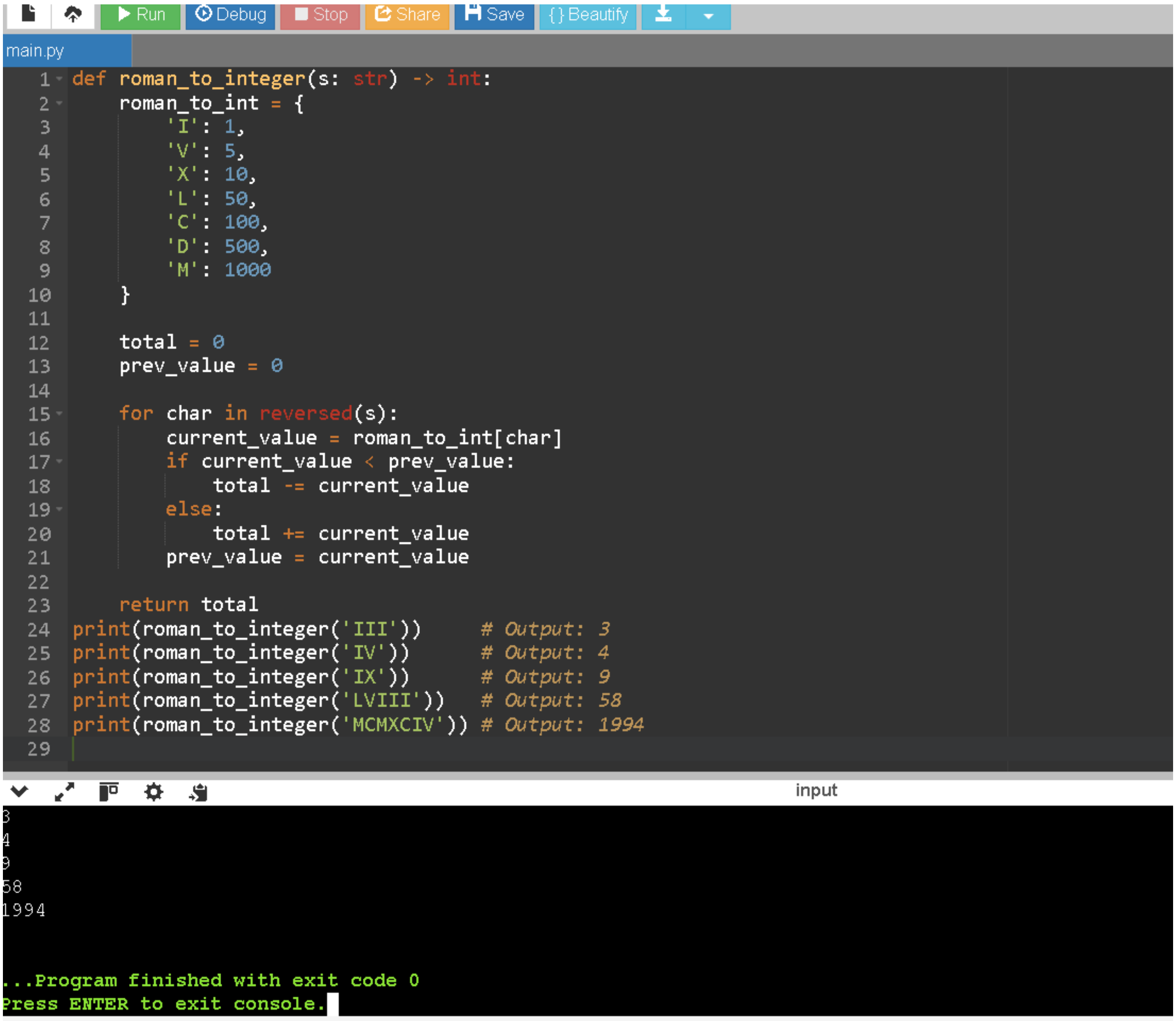
Assignment 1&2

Saveetha school of engineering

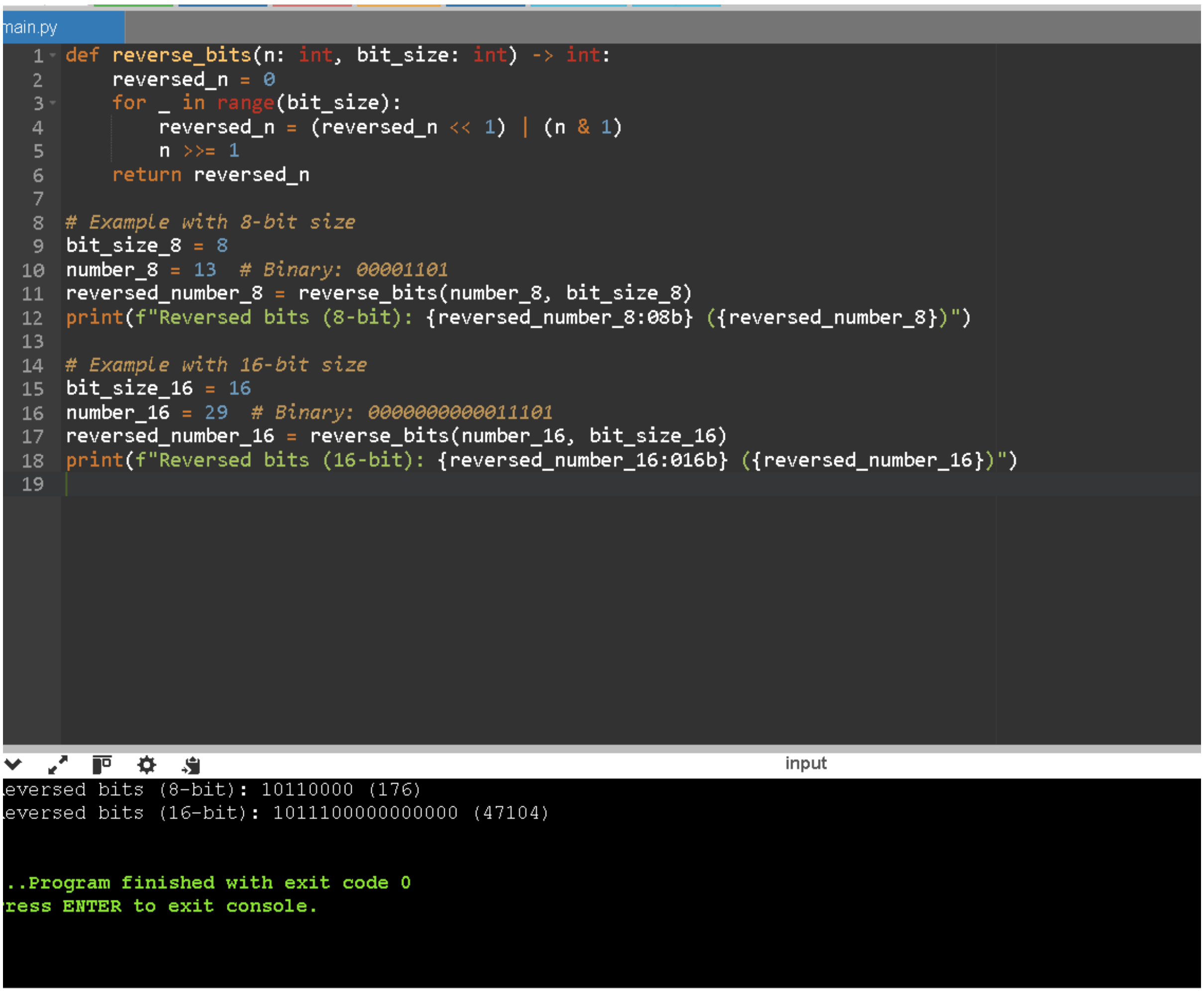
1.Converting Roman Numbers to integers



```
def roman_to_integer(s: str) -> int:
    roman_to_int = {
        '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[char]
        if current_value < prev_value:
            total -= current_value
        else:
            total += current_value
        prev_value = current_value

    return total
print(roman_to_integer('III'))     # Output: 3
print(roman_to_integer('IV'))      # Output: 4
print(roman_to_integer('IX'))      # Output: 9
print(roman_to_integer('LVIII'))   # Output: 58
print(roman_to_integer('MCMXCIV')) # Output: 1994

```

```
3
4
9
58
1994


