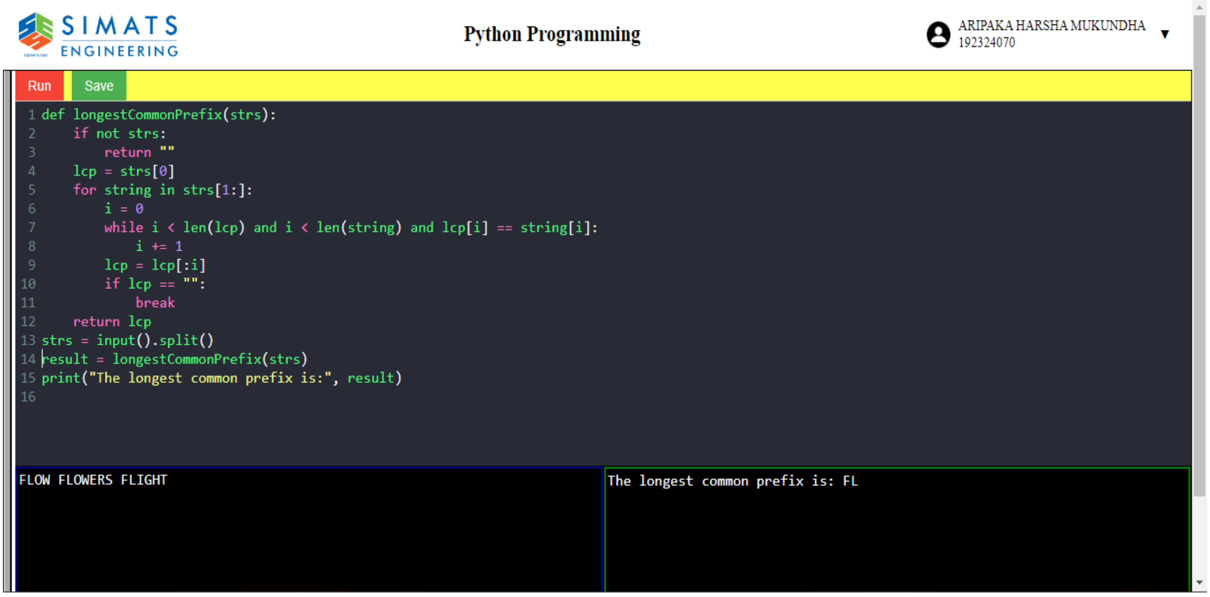
```
Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".


Example 1:

Input: strs = ["flower","flow","flight"]
Output: "fl"
Example 2:

Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.
```



Python Programming

ARIPAKA HARSHA MUKUNDHA
192324070

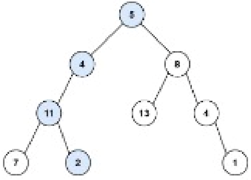
```
def longestCommonPrefix(strs):
    if not strs:
        return ""
    lcp = strs[0]
    for string in strs[1:]:
        i = 0
        while i < len(lcp) and i < len(string) and lcp[i] == string[i]:
            i += 1
        lcp = lcp[:i]
        if lcp == "":
            break
    return lcp
strs = input().split()
result = longestCommonPrefix(strs)
print("The longest common prefix is:", result)
```

```
FLOW FLOWERS FLIGHT
```

```
The longest common prefix is: FL
```

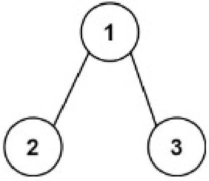
3)

Example 1:

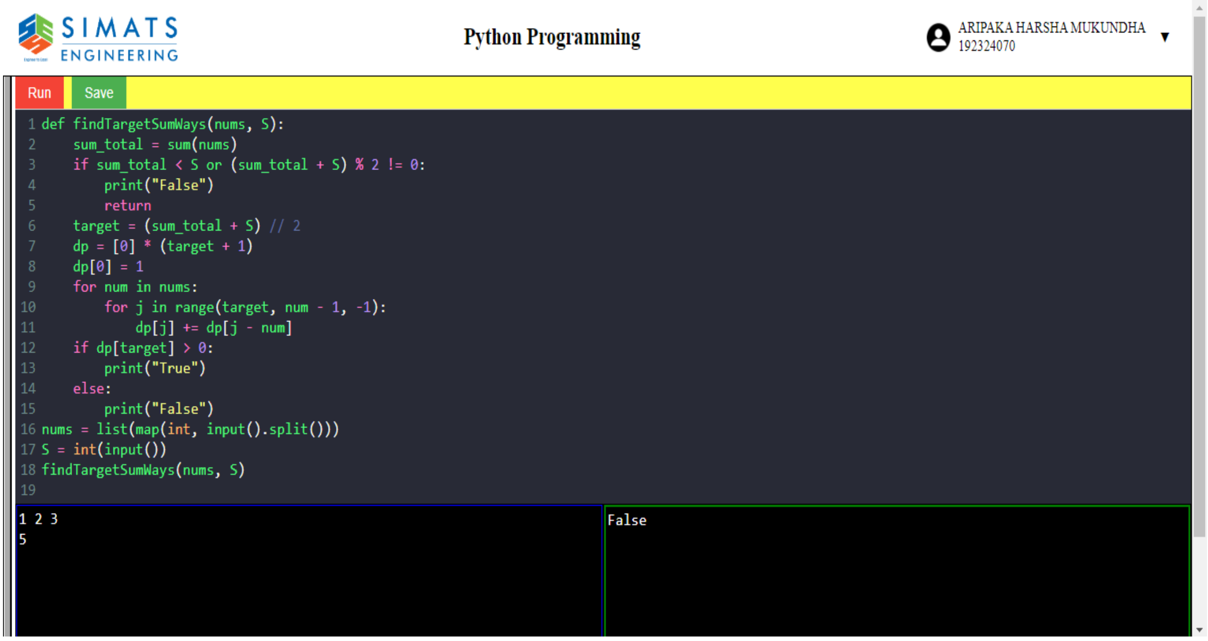


```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
Output: true
Explanation: The root-to-leaf path with the target sum is shown.
```

Example 2:



```
Input: root = [1,2,3], targetSum = 5
Output: false
Explanation: There two root-to-leaf paths in the tree:
(1 --> 2): The sum is 3.
(1 --> 3): The sum is 4.
There is no root-to-leaf path with sum = 5.
```

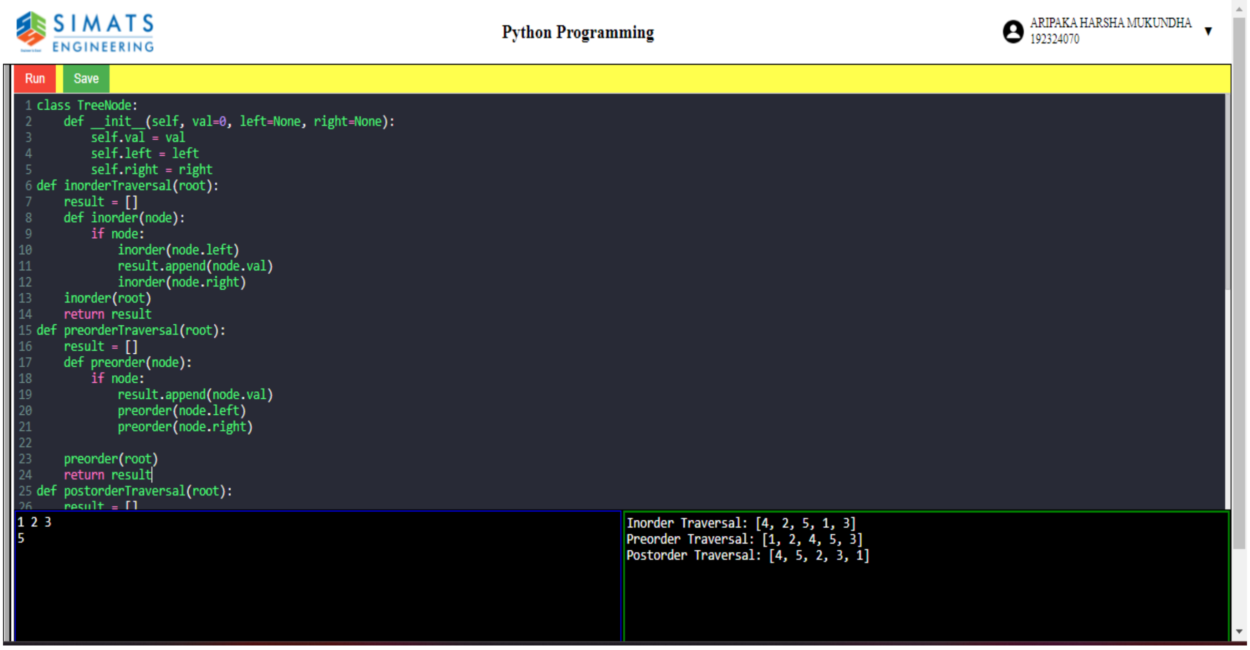


```python
def findTargetSumWays(nums, S):
    sum_total = sum(nums)
    if sum_total < S or (sum_total + S) % 2 != 0:
        print("False")
        return
    target = (sum_total + S) // 2
    dp = [0] * (target + 1)
    dp[0] = 1
    for num in nums:
        for j in range(target, num - 1, -1):
            dp[j] += dp[j - num]
    if dp[target] > 0:
        print("True")
    else:
        print("False")
nums = list(map(int, input().split()))
S = int(input())
findTargetSumWays(nums, S)
```

```
1 2 3
5
```

```
False
```

