ASSIGNMENT 02

1. Converting Roman Numbers to integers

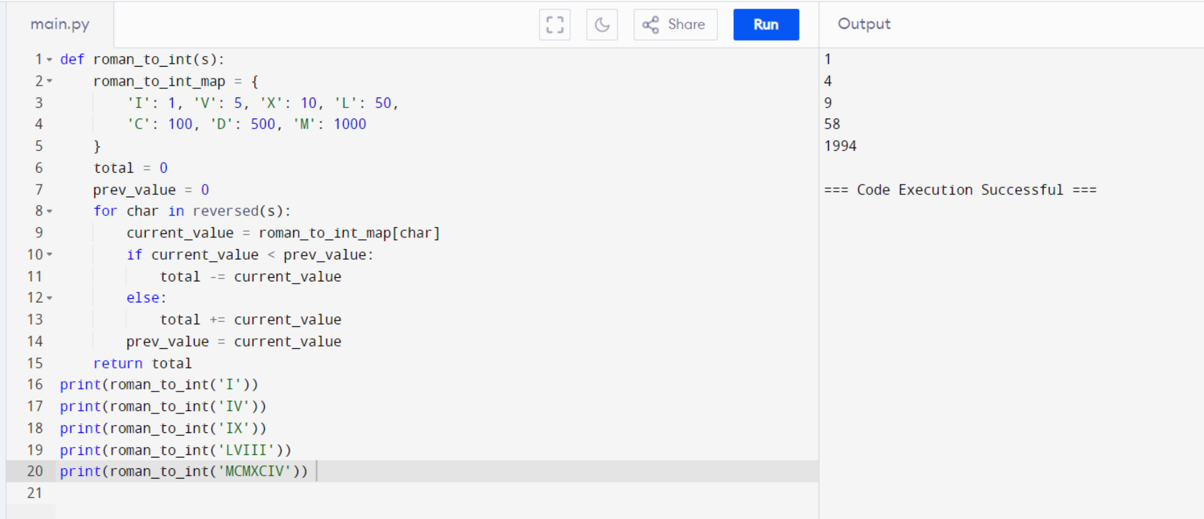
%

%%

%%%

Test case:

1. 1,0
2. -1,-1
3. $,5
4. ^,7
5. @,-1



```
def roman_to_int(s):
    roman_to_int_map = {
        'I': 1, 'V': 5, 'X': 10, 'L': 50,
        'C': 100, 'D': 500, 'M': 1000
    }
    total = 0
    prev_value = 0
    for char in reversed(s):
        current_value = roman_to_int_map[char]
        if current_value < prev_value:
            total -= current_value
        else:
            total += current_value
        prev_value = current_value
    return total
print(roman_to_int('I'))
print(roman_to_int('IV'))
print(roman_to_int('IX'))
print(roman_to_int('LVIII'))
print(roman_to_int('MCMXCIV'))
```

```
1
4
9
58
1994

=== Code Execution Successful ===
```