...Program finished with exit code 0
Press ENTER to exit console.
```

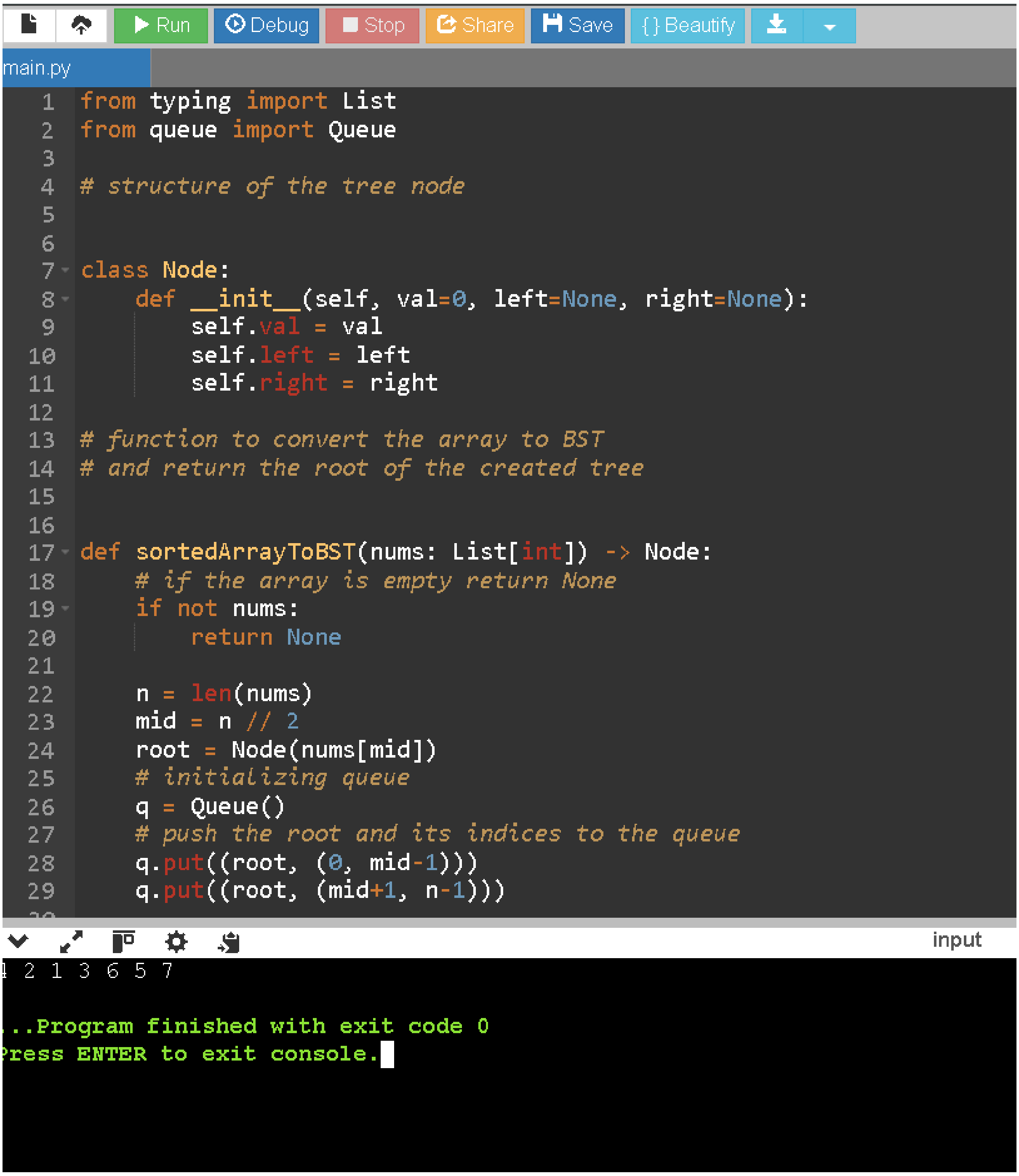
2.Bit Reserving



```python
def reverse_bits(n: int, bit_size: int) -> int:
    reversed_n = 0
    for _ in range(bit_size):
        reversed_n = (reversed_n << 1) | (n & 1)
        n >>= 1
    return reversed_n

# Example with 8-bit size
bit_size_8 = 8
number_8 = 13  # Binary: 00001101
reversed_number_8 = reverse_bits(number_8, bit_size_8)
print(f"Reversed bits (8-bit): {reversed_number_8:08b} ({reversed_number_8})")

# Example with 16-bit size
bit_size_16 = 16
number_16 = 29  # Binary: 0000000000011101
reversed_number_16 = reverse_bits(number_16, bit_size_16)
print(f"Reversed bits (16-bit): {reversed_number_16:016b} ({reversed_number_16})")

