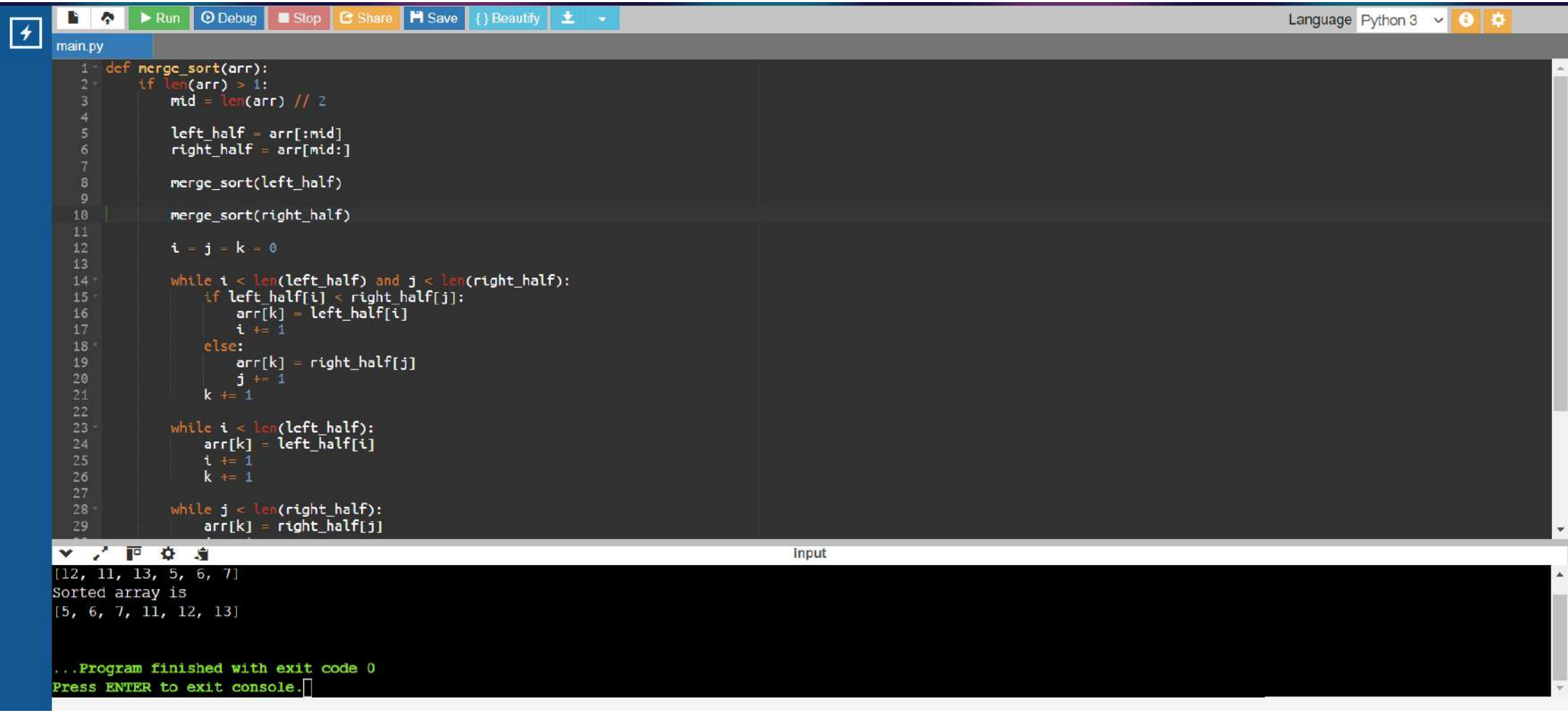
