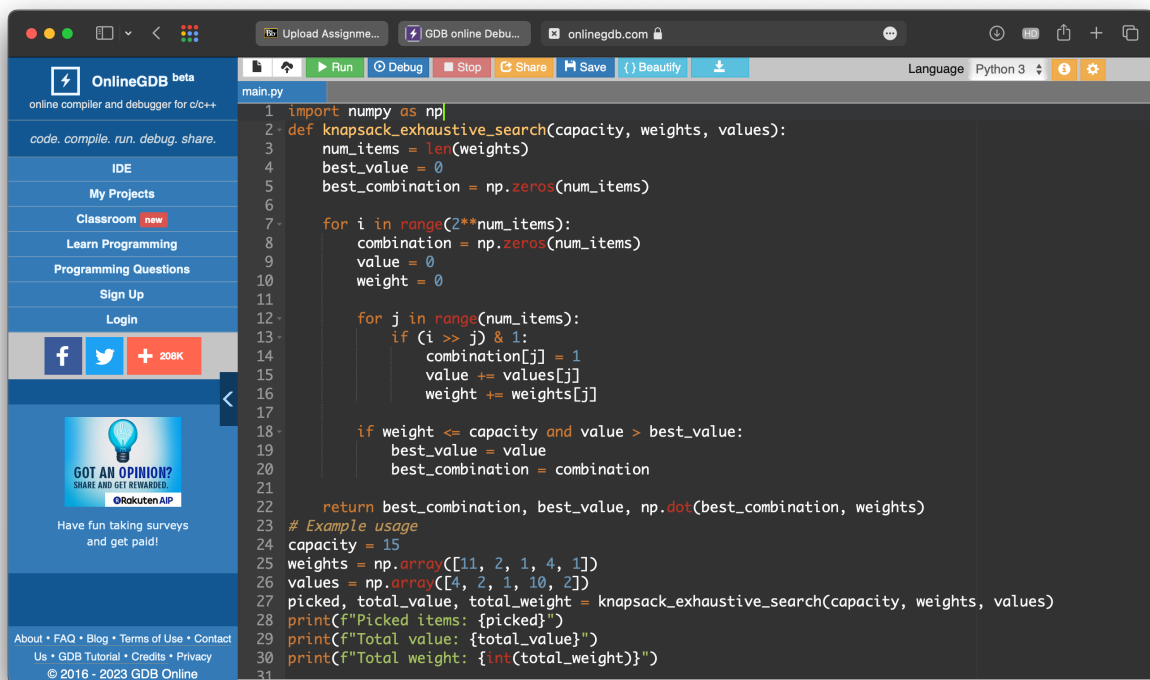


Lab 3
Student name: Khalid Nimri
Student ID: 2140145

Exercise 1: (Exhaustive Search: Knapsack Problem)

The screenshot shows the OnlineGDB web interface. On the left is a sidebar with navigation links like 'My Projects', 'Classroom', 'Learn Programming', etc. The main area displays a Python script for an exhaustive search knapsack algorithm. The script defines a function 'knapsack_exhaustive_search' that takes capacity, weights, and values as input. It iterates through all possible combinations of items (represented by a binary vector 'combination') and calculates the total value and weight for each. The combination with the highest value that does not exceed the capacity is returned. The script also includes example usage with specific capacity, weights, and values, and prints the results.

```
1 import numpy as np
2 def knapsack_exhaustive_search(capacity, weights, values):
3     num_items = len(weights)
4     best_value = 0
5     best_combination = np.zeros(num_items)
6
7     for i in range(2**num_items):
8         combination = np.zeros(num_items)
9         value = 0
10        weight = 0
11
12        for j in range(num_items):
13            if (i >> j) & 1:
14                combination[j] = 1
15                value += values[j]
16                weight += weights[j]
17
18        if weight <= capacity and value > best_value:
19            best_value = value
20            best_combination = combination
21
22    return best_combination, best_value, np.dot(best_combination, weights)
23
24 # Example usage
25 capacity = 15
26 weights = np.array([11, 2, 1, 4, 1])
27 values = np.array([4, 2, 1, 10, 2])
28 picked, total_value, total_weight = knapsack_exhaustive_search(capacity, weights, values)
29 print(f"Picked items: {picked}")
30 print(f"Total value: {total_value}")
31 print(f"Total weight: {int(total_weight)}")
```

```
import numpy as np
def knapsack_exhaustive_search(capacity, weights, values):
    num_items = len(weights)
    best_value = 0
    best_combination = np.zeros(num_items)

    for i in range(2**num_items):
        combination = np.zeros(num_items)
        value = 0
        weight = 0

        for j in range(num_items):
            if (i >> j) & 1:
                combination[j] = 1
                value += values[j]
                weight += weights[j]
```

```
if weight <= capacity and value > best_value:
    best_value = value
    best_combination = combination
```

```
return best_combination, best_value, np.dot(best_combination, weights)
```