```

```
eversed bits (8-bit): 10110000 (176)
eversed bits (16-bit): 1011100000000000 (47104)


..Program finished with exit code 0
ress ENTER to exit console.
```

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



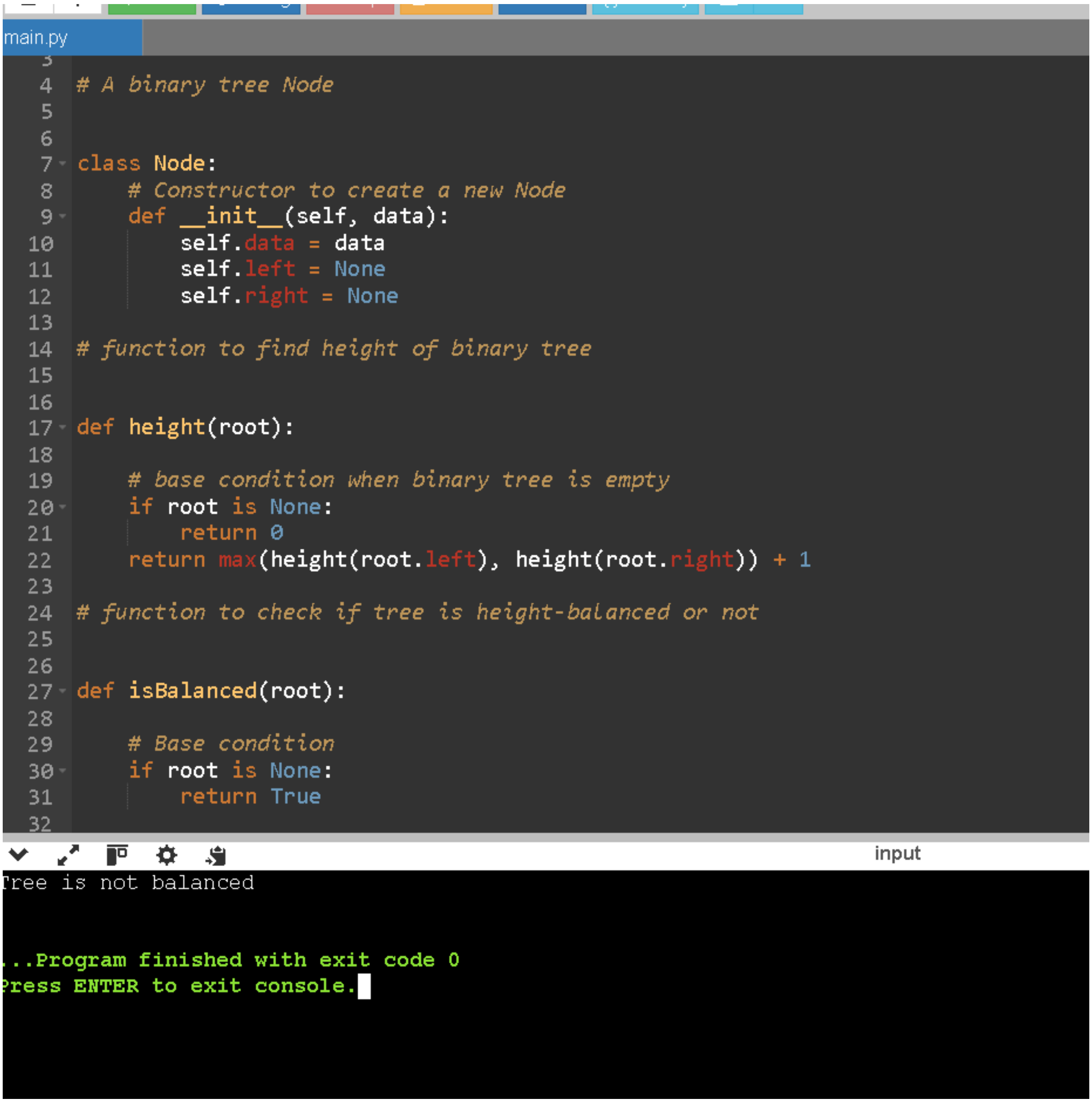
```python
from typing import List
from queue import Queue

# structure of the tree node


class Node:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

# function to convert the array to BST
# and return the root of the created tree


def sortedArrayToBST(nums: List[int]) -> Node:
    # if the array is empty return None
    if not nums:
        return None

    n = len(nums)
    mid = n // 2
    root = Node(nums[mid])
    # initializing queue
    q = Queue()
    # push the root and its indices to the queue
    q.put((root, (0, mid-1)))
    q.put((root, (mid+1, n-1)))
```

```
4 2 1 3 6 5 7

...Program finished with exit code 0
Press ENTER to exit console.
```