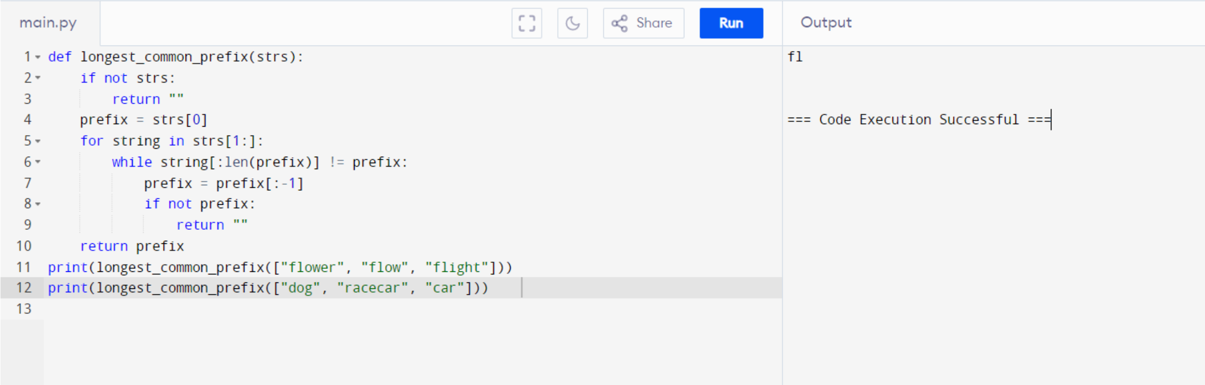
2) longest common prefix

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.
```

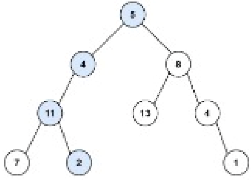


```python
def longest_common_prefix(strs):
    if not strs:
        return ""
    prefix = strs[0]
    for string in strs[1:]:
        while string[:len(prefix)] != prefix:
            prefix = prefix[:-1]
            if not prefix:
                return ""
    return prefix
print(longest_common_prefix(["flower", "flow", "flight"]))
print(longest_common_prefix(["dog", "racecar", "car"]))
```

