

Assignment 1

Saveetha school of engineering

1.write a python program to interchange first and last elements in a list

```
main.py
1  # Python3 program to swap first
2  # and last element of a list
3
4  # Swap function
5  def swapList(newList):
6      size = len(newList)
7
8      # Swapping
9      temp = newList[0]
10     newList[0] = newList[size - 1]
11     newList[size - 1] = temp
12
13     return newList
14
15 # Driver code
16 newList = [12, 35, 9, 56, 24]
17
18 print(swapList(newList))
19
```

input

24, 35, 9, 56, 12]

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

2. Python program to swap 2 elements in a list

```
# at given positions

# Swap function
def swapPositions(list, pos1, pos2):

    list[pos1], list[pos2] = list[pos2], list[pos1]
    return list

# Driver function
List = [23, 65, 19, 90]
pos1, pos2 = 1, 3

print(swapPositions(List, pos1-1, pos2-1))
```

Output:

```
[19, 65, 23, 90]
```

3. Reverse words in a given string in python

Python3

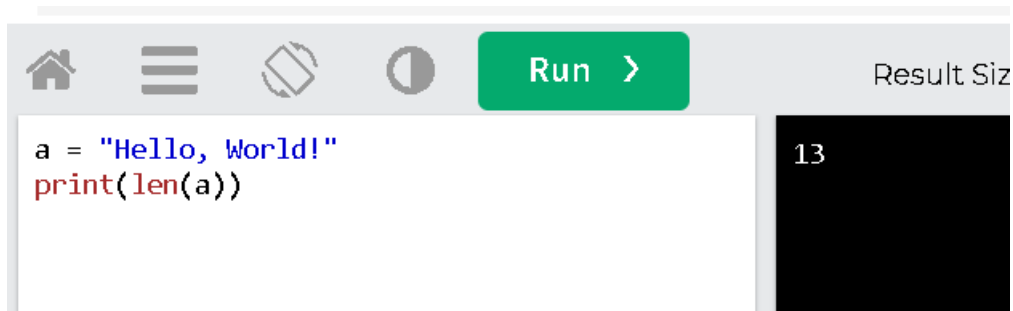
```
# Python code
# To reverse words in a given string

# input string
string = "geeks quiz practice code"
# reversing words in a given string
s = string.split()[::-1]
l = []
for i in s:
    # appending reversed words to l
    l.append(i)
# printing reverse words
print(" ".join(l))
```

Output

code practice quiz geeks

4. Find the length of a string in python



The image shows a Python IDE interface. At the top, there is a toolbar with icons for home, menu, copy, and a green 'Run' button with a right arrow. To the right of the 'Run' button, the text 'Result Size' is visible. Below the toolbar, the code editor contains two lines of Python code: `a = "Hello, World!"` and `print(len(a))`. To the right of the code editor, a black output box displays the number '13'.

```
a = "Hello, World!"  
print(len(a))
```

