

ASSISMENT = 1

P.,MALREDDY(192372015)

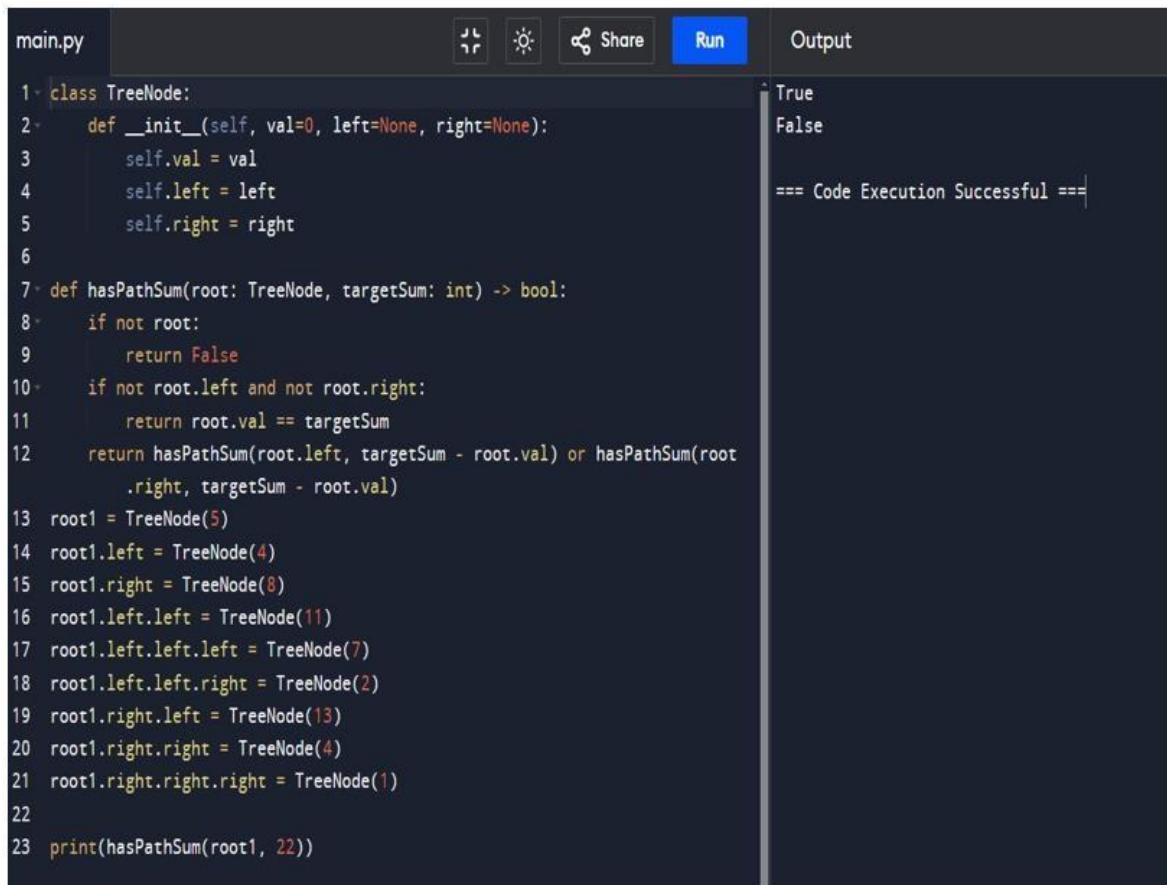
1.Converting Roman Numbers to integers?

main.py	Run	Output
<pre>1 def roman_to_int(s): 2     roman_dict = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000} 3     result = 0 4     prev_value = 0 5 6     for char in s: 7         value = roman_dict[char] 8         result += value 9         if value &gt; prev_value: 10             result -= 2 * prev_value 11             prev_value = value 12 13     return result 14 print(roman_to_int("IV")) 15 print(roman_to_int("IX")) 16 print(roman_to_int("LVIII")) 17 print(roman_to_int("MCMXCIV")) 18</pre>		<pre>4 58 1994  === Code Execution Successful ===</pre>

2.

main.py	Run	Output
<pre>1 def longestCommonPrefix(strs): 2     if not strs: 3         return "" 4 5     prefix = strs[0] 6 7     for s in strs[1:]: 8         while s[:len(prefix)] != prefix: 9             prefix = prefix[:-1] 10        if not prefix: 11            return "" 12 13    return prefix 14 15 16 print(longestCommonPrefix(["flower", "flow", "flight"])) 17 print(longestCommonPrefix(["dog", "racecar", "car"])) 18</pre>		<pre>f1  === Code Execution Successful ===</pre>