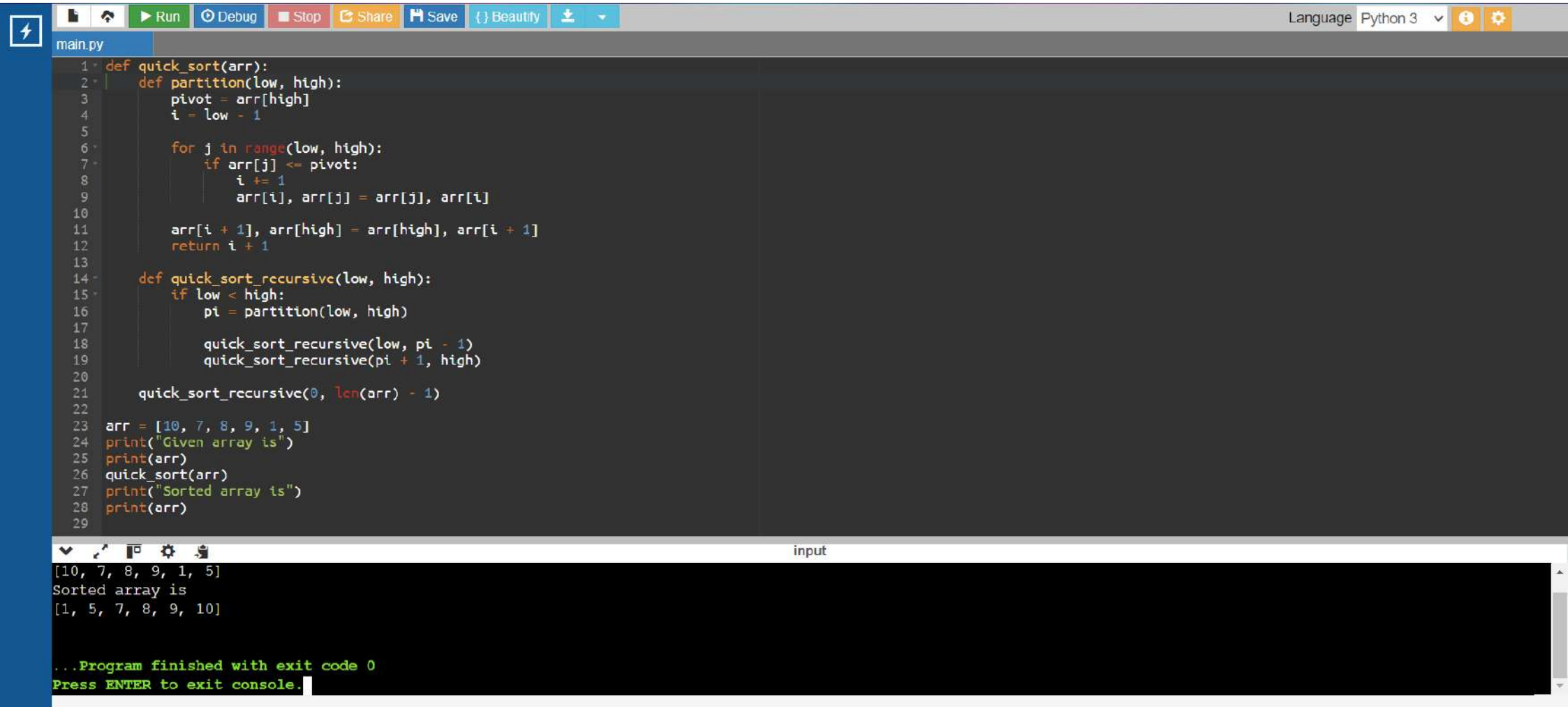
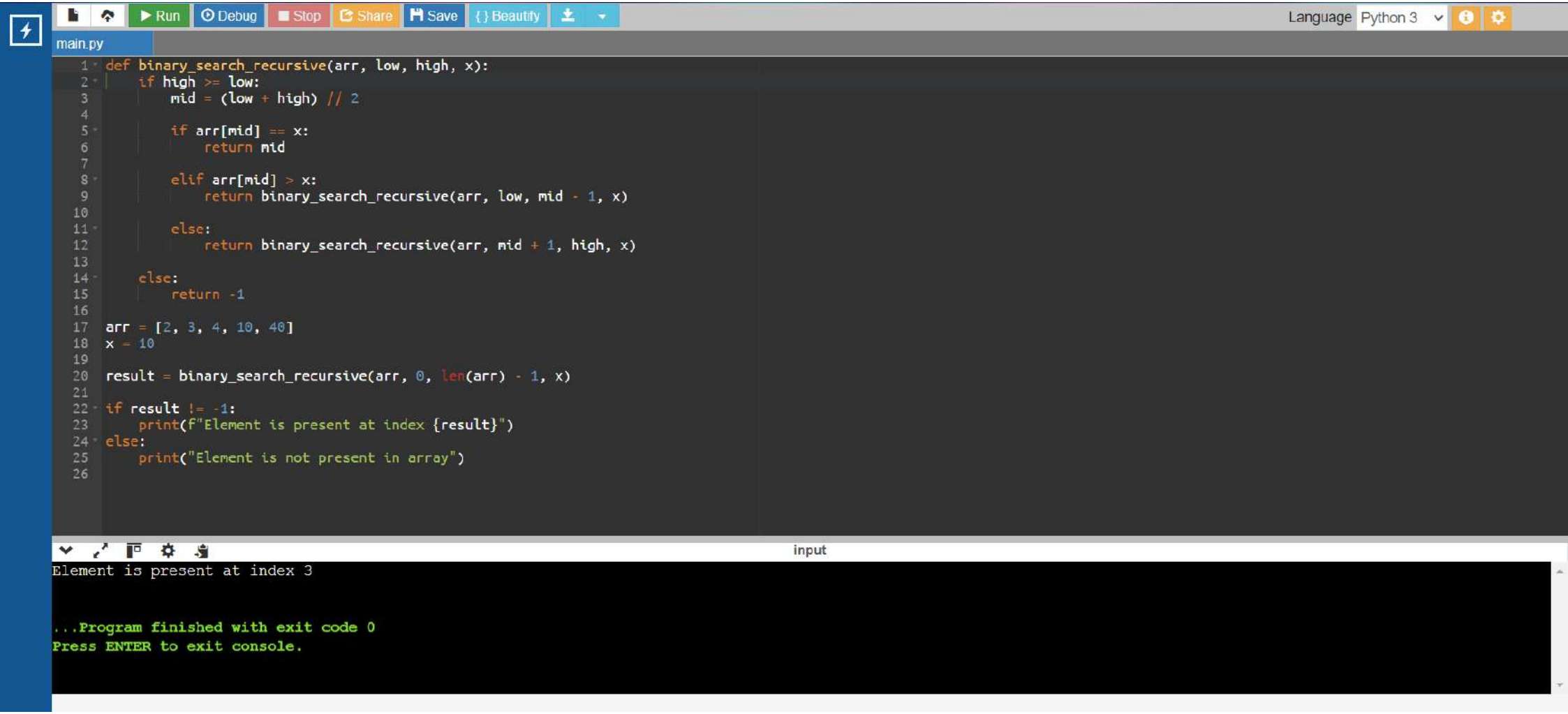