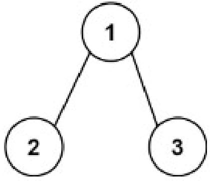
```
fl


=== Code Execution Successful ===
```

3) Given the root of a binary tree and an integer of target sum return true if the tree has a root to leaf such that adding up all the values

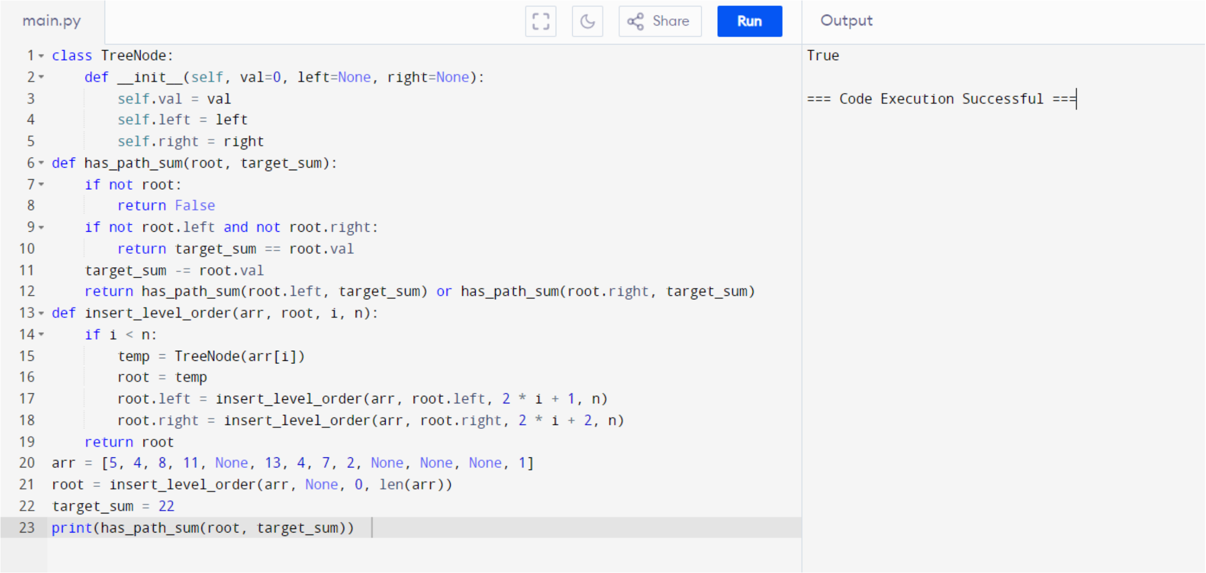
Example 1:



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:



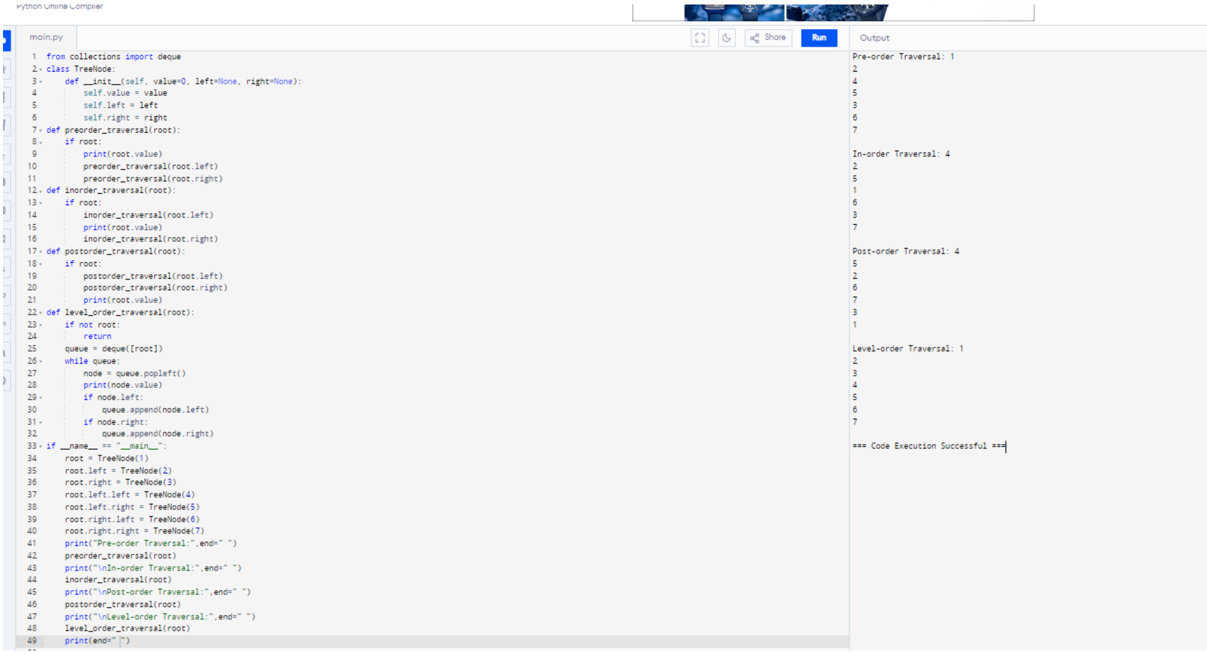
Input: root = [1,2,3], targetSum = 5
Output: false
Explanation: There two root-to-leaf paths in the tree:
(1 --> 2): The sum is 3.
(1 --> 3): The sum is 4.
There is no root-to-leaf path with sum = 5.



