




main.py



Share


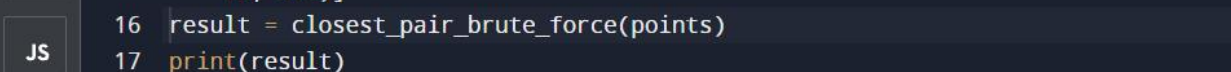
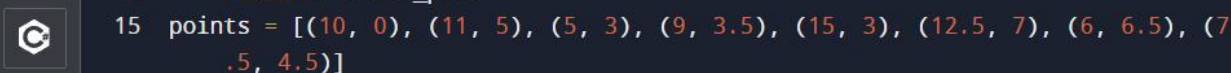
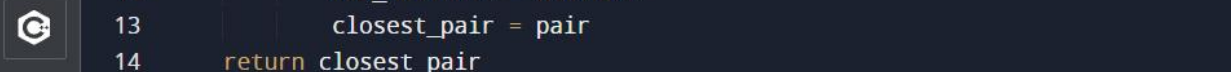
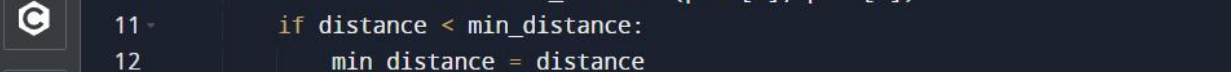
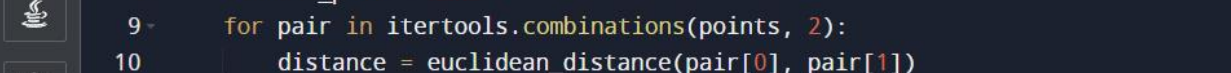
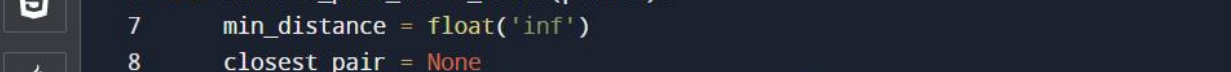
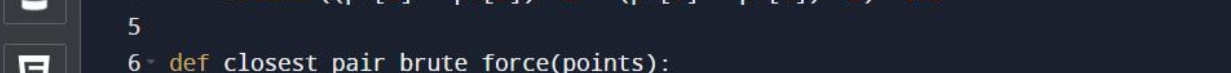
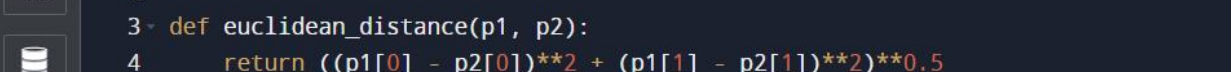
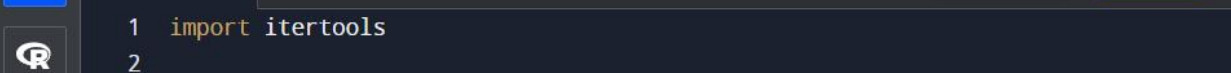

Run

```
1 import itertools
2 import math
3
4 points = [(1, 2), (4, 5), (7, 8), (3, 1)]
5 closest_pair, min_distance = min(((p1, p2), math.dist(p1, p2)) for p1, p2 in
    itertools.combinations(points, 2))
6 print(f"Closest pair: {closest_pair} Minimum distance: {min_distance}")
7
```

Output

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

=== Code Execution Successful ===



main.py

Share









Run

```
1 import itertools
2
3 def euclidean_distance(p1, p2):
4     return ((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)**0.5
5
6 def closest_pair_brute_force(points):
7     min_distance = float('inf')
8     closest_pair = None
9     for pair in itertools.combinations(points, 2):
10         distance = euclidean_distance(pair[0], pair[1])
11         if distance < min_distance:
12             min_distance = distance
13             closest_pair = pair
14     return closest_pair
15 points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (7
16         .5, 4.5)]
17 result = closest_pair_brute_force(points)
18 print(result)
```