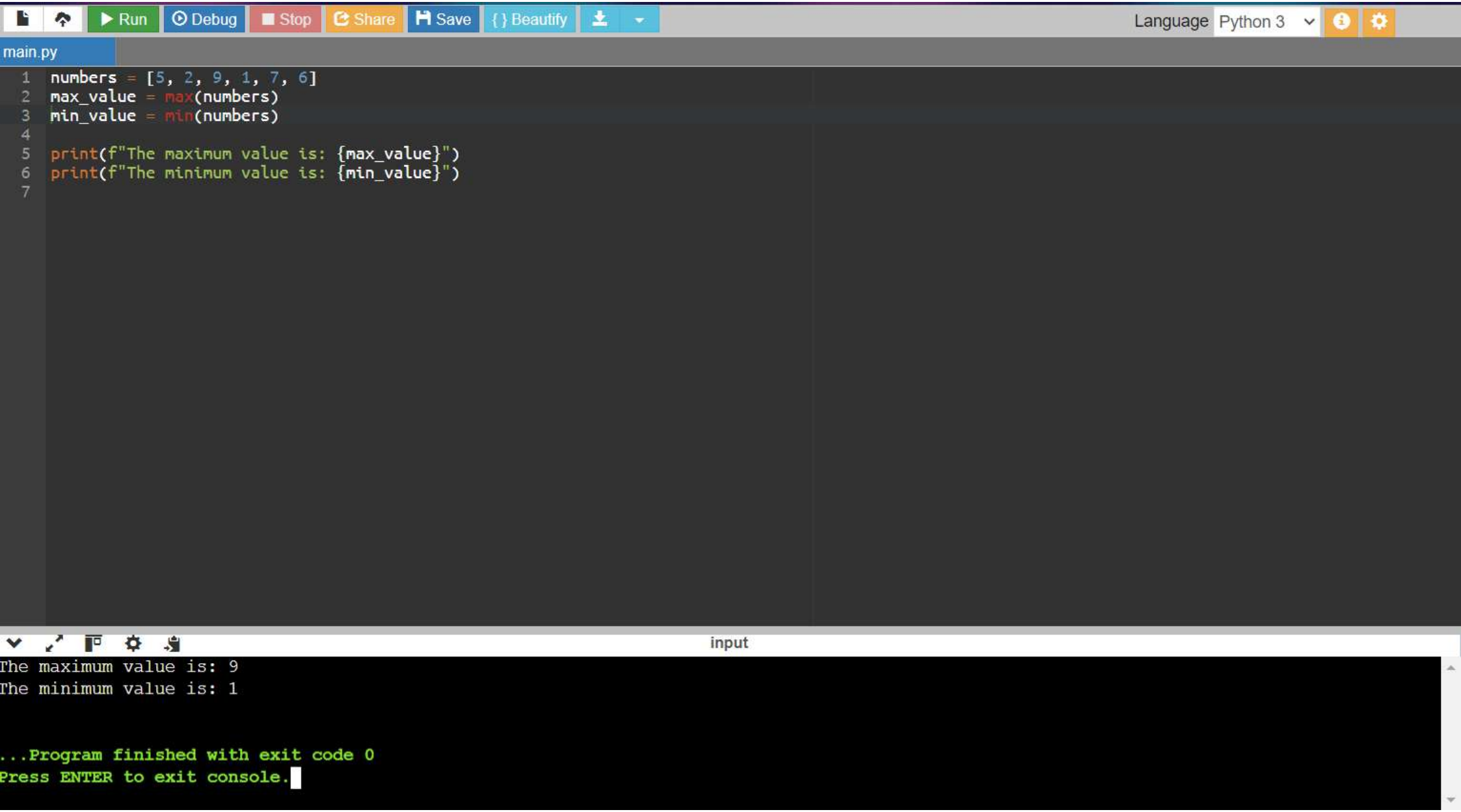