13

5. Python program to find the maximum and minimum k elements in a tuple

```
# initializing tuple
test_tup = (5, 20, 3, 7, 6, 8)

# printing original tuple
print("The original tuple is : " + str(test_tup))

# initializing K
K = 2

# Maximum and Minimum K elements in Tuple
# Using sorted() + loop
res = []
test_tup = list(sorted(test_tup))

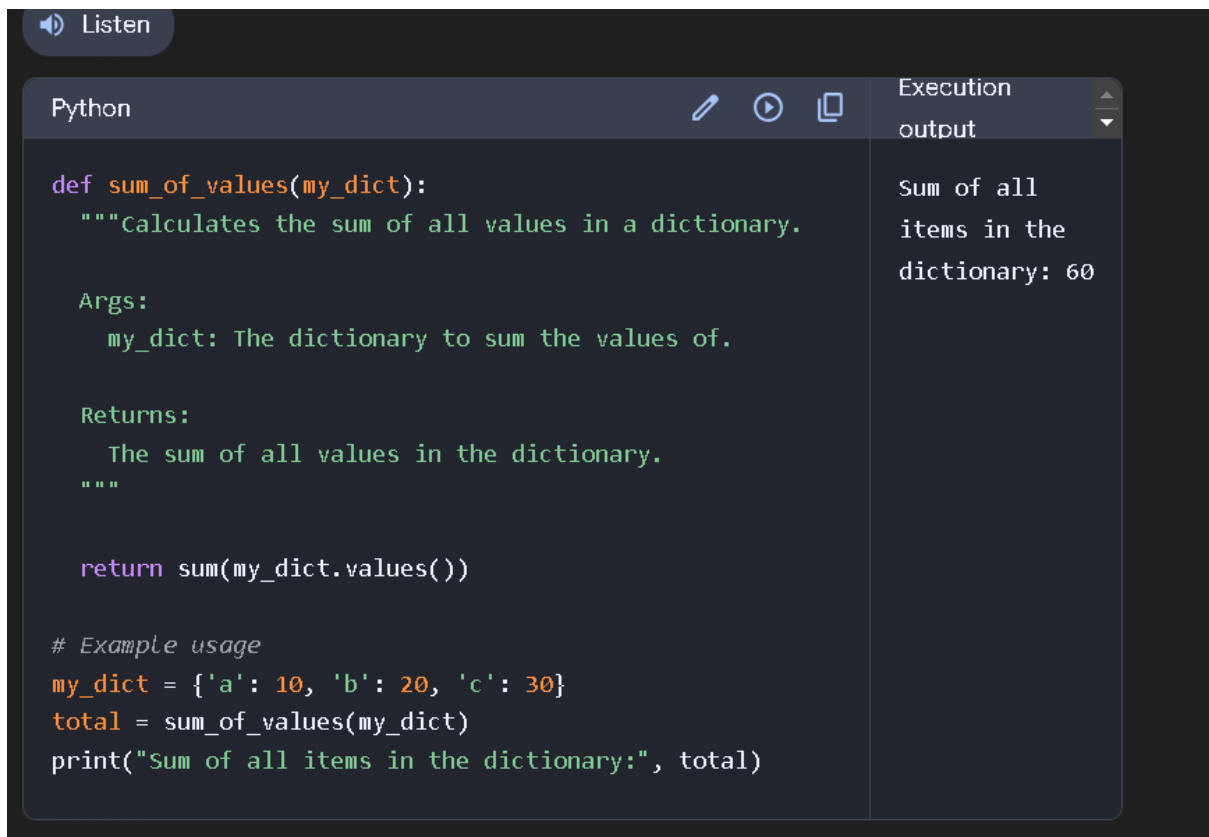
for idx, val in enumerate(test_tup):
    if idx < K or idx >= len(test_tup) - K:
        res.append(val)
res = tuple(res)

# printing result
print("The extracted values : " + str(res))
```

Output

```
The original tuple is : (5, 20, 3, 7, 6, 8)
The extracted values : (3, 5, 8, 20)
```

6. Python program to find sum of all item in a dictionary python



The image shows a Python IDE interface with a dark theme. At the top left, there is a 'Listen' button with a speaker icon. The main editor area is titled 'Python' and contains the following code:

```
def sum_of_values(my_dict):  
    """Calculates the sum of all values in a dictionary.  
  
    Args:  
        my_dict: The dictionary to sum the values of.  
  
    Returns:  
        The sum of all values in the dictionary.  
    """  
  
    return sum(my_dict.values())  
  
# Example usage  
my_dict = {'a': 10, 'b': 20, 'c': 30}  
total = sum_of_values(my_dict)  
print("Sum of all items in the dictionary:", total)
```

On the right side of the IDE, there is a panel titled 'Execution output'. It displays the result of running the code:

```
Sum of all  
items in the  
dictionary: 60
```

7. Python program to find sum of all item in a dictionary python

Python3

```
def Remove(initial_set):  
    while initial_set:  
        initial_set.pop()  
        print(initial_set)  
  
initial_set = set([12, 10, 13, 15, 8, 9])  
Remove(initial_set)
```