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

```
"""
Python3 program to check if a tree is height-balanced
"""
# A binary tree Node


class Node:
	# Constructor to create a new Node
	def __init__(self, data):
		self.data = data
		self.left = None
		self.right = None

# function to find height of binary tree


def height(root):

	# base condition when binary tree is empty
	if root is None:
		return 0
	return max(height(root.left), height(root.right)) + 1

# function to check if tree is height-balanced or not


def isBalanced(root):

	# Base condition
	if root is None:
		return True

	# for left and right subtree height
```

```
    lh = height(root.left)
    rh = height(root.right)

    # allowed values for (lh - rh) are 1, -1, 0
    if (abs(lh - rh) <= 1) and isBalanced(
            root.left) is True and isBalanced(root.right) is True:
        return True

    # if we reach here means tree is not
    # height-balanced tree
    return False



# Driver function to test the above function
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.left.left.left = Node(8)
if isBalanced(root):
    print("Tree is balanced")
else:
    print("Tree is not balanced")

# This code is contributed by Shweta Singh
```



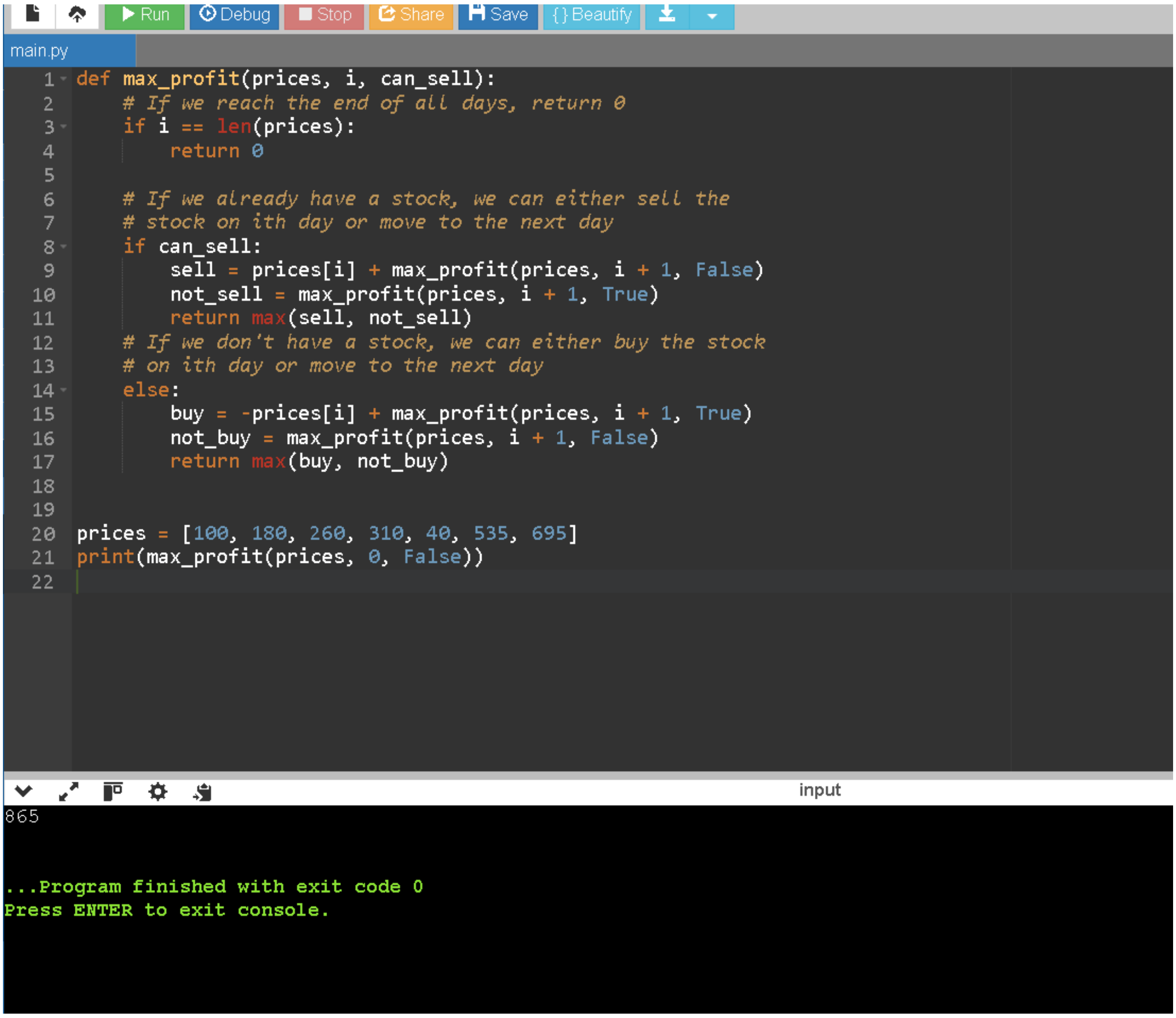
```python
# A binary tree Node


class Node:
    # Constructor to create a new Node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# function to find height of binary tree


def height(root):

    # base condition when binary tree is empty
    if root is None:
        return 0
    return max(height(root.left), height(root.right)) + 1

# function to check if tree is height-balanced or not


def isBalanced(root):

    # Base condition
    if root is None:
        return True

```

```
Tree is not balanced


...Program finished with exit code 0
Press ENTER to exit console.
```

5.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.



