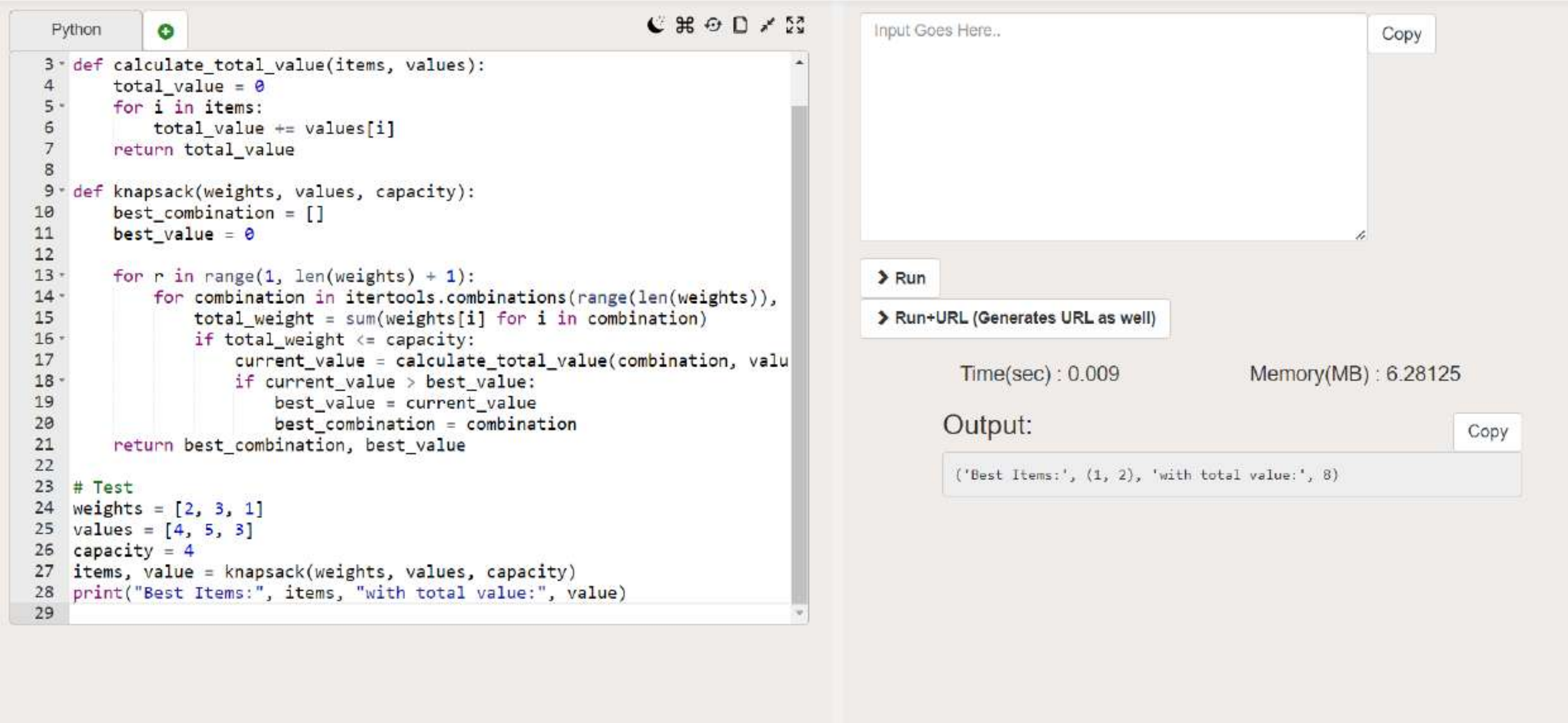
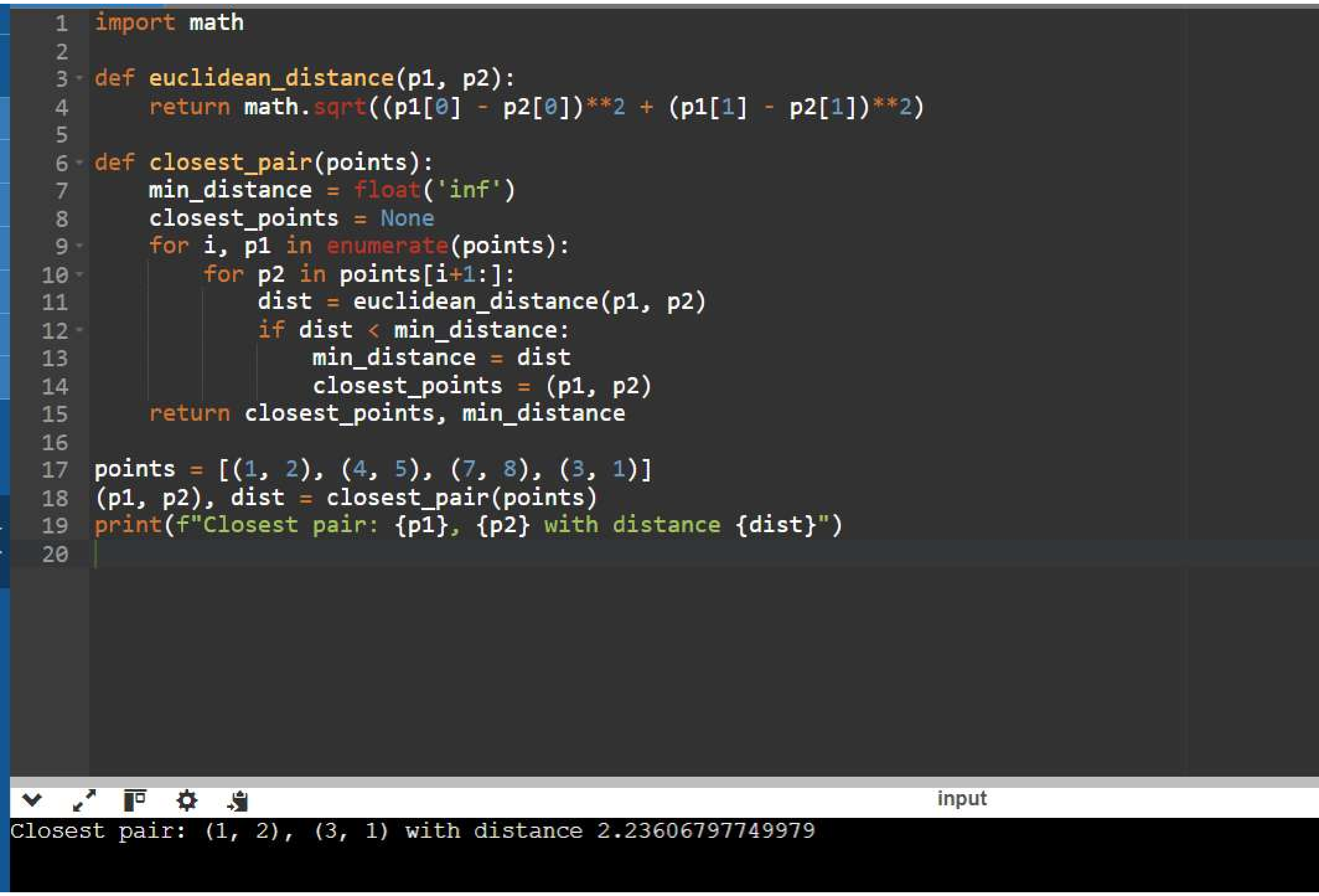


```python
def calculate_total_value(items, values):
    total_value = 0
    for i in items:
        total_value += values[i]
    return total_value

def knapsack(weights, values, capacity):
    best_combination = []
    best_value = 0

    for r in range(1, len(weights) + 1):
        for combination in itertools.combinations(range(len(weights)),
            total_weight = sum(weights[i] for i in combination)
            if total_weight <= capacity:
                current_value = calculate_total_value(combination, valu
                if current_value > best_value:
                    best_value = current_value
                    best_combination = combination
    return best_combination, best_value

# Test
weights = [2, 3, 1]
values = [4, 5, 3]
capacity = 4
items, value = knapsack(weights, values, capacity)
print("Best Items:", items, "with total value:", value)
```