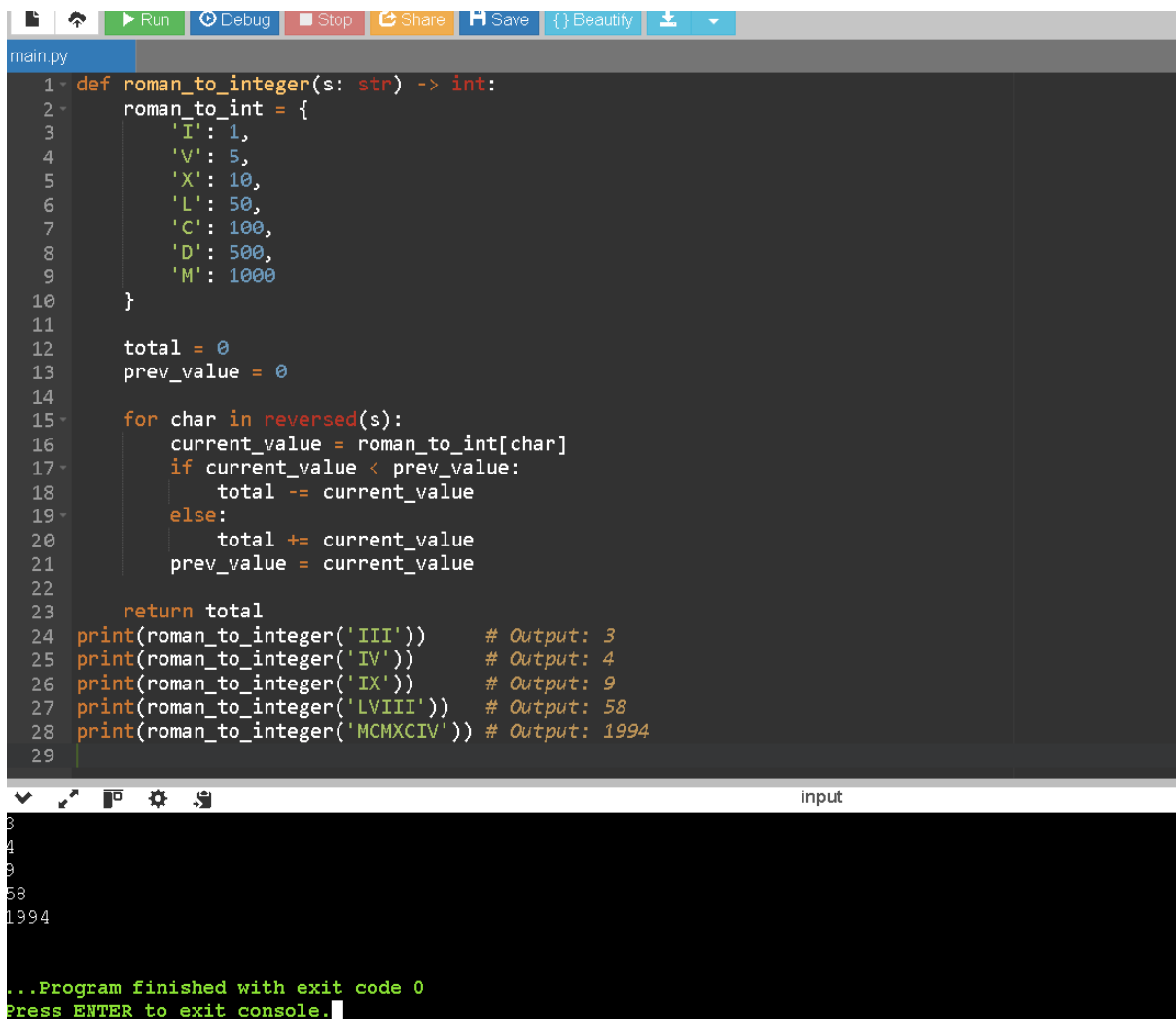
Output:

```
{9, 10, 12, 13, 15}  
{10, 12, 13, 15}  
{12, 13, 15}  
{13, 15}  
{15}  
set()
```

ASSIGNMENT 2

SAVEETHA SCHOOL OF ENGINEERING

1. Converting Roman Numbers to integers



```
main.py
1 def roman_to_integer(s: str) -> int:
2     roman_to_int = {
3         'I': 1,
4         'V': 5,
5         'X': 10,
6         'L': 50,
7         'C': 100,
8         'D': 500,
9         'M': 1000
10    }
11
12    total = 0
13    prev_value = 0
14
15    for char in reversed(s):
16        current_value = roman_to_int[char]
17        if current_value < prev_value:
18            total -= current_value
19        else:
20            total += current_value
21        prev_value = current_value
22
23    return total
24 print(roman_to_integer('III'))      # Output: 3
25 print(roman_to_integer('IV'))     # Output: 4
26 print(roman_to_integer('IX'))     # Output: 9
27 print(roman_to_integer('LVIII'))  # Output: 58
28 print(roman_to_integer('MCMXCIV')) # Output: 1994
29
```