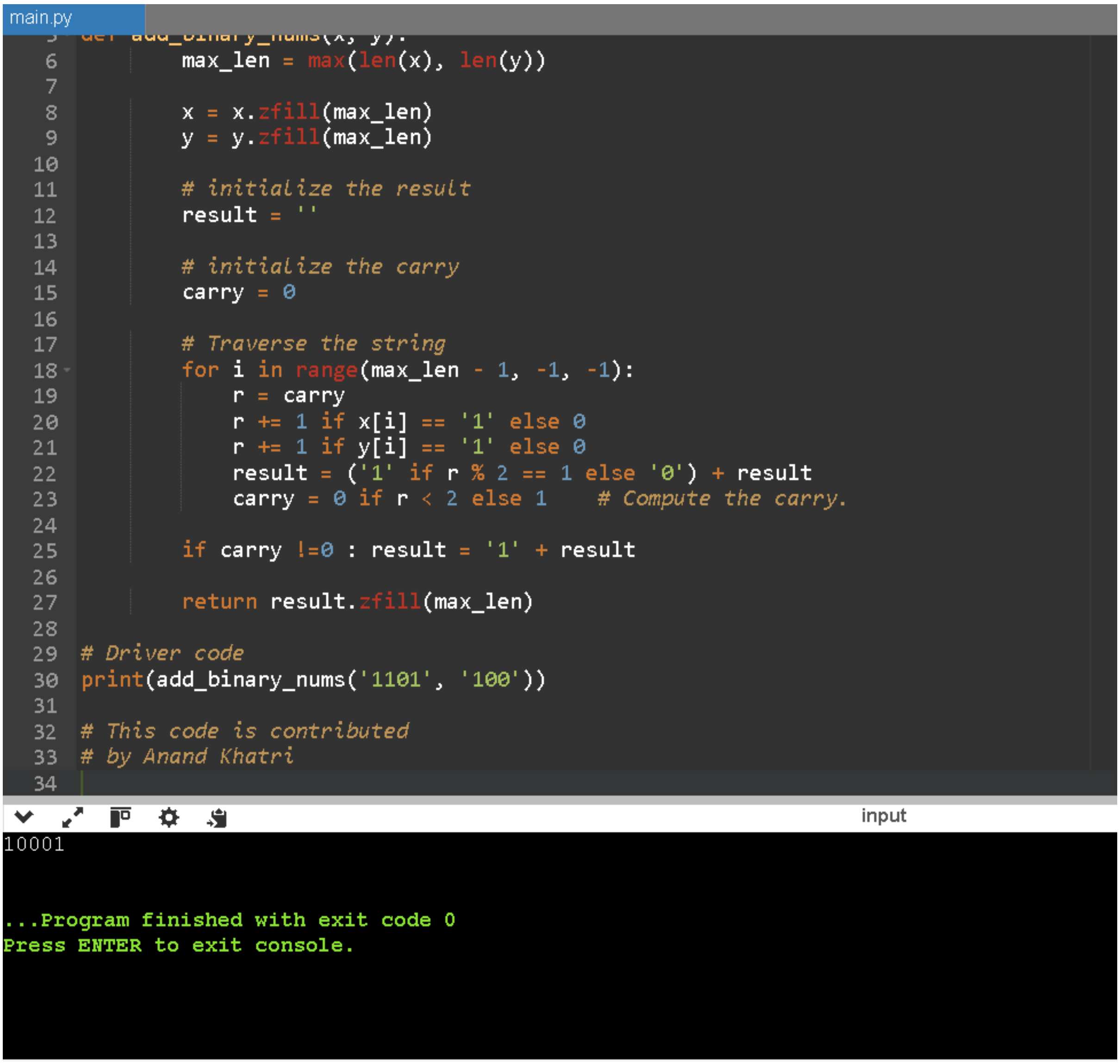
```
def max_profit(prices, i, can_sell):
    # If we reach the end of all days, return 0
    if i == len(prices):
        return 0

    # If we already have a stock, we can either sell the
    # stock on ith day or move to the next day
    if can_sell:
        sell = prices[i] + max_profit(prices, i + 1, False)
        not_sell = max_profit(prices, i + 1, True)
        return max(sell, not_sell)
    # If we don't have a stock, we can either buy the stock
    # on ith day or move to the next day
    else:
        buy = -prices[i] + max_profit(prices, i + 1, True)
        not_buy = max_profit(prices, i + 1, False)
        return max(buy, not_buy)


prices = [100, 180, 260, 310, 40, 535, 695]
print(max_profit(prices, 0, False))

```

```
865


...Program finished with exit code 0
Press ENTER to exit console.
```