4) Binary tree traversal



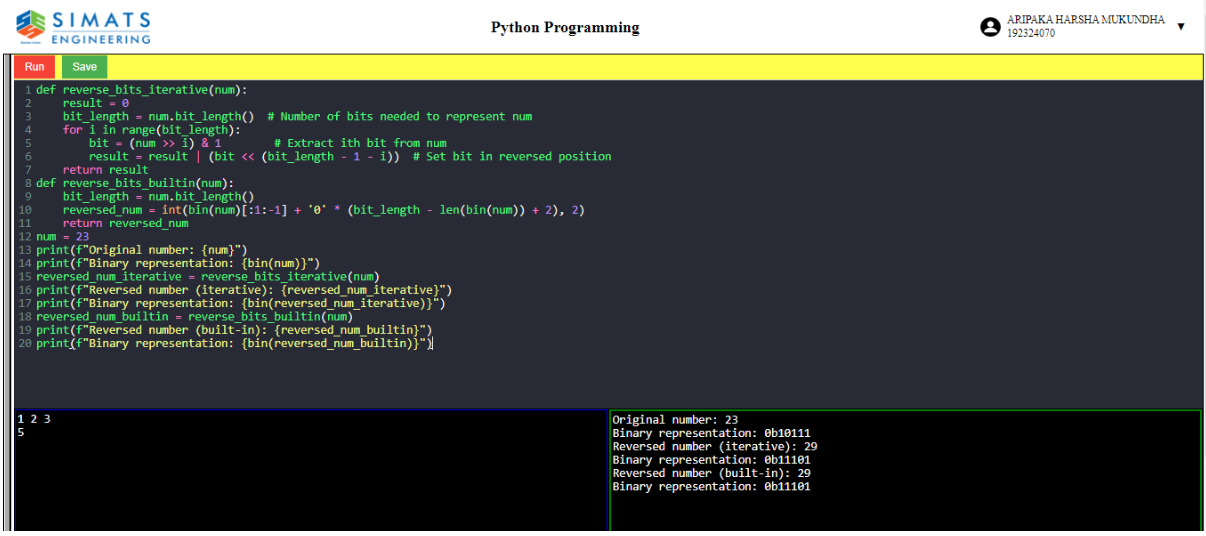
```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def inorderTraversal(root):
    result = []
    def inorder(node):
        if node:
            inorder(node.left)
            result.append(node.val)
            inorder(node.right)
    inorder(root)
    return result
def preorderTraversal(root):
    result = []
    def preorder(node):
        if node:
            result.append(node.val)
            preorder(node.left)
            preorder(node.right)

    preorder(root)
    return result
def postorderTraversal(root):
    result = []
```

```
1 2 3
5
```

```
Inorder Traversal: [4, 2, 5, 1, 3]
Preorder Traversal: [1, 2, 4, 5, 3]
Postorder Traversal: [4, 5, 2, 3, 1]
```

