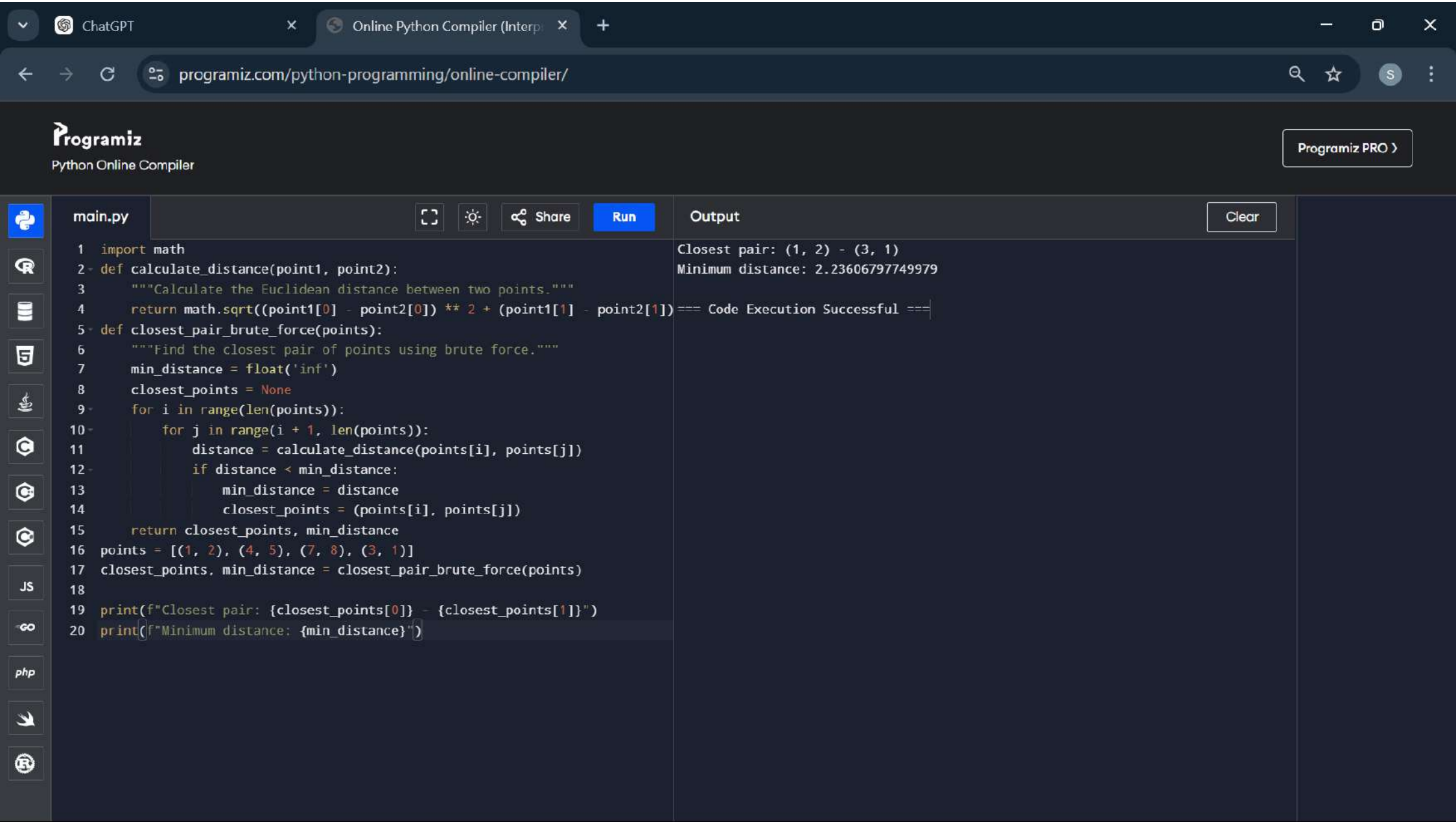
