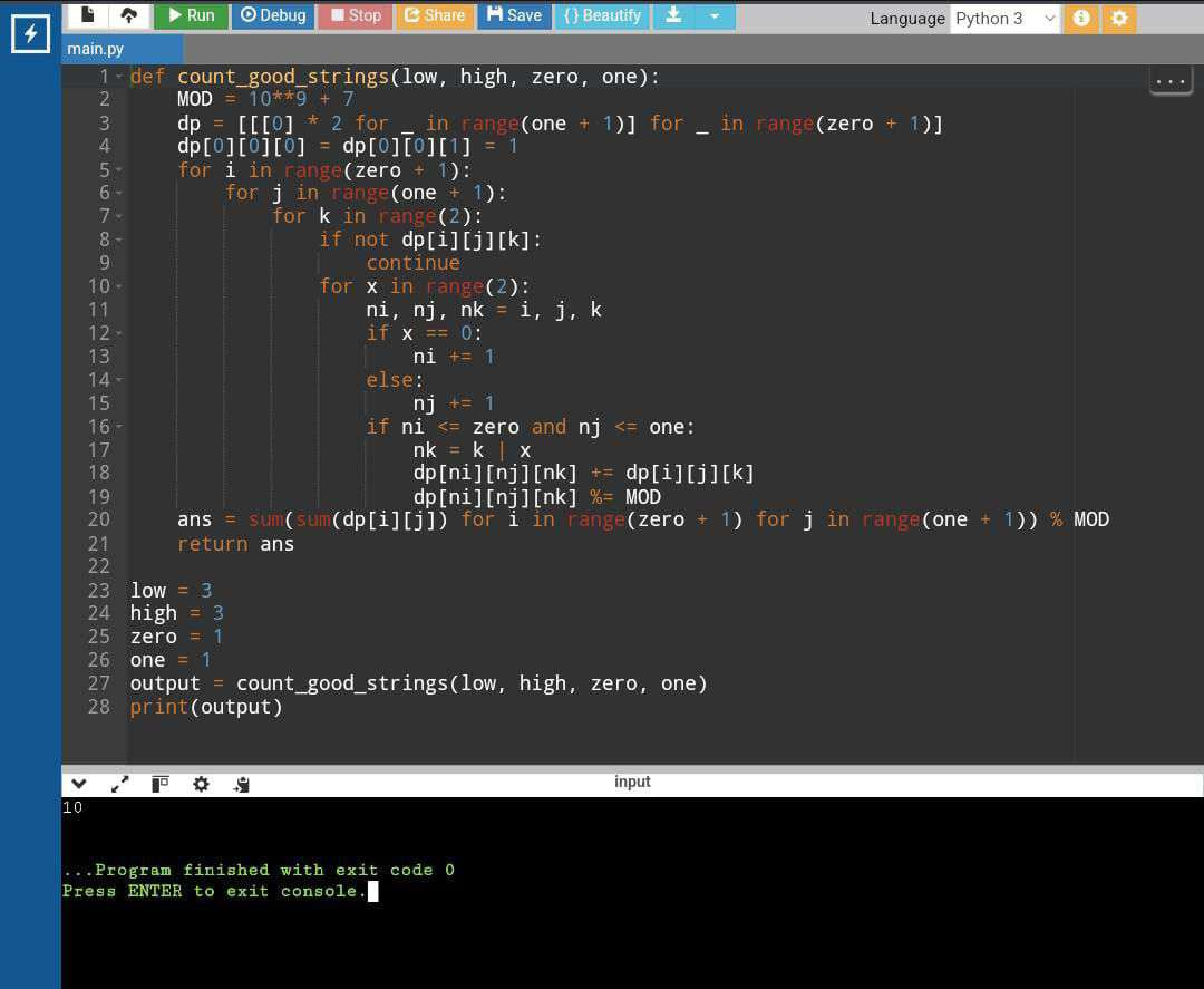
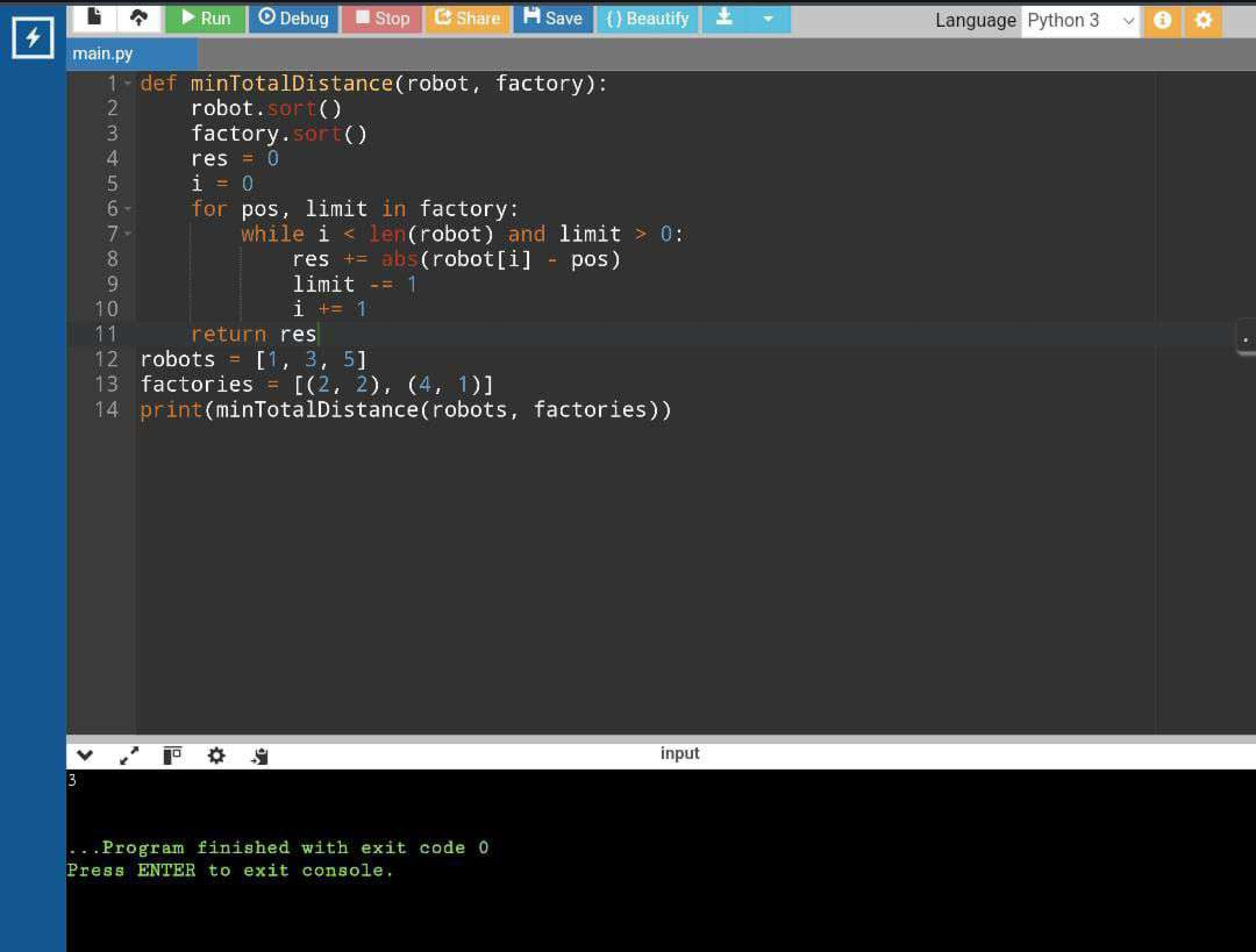