input

3
4
9
58
1994

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

2.Bit Reserving

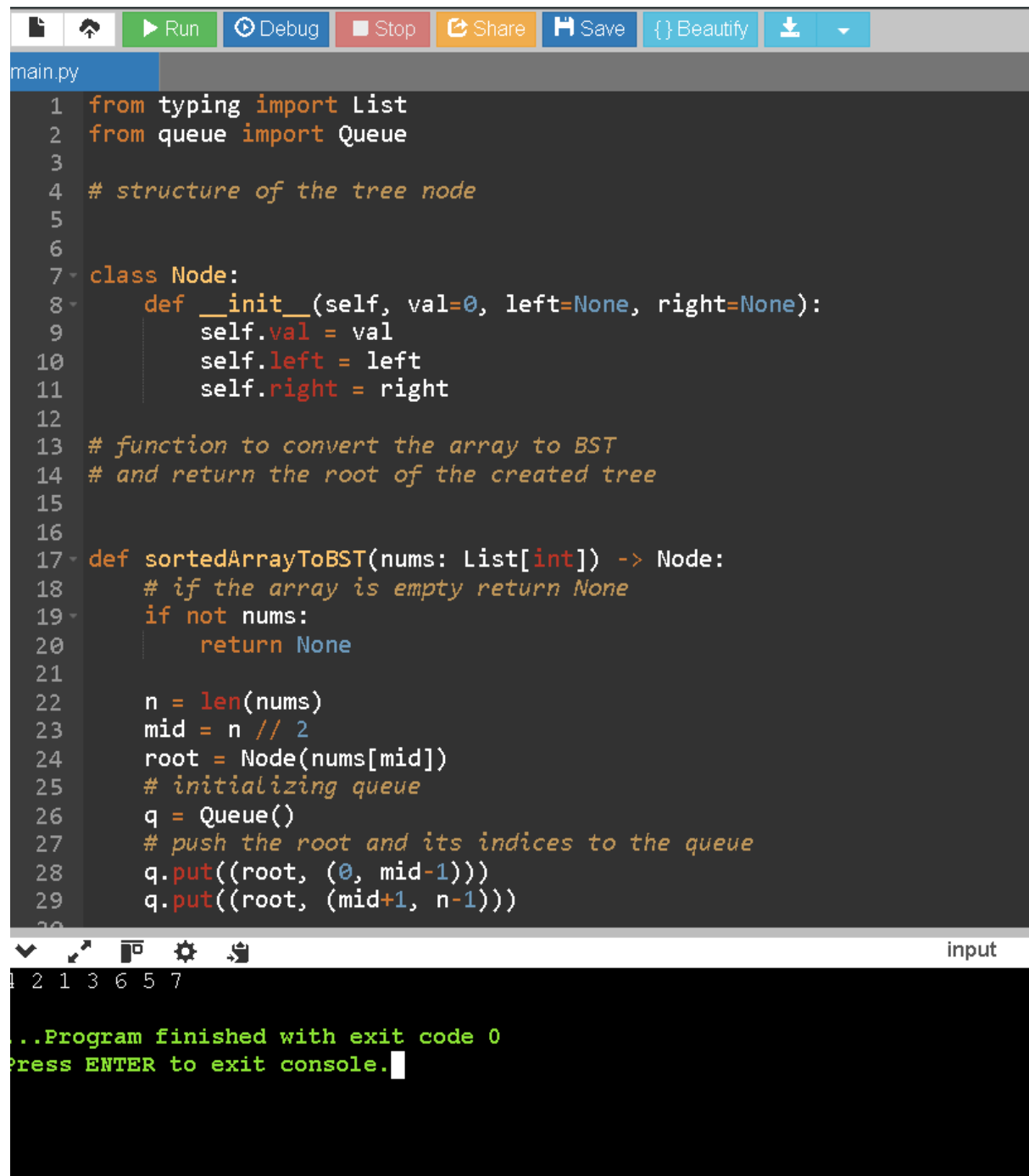
```
main.py
1 def reverse_bits(n: int, bit_size: int) -> int:
2     reversed_n = 0
3     for _ in range(bit_size):
4         reversed_n = (reversed_n << 1) | (n & 1)
5         n >>= 1
6     return reversed_n
7
8 # Example with 8-bit size
9 bit_size_8 = 8
10 number_8 = 13 # Binary: 00001101
11 reversed_number_8 = reverse_bits(number_8, bit_size_8)
12 print(f"Reversed bits (8-bit): {reversed_number_8:08b} ({reversed_number_8})")
13
14 # Example with 16-bit size
15 bit_size_16 = 16
16 number_16 = 29 # Binary: 0000000000011101
17 reversed_number_16 = reverse_bits(number_16, bit_size_16)
18 print(f"Reversed bits (16-bit): {reversed_number_16:016b} ({reversed_number_16})")
19
```