5) Bit Reserving



```python
def reverse_bits_iterative(num):
    result = 0
    bit_length = num.bit_length()  # Number of bits needed to represent num
    for i in range(bit_length):
        bit = (num >> i) & 1        # Extract ith bit from num
        result = result | (bit << (bit_length - 1 - i))  # Set bit in reversed position
    return result
def reverse_bits_builtin(num):
    bit_length = num.bit_length()
    reversed_num = int(bin(num)[:1:-1] + '0' * (bit_length - len(bin(num)) + 2), 2)
    return reversed_num
num = 23
print(f"Original number: {num}")
print(f"Binary representation: {bin(num)}")
reversed_num_iterative = reverse_bits_iterative(num)
print(f"Reversed number (iterative): {reversed_num_iterative}")
print(f"Binary representation: {bin(reversed_num_iterative)}")
reversed_num_builtin = reverse_bits_builtin(num)
print(f"Reversed number (built-in): {reversed_num_builtin}")
print(f"Binary representation: {bin(reversed_num_builtin)}")
```

```
1 2 3
5
```

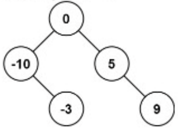
```
Original number: 23
Binary representation: 0b10111
Reversed number (iterative): 29
Binary representation: 0b11101
Reversed number (built-in): 29
Binary representation: 0b11101
```

6)

Convert Sorted Array to Binary Search Tree

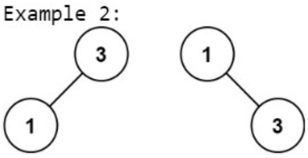
Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced
 binary search tree.

Example 1:

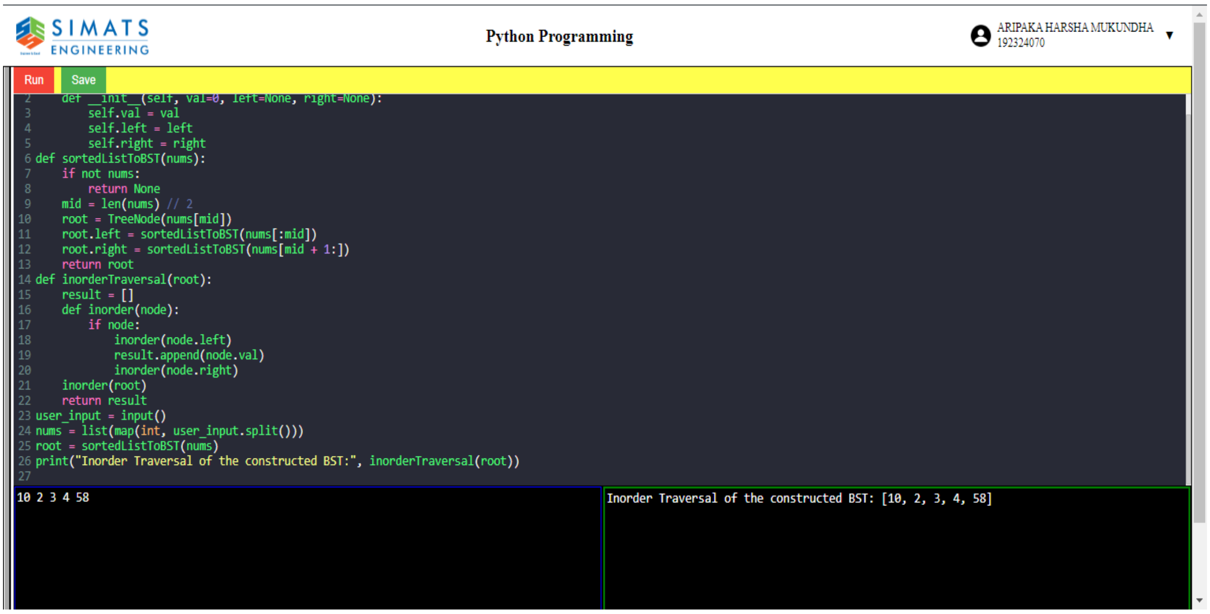


```
Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:
```

Example 2:



```
Input: nums = [1,3]
Output: [3,1]
Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.
```



```
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def sortedListToBST(nums):
    if not nums:
        return None
    mid = len(nums) // 2
    root = TreeNode(nums[mid])
    root.left = sortedListToBST(nums[:mid])
    root.right = sortedListToBST(nums[mid + 1:])
    return root
def inorderTraversal(root):
    result = []
    def inorder(node):
        if node:
            inorder(node.left)
            result.append(node.val)
            inorder(node.right)
    inorder(root)
    return result
user_input = input()
nums = list(map(int, user_input.split()))
root = sortedListToBST(nums)
print("Inorder Traversal of the constructed BST:", inorderTraversal(root))
```

```
10 2 3 4 58
```

```
Inorder Traversal of the constructed BST: [10, 2, 3, 4, 58]
```

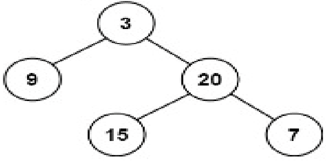
7)

Balanced Binary Tree

Given a binary tree, determine if it is height-balanced

.