```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def has_path_sum(root, target_sum):
    if not root:
        return False
    if not root.left and not root.right:
        return target_sum == root.val
    target_sum -= root.val
    return has_path_sum(root.left, target_sum) or has_path_sum(root.right, target_sum)
def insert_level_order(arr, root, i, n):
    if i < n:
        temp = TreeNode(arr[i])
        root = temp
        root.left = insert_level_order(arr, root.left, 2 * i + 1, n)
        root.right = insert_level_order(arr, root.right, 2 * i + 2, n)
    return root
arr = [5, 4, 8, 11, None, 13, 4, 7, 2, None, None, None, 1]
root = insert_level_order(arr, None, 0, len(arr))
target_sum = 22
print(has_path_sum(root, target_sum))
```

```
True

=== Code Execution Successful ===
```

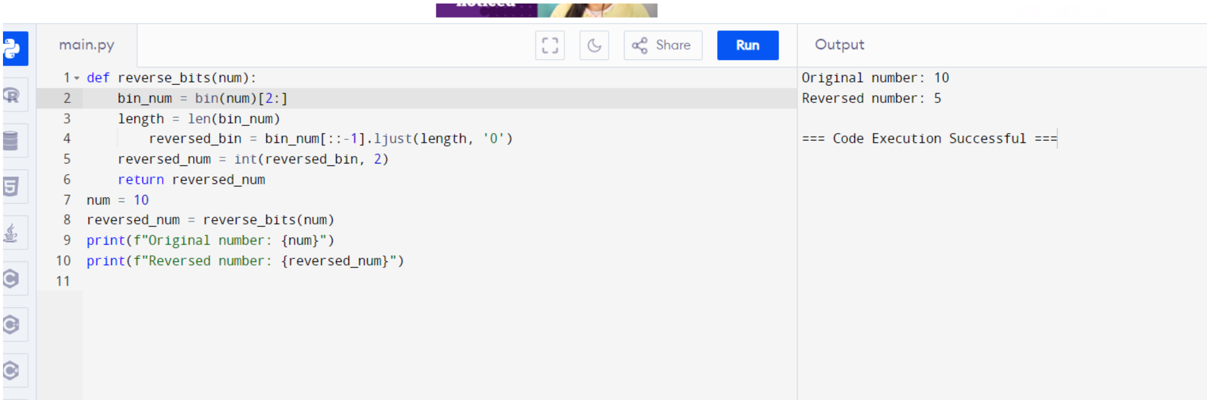
4) Binary tree traversal



```python
from collections import deque
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right
def preorder_traversal(root):
    if root:
        print(root.value)
        preorder_traversal(root.left)
        preorder_traversal(root.right)
def inorder_traversal(root):
    if root:
        inorder_traversal(root.left)
        print(root.value)
        inorder_traversal(root.right)
def postorder_traversal(root):
    if root:
        postorder_traversal(root.left)
        postorder_traversal(root.right)
        print(root.value)
def level_order_traversal(root):
    if not root:
        return
    queue = deque([root])
    while queue:
        node = queue.popleft()
        print(node.value)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
if __name__ == "__main__":
    root = TreeNode(1)
    root.left = TreeNode(2)
    root.right = TreeNode(3)
    root.left.left = TreeNode(4)
    root.left.right = TreeNode(5)
    root.right.left = TreeNode(6)
    root.right.right = TreeNode(7)
    print("Pre-order Traversal:",end=" ")
    preorder_traversal(root)
    print("\nIn-order Traversal:",end=" ")
    inorder_traversal(root)
    print("\nPost-order Traversal:",end=" ")
    postorder_traversal(root)
    print("\nLevel-order Traversal:",end=" ")
    level_order_traversal(root)
    print(end=" ")
```

```
Pre-order Traversal: 1
2
4
5
3
6
7

In-order Traversal: 4
2
5
1
6
3
7

Post-order Traversal: 4
5
2
6
7
3
1

Level-order Traversal: 1
2
3
4
5
6
7

=== Code Execution Successful ===
```

5) Bit Reversing



```python
def reverse_bits(num):
    bin_num = bin(num)[2:]
    length = len(bin_num)
        reversed_bin = bin_num[::-1].ljust(length, '0')
    reversed_num = int(reversed_bin, 2)
    return reversed_num
num = 10
reversed_num = reverse_bits(num)
print(f"Original number: {num}")
print(f"Reversed number: {reversed_num}")
```

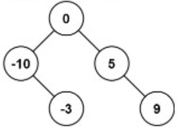
```
Original number: 10
Reversed number: 5