3.



```
main.py  [Icons] [Share] [Run] [Output]

1 class TreeNode:
2     def __init__(self, val=0, left=None, right=None):
3         self.val = val
4         self.left = left
5         self.right = right
6
7 def hasPathSum(root: TreeNode, targetSum: int) -> bool:
8     if not root:
9         return False
10    if not root.left and not root.right:
11        return root.val == targetSum
12    return hasPathSum(root.left, targetSum - root.val) or hasPathSum(root
        .right, targetSum - root.val)
13 root1 = TreeNode(5)
14 root1.left = TreeNode(4)
15 root1.right = TreeNode(8)
16 root1.left.left = TreeNode(11)
17 root1.left.left.left = TreeNode(7)
18 root1.left.left.right = TreeNode(2)
19 root1.right.left = TreeNode(13)
20 root1.right.right = TreeNode(4)
21 root1.right.right.right = TreeNode(1)
22
23 print(hasPathSum(root1, 22))
```

True  
False  
=== Code Execution Successful ===

4. Given the root of a binary tree and an integer of targetsum return true if the tree has a root to leaf such that adding up all the values

main.py

Share

Run


```
1 class TreeNode:
2     def __init__(self, val=0, left=None, right=None):
3         self.val = val
4         self.left = left
5         self.right = right
6
7 def hasPathSum(root: TreeNode, targetSum: int) -> bool:
8     if not root:
9         return False
10    if not root.left and not root.right:
11        return root.val == targetSum
12    return hasPathSum(root.left, targetSum - root.val) or hasPathSum(root
        .right, targetSum - root.val)
13 root1 = TreeNode(5)
14 root1.left = TreeNode(4)
15 root1.right = TreeNode(8)
16 root1.left.left = TreeNode(11)
17 root1.left.left.left = TreeNode(7)
18 root1.left.left.right = TreeNode(2)
19 root1.right.left = TreeNode(13)
20 root1.right.right = TreeNode(4)
21 root1.right.right.right = TreeNode(1)
22
23 print(hasPathSum(root1, 22))
```