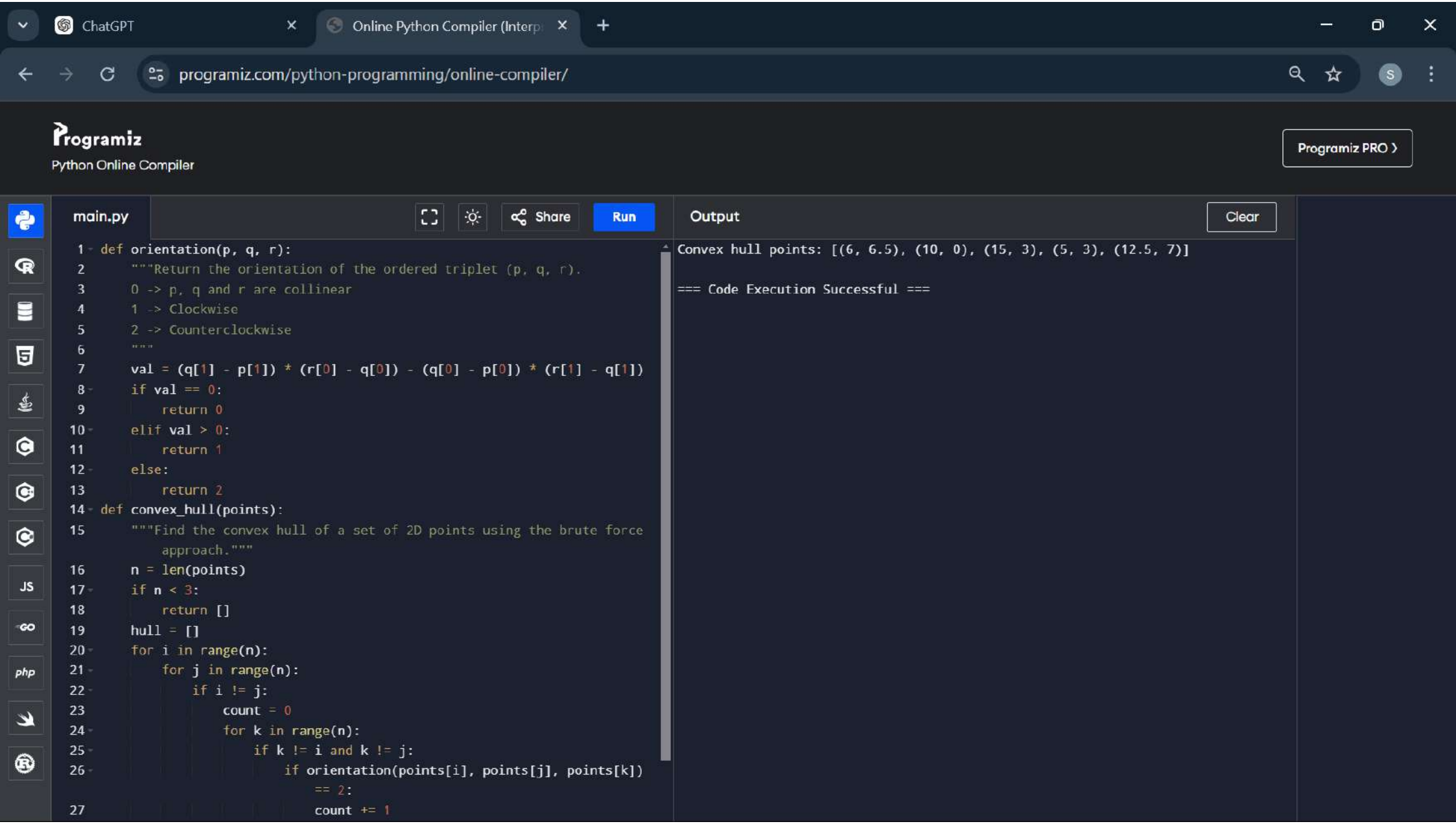
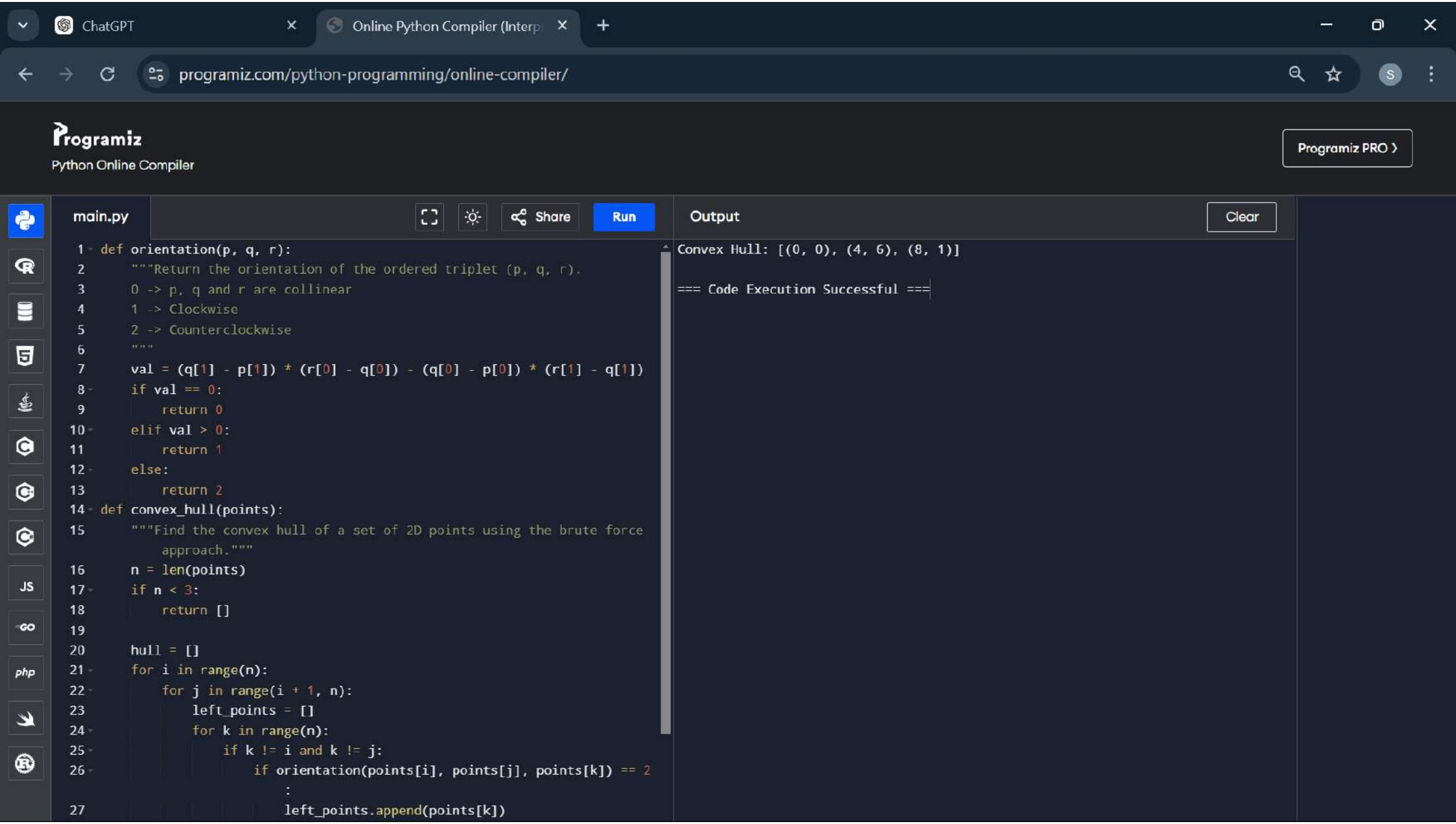