=== Code Execution Successful ===
```

6) Convert sorted array into binary search tree

Convert Sorted Array to Binary Search Tree

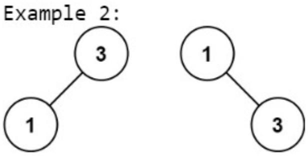
Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced
 binary search tree.

Example 1:

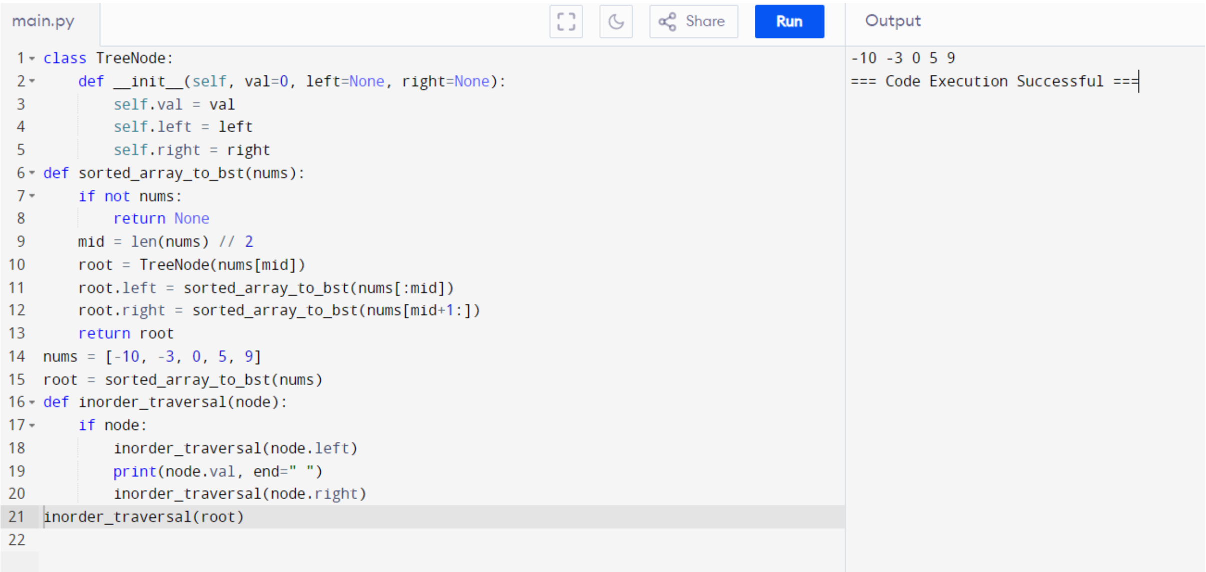


```
Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:
```

Example 2:



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



```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def sorted_array_to_bst(nums):
    if not nums:
        return None
    mid = len(nums) // 2
    root = TreeNode(nums[mid])
    root.left = sorted_array_to_bst(nums[:mid])
    root.right = sorted_array_to_bst(nums[mid+1:])
    return root
nums = [-10, -3, 0, 5, 9]
root = sorted_array_to_bst(nums)
def inorder_traversal(node):
    if node:
        inorder_traversal(node.left)
        print(node.val, end=" ")
        inorder_traversal(node.right)
inorder_traversal(root)
```