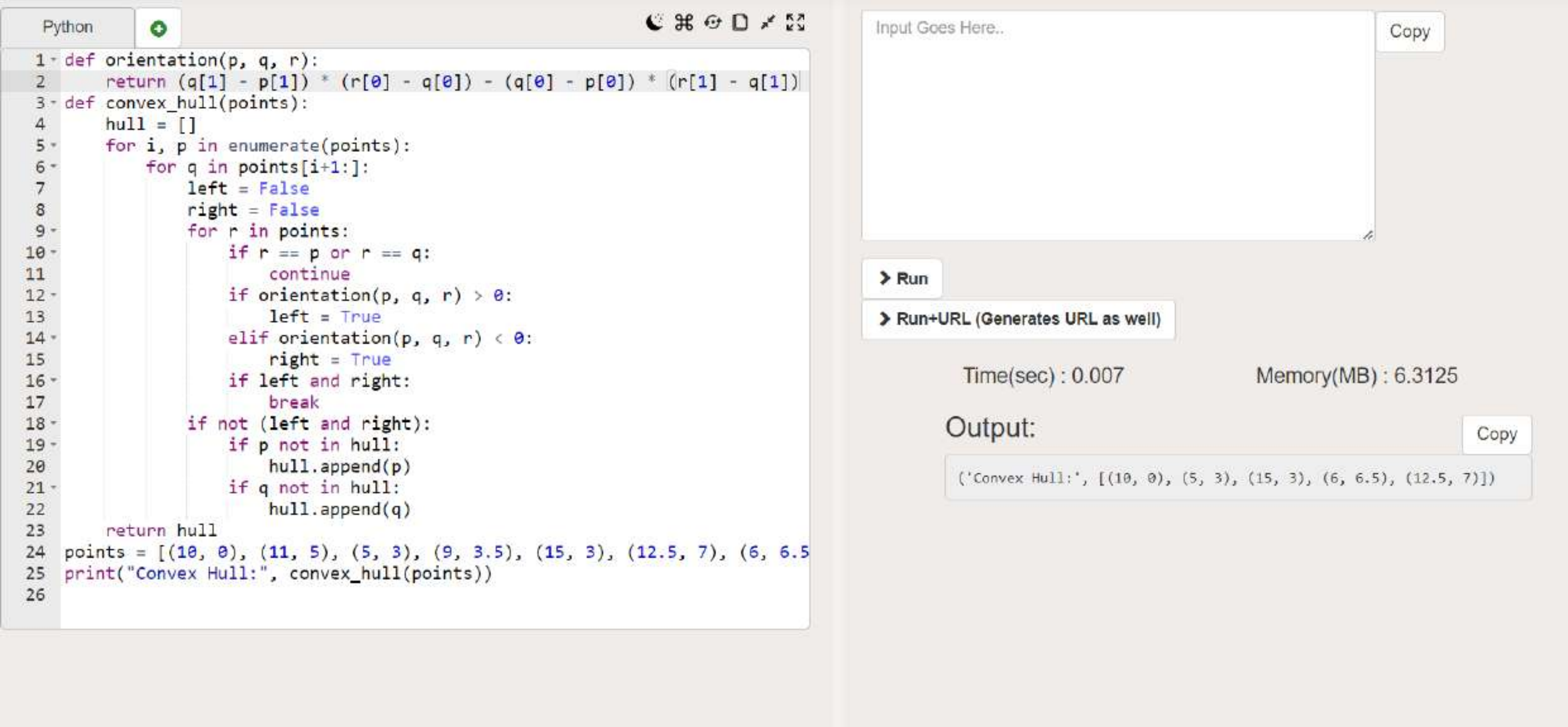
Time(sec) : 0.009   Memory(MB) : 6.28125

```
('Best Items:', (1, 2), 'with total value:', 8)
```



```python
import math

def euclidean_distance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def closest_pair(points):
    min_distance = float('inf')
    closest_points = None
    for i, p1 in enumerate(points):
        for p2 in points[i+1:]:
            dist = euclidean_distance(p1, p2)
            if dist < min_distance:
                min_distance = dist
                closest_points = (p1, p2)
    return closest_points, min_distance

points = [(1, 2), (4, 5), (7, 8), (3, 1)]
(p1, p2), dist = closest_pair(points)
print(f"Closest pair: {p1}, {p2} with distance {dist}")