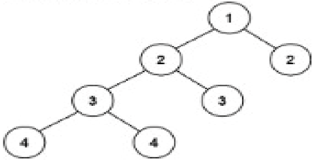
Example 1:



Input: root = [3,9,20,null,null,15,7]

Output: true

Example 2:



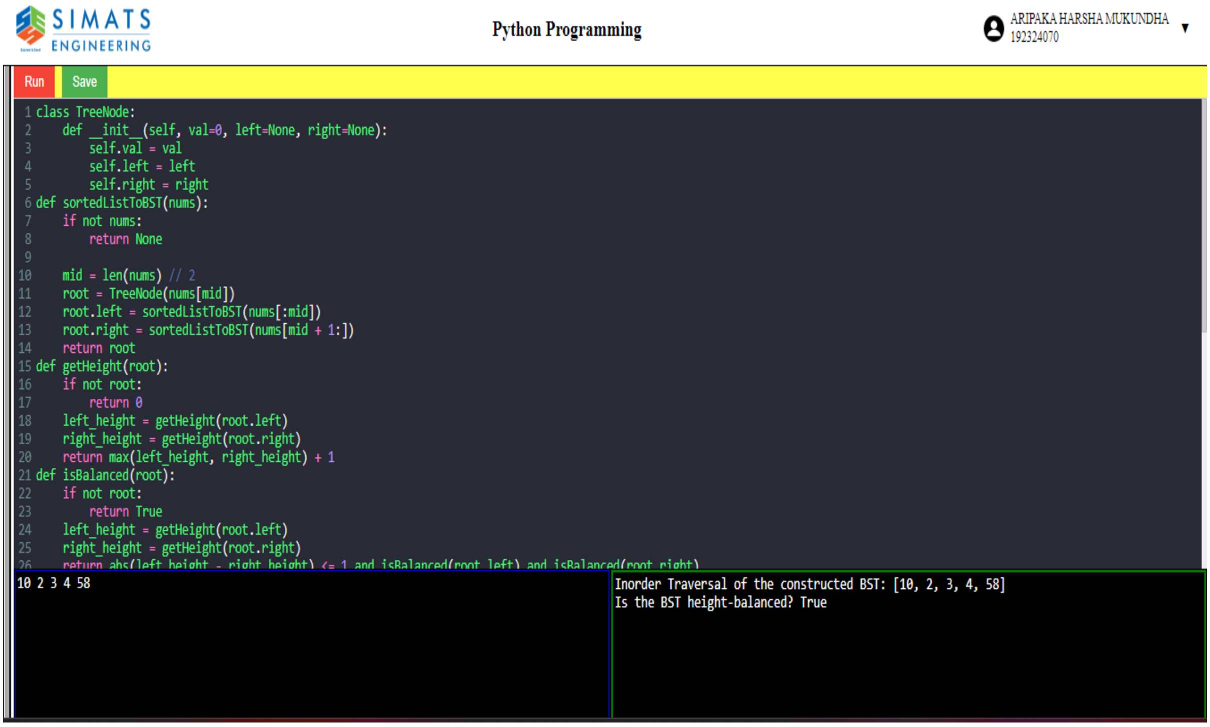
Input: root = [1,2,2,3,3,null,null,4,4]

Output: false

Example 3:

Input: root = []

Output: true



```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def sortedListToBST(nums):
    if not nums:
        return None

    mid = len(nums) // 2
    root = TreeNode(nums[mid])
    root.left = sortedListToBST(nums[:mid])
    root.right = sortedListToBST(nums[mid + 1:])
    return root
def getHeight(root):
    if not root:
        return 0
    left_height = getHeight(root.left)
    right_height = getHeight(root.right)
    return max(left_height, right_height) + 1
def isBalanced(root):
    if not root:
        return True
    left_height = getHeight(root.left)
    right_height = getHeight(root.right)
    return abs(left_height - right_height) <= 1 and isBalanced(root.left) and isBalanced(root.right)
```

```
10 2 3 4 58
```

```
Inorder Traversal of the constructed BST: [10, 2, 3, 4, 58]
Is the BST height-balanced? True
```