input

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

..Program finished with exit code 0
Press 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.



```
1 from typing import List
2 from queue import Queue
3
4 # structure of the tree node
5
6
7 class Node:
8     def __init__(self, val=0, left=None, right=None):
9         self.val = val
10        self.left = left
11        self.right = right
12
13 # function to convert the array to BST
14 # and return the root of the created tree
15
16
17 def sortedArrayToBST(nums: List[int]) -> Node:
18     # if the array is empty return None
19     if not nums:
20         return None
21
22     n = len(nums)
23     mid = n // 2
24     root = Node(nums[mid])
25     # initializing queue
26     q = Queue()
27     # push the root and its indices to the queue
28     q.put((root, (0, mid-1)))
29     q.put((root, (mid+1, n-1)))
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

main.py

Run Debug Stop Share Save {} Beautify

input

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
```

```
main.py
3
4 # A binary tree Node
5
6
7 class Node:
8     # Constructor to create a new Node
9     def __init__(self, data):
10         self.data = data
11         self.left = None
12         self.right = None
13
14 # function to find height of binary tree
15
16
17 def height(root):
18     # base condition when binary tree is empty
19     if root is None:
20         return 0
21     return max(height(root.left), height(root.right)) + 1
22
23
24 # function to check if tree is height-balanced or not
25
26
27 def isBalanced(root):
28     # Base condition
29     if root is None:
30         return True
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

input

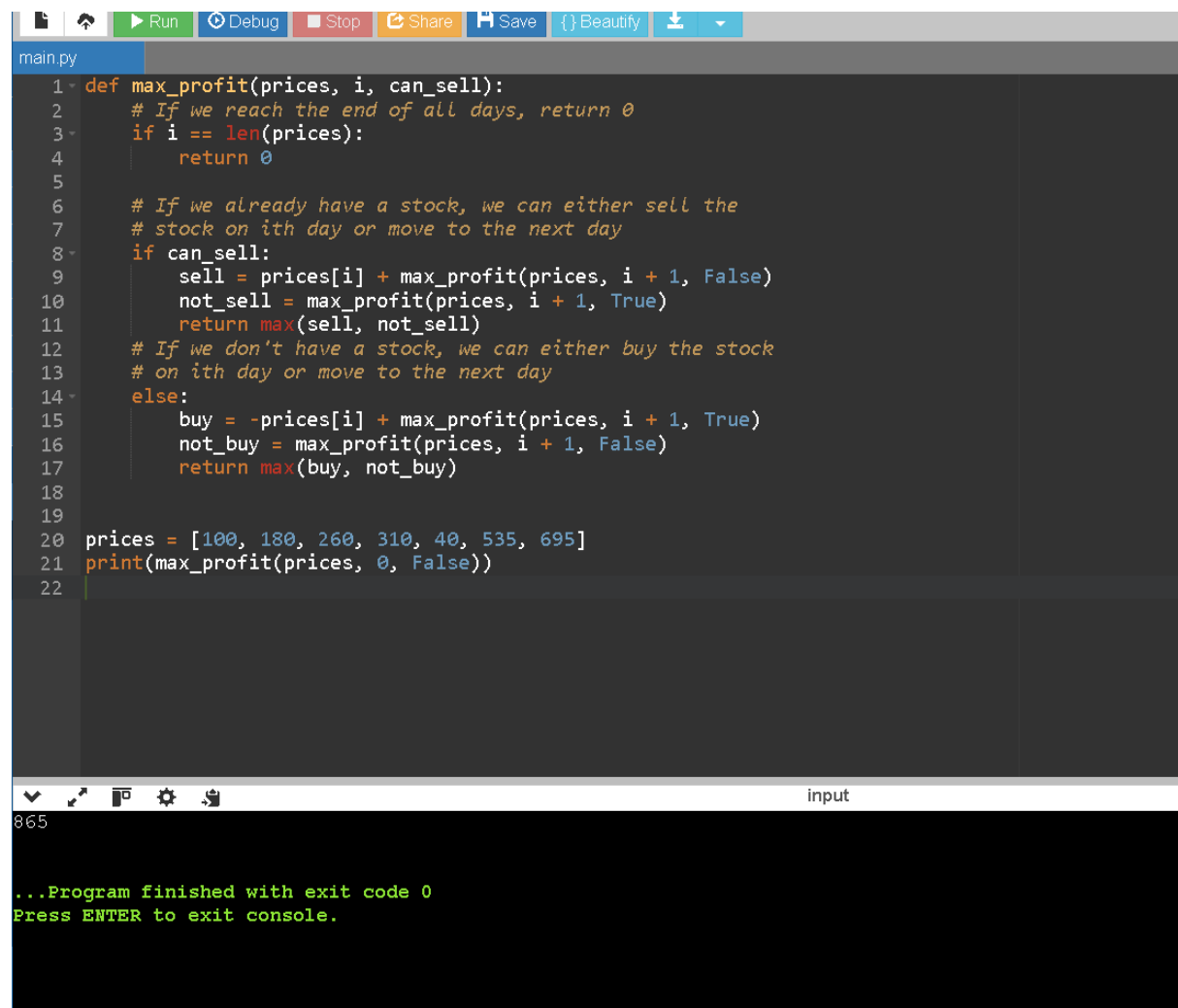
Tree is not balanced

...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 i th 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.