```

```
Closest pair: (1, 2), (3, 1) with distance 2.23606797749979
```



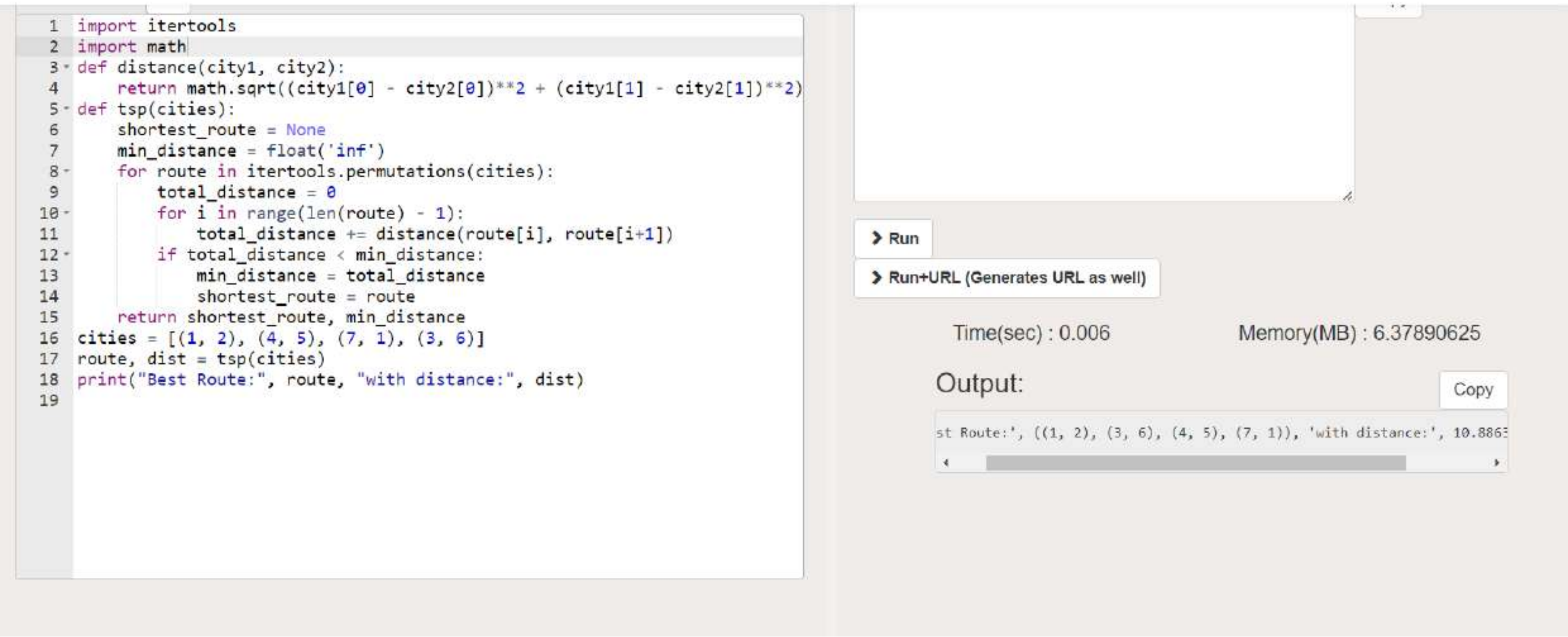
```python
def orientation(p, q, r):
    return (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1])
def convex_hull(points):
    hull = []
    for i, p in enumerate(points):
        for q in points[i+1:]:
            left = False
            right = False
            for r in points:
                if r == p or r == q:
                    continue
                if orientation(p, q, r) > 0:
                    left = True
                elif orientation(p, q, r) < 0:
                    right = True
                if left and right:
                    break
            if not (left and right):
                if p not in hull:
                    hull.append(p)
                if q not in hull:
                    hull.append(q)
    return hull
points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5
print("Convex Hull:", convex_hull(points))
```

Time(sec) : 0.007

Memory(MB) : 6.3125

Output:

```
('Convex Hull:', [(10, 0), (5, 3), (15, 3), (6, 6.5), (12.5, 7)])
```