8)

Climbing Stairs

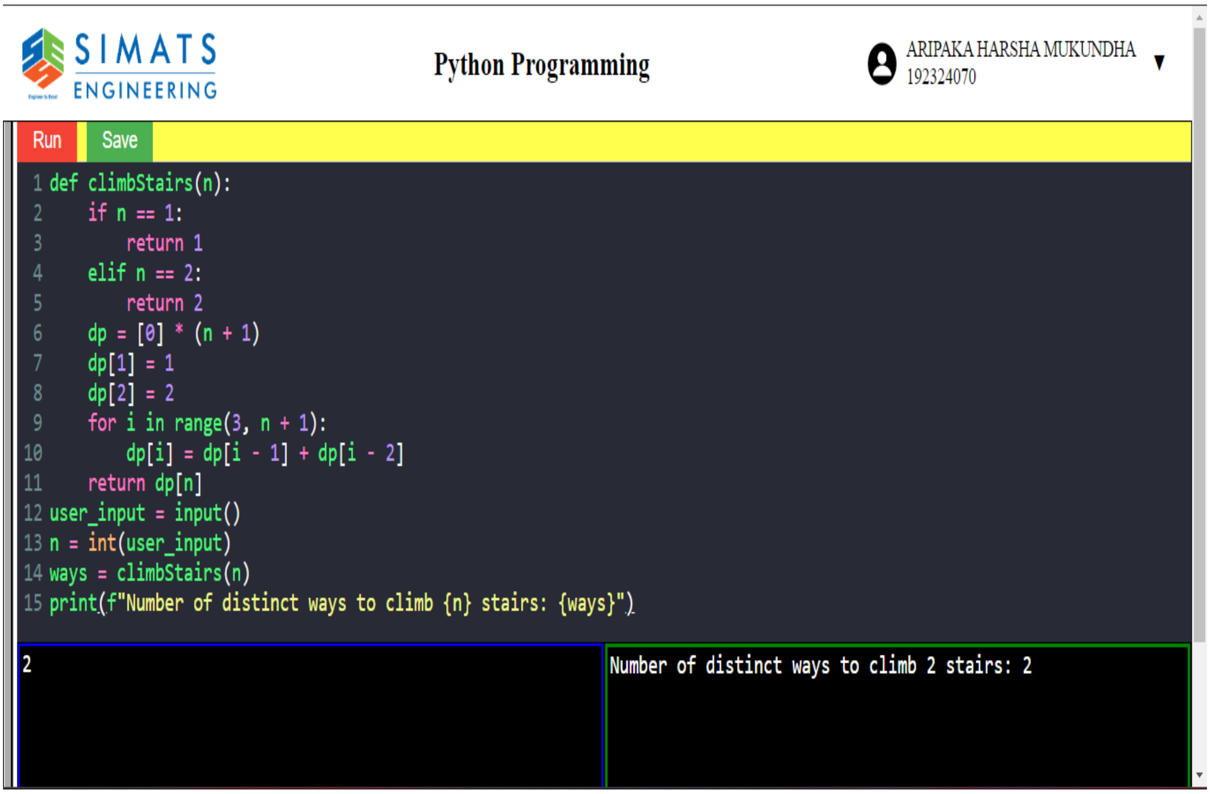
You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
Example 2:

Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
```



Python Programming

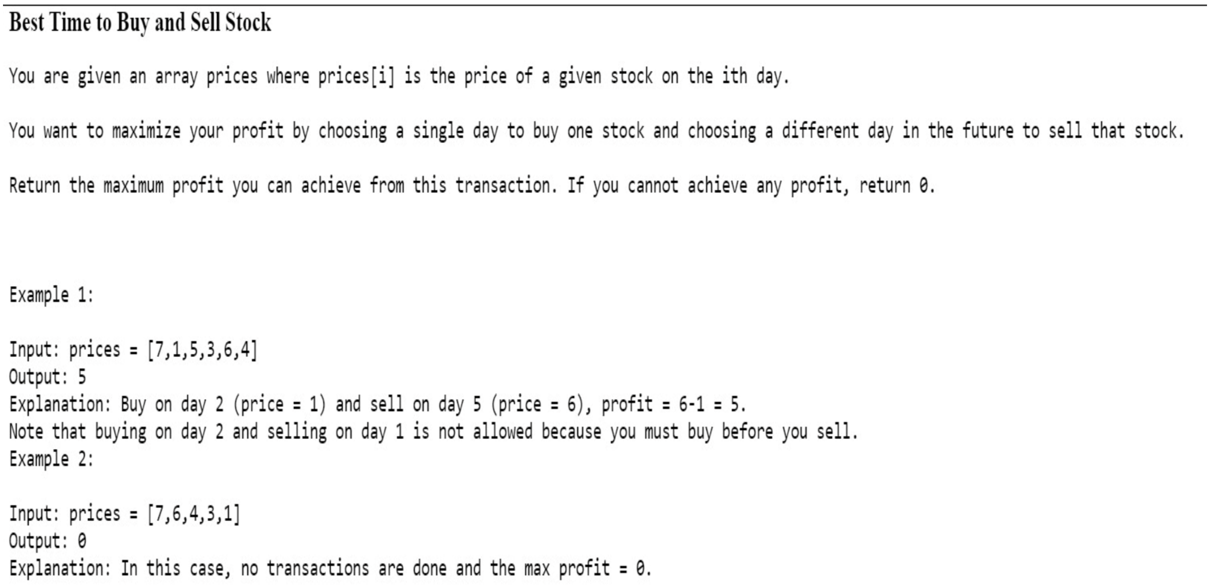
ARIPAKA HARSHA MUKUNDHA
192324070

```python
def climbStairs(n):
    if n == 1:
        return 1
    elif n == 2:
        return 2
    dp = [0] * (n + 1)
    dp[1] = 1
    dp[2] = 2
    for i in range(3, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]
user_input = input()
n = int(user_input)
ways = climbStairs(n)
print(f"Number of distinct ways to climb {n} stairs: {ways}")
```

```
2
```

```
Number of distinct ways to climb 2 stairs: 2
```

9)



Best Time to Buy and Sell Stock

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

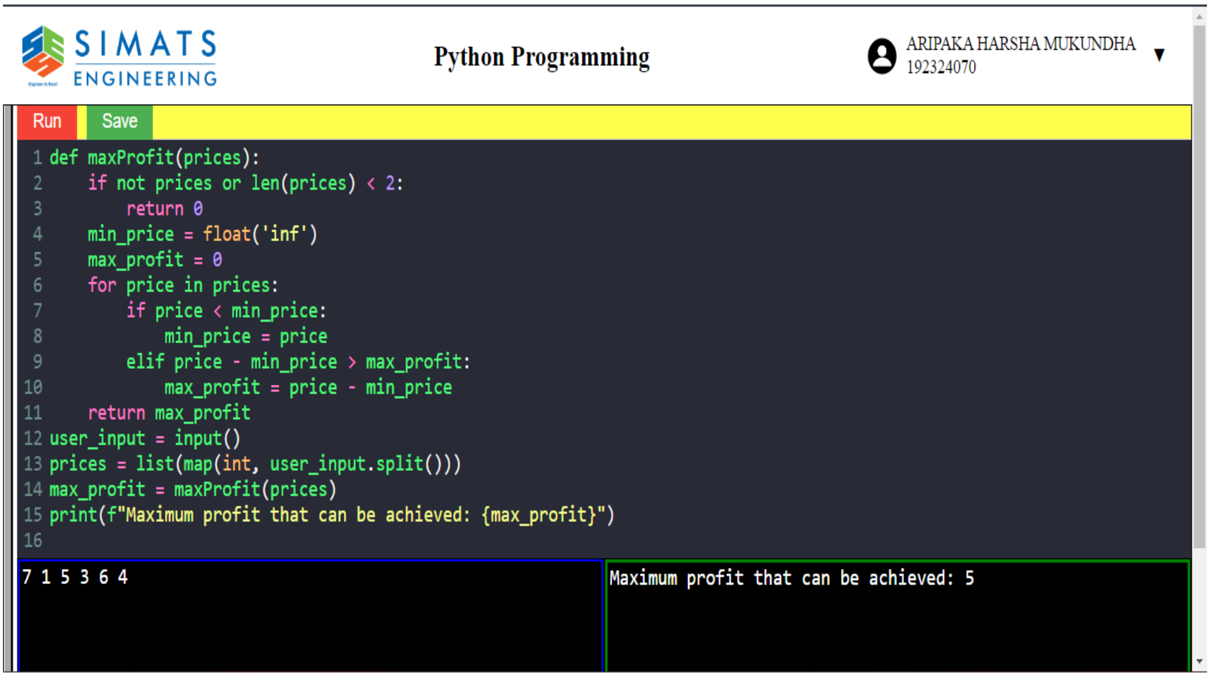
Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```