```
main.py
1 def max_profit(prices, i, can_sell):
2     # If we reach the end of all days, return 0
3     if i == len(prices):
4         return 0
5
6     # If we already have a stock, we can either sell the
7     # stock on ith day or move to the next day
8     if can_sell:
9         sell = prices[i] + max_profit(prices, i + 1, False)
10        not_sell = max_profit(prices, i + 1, True)
11        return max(sell, not_sell)
12    # If we don't have a stock, we can either buy the stock
13    # on ith day or move to the next day
14    else:
15        buy = -prices[i] + max_profit(prices, i + 1, True)
16        not_buy = max_profit(prices, i + 1, False)
17        return max(buy, not_buy)
18
19
20 prices = [100, 180, 260, 310, 40, 535, 695]
21 print(max_profit(prices, 0, False))
22
```

865

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

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

```
main.py
1 def add_binary_nums(x, y):
2     max_len = max(len(x), len(y))
3
4     x = x.zfill(max_len)
5     y = y.zfill(max_len)
6
7     # initialize the result
8     result = ''
9
10    # initialize the carry
11    carry = 0
12
13    # Traverse the string
14    for i in range(max_len - 1, -1, -1):
15        r = carry
16        r += 1 if x[i] == '1' else 0
17        r += 1 if y[i] == '1' else 0
18        result = ('1' if r % 2 == 1 else '0') + result
19        carry = 0 if r < 2 else 1    # Compute the carry.
20
21    if carry != 0 : result = '1' + result
22
23    return result.zfill(max_len)
24
25 # Driver code
26 print(add_binary_nums('1101', '100'))
27
28 # This code is contributed
29 # by Anand Khatni
30
31
32
33
34
```

input

10001

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

7. 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?

```
main.py
1 def climb_stairs(n: int) -> int:
2     if n == 0:
3         return 1
4     if n == 1:
5         return 1
6
7     prev1, prev2 = 1, 1
8
9     for _ in range(2, n + 1):
10        current = prev1 + prev2
11        prev2 = prev1
12        prev1 = current
13
14    return prev1
15
16 # Example usage
17 print(climb_stairs(2)) # Output: 2
18 print(climb_stairs(3)) # Output: 3
19 print(climb_stairs(4)) # Output: 5
20
```

input

```
2
3
5

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


8. 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"]