```
-10 -3 0 5 9
=== Code Execution Successful ===
```

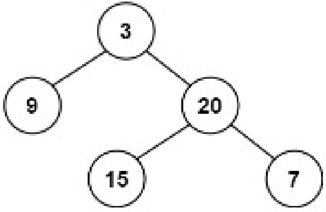
7) Balanced Binary Tree

Balanced Binary Tree

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

.

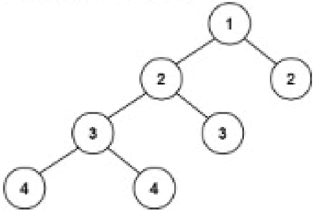
Example 1:



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

Output: true

Example 2:



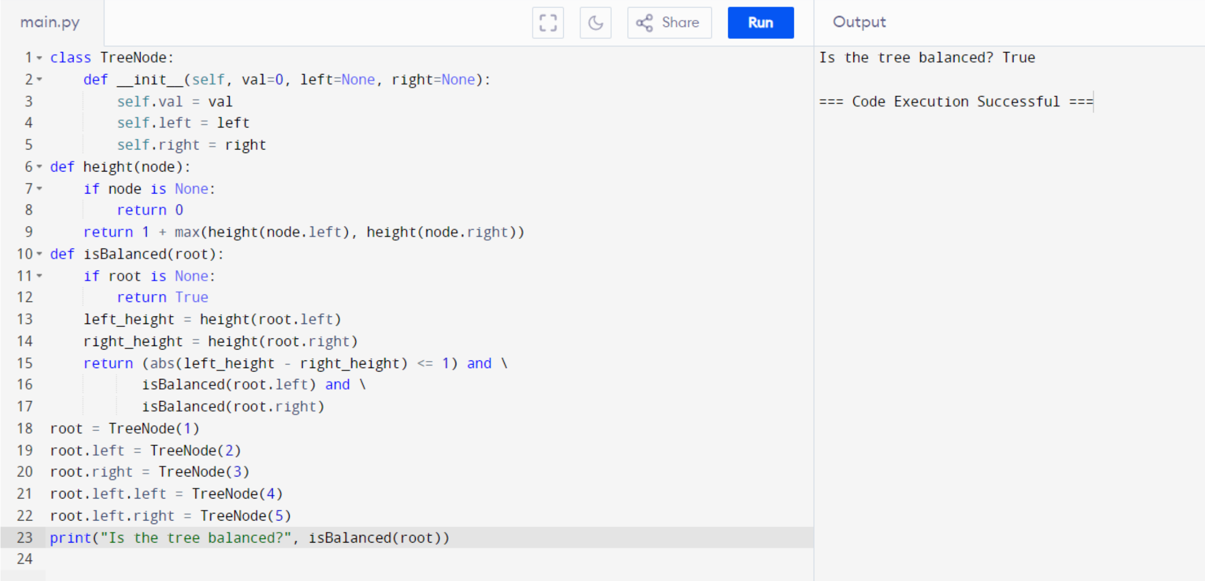
Input: root = [1,2,2,3,3,null,null,4,4]

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 height(node):
    if node is None:
        return 0
    return 1 + max(height(node.left), height(node.right))
def isBalanced(root):
    if root is None:
        return True
    left_height = height(root.left)
    right_height = height(root.right)
    return (abs(left_height - right_height) <= 1) and \
           isBalanced(root.left) and \
           isBalanced(root.right)
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
print("Is the tree balanced?", isBalanced(root))
```

```
Is the tree balanced? True

=== Code Execution Successful ===
```

8) climbing stairs

Climbing Stairs

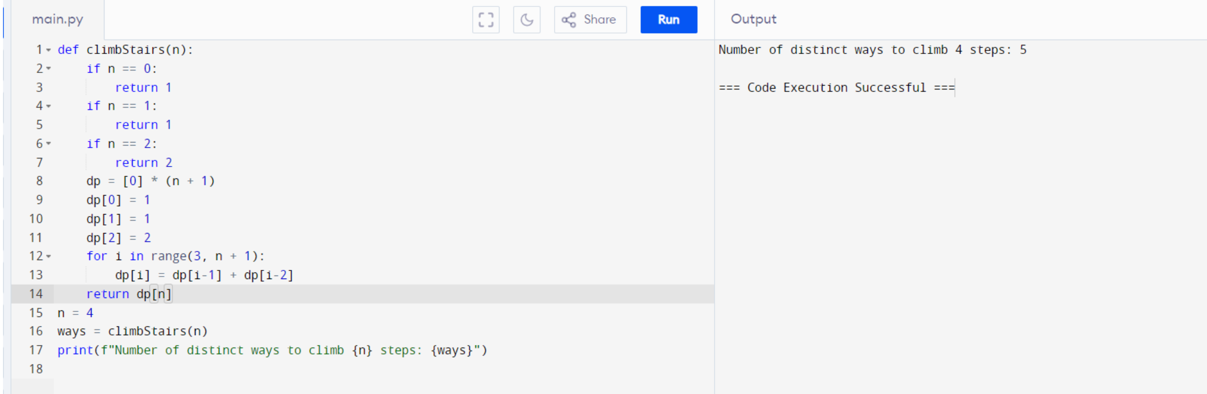
You are climbing a staircase. It takes n steps to reach the top.