ASSIGNMENT 2

SAVEETHA SCHOOL OF ENGINEERING

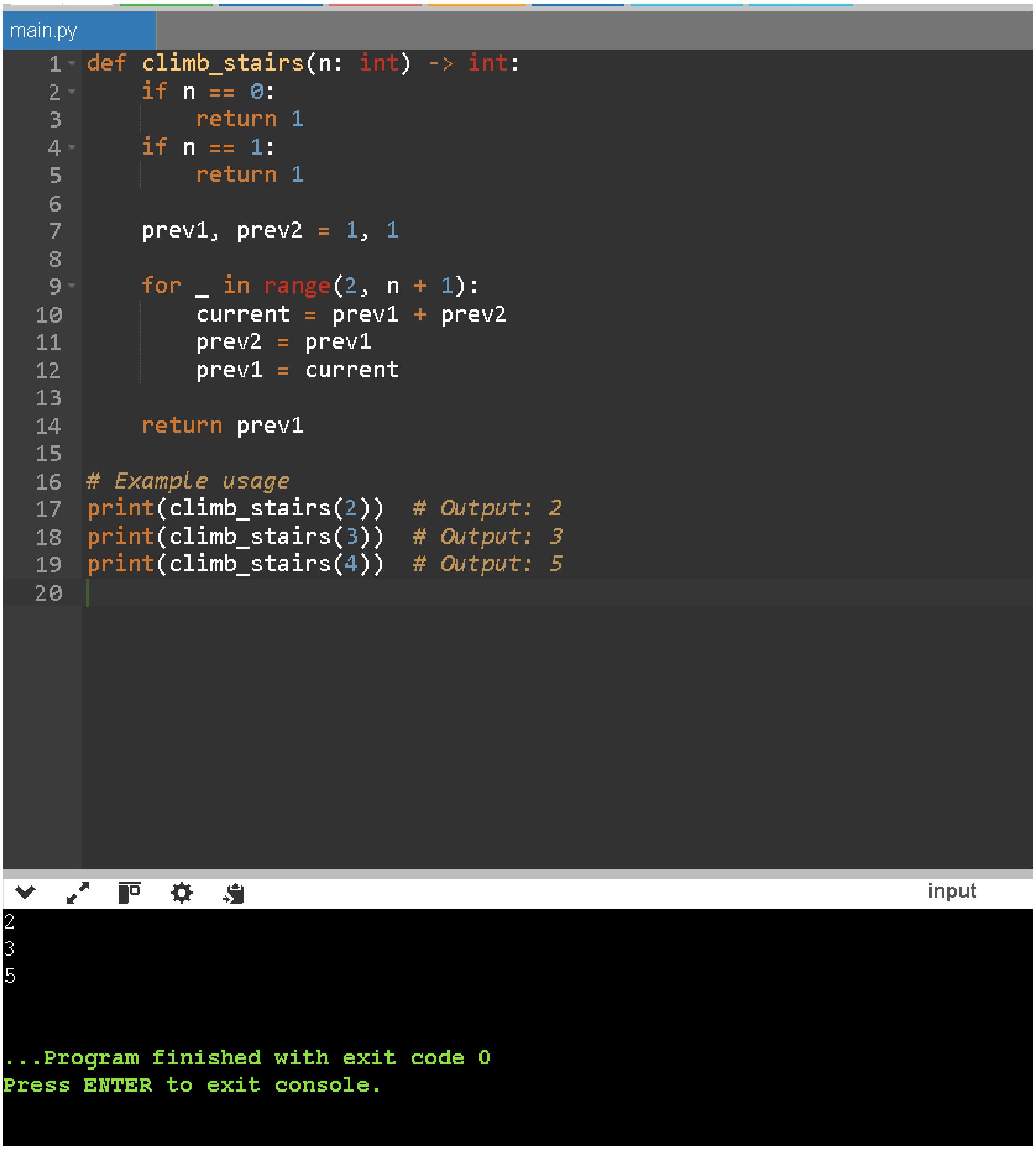
1.Given two binary strings a and b, return their sum as a binary string.



```python
def add_binary_nums(x, y):
        max_len = max(len(x), len(y))

        x = x.zfill(max_len)
        y = y.zfill(max_len)

        # initialize the result
        result = ''

        # initialize the carry
        carry = 0

        # Traverse the string
        for i in range(max_len - 1, -1, -1):
            r = carry
            r += 1 if x[i] == '1' else 0
            r += 1 if y[i] == '1' else 0
            result = ('1' if r % 2 == 1 else '0') + result
            carry = 0 if r < 2 else 1     # Compute the carry.

        if carry !=0 : result = '1' + result

        return result.zfill(max_len)

# Driver code
print(add_binary_nums('1101', '100'))

# This code is contributed
# by Anand Khatri
```

```
10001


...Program finished with exit code 0
Press ENTER to exit console.
```

2.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?



```python
def climb_stairs(n: int) -> int:
    if n == 0:
        return 1
    if n == 1:
        return 1

    prev1, prev2 = 1, 1

    for _ in range(2, n + 1):
        current = prev1 + prev2
        prev2 = prev1
        prev1 = current

    return prev1

# Example usage
print(climb_stairs(2))  # Output: 2
print(climb_stairs(3))  # Output: 3
print(climb_stairs(4))  # Output: 5
```

```
2
3
5


...Program finished with exit code 0
Press ENTER to exit console.
```

3. 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"]

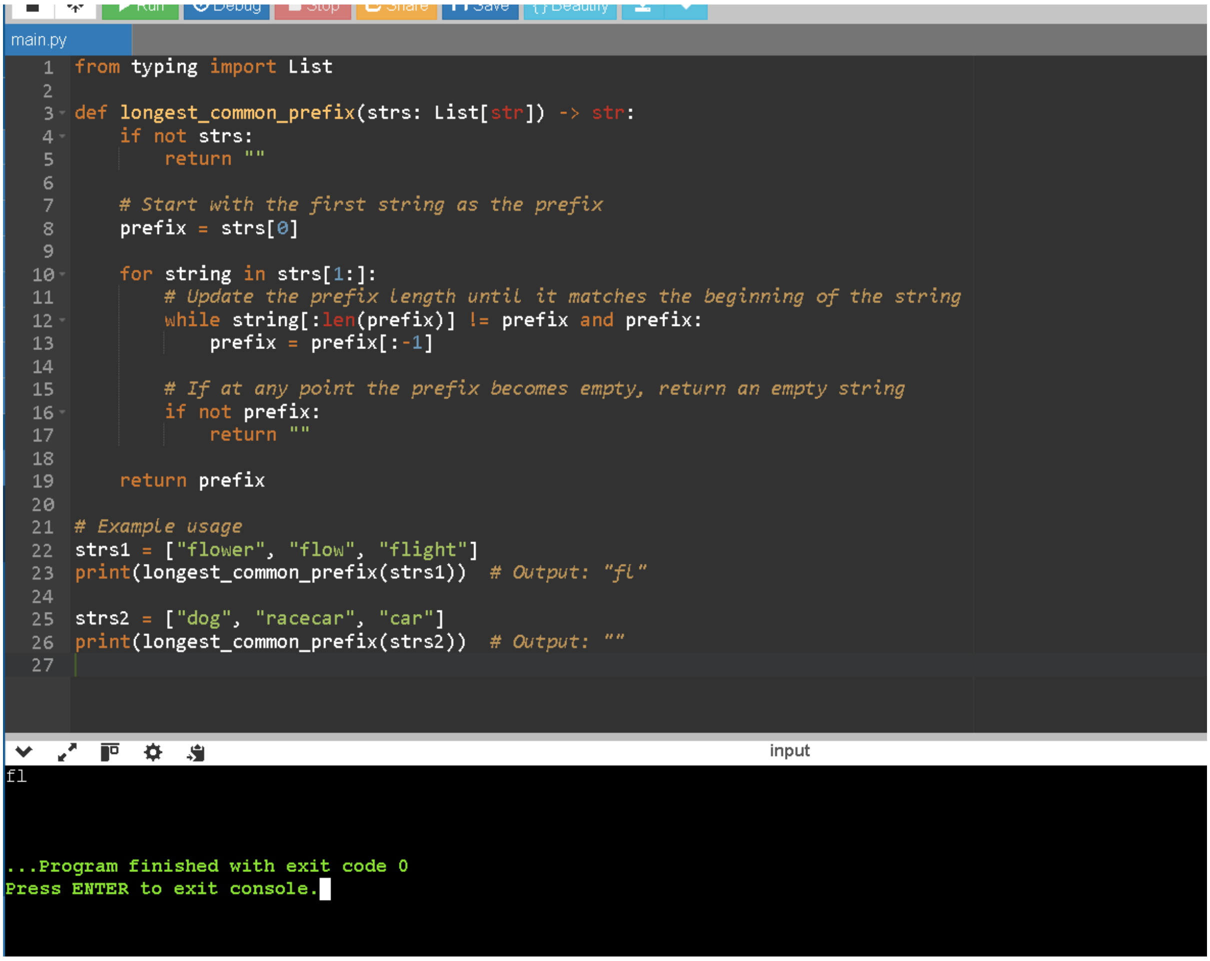
Output: "fl"

