```
1 import math
 2 - def calculate_distance(point1, point2):
        return math.sqrt((point1[0] - point2[0]) ** 2 - (point1[1] -
            point2[1]) ** 2)
4 - def closest_pair(points):
        min_distance = float('inf')
        closest_points = (None, None)
       for i in range(len(points)):
            for j in range(i + 1, len(points)):
 8 +
                distance = calculate_distance(points[i], points[j])
                if distance < min_distance:</pre>
10 -
                    min_distance = distance
11
12
                    closest_points = (points[i], points[j])
        return closest_points, min_distance
13
    points = [(1, 2), (4, 5), (7, 8), (3, 1)]
15 closest_points, min_distance = closest_pair(points)
16 print(f"Closest pair: {closest_points[0]} - {closest_points[1]}")
17 print(f"Minimum distance: {min_distance}")
18
```

main.py

[] Share

Output

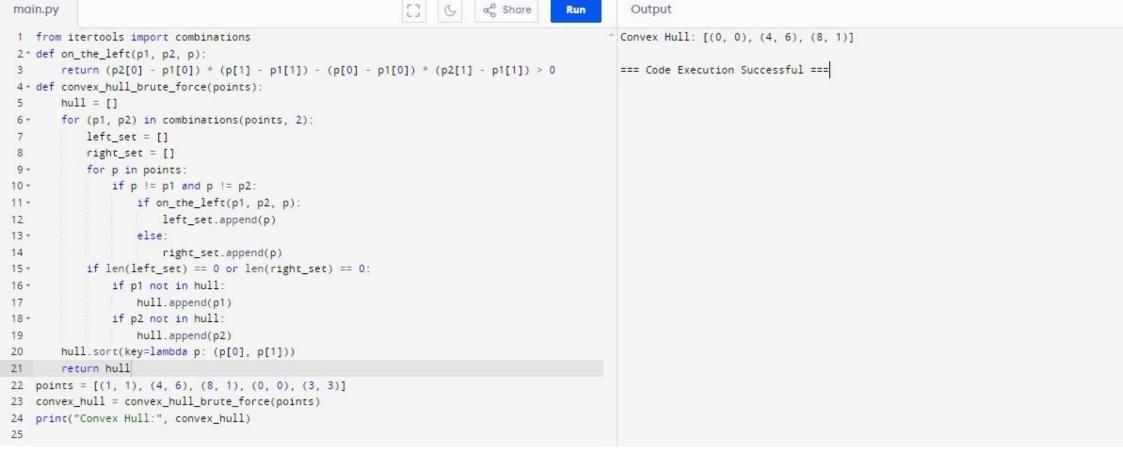
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979