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:

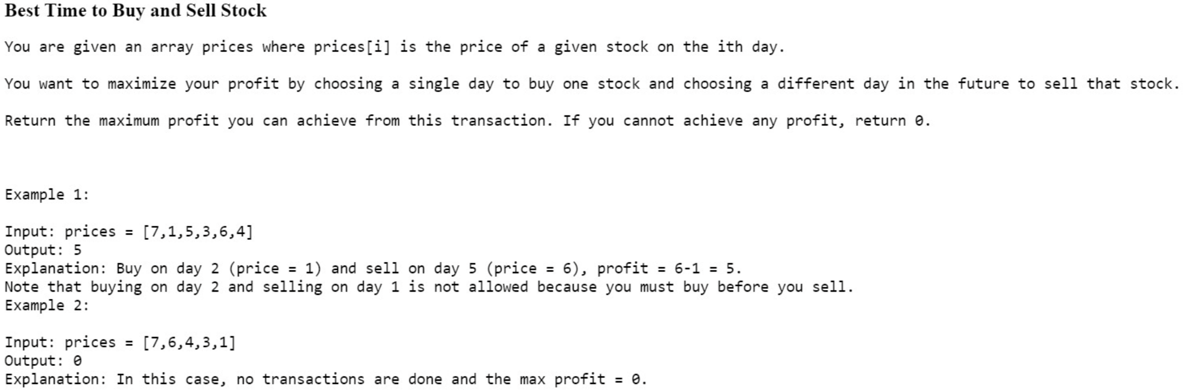
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step



```python
def climbStairs(n):
    if n == 0:
        return 1
    if n == 1:
        return 1
    if n == 2:
        return 2
    dp = [0] * (n + 1)
    dp[0] = 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]
n = 4
ways = climbStairs(n)
print(f"Number of distinct ways to climb {n} steps: {ways}")
```

```
Number of distinct ways to climb 4 steps: 5

=== Code Execution Successful ===
```

9) Best time to buy and sell stock



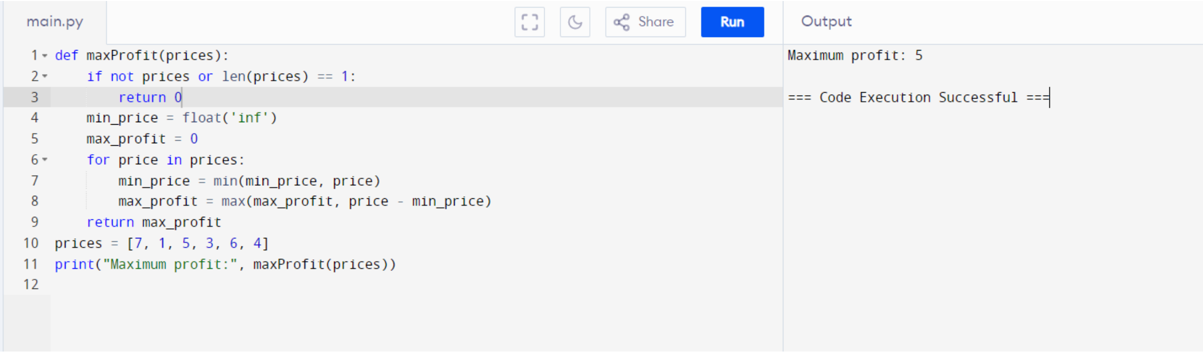
Best Time to Buy and Sell Stock

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.