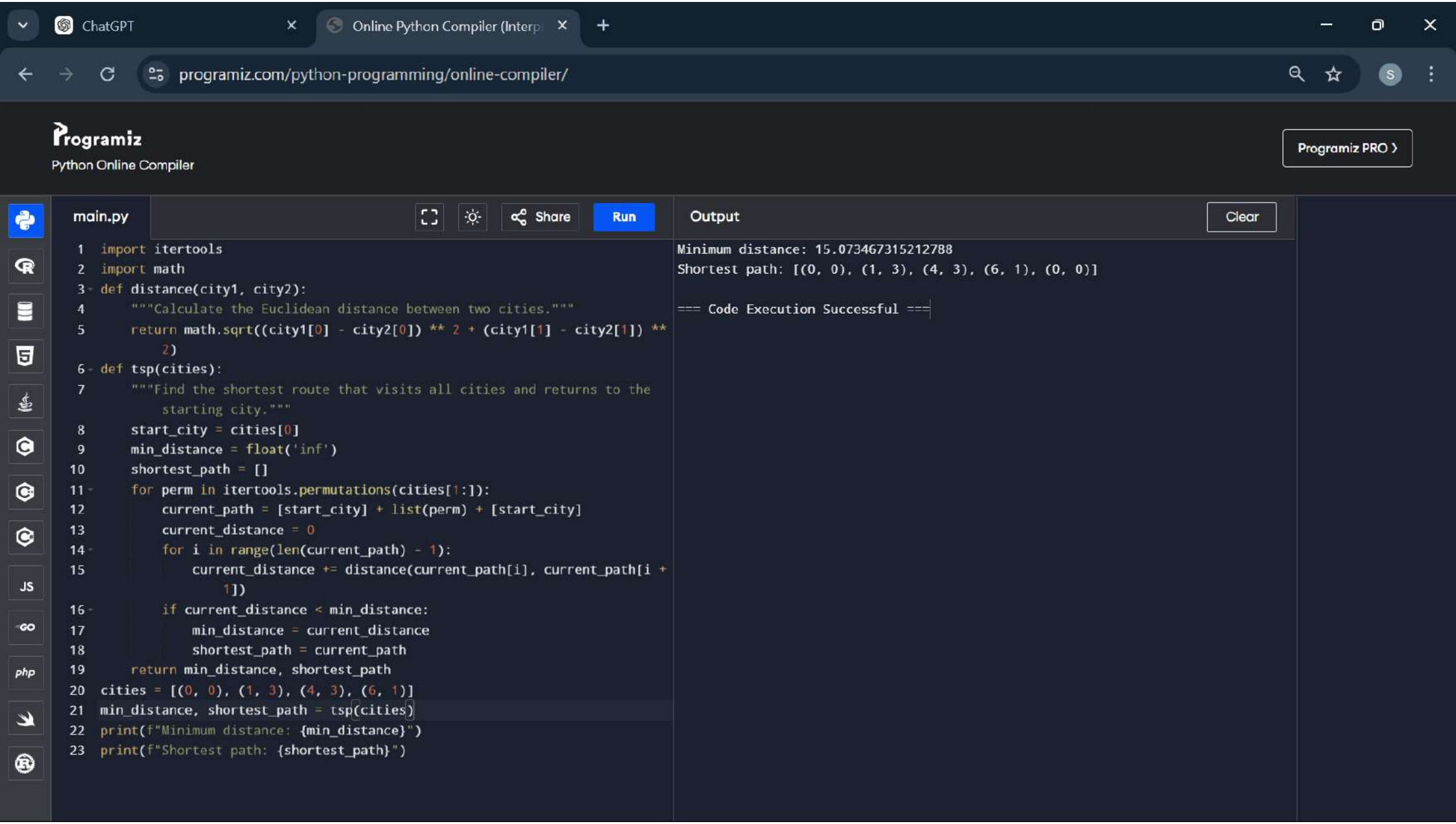