Run
Debug
Stop
Share
Save
{} Beautify
Language
Python 3
main.py
def merge_sort(arr):
if len(arr) > 1:
mid = len(arr) // 2
left_half = arr[:mid]
right_half = arr[mid:]
merge_sort(left_half)
merge_sort(right_half)
i = j = k = 0
while i < len(left_half) and j < len(right_half):
if left_half[i] < right_half[j]:
arr[k] = left_half[i]
i += 1
else:
arr[k] = right_half[j]
j += 1
k += 1
while i < len(left_half):
arr[k] = left_half[i]
i += 1
k += 1
while j < len(right_half):
arr[k] = right_half[j]
Input
[12, 11, 13, 5, 6, 7]
Sorted array is
[5, 6, 7, 11, 12, 13]
...Program finished with exit code 0
Press ENTER to exit console.


Run
Debug
Stop
Share
Save
{} Beautify
Language
Python 3
main.py
def quick_sort(arr):
def partition(low, high):
pivot = arr[high]
i = low - 1
for j in range(low, high):
if arr[j] <= pivot:
i += 1
arr[i], arr[j] = arr[j], arr[i]
arr[i + 1], arr[high] = arr[high], arr[i + 1]
return i + 1
def quick_sort_recursive(low, high):
if low < high:
pi = partition(low, high)
quick_sort_recursive(low, pi - 1)
quick_sort_recursive(pi + 1, high)
quick_sort_recursive(0, len(arr) - 1)
arr = [10, 7, 8, 9, 1, 5]
print("Given array is")
print(arr)
quick_sort(arr)
print("Sorted array is")
print(arr)
input
[10, 7, 8, 9, 1, 5]
Sorted array is
[1, 5, 7, 8, 9, 10]
...Program finished with exit code 0
Press ENTER to exit console.