Example 2:

Input: strs = ["dog", "racecar", "car"]

Output: ""

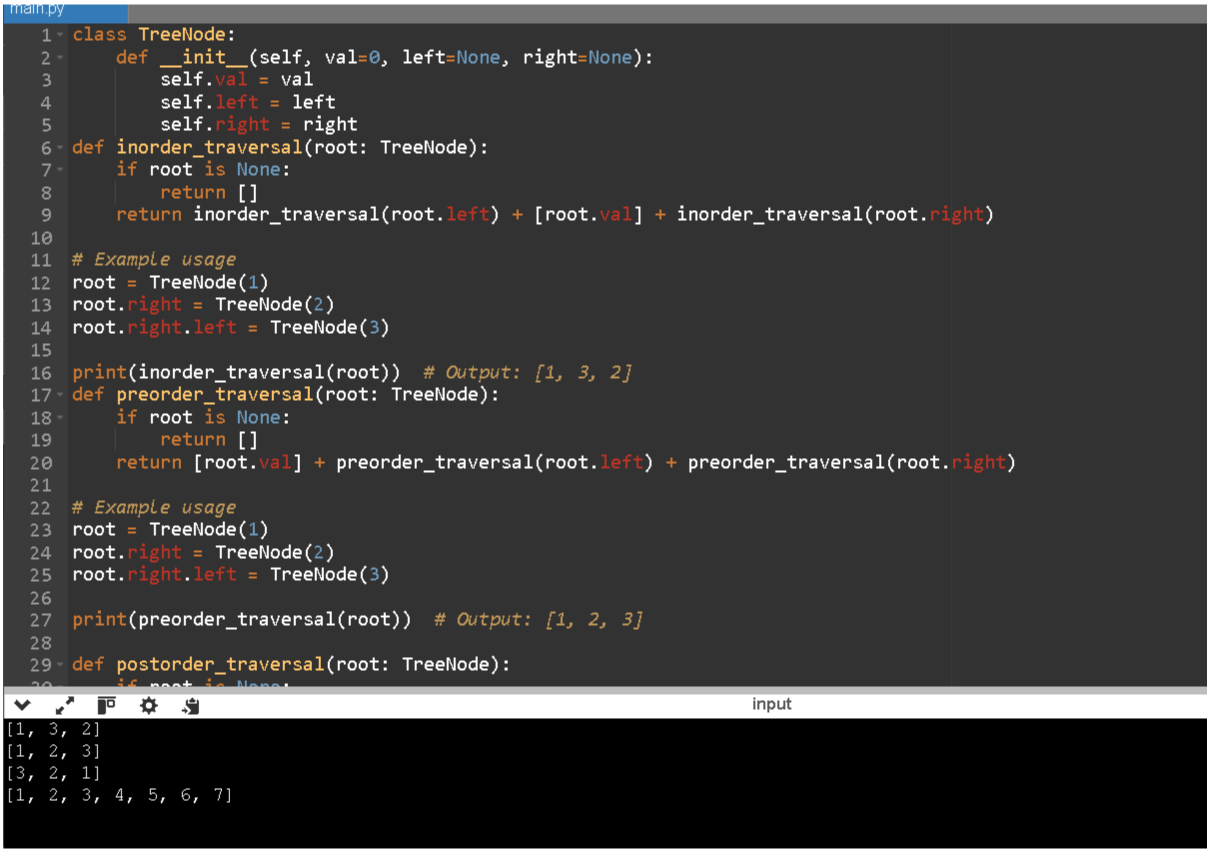
Explanation: There is no common prefix among the input strings



```python
from typing import List

def longest_common_prefix(strs: List[str]) -> str:
    if not strs:
        return ""

    # Start with the first string as the prefix
    prefix = strs[0]

    for string in strs[1:]:
        # Update the prefix length until it matches the beginning of the string
        while string[:len(prefix)] != prefix and prefix:
            prefix = prefix[:-1]

        # If at any point the prefix becomes empty, return an empty string
        if not prefix:
            return ""

    return prefix

# Example usage
strs1 = ["flower", "flow", "flight"]
print(longest_common_prefix(strs1))  # Output: "fl"

strs2 = ["dog", "racecar", "car"]
print(longest_common_prefix(strs2))  # Output: ""
```

```
fl


...Program finished with exit code 0
Press ENTER to exit console.
```