```python
def maxProfit(prices):
    if not prices or len(prices) < 2:
        return 0
    min_price = float('inf')
    max_profit = 0
    for price in prices:
        if price < min_price:
            min_price = price
        elif price - min_price > max_profit:
            max_profit = price - min_price
    return max_profit
user_input = input()
prices = list(map(int, user_input.split()))
max_profit = maxProfit(prices)
print(f"Maximum profit that can be achieved: {max_profit}")
```

```
7 1 5 3 6 4
```

```
Maximum profit that can be achieved: 5
```

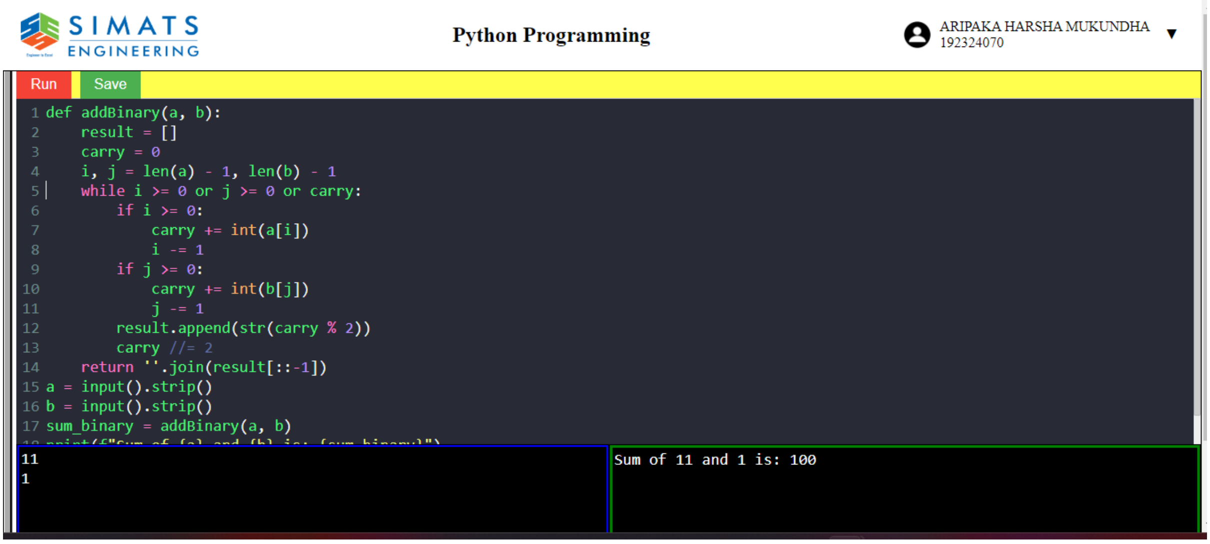
10
Add Binary

Given two binary strings a and b, return their sum as a binary string.

```
Example 1:

Input: a = "11", b = "1"
Output: "100"
Example 2:

Input: a = "1010", b = "1011"
Output: "10101"
```



```python
def addBinary(a, b):
    result = []
    carry = 0
    i, j = len(a) - 1, len(b) - 1
    while i >= 0 or j >= 0 or carry:
        if i >= 0:
            carry += int(a[i])
            i -= 1
        if j >= 0:
            carry += int(b[j])
            j -= 1
        result.append(str(carry % 2))
        carry //= 2
    return ''.join(result[::-1])
a = input().strip()
b = input().strip()
sum_binary = addBinary(a, b)
print(f"Sum of {a} and {b} is: {sum_binary}")
```

```
11
1
```

```
Sum of 11 and 1 is: 100
```