Run
Debug
Stop
Share
Save
{} Beautify
Language
Python 3
main.py
def binary_search_recursive(arr, low, high, x):
if high >= low:
mid = (low + high) // 2
if arr[mid] == x:
return mid
elif arr[mid] > x:
return binary_search_recursive(arr, low, mid - 1, x)
else:
return binary_search_recursive(arr, mid + 1, high, x)
else:
return -1
arr = [2, 3, 4, 10, 40]
x = 10
result = binary_search_recursive(arr, 0, len(arr) - 1, x)
if result != -1:
print(f"Element is present at index {result}")
else:
print("Element is not present in array")
input
Element is present at index 3
...Program finished with exit code 0
Press ENTER to exit console.


Run
Debug
Stop
Share
Save
{ } Beautify
Language
Python 3
main.py
1 numbers = [5, 2, 9, 1, 7, 6]
2 max_value = max(numbers)
3 min_value = min(numbers)
4
5 print(f"The maximum value is: {max_value}")
6 print(f"The minimum value is: {min_value}")
7
input
The maximum value is: 9
The minimum value is: 1
...Program finished with exit code 0
Press ENTER to exit console.

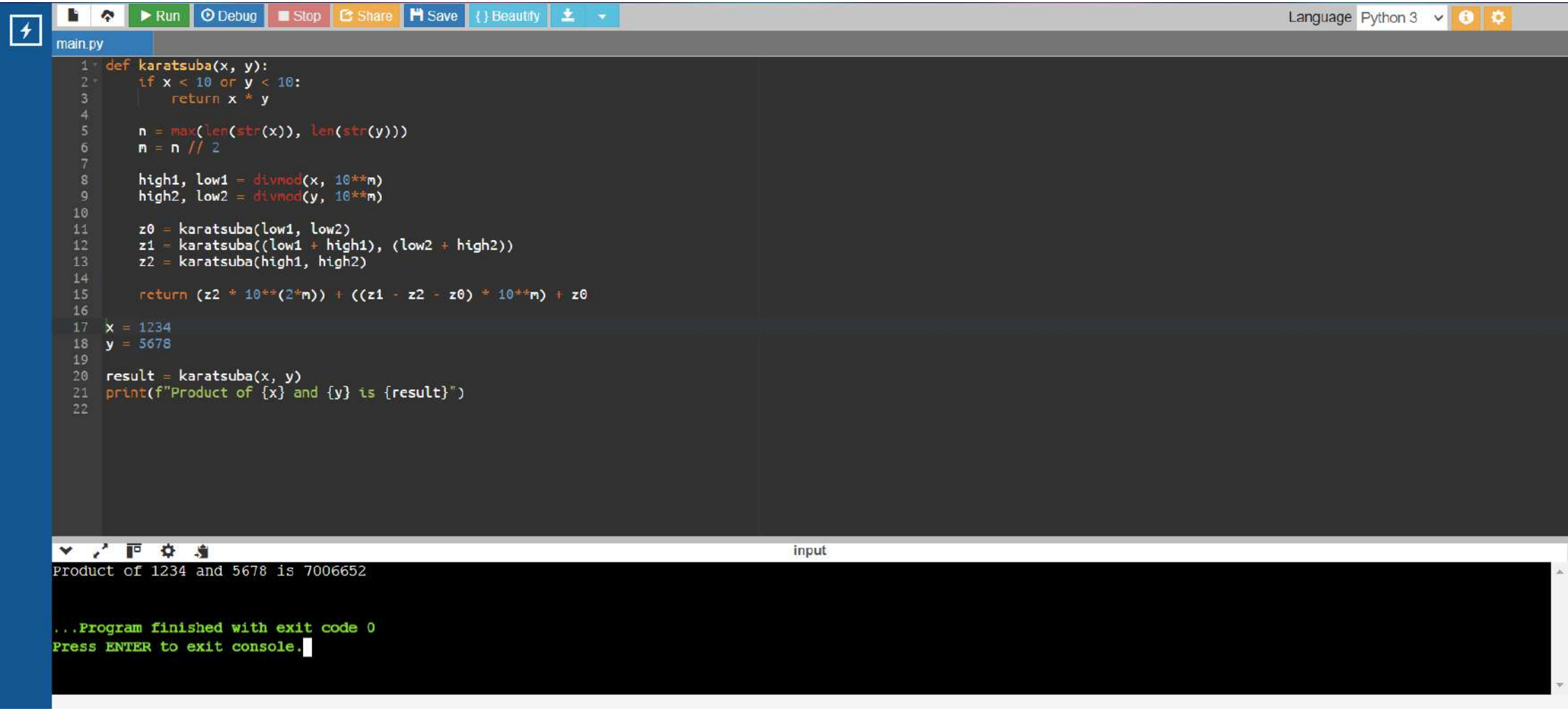
Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
import numpy as np

def strassen(A, B):
    n = len(A)

    if n == 1:
        return A * B

    k = n // 2
    A11, A12, A21, A22 = A[:k, :k], A[:k, k:], A[k:, :k], A[k:, k:]
    B11, B12, B21, B22 = B[:k, :k], B[:k, k:], B[k:, :k], B[k:, k:]