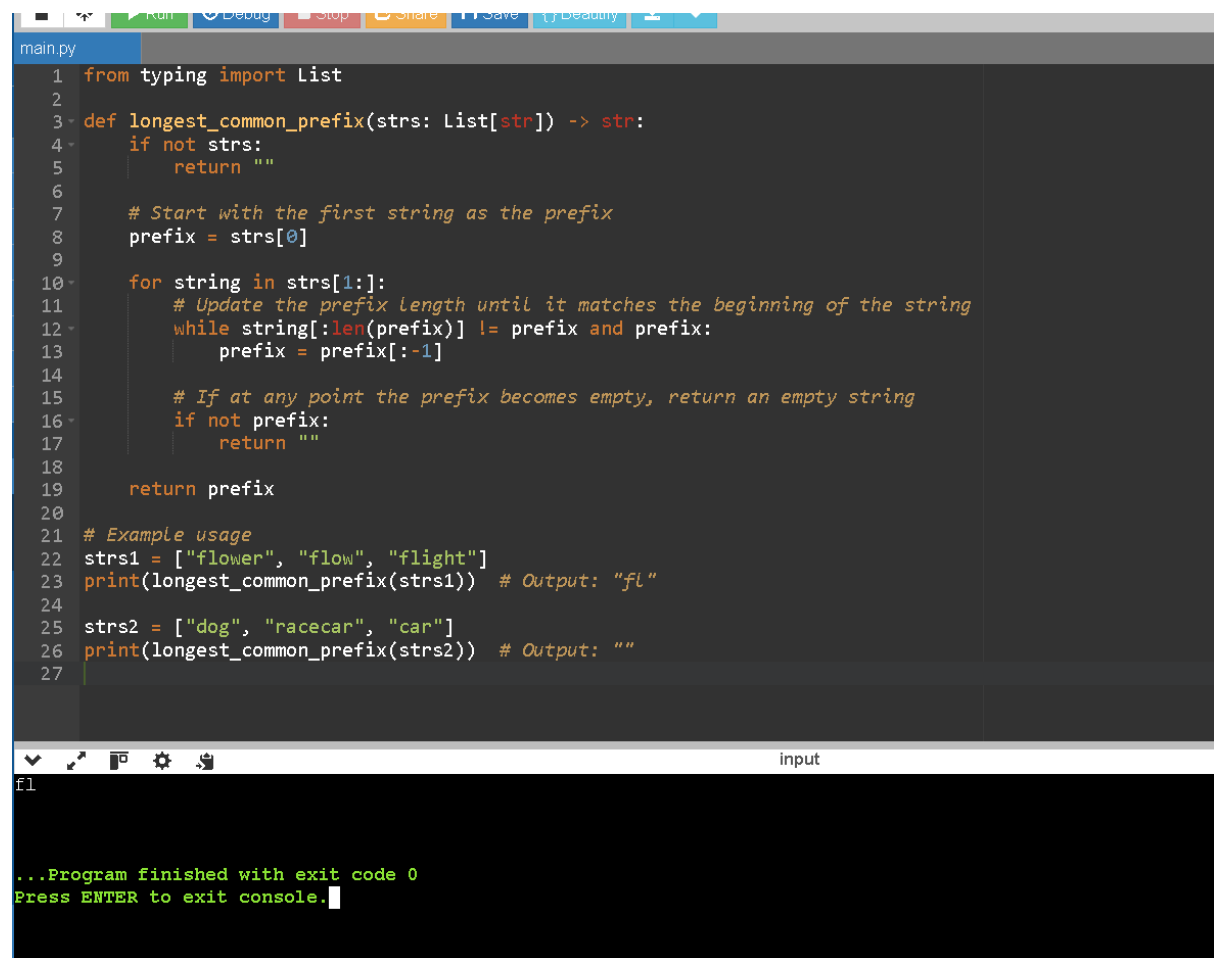
Output: "fl"

Example 2:

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

Output: ""

Explanation: There is no common prefix among the input strings



```
main.py
1 from typing import List
2
3 def longest_common_prefix(strs: List[str]) -> str:
4     if not strs:
5         return ""
6
7     # Start with the first string as the prefix
8     prefix = strs[0]
9
10    for string in strs[1:]:
11        # Update the prefix length until it matches the beginning of the string
12        while string[:len(prefix)] != prefix and prefix:
13            prefix = prefix[:-1]
14
15        # If at any point the prefix becomes empty, return an empty string
16        if not prefix:
17            return ""
18
19    return prefix
20
21 # Example usage
22 strs1 = ["flower", "flow", "flight"]
23 print(longest_common_prefix(strs1)) # Output: "fl"
24
25 strs2 = ["dog", "racecar", "car"]
26 print(longest_common_prefix(strs2)) # Output: ""
27
```