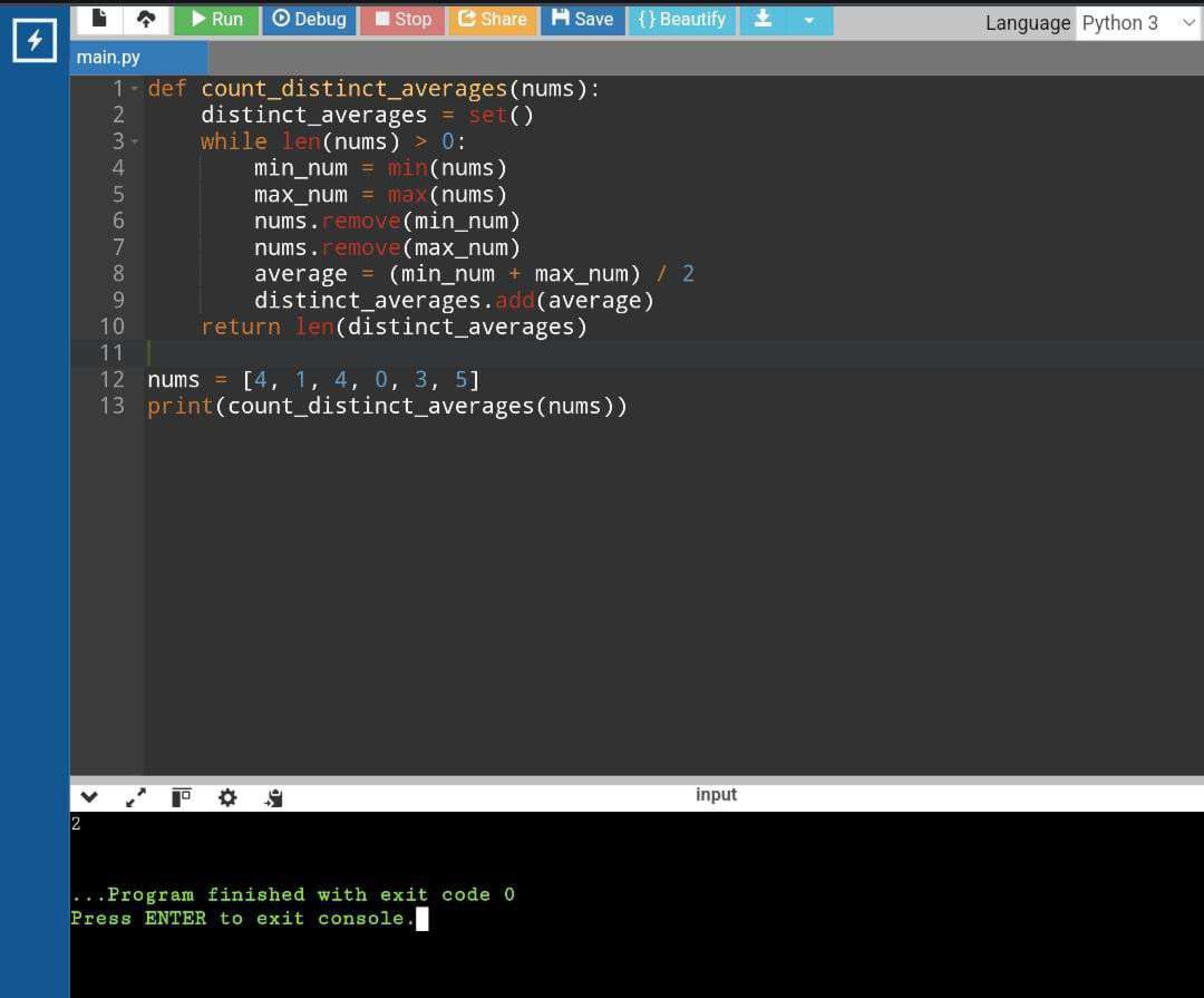
```python
def count_good_strings(low, high, zero, one):
    MOD = 10**9 + 7
    dp = [[[0] * 2 for _ in range(one + 1)] for _ in range(zero + 1)]
    dp[0][0][0] = dp[0][0][1] = 1
    for i in range(zero + 1):
        for j in range(one + 1):
            for k in range(2):
                if not dp[i][j][k]:
                    continue
                for x in range(2):
                    ni, nj, nk = i, j, k
                    if x == 0:
                        ni += 1
                    else:
                        nj += 1
                    if ni <= zero and nj <= one:
                        nk = k | x
                        dp[ni][nj][nk] += dp[i][j][k]