ChatGPT
Online Python Compiler (Interp
programiz.com/python-programming/online-compiler/
Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
import math
def calculate_distance(point1, point2):
"""Calculate the Euclidean distance between two points."""
return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1])
def closest_pair_brute_force(points):
"""Find the closest pair of points using brute force."""
min_distance = float('inf')
closest_points = None
for i in range(len(points)):
for j in range(i + 1, len(points)):
distance = calculate_distance(points[i], points[j])
if distance < min_distance:
min_distance = distance
closest_points = (points[i], points[j])
return closest_points, min_distance
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_points, min_distance = closest_pair_brute_force(points)
print(f"Closest pair: {closest_points[0]} - {closest_points[1]}")
print(f"Minimum distance: {min_distance}")
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979
=== Code Execution Successful ===


ChatGPT
Online Python Compiler (Interp
programiz.com/python-programming/online-compiler/
Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
def orientation(p, q, r):
"""Return the orientation of the ordered triplet (p, q, r).
0 -> p, q and r are collinear
1 -> Clockwise
2 -> Counterclockwise
"""
val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1])
if val == 0:
return 0
elif val > 0:
return 1
else:
return 2
def convex_hull(points):
"""Find the convex hull of a set of 2D points using the brute force approach."""
n = len(points)
if n < 3:
return []
hull = []
for i in range(n):
for j in range(n):
if i != j:
count = 0
for k in range(n):
if k != i and k != j:
if orientation(points[i], points[j], points[k]) == 2:
count += 1
Convex hull points: [(6, 6.5), (10, 0), (15, 3), (5, 3), (12.5, 7)]
=== Code Execution Successful ===


