# 1.converting roman numbers to integers.

Python Online Compiler

main.py

Share Run

```python
def roman_to_integer(roman):
    roman_numerals = {
        'I': 1,
        'V': 5,
        'X': 10,
        'L': 50,
        'C': 100,
        'D': 500,
        'M': 1000
    }
    integer_value = 0
    prev_value = 0

    for numeral in reversed(roman):
        value = roman_numerals[numeral]
        if value >= prev_value:
            integer_value += value
        else:
            integer_value -= value
        prev_value = value

    return integer_value
roman_numeral = "MCMXCIV"
integer_value = roman_to_integer(roman_numeral)
print(f"The integer value of the Roman numeral {roman_numeral} is {integer_value}")
```

Output

```
The integer value of the Roman numeral MCMXCIV is 1994

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

# 2.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”.

Python Online Compiler



main.py

Share Run

```python
def longest_common_prefix(strs):
    if not strs:
        return ""


    prefix = strs[0]

    for s in strs[1:]:
        while s[:len(prefix)] != prefix and prefix:
            prefix = prefix[:-1]
            if not prefix:
                return ""

    return prefix

strings = ["flower", "flow", "flight"]
print(longest_common_prefix(strings))
```

Output

```
fl

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

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

main.py

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def hasPathSum(root, sum):
    if not root:
        return False

    if not root.left and not root.right:
        return sum == root.val

    sum -= root.val
    return hasPathSum(root.left, sum) or hasPathSum(root.right, sum)

root = TreeNode(5)
root.left = TreeNode(4)
root.right = TreeNode(8)
root.left.left = TreeNode(11)
root.left.left.left = TreeNode(7)
root.left.left.right = TreeNode(2)
root.right.left = TreeNode(13)
root.right.right = TreeNode(4)
root.right.right.right = TreeNode(1)

print(hasPathSum(root, 22))
```

Share Run

Output

```
True

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

# 4.Binary tree traversal.

main.py

```python
class TreeNode:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

def inorder_traversal(root):
    result = []

    def traverse(node):
        if node:
            traverse(node.left)
            result.append(node.val)
            traverse(node.right)

    traverse(root)
    return result

def preorder_traversal(root):
    result = []

    def traverse(node):
        if node:
            result.append(node.val)
            traverse(node.left)
            traverse(node.right)

    traverse(root)
    return result

def postorder_traversal(root):
    result = []

    def traverse(node):
        if node:
            traverse(node.left)
            traverse(node.right)
            result.append(node.val)

    traverse(root)
    return result

from collections import deque

def levelorder_traversal(root):
    result = []
    if not root:
        return result

    queue = deque([root])

    while queue:
        node = queue.popleft()
        result.append(node.val)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)

    return result


root = TreeNode(1)
```

Output

```
In-order Traversal: [4, 2, 5, 1, 3]
Pre-order Traversal: [1, 2, 4, 5, 3]
Post-order Traversal: [4, 5, 2, 3, 1]
Level-order Traversal: [1, 2, 3, 4, 5]

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

# 5.Bit Reserving.

main.py

```python
def reserve_bits(num, mask):
    """
    Reserves (sets) bits in 'num' according to 'mask'.

    :param num: Integer, the number whose bits are to be reserved.
    :param mask: Integer, the mask indicating which bits to reserve.
    :return: Integer, the result after reserving the bits.
    """
    return num | mask

num = 0b00101100
mask = 0b00010010


result = reserve_bits(num, mask)

print(f"Original number: {bin(num)} ({num})")
print(f"Mask: {bin(mask)} ({mask})")
print(f"Result after reserving bits: {bin(result)} ({result})")
```

Output

```
Original number: 0b101100 (44)
Mask: 0b10010 (18)
Result after reserving bits: 0b111110 (62)

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

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

main.py

```python
class TreeNode:
    def _init_(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def isBalanced(root: TreeNode) -> bool:
    def checkHeight(node: TreeNode) -> int:
        if not node:
            return 0

        left_height = checkHeight(node.left)
        if left_height == -1:
            return -1

        right_height = checkHeight(node.right)
        if right_height == -1:
            return -1

        if abs(left_height - right_height) > 1:
            return -1

        return max(left_height, right_height) + 1

    return checkHeight(root) != -1

root = TreeNode
```

Output

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

# 7.Given a binary tree ,determine if it is heighted balanced.

main.py

```python
class TreeNode:
    def __init__(self, value=0, left=None, right=None):
        self.value = value
        self.left = left
        self.right = right

def is_balanced(root):
    """
    Determine if a binary tree is height-balanced.

    :param root: TreeNode, the root of the binary tree.
    :return: True if the tree is height-balanced, False otherwise.
    """
    def check_height(node):
        if node is None:
            return 0

        left_height = check_height(node.left)
        if left_height == -1:
            return -1

        right_height = check_height(node.right)
        if right_height == -1:
            return -1

        if abs(left_height - right_height) > 1:
            return -1

        return max(left_height, right_height) + 1

    return check_height(root) != -1


root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)

print(is_balanced(root))
```

Output

```
True

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

# 8.Climbing stairs, your are climbing a staircase it takes n steps to reach the top.

# Sach time you can either climb 1 or 2 steps.in how many distinct ways can you climb to the top?

main.py

```python
def climb_stairs(n):
    """
    Calculate the number of distinct ways to climb to the top of a staircase with `n` steps.

    :param n: Integer, the number of steps in the staircase.
    :return: Integer, the number of distinct ways to reach the top.
    """
    if n == 1:
        return 1
    if 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]

n = 5
print(climb_stairs(n))
```

Output

```
8

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

# 9. Best these to buy and sell stock, you are given an array prices where prices[i] day. you want to max your

**profit by choosing a single day to buy one stock and choosing a diffrent day in the future to sell stocks.if you cannot achieve any profit ,return 0. give python programming.**

main.py

```python
def max_profit(prices):
    """
    Calculate the maximum profit that can be achieved by buying and selling stocks on different days.

    :param prices: List of integers where prices[i] is the price of the stock on the ith day.
    :return: Integer representing the maximum profit. If no profit can be achieved, returns 0.
    """
    if not prices:
        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


prices = [7, 1, 5, 3, 6, 4]
print(max_profit(prices))
```

Output

```
5

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

# 10. Given two binary strings A and B ,return their sum as a binary string give python code.

main.py

```python
def add_binary(a, b):
    """
    Add two binary strings and return their sum as a binary string.

    :param a: String representing the first binary number.
    :param b: String representing the second binary number.
    :return: String representing the sum of the binary numbers.
    """
    num1 = int(a, 2)
    num2 = int(b, 2)

    total = num1 + num2

    return bin(total)[2:]

a = "1010"
b = "1101"
print(add_binary(a, b))
```

Output

```
10111

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