                        dp[ni][nj][nk] %= MOD
    ans = sum(sum(dp[i][j]) for i in range(zero + 1) for j in range(one + 1)) % MOD
    return ans

low = 3
high = 3
zero = 1
one = 1
output = count_good_strings(low, high, zero, one)
print(output)
```

```
10


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



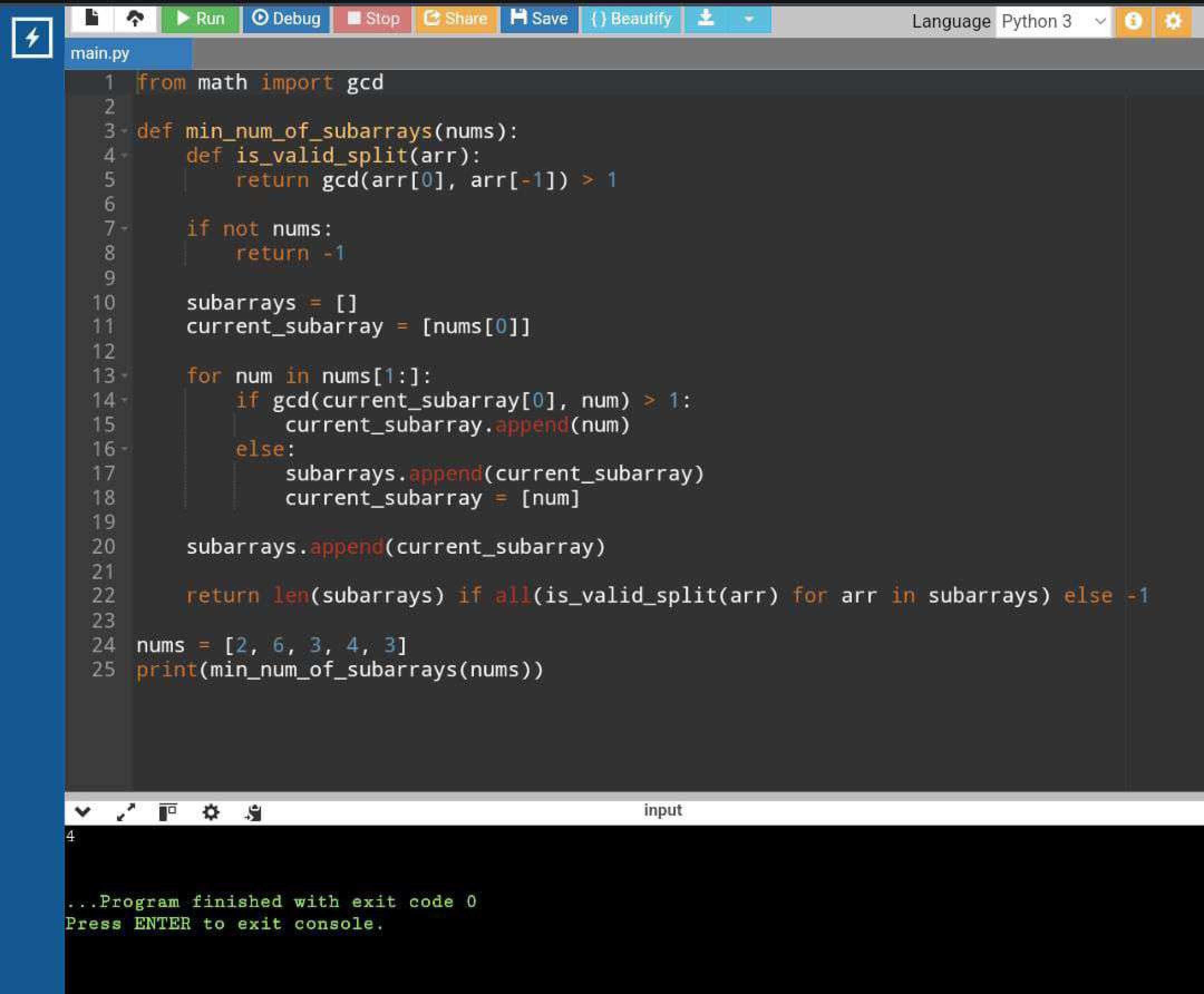
```python
def minTotalDistance(robot, factory):
    robot.sort()
    factory.sort()
    res = 0
    i = 0
    for pos, limit in factory:
        while i < len(robot) and limit > 0:
            res += abs(robot[i] - pos)
            limit -= 1
            i += 1
    return res
robots = [1, 3, 5]
factories = [(2, 2), (4, 1)]
print(minTotalDistance(robots, factories))
```