ChatGPT
Online Python Compiler (Interp
programiz.com/python-programming/online-compiler/
Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
Convex Hull: [(0, 0), (4, 6), (8, 1)]
=== Code Execution Successful ===
def orientation(p, q, r):
"""Return the orientation of the ordered triplet (p, q, r).
0 -> p, q and r are collinear
1 -> Clockwise
2 -> Counterclockwise
"""
val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1])
if val == 0:
return 0
elif val > 0:
return 1
else:
return 2
def convex_hull(points):
"""Find the convex hull of a set of 2D points using the brute force approach."""
n = len(points)
if n < 3:
return []
hull = []
for i in range(n):
for j in range(i + 1, n):
left_points = []
for k in range(n):
if k != i and k != j:
if orientation(points[i], points[j], points[k]) == 2:
left_points.append(points[k])


ChatGPT
Online Python Compiler (Interp
programiz.com/python-programming/online-compiler/
Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
import itertools
import math
def distance(city1, city2):
"""Calculate the Euclidean distance between two cities."""
return math.sqrt((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1]) ** 2)
def tsp(cities):
"""Find the shortest route that visits all cities and returns to the starting city."""
start_city = cities[0]
min_distance = float('inf')
shortest_path = []
for perm in itertools.permutations(cities[1:]):
current_path = [start_city] + list(perm) + [start_city]
current_distance = 0
for i in range(len(current_path) - 1):
current_distance += distance(current_path[i], current_path[i + 1])
if current_distance < min_distance:
min_distance = current_distance
shortest_path = current_path
return min_distance, shortest_path
cities = [(0, 0), (1, 3), (4, 3), (6, 1)]
min_distance, shortest_path = tsp(cities)
print(f"Minimum distance: {min_distance}")
print(f"Shortest path: {shortest_path}")
Minimum distance: 15.073467315212788
Shortest path: [(0, 0), (1, 3), (4, 3), (6, 1), (0, 0)]
=== Code Execution Successful ===