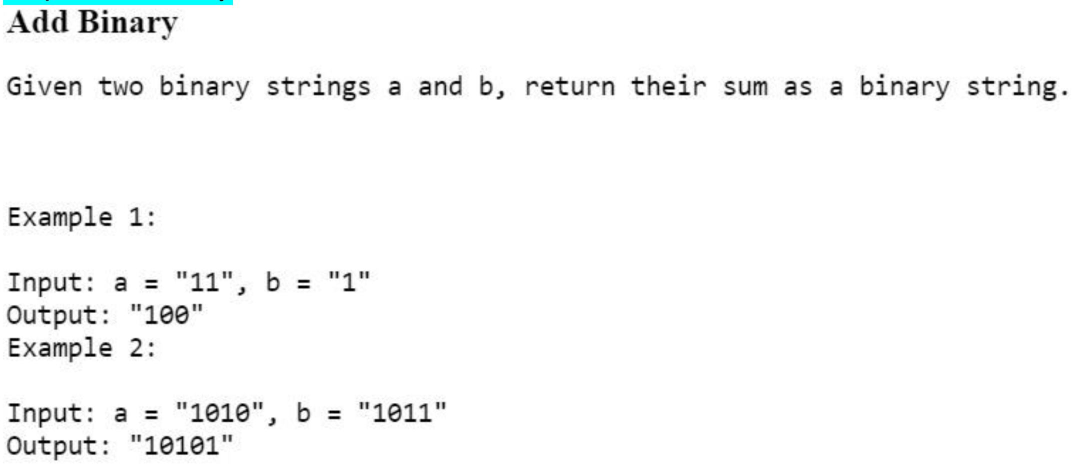
```
Example 1:

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) == 1:
        return 0
    min_price = float('inf')
    max_profit = 0
    for price in prices:
        min_price = min(min_price, price)
        max_profit = max(max_profit, price - min_price)
    return max_profit
prices = [7, 1, 5, 3, 6, 4]
print("Maximum profit:", maxProfit(prices))
```

```
Maximum profit: 5

=== Code Execution Successful ===
```

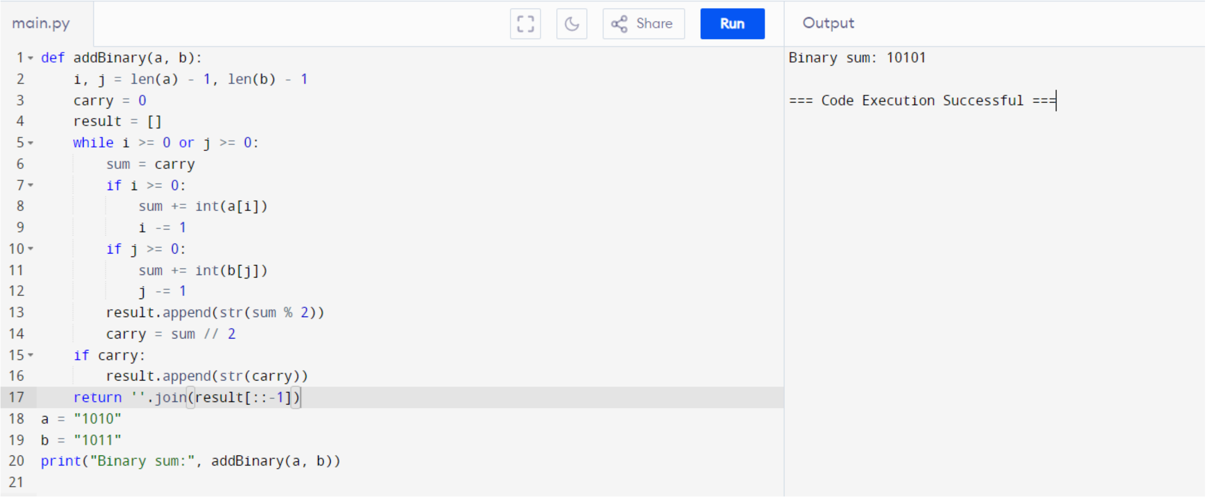
10)Add binary



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):
    i, j = len(a) - 1, len(b) - 1
    carry = 0
    result = []
    while i >= 0 or j >= 0:
        sum = carry
        if i >= 0:
            sum += int(a[i])
            i -= 1
        if j >= 0:
            sum += int(b[j])
            j -= 1
        result.append(str(sum % 2))
        carry = sum // 2
    if carry:
        result.append(str(carry))
    return ''.join(result[::-1])
a = "1010"
b = "1011"
print("Binary sum:", addBinary(a, b))
```

```
Binary sum: 10101

=== Code Execution Successful ===
```