    M1 = strassen(A11 + A22, B11 + B22)
    M2 = strassen(A21 + A22, B11)
    M3 = strassen(A11, B12 - B22)
    M4 = strassen(A22, B21 - B11)
    M5 = strassen(A11 + A12, B22)
    M6 = strassen(A21 - A11, B11 + B12)
    M7 = strassen(A12 - A22, B21 + B22)

    C11 = M1 + M4 - M5 + M7
    C12 = M3 + M5
    C21 = M2 + M4
    C22 = M1 - M2 + M3 + M6

    C = np.vstack((np.hstack((C11, C12)), np.hstack((C21, C22))))

    return C
```

input

```
Product of A and B is:
[[19 22]
 [43 50]]


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


Run
Debug
Stop
Share
Save
{} Beautify
Language
Python 3
main.py
def karatsuba(x, y):
if x < 10 or y < 10:
return x * y
n = max(len(str(x)), len(str(y)))
m = n // 2
high1, low1 = divmod(x, 10**m)
high2, low2 = divmod(y, 10**m)
z0 = karatsuba(low1, low2)
z1 = karatsuba((low1 + high1), (low2 + high2))
z2 = karatsuba(high1, high2)
return (z2 * 10**(2*m)) + ((z1 - z2 - z0) * 10**m) + z0
x = 1234
y = 5678
result = karatsuba(x, y)
print(f"Product of {x} and {y} is {result}")
input
Product of 1234 and 5678 is 7006652
...Program finished with exit code 0
Press ENTER to exit console.

Run | Debug | Stop | Share | Save | {} Beautify | Language Python 3

main.py

```python
import math
def distance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def brute_force(P):
    min_dist = float('inf')
    n = len(P)
    for i in range(n):
        for j in range(i + 1, n):
            if distance(P[i], P[j]) < min_dist:
                min_dist = distance(P[i], P[j])
    return min_dist

def strip_closest(strip, d):
    min_dist = d
    n = len(strip)
    strip.sort(key=lambda x: x[1])

    for i in range(n):
        j = i + 1
        while j < n and (strip[j][1] - strip[i][1]) < min_dist:
            min_dist = min(min_dist, distance(strip[i], strip[j]))
            j += 1

    return min_dist

def closest_pair(P):
    def closest_pair_recursive(Px, Py):
        n = len(Px)
```