```
3


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



```python
def count_distinct_averages(nums):
    distinct_averages = set()
    while len(nums) > 0:
        min_num = min(nums)
        max_num = max(nums)
        nums.remove(min_num)
        nums.remove(max_num)
        average = (min_num + max_num) / 2
        distinct_averages.add(average)
    return len(distinct_averages)

nums = [4, 1, 4, 0, 3, 5]
print(count_distinct_averages(nums))
```

```
2


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

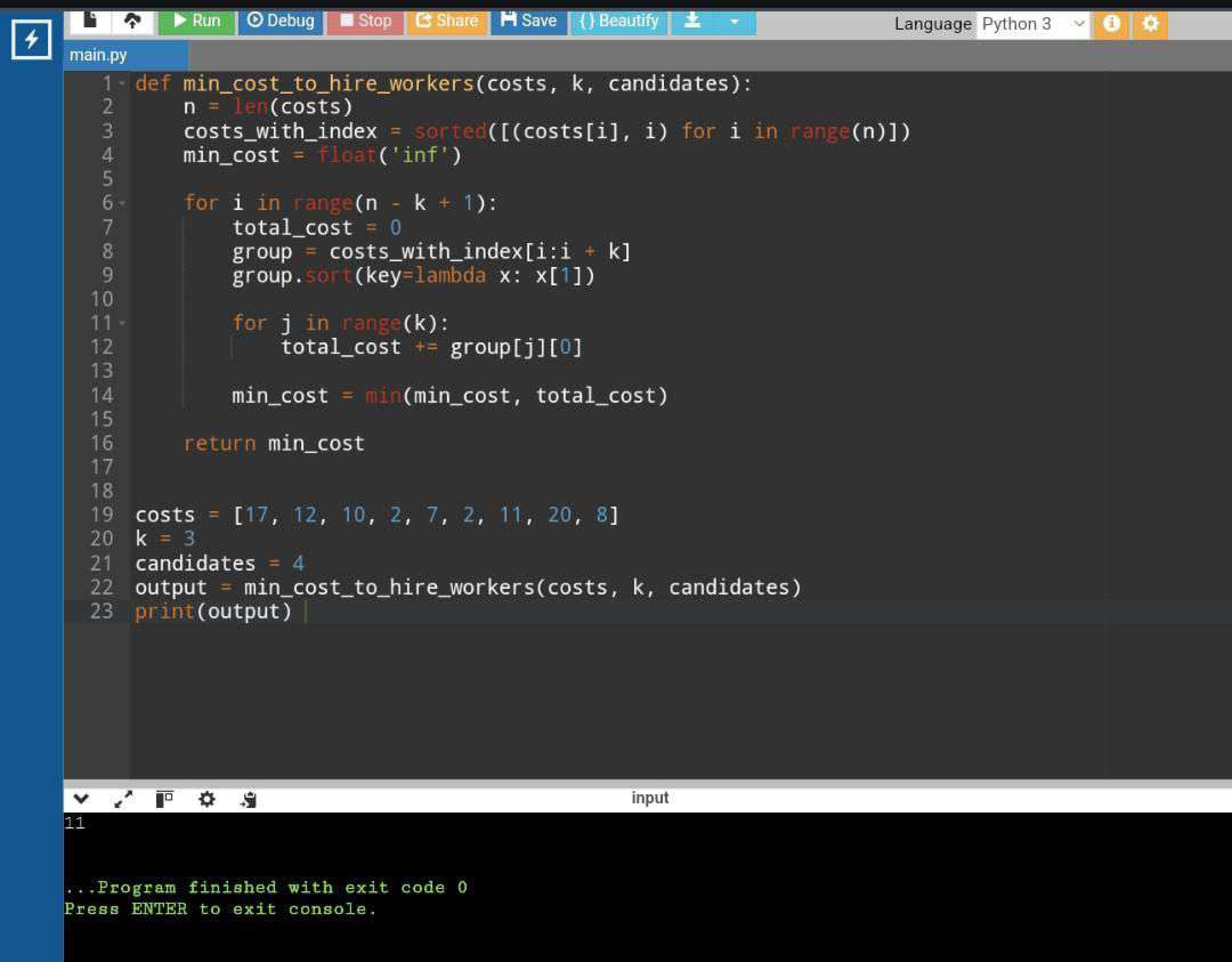


Language Python 3

main.py

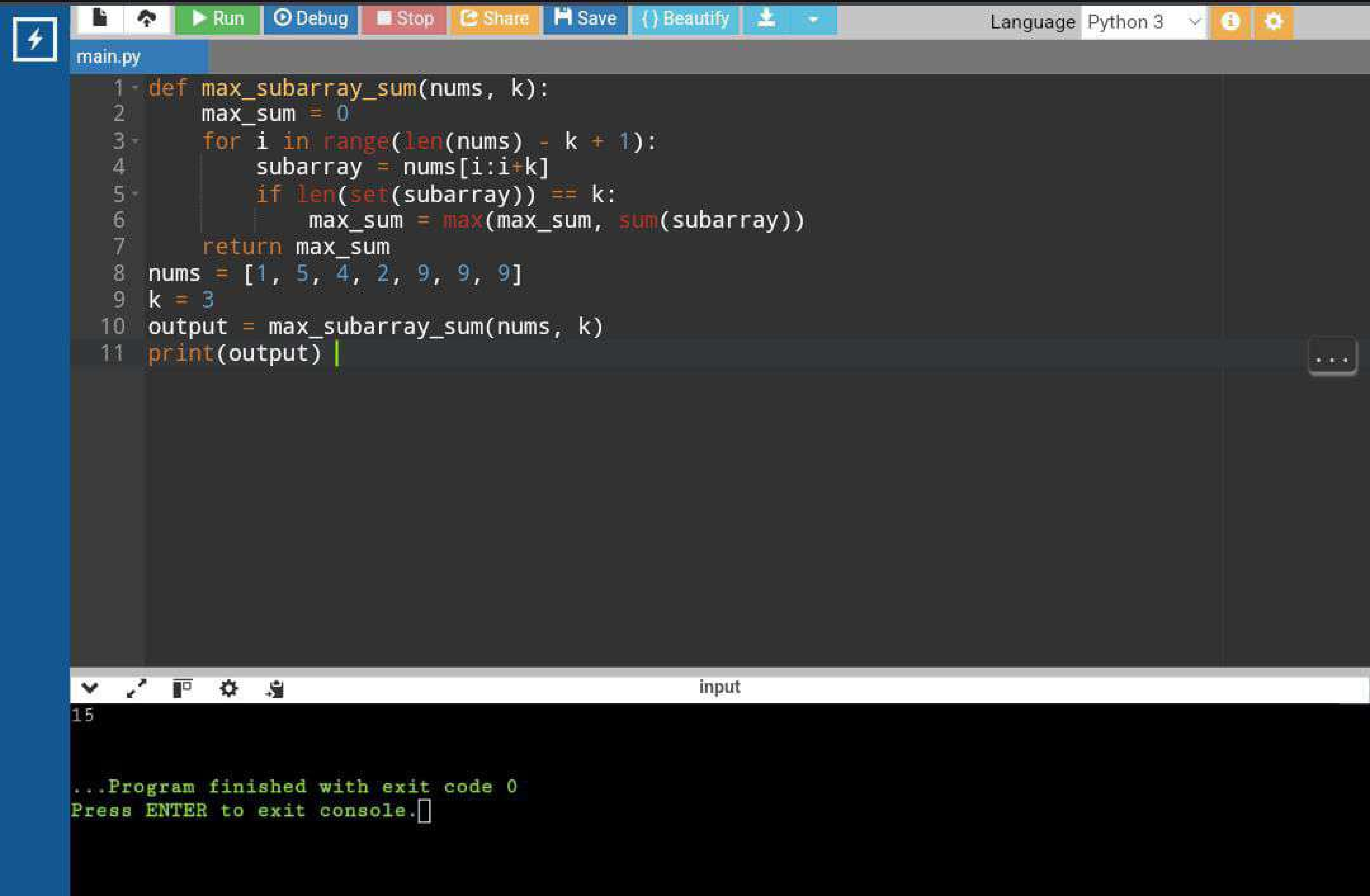
```python
from math import gcd