Example usage

```
capacity = 15
```

```
weights = np.array([11, 2, 1, 4, 1])
```

```
values = np.array([4, 2, 1, 10, 2])
```

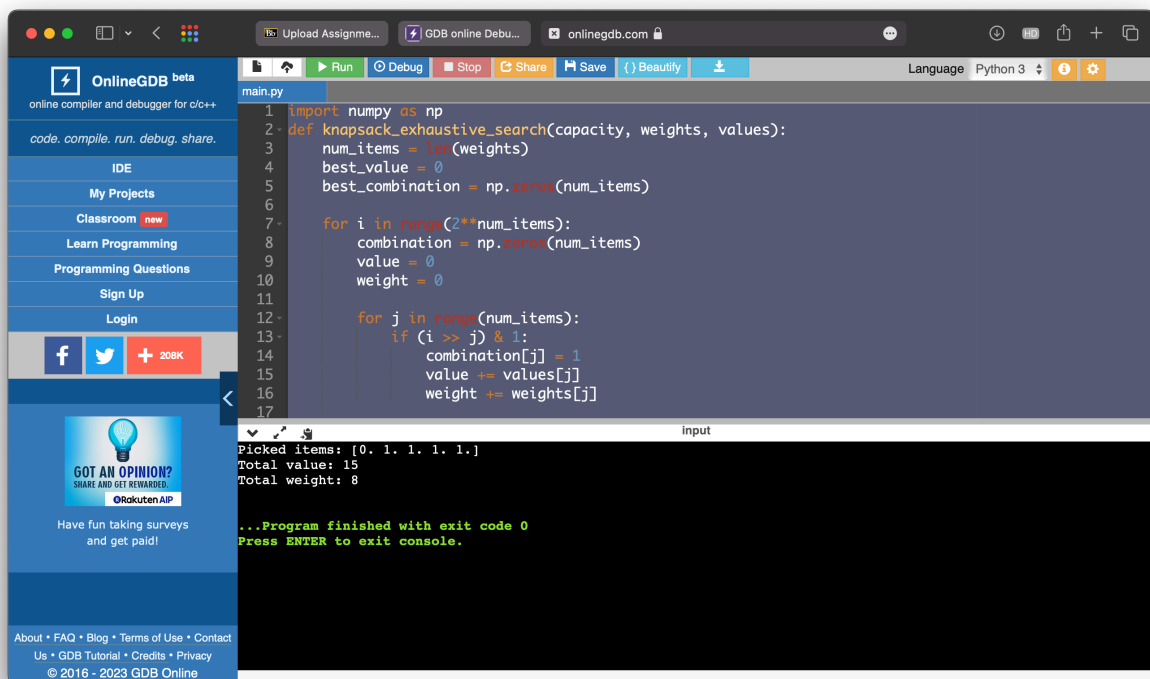
```
picked, total_value, total_weight = knapsack_exhaustive_search(capacity, weights, values)
```

```
print(f"Picked items: {picked}")
```

```
print(f"Total value: {total_value}")
```

```
print(f"Total weight: {int(total_weight)}")
```

Output:



```
1 import numpy as np
2 def knapsack_exhaustive_search(capacity, weights, values):
3     num_items = len(weights)
4     best_value = 0
5     best_combination = np.zeros(num_items)
6
7     for i in range(2**num_items):
8         combination = np.zeros(num_items)
9         value = 0
10        weight = 0
11
12        for j in range(num_items):
13            if (i >> j) & 1:
14                combination[j] = 1
15                value += values[j]
16                weight += weights[j]
17
18        if weight <= capacity and value > best_value:
19            best_value = value
20            best_combination = combination
21
22    return best_combination, best_value, np.dot(best_combination, weights)
23
24 capacity = 15
25 weights = np.array([11, 2, 1, 4, 1])
26 values = np.array([4, 2, 1, 10, 2])
27 picked, total_value, total_weight = knapsack_exhaustive_search(capacity, weights, values)
28 print(f"Picked items: {picked}")
29 print(f"Total value: {total_value}")
30 print(f"Total weight: {int(total_weight)}")
```