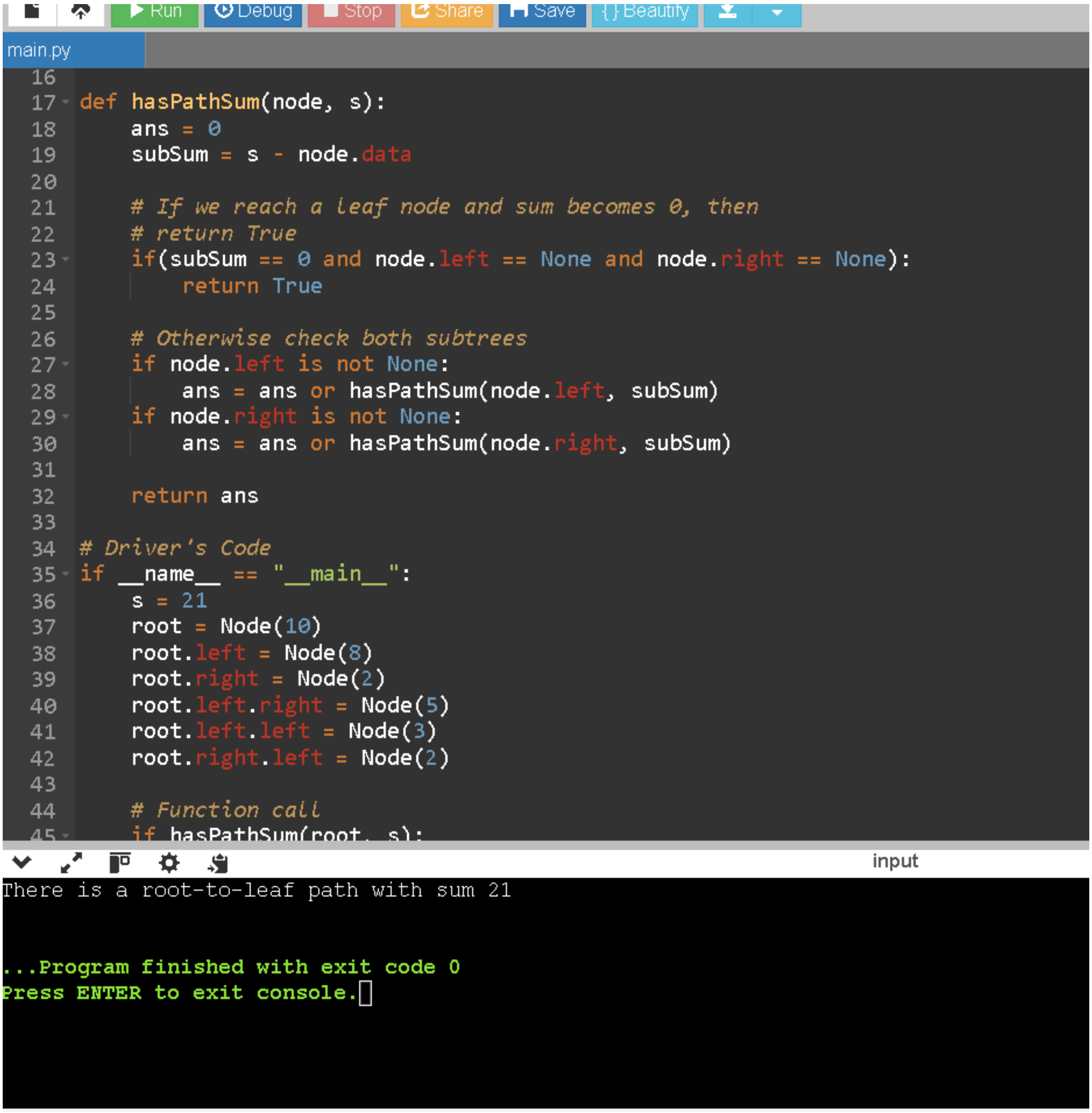
4. .Binary tree traversal



```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
def inorder_traversal(root: TreeNode):
    if root is None:
        return []
    return inorder_traversal(root.left) + [root.val] + inorder_traversal(root.right)

# Example usage
root = TreeNode(1)
root.right = TreeNode(2)
root.right.left = TreeNode(3)

print(inorder_traversal(root))  # Output: [1, 3, 2]
def preorder_traversal(root: TreeNode):
    if root is None:
        return []
    return [root.val] + preorder_traversal(root.left) + preorder_traversal(root.right)

# Example usage
root = TreeNode(1)
root.right = TreeNode(2)
root.right.left = TreeNode(3)

print(preorder_traversal(root))  # Output: [1, 2, 3]

def postorder_traversal(root: TreeNode):
    if root is None:
```

```
[1, 3, 2]
[1, 2, 3]
[3, 2, 1]
[1, 2, 3, 4, 5, 6, 7]
```

5. 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



```python

def hasPathSum(node, s):
    ans = 0
    subSum = s - node.data

    # If we reach a leaf node and sum becomes 0, then
    # return True
    if(subSum == 0 and node.left == None and node.right == None):
        return True

    # Otherwise check both subtrees
    if node.left is not None:
        ans = ans or hasPathSum(node.left, subSum)
    if node.right is not None:
        ans = ans or hasPathSum(node.right, subSum)

    return ans

# Driver's Code
if __name__ == "__main__":
    s = 21
    root = Node(10)
    root.left = Node(8)
    root.right = Node(2)
    root.left.right = Node(5)
    root.left.left = Node(3)
    root.right.left = Node(2)

    # Function call
    if hasPathSum(root, s):
```

```
There is a root-to-leaf path with sum 21


...Program finished with exit code 0
Press ENTER to exit console.
```