True

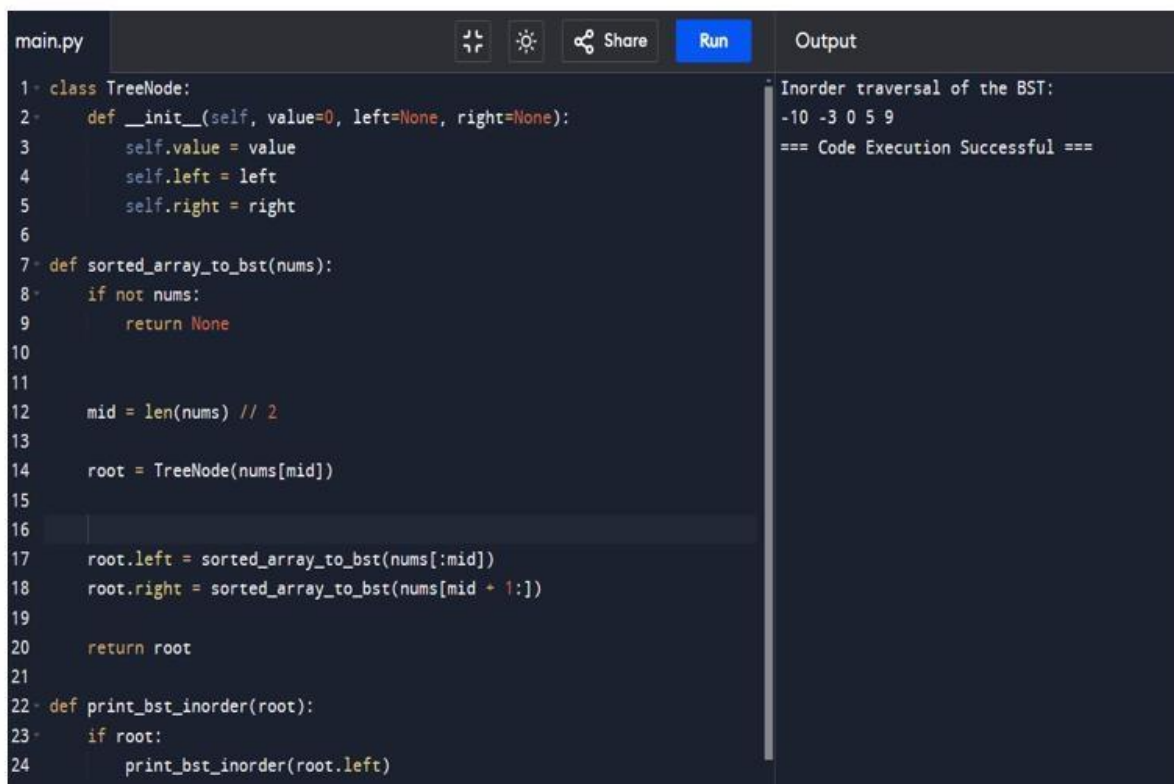
False

=== Code Execution Successful ===

## 5.bit reversing?

main.py	Run	Output
<pre>1 def reverseBits(n: int) -&gt; int: 2     result = 0 3     for i in range(32): 4         result = (result &lt;&lt; 1)   (n &amp; 1) 5         n &gt;&gt;= 1 6     return result 7 8 9 num = 43261596 10 reversed_num = reverseBits(num) 11 print(reversed_num) 12</pre>		964176192  === Code Execution Successful ===

6. convert sorted array to binary search tree?



```
main.py  [Icons] [Share] [Run] [Output]

1 class TreeNode:
2     def __init__(self, value=0, left=None, right=None):
3         self.value = value
4         self.left = left
5         self.right = right
6
7 def sorted_array_to_bst(nums):
8     if not nums:
9         return None
10
11
12     mid = len(nums) // 2
13
14     root = TreeNode(nums[mid])
15
16
17     root.left = sorted_array_to_bst(nums[:mid])
18     root.right = sorted_array_to_bst(nums[mid + 1:])
19
20     return root
21
22 def print_bst_inorder(root):
23     if root:
24         print_bst_inorder(root.left)
```

Output

Inorder traversal of the BST:  
-10 -3 0 5 9  
=== Code Execution Successful ===

7. given a binary tree, determine if it is height-balanced?

```
main.py  [Icons] [Share] [Run] Output
1 class TreeNode:
2     def __init__(self, value=0, left=None, right=None):
3         self.value = value
4         self.left = left
5         self.right = right
6
7 def is_balanced(root):
8     def check_height(node):
9         if not node:
10            return 0
11
12            left_height = check_height(node.left)
13            if left_height == -1:
14                return -1
15
16            right_height = check_height(node.right)
17            if right_height == -1:
18                return -1
19
20            if abs(left_height - right_height) > 1:
21                return -1
22
23            return max(left_height, right_height) + 1
24
25 Is the tree balanced? True
26
27 === Code Execution Successful ===
```

## 8. Climbing stairs?

```
main.py  [Icons] [Share] [Run] Output
1 def climb_stairs(n):
2     if n <= 1:
3         return 1
4
5
6     dp = [0] * (n + 1)
7
8
9     dp[0] = 1
10    dp[1] = 1
11
12
13    for i in range(2, n + 1):
14        dp[i] = dp[i - 1] + dp[i - 2]
15
16    return dp[n]
17
18
19 n = 5
20 print("Number of ways to climb the stairs:", climb_stairs(n))
21
22 Number of ways to climb the stairs: 8
23
24 === Code Execution Successful ===
```

## 9. best time to buy and sell stock?

main.py	Run	Output
<pre>1 def max_profit(prices): 2     if not prices or len(prices) &lt; 2: 3         return 0 4 5     min_price = float('inf') 6     max_profit = 0 7 8     for price in prices: 9 10        if price &lt; min_price: 11            min_price = price 12 13 14        profit = price - min_price 15 16 17        if profit &gt; max_profit: 18            max_profit = profit 19 20    return max_profit 21 22 23 prices = [7, 1, 5, 3, 6, 4] 24 print("Maximum profit:", max_profit(prices))</pre>		<pre>Maximum profit: 5 === Code Execution Successful ===</pre>

10.sum of two binary strings?

main.py	Run	Output
<pre>1 def add_binary(a: str, b: str) -&gt; str: 2 3     int_a = int(a, 2) 4     int_b = int(b, 2) 5 6 7     sum_int = int_a + int_b 8 9 10    return bin(sum_int)[2:] 11 12 13 binary_str1 = "1010" 14 binary_str2 = "1101" 15 print("Sum of binary strings:", add_binary(binary_str1, binary_str2)) 16</pre>		<pre>Sum of binary strings: 10111 === Code Execution Successful ===</pre>