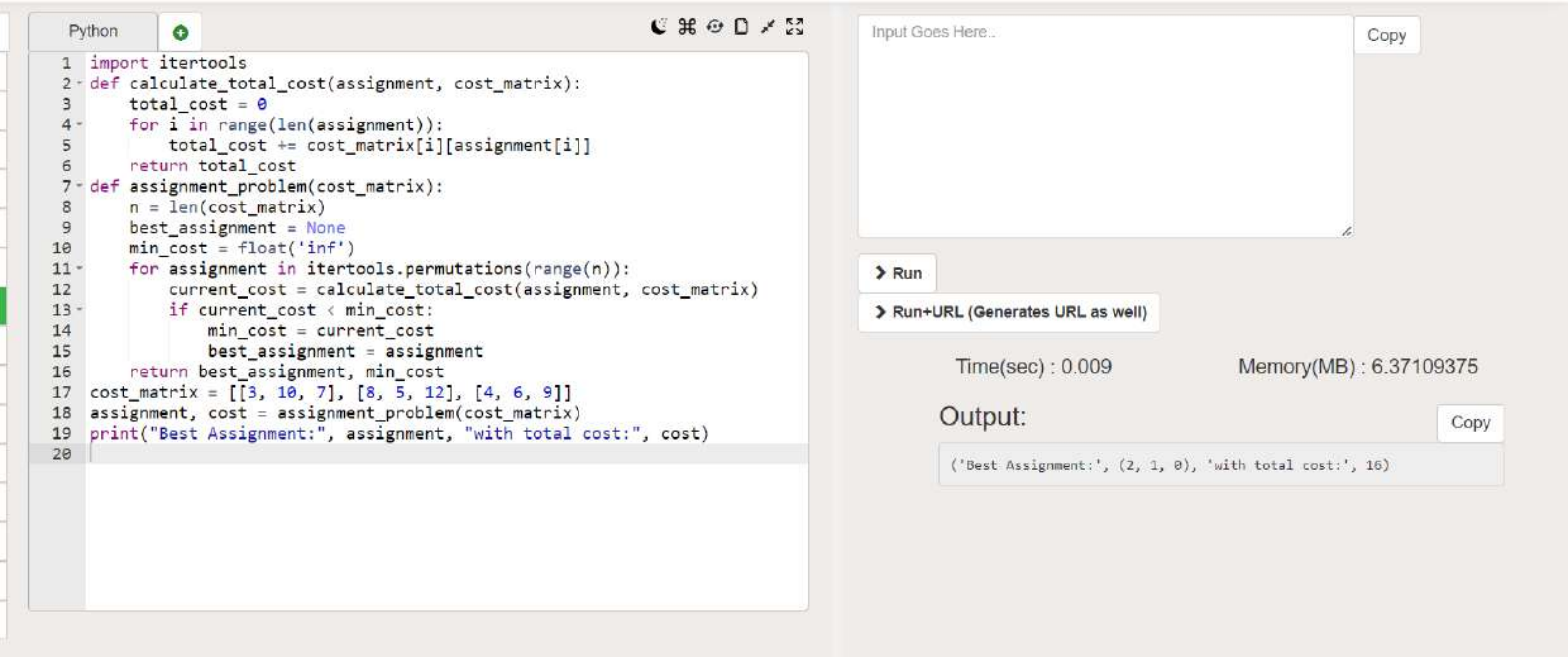


```python
import itertools
import math
def distance(city1, city2):
    return math.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)
def tsp(cities):
    shortest_route = None
    min_distance = float('inf')
    for route in itertools.permutations(cities):
        total_distance = 0
        for i in range(len(route) - 1):
            total_distance += distance(route[i], route[i+1])
        if total_distance < min_distance:
            min_distance = total_distance
            shortest_route = route
    return shortest_route, min_distance
cities = [(1, 2), (4, 5), (7, 1), (3, 6)]
route, dist = tsp(cities)
print("Best Route:", route, "with distance:", dist)
```

Time(sec) : 0.006

Memory(MB) : 6.37890625

```
st Route:', ((1, 2), (3, 6), (4, 5), (7, 1)), 'with distance:', 10.8863
```



```python
import itertools
def calculate_total_cost(assignment, cost_matrix):
    total_cost = 0
    for i in range(len(assignment)):
        total_cost += cost_matrix[i][assignment[i]]
    return total_cost
def assignment_problem(cost_matrix):
    n = len(cost_matrix)
    best_assignment = None
    min_cost = float('inf')
    for assignment in itertools.permutations(range(n)):
        current_cost = calculate_total_cost(assignment, cost_matrix)
        if current_cost < min_cost:
            min_cost = current_cost
            best_assignment = assignment
    return best_assignment, min_cost
cost_matrix = [[3, 10, 7], [8, 5, 12], [4, 6, 9]]
assignment, cost = assignment_problem(cost_matrix)
print("Best Assignment:", assignment, "with total cost:", cost)
```

Time(sec) : 0.009

Memory(MB) : 6.37109375

```
('Best Assignment:', (2, 1, 0), 'with total cost:', 16)
```