def min_num_of_subarrays(nums):
    def is_valid_split(arr):
        return gcd(arr[0], arr[-1]) > 1

    if not nums:
        return -1

    subarrays = []
    current_subarray = [nums[0]]

    for num in nums[1:]:
        if gcd(current_subarray[0], num) > 1:
            current_subarray.append(num)
        else:
            subarrays.append(current_subarray)
            current_subarray = [num]

    subarrays.append(current_subarray)

    return len(subarrays) if all(is_valid_split(arr) for arr in subarrays) else -1

nums = [2, 6, 3, 4, 3]
print(min_num_of_subarrays(nums))
```

```
4


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



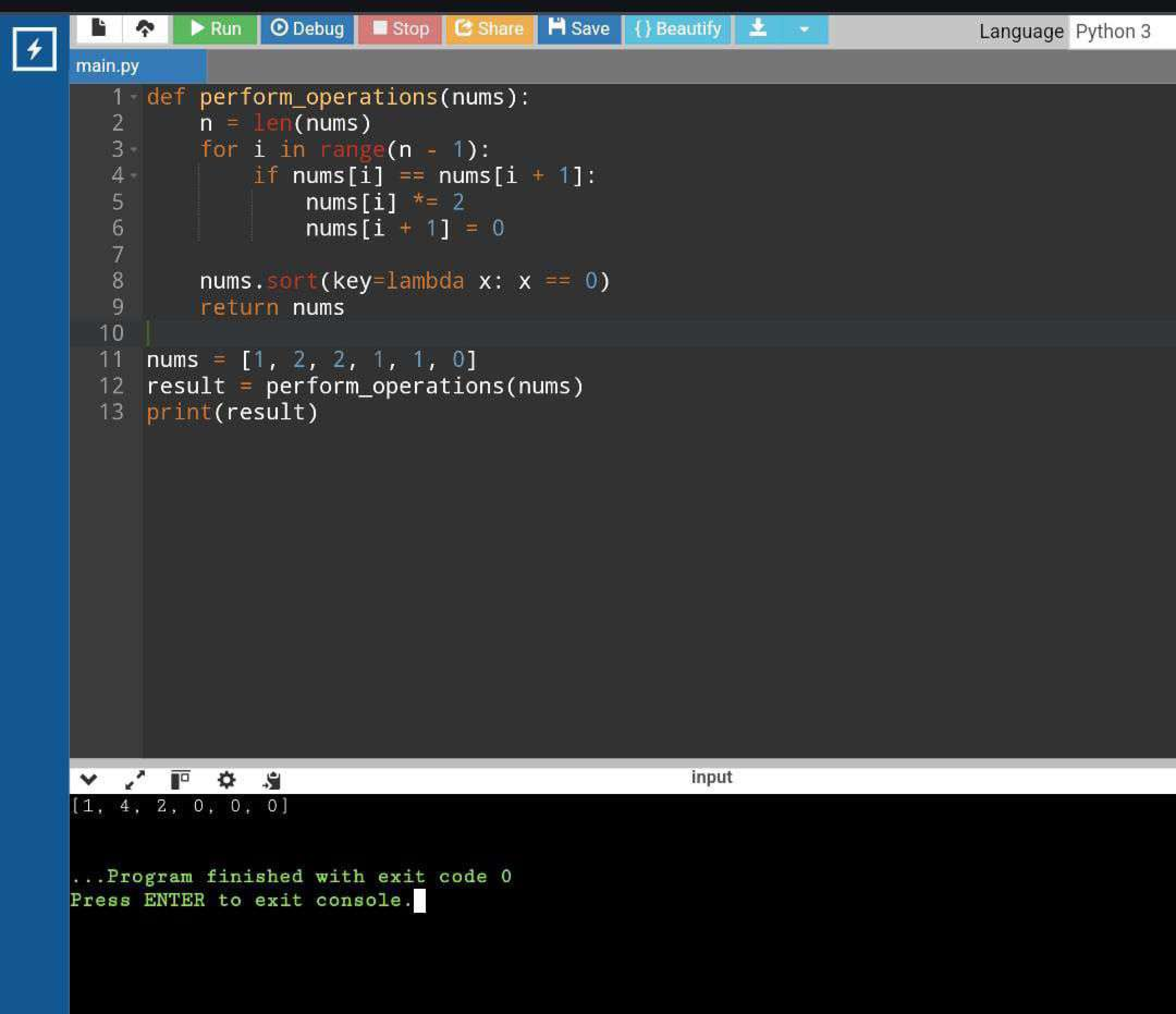
```python
def min_cost_to_hire_workers(costs, k, candidates):
    n = len(costs)
    costs_with_index = sorted([(costs[i], i) for i in range(n)])