Output

```
((9, 3.5), (7.5, 4.5))



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






JS

GO

php



main.py

 Share














Run

```
1 from itertools import combinations
2
3 def orientation(p, q, r):
4     val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1])
5     if val == 0:
6         return 0
7     return 1 if val > 0 else -1
8
9 def convex_hull(points):
10     n = len(points)
11     if n < 3:
12         return points
13
14     hull = []
15     for p, q in combinations(points, 2):
16         side = 0
17         for r in points:
18             side = max(side, orientation(p, q, r))
19         if side == 0:
20             hull.extend([p, q])
21
22     return sorted(list(set(hull)))
23 points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
24 convex_hull_points = convex_hull(points)
25 print(convex_hull_points)
26
```




Output

[(0, 0), (4, 6)]

=== Code Execution Successful ===



main.py

 Share

Run

```
1 import itertools
2
3 def distance(city1, city2):
4     return ((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)**0.5
5
6 def tsp(cities):
7     min_distance = float('inf')
8     shortest_path = None
9
10    for perm in itertools.permutations(cities[1:]):
11        total_distance = 0
12        path = [cities[0]] + list(perm) + [cities[0]]
13
14        for i in range(len(path) - 1):
15            total_distance += distance(path[i], path[i + 1])
16
17        if total_distance < min_distance:
18            min_distance = total_distance
19            shortest_path = path
20
21    return min_distance, shortest_path
22 cities = [(0, 0), (1, 2), (3, 1), (5, 3)]
23 min_dist, shortest_route = tsp(cities)
24 print(f"Minimum Distance: {min_dist}")
25 print(f"Shortest Route: {shortest_route}")
26
```

Output

Minimum Distance: 12.349878388032021
Shortest Route: [(0, 0), (1, 2), (5, 3), (3, 1), (0, 0)]

=== Code Execution Successful ===

main.py



Share

Run

```
1 from itertools import permutations
2
3 def total_cost(assignment, cost_matrix):
4     return sum(cost_matrix[worker][task] for worker, task in assignment)
5
6 def assignment_problem(cost_matrix):
7     workers = range(len(cost_matrix))
8     min_cost = float('inf')
9     optimal_assignment = None
10
11     for perm in permutations(workers):
12         assignment = list(zip(perm, range(len(cost_matrix))))
13         cost = total_cost(assignment, cost_matrix)
14
15         if cost < min_cost:
16             min_cost = cost
17             optimal_assignment = assignment
18
19     return optimal_assignment, min_cost
20 cost_matrix_1 = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
21 cost_matrix_2 = [[15, 9, 4], [8, 7, 18], [6, 12, 11]]
22
23 optimal_assignment_1, total_cost_1 = assignment_problem(cost_matrix_1)
24 optimal_assignment_2, total_cost_2 = assignment_problem(cost_matrix_2)
25 print("Test Case 1:")
26 print("Optimal Assignment:", [(f"worker {worker + 1}", f"task {task + 1}")
27     for worker, task in optimal_assignment_1])
28 print("Total Cost:", total_cost_1)
```

Output

Clear

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

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

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

main.py



Share

Run

Output

```
1 def total_value(items, values):
2     return sum(values[i] for i in items)
3
4 def is_feasible(items, weights, capacity):
5     return sum(weights[i] for i in items) <= capacity
6 items = 3
7 weights = [2, 3, 1]
8 values = [4, 5, 3]
9 capacity = 4
10
11 max_value = 0
12 optimal_selection = []
13
14 for i in range(1 << items):
15     selected_items = [j for j in range(items) if i & (1 << j)]
16     if is_feasible(selected_items, weights, capacity):
17         total = total_value(selected_items, values)
18         if total > max_value:
19             max_value = total
20             optimal_selection = selected_items
21
22 print("Optimal Selection:", optimal_selection)
23 print("Total Value:", max_value)
24
```

Optimal Selection: [1, 2]

Total Value: 8

=== Code Execution Successful ===