ChatGPT | Online Python Compiler (Interp

programiz.com/python-programming/online-compiler/

Programiz
Python Online Compiler

Programiz PRO ›

main.py | Share | Run

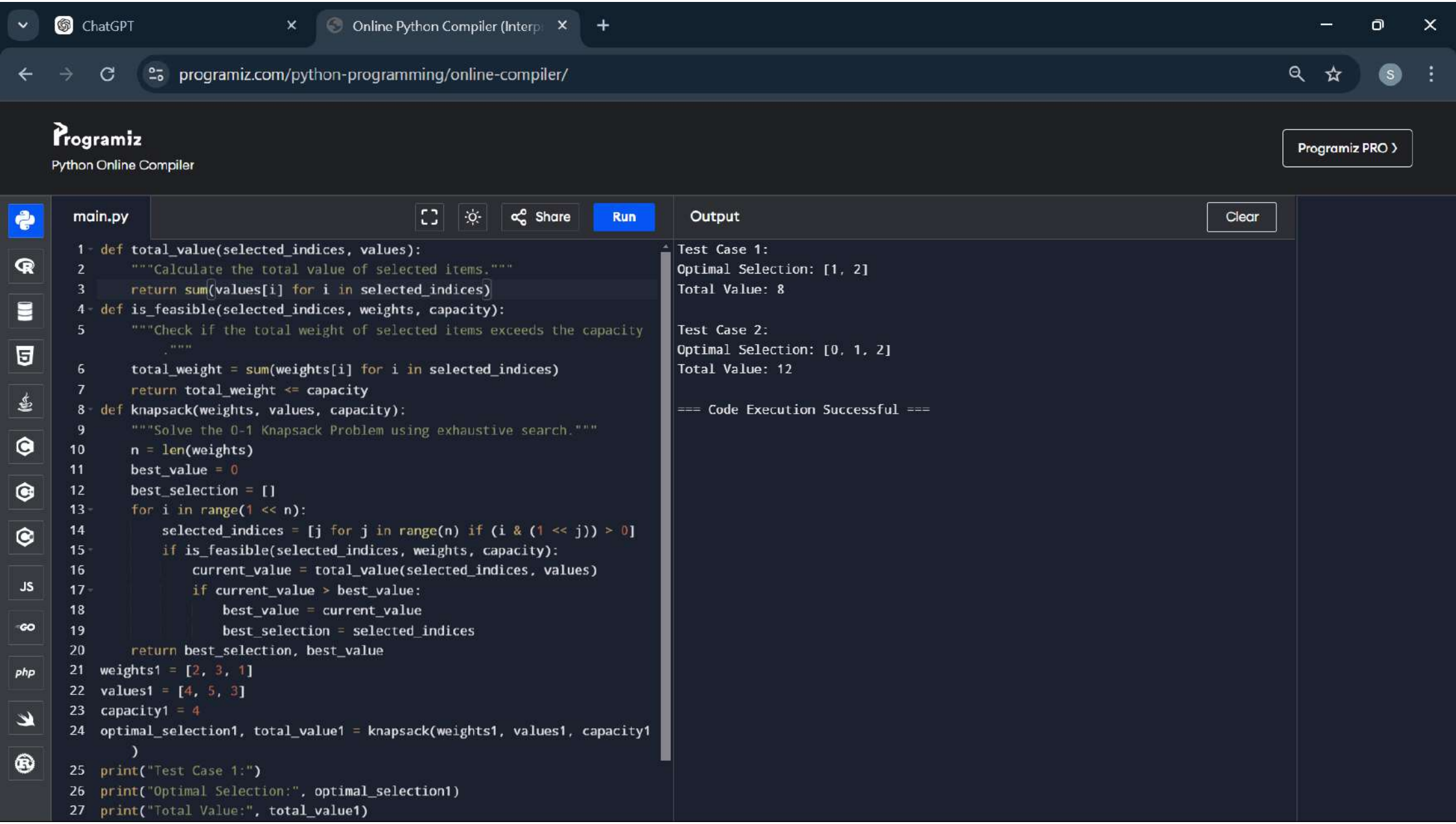
```python
import itertools
def total_cost(assignment, cost_matrix):
    """Calculate the total cost of a given assignment based on the cost matrix."""
    total = 0
    for worker, task in enumerate(assignment):
        total += cost_matrix[worker][task]
    return total
def assignment_problem(cost_matrix):
    """Solve the assignment problem using exhaustive search."""
    n = len(cost_matrix)
    min_cost = float('inf')
    best_assignment = None
    for assignment in itertools.permutations(range(n)):
        current_cost = total_cost(assignment, cost_matrix)
        if current_cost < min_cost:
            min_cost = current_cost
            best_assignment = assignment
    optimal_assignment = [(f'worker {i + 1}', f'task {best_assignment[i] + 1}') for i in range(n)]
    return optimal_assignment, min_cost
cost_matrix1 = [[3, 10, 7],
                [8, 5, 12],
                [4, 6, 9]]
optimal_assignment1, total_cost1 = assignment_problem(cost_matrix1)
print("Test Case 1:")
print("Optimal Assignment:", optimal_assignment1)
print("Total Cost:", total_cost1)
cost_matrix2 = [[15, 9, 4],
```

Output | Clear

```
Test Case 1:
Optimal Assignment: [('worker 1', 'task 3'), ('worker 2', 'task 2'), ('worker 3', 'task 1')]
Total Cost: 16

Test Case 2:
Optimal Assignment: [('worker 1', 'task 3'), ('worker 2', 'task 2'), ('worker 3', 'task 1')]
Total Cost: 17

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


ChatGPT
Online Python Compiler (Interp
programiz.com/python-programming/online-compiler/
Programiz
Python Online Compiler
Programiz PRO
main.py
Share
Run
Output
Clear
def total_value(selected_indices, values):
"""Calculate the total value of selected items."""
return sum(values[i] for i in selected_indices)
def is_feasible(selected_indices, weights, capacity):
"""Check if the total weight of selected items exceeds the capacity."""
total_weight = sum(weights[i] for i in selected_indices)
return total_weight <= capacity
def knapsack(weights, values, capacity):
"""Solve the 0-1 Knapsack Problem using exhaustive search."""
n = len(weights)
best_value = 0
best_selection = []
for i in range(1 << n):
selected_indices = [j for j in range(n) if (i & (1 << j)) > 0]
if is_feasible(selected_indices, weights, capacity):
current_value = total_value(selected_indices, values)
if current_value > best_value:
best_value = current_value
best_selection = selected_indices
return best_selection, best_value
weights1 = [2, 3, 1]
values1 = [4, 5, 3]
capacity1 = 4
optimal_selection1, total_value1 = knapsack(weights1, values1, capacity1)
print("Test Case 1:")
print("Optimal Selection:", optimal_selection1)
print("Total Value:", total_value1)
Test Case 1:
Optimal Selection: [1, 2]
Total Value: 8
Test Case 2:
Optimal Selection: [0, 1, 2]
Total Value: 12
=== Code Execution Successful ===