    min_cost = float('inf')

    for i in range(n - k + 1):
        total_cost = 0
        group = costs_with_index[i:i + k]
        group.sort(key=lambda x: x[1])

        for j in range(k):
            total_cost += group[j][0]

        min_cost = min(min_cost, total_cost)

    return min_cost


costs = [17, 12, 10, 2, 7, 2, 11, 20, 8]
k = 3
candidates = 4
output = min_cost_to_hire_workers(costs, k, candidates)
print(output)
```

```
11


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



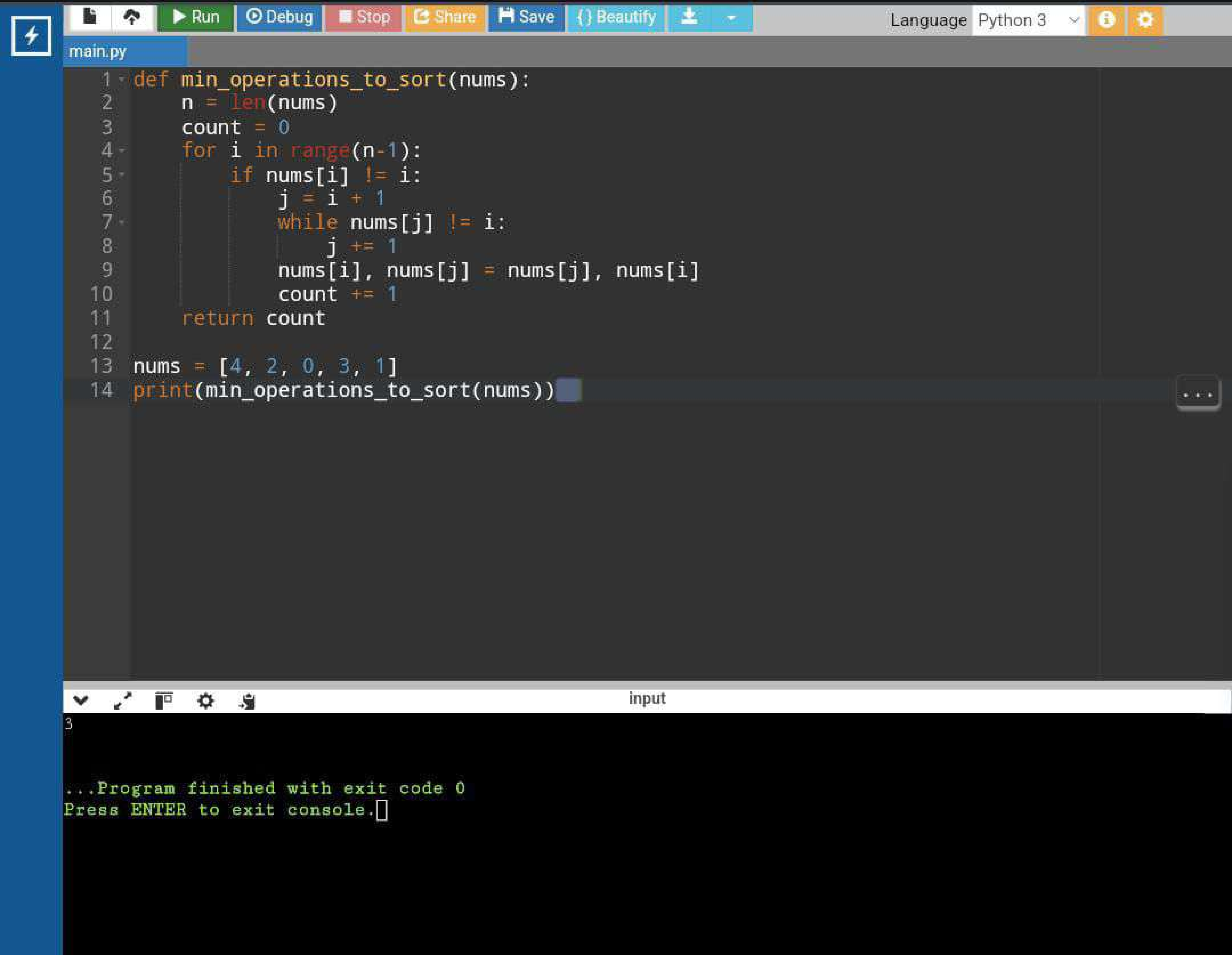
```python
def max_subarray_sum(nums, k):
    max_sum = 0
    for i in range(len(nums) - k + 1):
        subarray = nums[i:i+k]
        if len(set(subarray)) == k:
            max_sum = max(max_sum, sum(subarray))
    return max_sum
nums = [1, 5, 4, 2, 9, 9, 9]
k = 3
output = max_subarray_sum(nums, k)
print(output)
```