=== Code Execution Successful ===

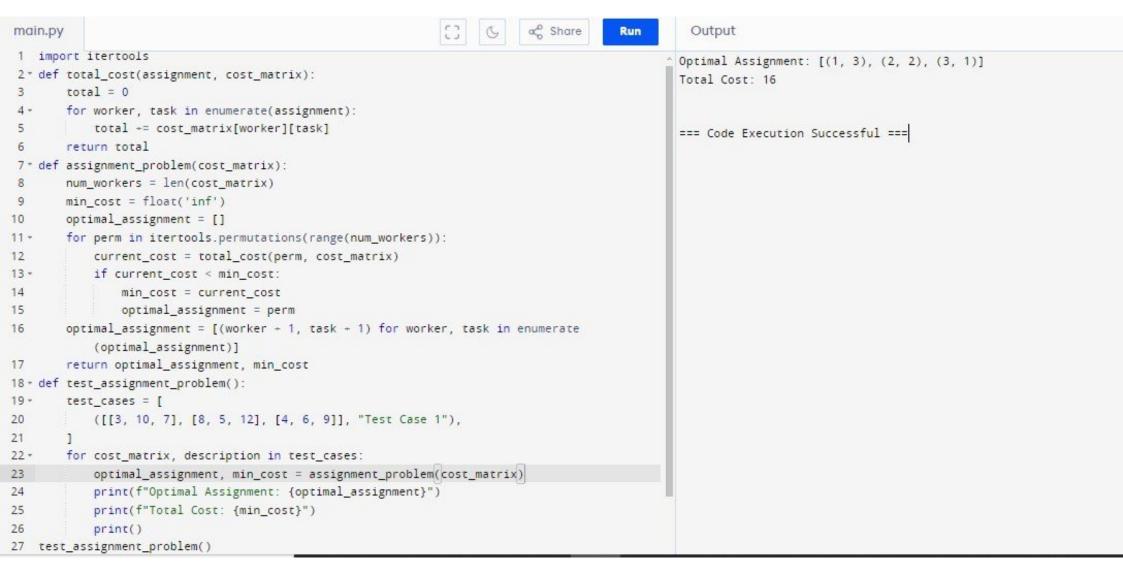
```
Output
main.py
 1 import math
                                                                                    Closest pair: (1, 2) - (3, 1)
                                                                                    Minimum distance: 2.23606797749979
 2 from itertools import combinations
 3 - def distance(point1, point2):
                                                                                    Convex hull: [(5, 3), (6, 6.5), (10, 0), (12.5, 7), (15, 3)]
        return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1])
                                                                                    === Code Execution Successful ===
 5 - def closest_pair_brute_force(points):
        min_distance = float('inf')
        closest_pair = (None, None)
       num_points = len(points)
 9+
       for i in range(num_points):
10 +
            for j in range(i + 1, num_points):
                d = distance(points[i], points[j])
11
12 -
               if d < min_distance:</pre>
13
                   min distance = d
14
                   closest_pair = (points[i], points[j])
15
        return closest_pair, min_distance
16 - def on_the_left(p1, p2, p):
17
        return (p2[0] - p1[0]) * (p[1] - p1[1]) - (p[0] - p1[0]) * (p2[1] - p1[1])
           ) > 0
18 - def convex_hull_brute_force(points):
       hull = []
19
       for (p1, p2) in combinations(points, 2):
20 -
21
           left_set = []
22
           right_set = []
```

```
right_set = []
        for p in points:
            if p != p1 and p != p2:
                if on_the_left(p1, p2, p):
                   left_set.append(p)
                else:
                    right_set.append(p)
        if len(left_set) == 0 or len(right_set) == 0:
            if p1 not in hull:
                hull.append(p1)
            if p2 not in hull:
                hull.append(p2)
    return hull
points_closest_pair = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_pair, min_distance = closest_pair_brute_force(points_closest_pair)
print(f"Closest pair: {closest_pair[0]} - {closest_pair[1]}")
print(f"Minimum distance: {min_distance}")
points_convex_hull = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7),
    (6, 6.5), (7.5, 4.5)]
convex_hull = convex_hull_brute_force(points_convex_hull)
convex_hull.sort(key=lambda p: (p[0], p[1]))
print("Convex hull:", convex_hull)
```

```
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979
Convex hull: [(5, 3), (6, 6.5), (10, 0), (12.5, 7), (15, 3)]
=== Code Execution Successful ===
```



```
α<sup>0</sup> Share
                                                                                                   Output
main.py
                                                                                         Run
1 import itertools
                                                                                                 Test Case 1:
 2 import math
                                                                                                 Shortest Distance: 16.969112047670894
3 - def distance(city1, city2):
                                                                                                 Shortest Path: [(1, 2), (7, 1), (4, 5), (3, 6), (1, 2)]
        return math.sqrt((city1[0] - city2[0]) ** 2 + (city1[1] - city2[1]) ** 2)
 5 * def tsp(cities):
        min_distance = float('inf')
                                                                                                 === Code Execution Successful ===
        shortest_path = []
       for perm in itertools.permutations(cities[1:]):
            current_path = [cities[0]] + list(perm) + [cities[0]]
10
            current_distance = 0
           for i in range(len(current_path) - 1):
11 -
                current_distance += distance(current_path[i], current_path[i + 1])
12
13 -
            if current_distance < min_distance:
                min_distance = current_distance
14
15
                shortest_path = current_path
16
        return min_distance, shortest_path
17 def test_tsp():
18 -
        test_cases = [
           ([(1, 2), (4, 5), (7, 1), (3, 6)], "Test Case 1"),
19
20
       for cities, description in test_cases:
21 -
22
           min_distance, shortest_path = tsp(cities)
23
            print(f"{description}:")
            print(f"Shortest Distance: {min distance}")
24
25
            print(f"Shortest Path: {shortest_path}")
26
            print()
27 test_tsp()
```



```
Optimal Selection: [1, 2]
1 import itertools
2 * def total_value(selected_items, values):
                                                                                                 Total Value: 8
       return sum(values[i] for i in selected_items)
4 - def is_feasible(selected_items, weights, capacity):
       return sum(weights[i] for i in selected_items) <= capacity
                                                                                                 === Code Execution Successful ===
6 - def knapsack_problem(weights, values, capacity):
       num_items = len(weights)
       max value = 0
       optimal_selection = []
9
       for r in range(num_items + 1):
10 -
           for combination in itertools.combinations(range(num_items), r):
11 -
12 -
                if is_feasible(combination, weights, capacity):
                    current value = total value(combination, values)
13
                    if current_value > max_value:
14 -
                        max_value = current_value
15
                       optimal_selection = combination
16
        return list(optimal selection), max value
17
18 - def test_knapsack_problem():
19 -
        test cases = [
           ([2, 3, 1], [4, 5, 3], 4, "Test Case 1"),
20
21
       for weights, values, capacity, description in test_cases:
22 -
           optimal_selection, max_value = knapsack_problem(weights, values, capacity)
23
           print(f"Optimal Selection: {optimal_selection}")
24
           print(f"Total Value: {max_value}")
25
26
           print()
27 test_knapsack_problem()
```

α⁰ Share

Run

main.py

Output