input

fl

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

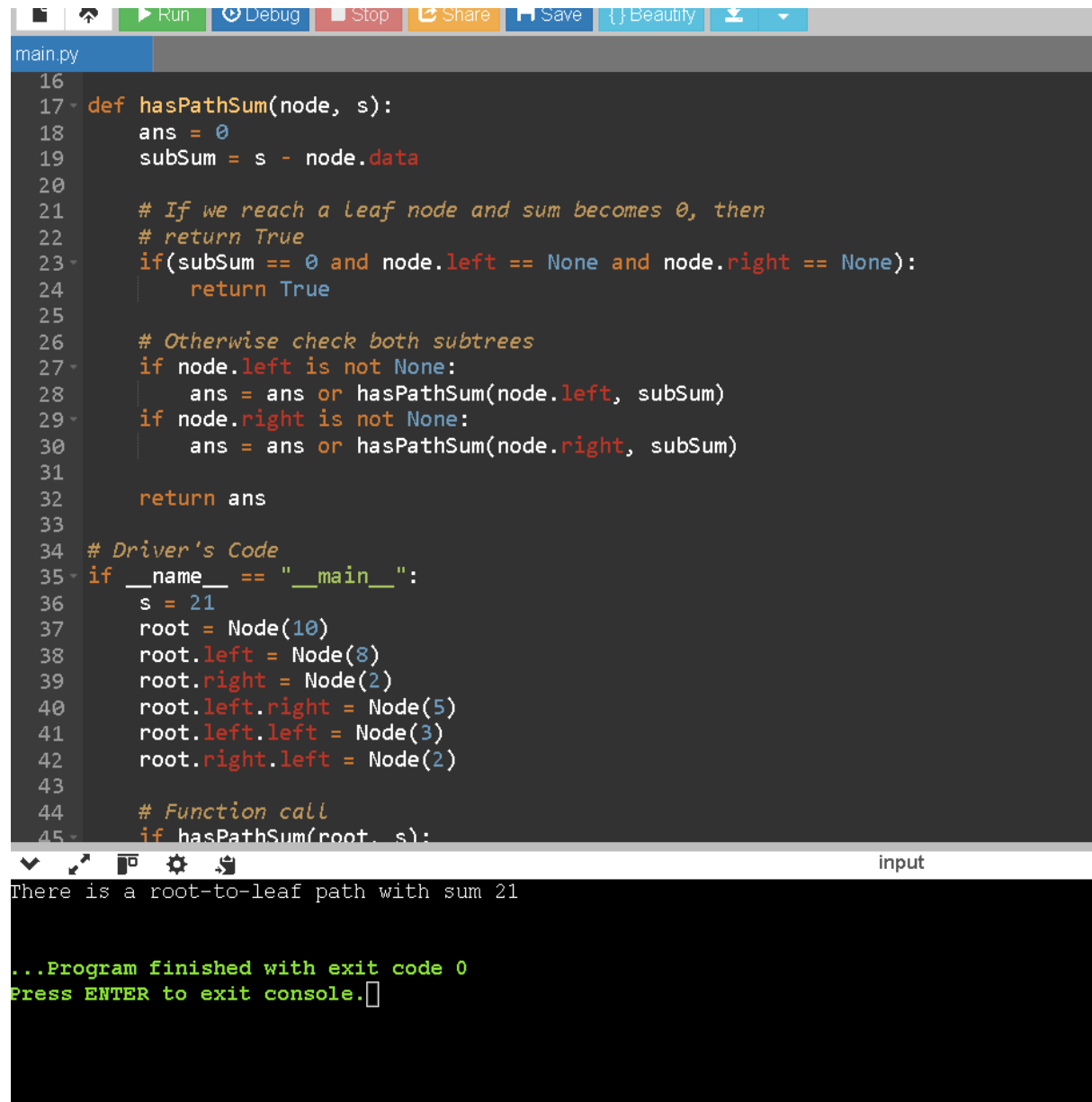
9.Binary tree traversal

```
main.py
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- def inorder_traversal(root: TreeNode):
7-     if root is None:
8-         return []
9-     return inorder_traversal(root.left) + [root.val] + inorder_traversal(root.right)
10
11 # Example usage
12 root = TreeNode(1)
13 root.right = TreeNode(2)
14 root.right.left = TreeNode(3)
15
16 print(inorder_traversal(root)) # Output: [1, 3, 2]
17 def preorder_traversal(root: TreeNode):
18     if root is None:
19         return []
20     return [root.val] + preorder_traversal(root.left) + preorder_traversal(root.right)
21
22 # Example usage
23 root = TreeNode(1)
24 root.right = TreeNode(2)
25 root.right.left = TreeNode(3)
26
27 print(preorder_traversal(root)) # Output: [1, 2, 3]
28
29 def postorder_traversal(root: TreeNode):
30     if root is None:
31         return []
32     return postorder_traversal(root.left) + postorder_traversal(root.right) + [root.val]
33
34 # Example usage
35 root = TreeNode(1)
36 root.right = TreeNode(2)
37 root.right.left = TreeNode(3)
38
39 print(postorder_traversal(root)) # Output: [3, 2, 1]
```

input

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

10. 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
16
17 def hasPathSum(node, s):
18     ans = 0
19     subSum = s - node.data
20
21     # If we reach a leaf node and sum becomes 0, then
22     # return True
23     if(subSum == 0 and node.left == None and node.right == None):
24         return True
25
26     # Otherwise check both subtrees
27     if node.left is not None:
28         ans = ans or hasPathSum(node.left, subSum)
29     if node.right is not None:
30         ans = ans or hasPathSum(node.right, subSum)
31
32     return ans
33
34 # Driver's Code
35 if __name__ == "__main__":
36     s = 21
37     root = Node(10)
38     root.left = Node(8)
39     root.right = Node(2)
40     root.left.right = Node(5)
41     root.left.left = Node(3)
42     root.right.left = Node(2)
43
44     # Function call
45     if hasPathSum(root, s):
```

input

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

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