```
15


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



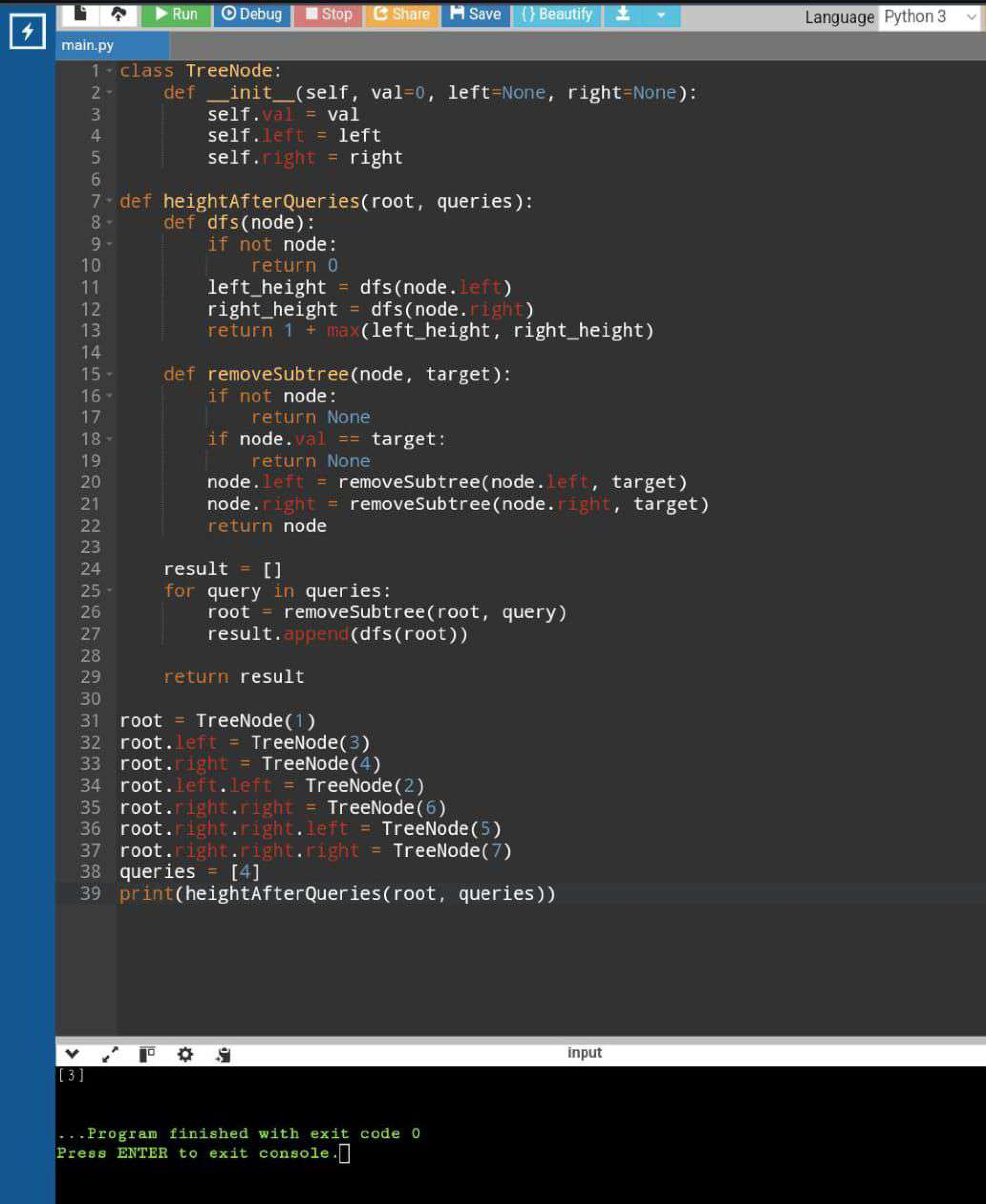
```python
def perform_operations(nums):
    n = len(nums)
    for i in range(n - 1):
        if nums[i] == nums[i + 1]:
            nums[i] *= 2
            nums[i + 1] = 0

    nums.sort(key=lambda x: x == 0)
    return nums

nums = [1, 2, 2, 1, 1, 0]
result = perform_operations(nums)
print(result)
```

```
[1, 4, 2, 0, 0, 0]


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



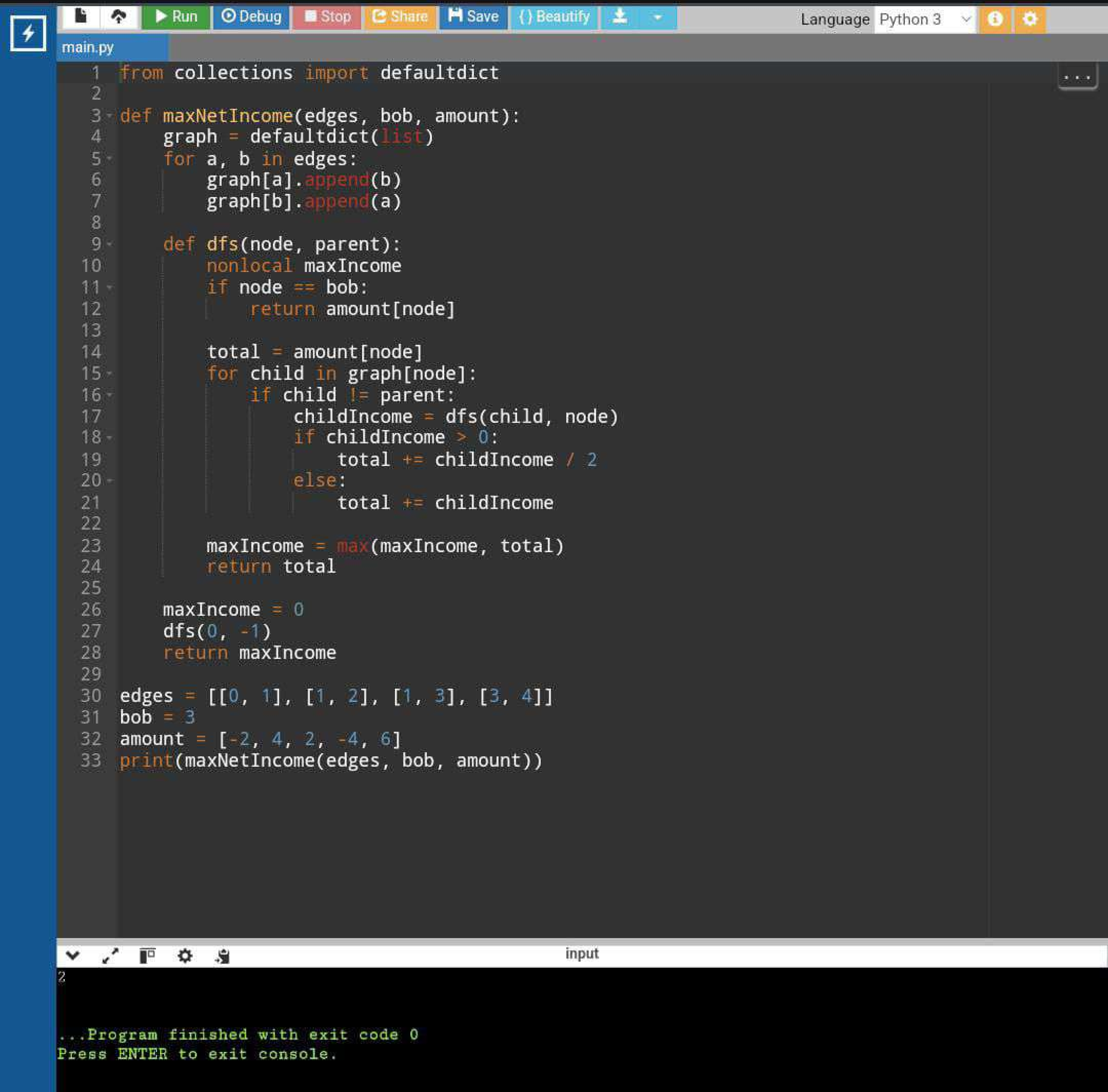
```python
def min_operations_to_sort(nums):
    n = len(nums)
    count = 0
    for i in range(n-1):
        if nums[i] != i:
            j = i + 1
            while nums[j] != i:
                j += 1
            nums[i], nums[j] = nums[j], nums[i]
            count += 1
    return count