input

```
The smallest distance is 1.4142135623730951


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

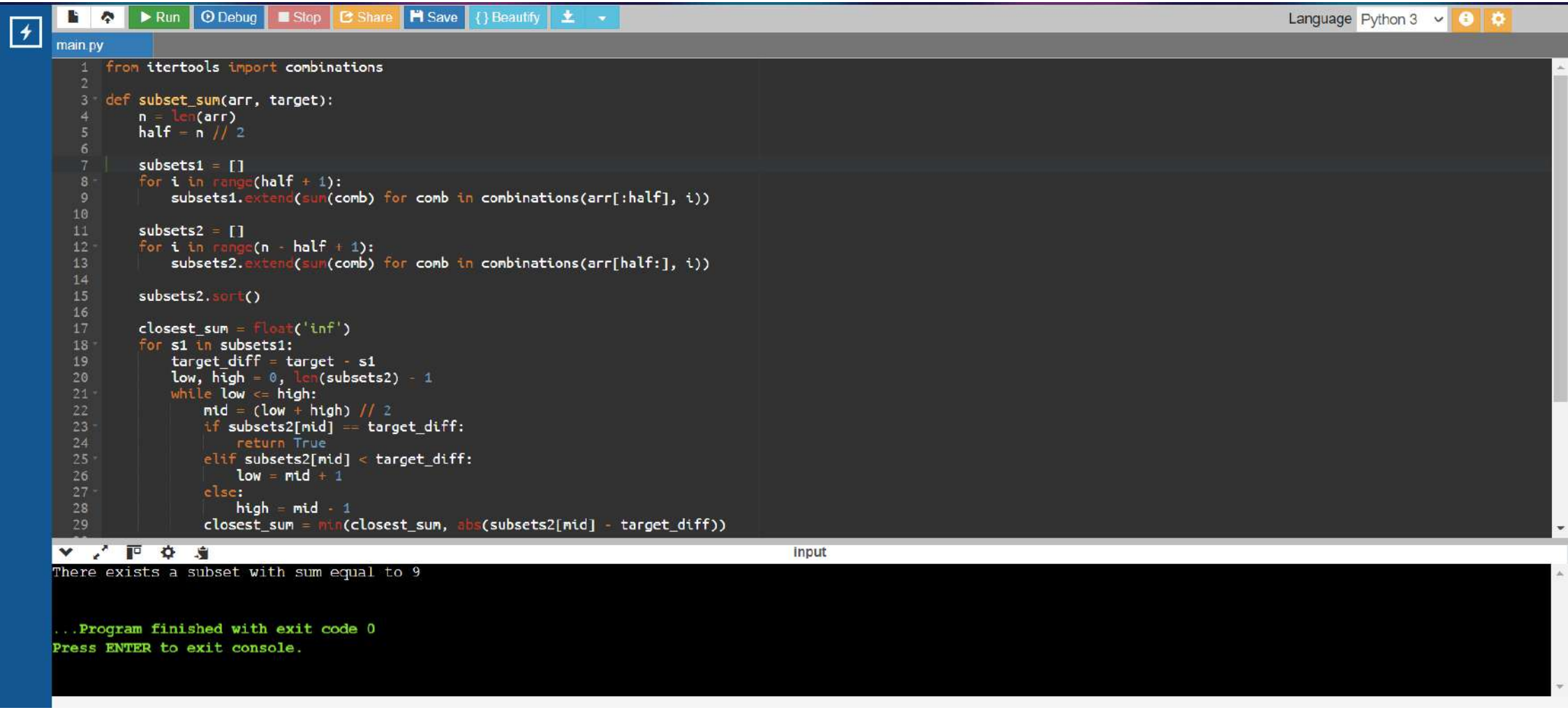
Run | Debug | Stop | Share | Save | { } Beautify | Language Python 3

main.py

```python
def partition(arr, low, high, pivot):
    pivot_index = arr.index(pivot)
    arr[pivot_index], arr[high] = arr[high], arr[pivot_index]
    i = low
    for j in range(low, high):
        if arr[j] < pivot:
            arr[i], arr[j] = arr[j], arr[i]
            i += 1
    arr[i], arr[high] = arr[high], arr[i]
    return i

def median_of_medians(arr, k):
    n = len(arr)

    if n <= 5:
        return sorted(arr)[k]

    medians = [sorted(arr[i:i+5])[len(arr[i:i+5]) // 2] for i in range(0, n, 5)]
    pivot = median_of_medians(medians, len(medians) // 2)

    pivot_index = partition(arr, 0, n - 1, pivot)

    if k < pivot_index:
        return median_of_medians(arr[:pivot_index], k)
    elif k > pivot_index:
        return median_of_medians(arr[pivot_index + 1:], k - pivot_index - 1)
    else:
        return arr[pivot_index]
```

input

```
The 4-th smallest element is 7


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


Run
Debug
Stop
Share
Save
{} Beautify
Language
Python 3
main.py
from itertools import combinations
def subset_sum(arr, target):
n = len(arr)
half = n // 2
subsets1 = []
for i in range(half + 1):
subsets1.extend(sum(comb) for comb in combinations(arr[:half], i))
subsets2 = []
for i in range(n - half + 1):
subsets2.extend(sum(comb) for comb in combinations(arr[half:], i))
subsets2.sort()
closest_sum = float('inf')
for s1 in subsets1:
target_diff = target - s1
low, high = 0, len(subsets2) - 1
while low <= high:
mid = (low + high) // 2
if subsets2[mid] == target_diff:
return True
elif subsets2[mid] < target_diff:
low = mid + 1
else:
high = mid - 1
closest_sum = min(closest_sum, abs(subsets2[mid] - target_diff))
Input
There exists a subset with sum equal to 9
...Program finished with exit code 0
Press ENTER to exit console.