nums = [4, 2, 0, 3, 1]
print(min_operations_to_sort(nums))
```

```
3


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



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

def heightAfterQueries(root, queries):
    def dfs(node):
        if not node:
            return 0
        left_height = dfs(node.left)
        right_height = dfs(node.right)
        return 1 + max(left_height, right_height)

    def removeSubtree(node, target):
        if not node:
            return None
        if node.val == target:
            return None
        node.left = removeSubtree(node.left, target)
        node.right = removeSubtree(node.right, target)
        return node

    result = []
    for query in queries:
        root = removeSubtree(root, query)
        result.append(dfs(root))

    return result

root = TreeNode(1)
root.left = TreeNode(3)
root.right = TreeNode(4)
root.left.left = TreeNode(2)
root.right.right = TreeNode(6)
root.right.right.left = TreeNode(5)
root.right.right.right = TreeNode(7)
queries = [4]
print(heightAfterQueries(root, queries))
```

```
[3]


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



```python
from collections import defaultdict

def maxNetIncome(edges, bob, amount):
    graph = defaultdict(list)
    for a, b in edges:
        graph[a].append(b)
        graph[b].append(a)

    def dfs(node, parent):
        nonlocal maxIncome
        if node == bob:
            return amount[node]

        total = amount[node]
        for child in graph[node]:
            if child != parent:
                childIncome = dfs(child, node)
                if childIncome > 0:
                    total += childIncome / 2
                else:
                    total += childIncome

        maxIncome = max(maxIncome, total)
        return total

    maxIncome = 0
    dfs(0, -1)
    return maxIncome

edges = [[0, 1], [1, 2], [1, 3], [3, 4]]
bob = 3
amount = [-2, 4, 2, -4, 6]
print(maxNetIncome(edges, bob, amount))
```

```
2


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