Picked items: [0. 1. 1. 1. 1.]
Total value: 15
Total weight: 8

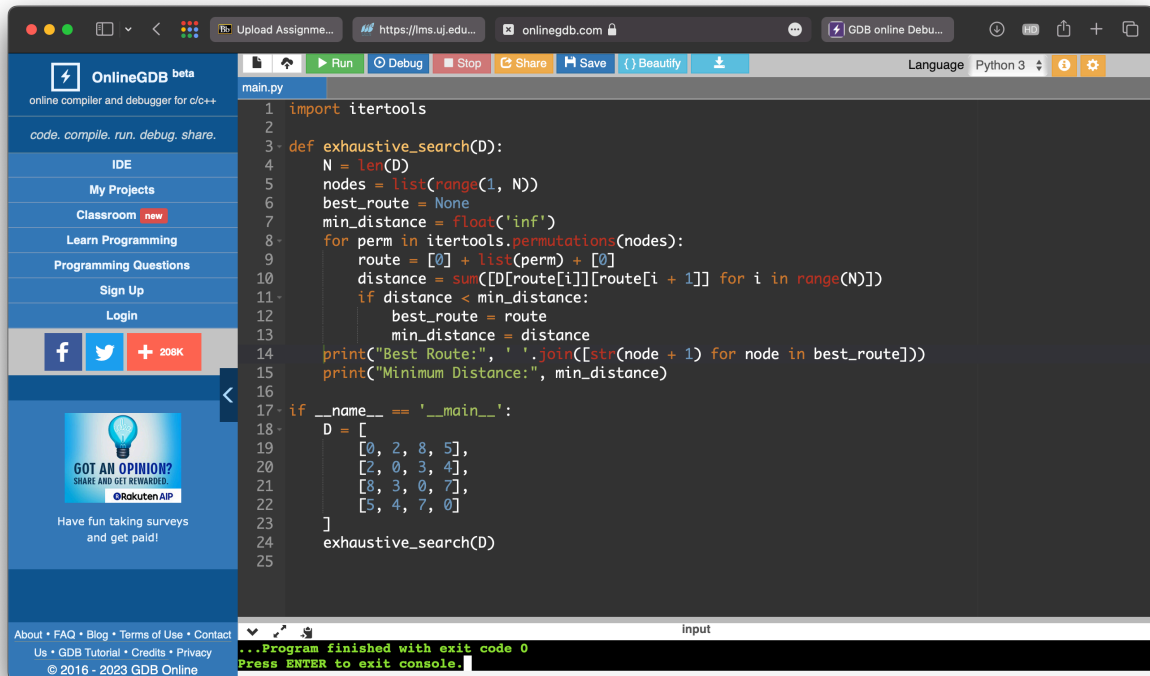
...Program finished with exit code 0
Press ENTER to exit console.

Items Picked $P = [0. 1. 1. 1. 1]$

Total value of picked items = 15

Total weight of picked items = 8

Exercise 2: (Exhaustive Search: Travel Salesperson Problem)



```
1 import itertools
2
3 def exhaustive_search(D):
4     N = len(D)
5     nodes = list(range(1, N))
6     best_route = None
7     min_distance = float('inf')
8     for perm in itertools.permutations(nodes):
9         route = [0] + list(perm) + [0]
10        distance = sum([D[route[i]][route[i + 1]] for i in range(N)])
11        if distance < min_distance:
12            best_route = route
13            min_distance = distance
14    print("Best Route:", ' '.join([str(node + 1) for node in best_route]))
15    print("Minimum Distance:", min_distance)
16
17 if __name__ == '__main__':
18     D = [
19         [0, 2, 8, 5],
20         [2, 0, 3, 4],
21         [8, 3, 0, 7],
22         [5, 4, 7, 0]
23     ]
24     exhaustive_search(D)
25
```

```
import itertools
```

```
def exhaustive_search(D):
```

```
    N = len(D)
```

```
    nodes = list(range(1, N))
```

```
    best_route = None
```

```
    min_distance = float('inf')
```

```
    for perm in itertools.permutations(nodes):
```

```
        route = [0] + list(perm) + [0]
```

```
        distance = sum([D[route[i]][route[i + 1]] for i in range(N)])
```

```
        if distance < min_distance:
```

```
            best_route = route
```

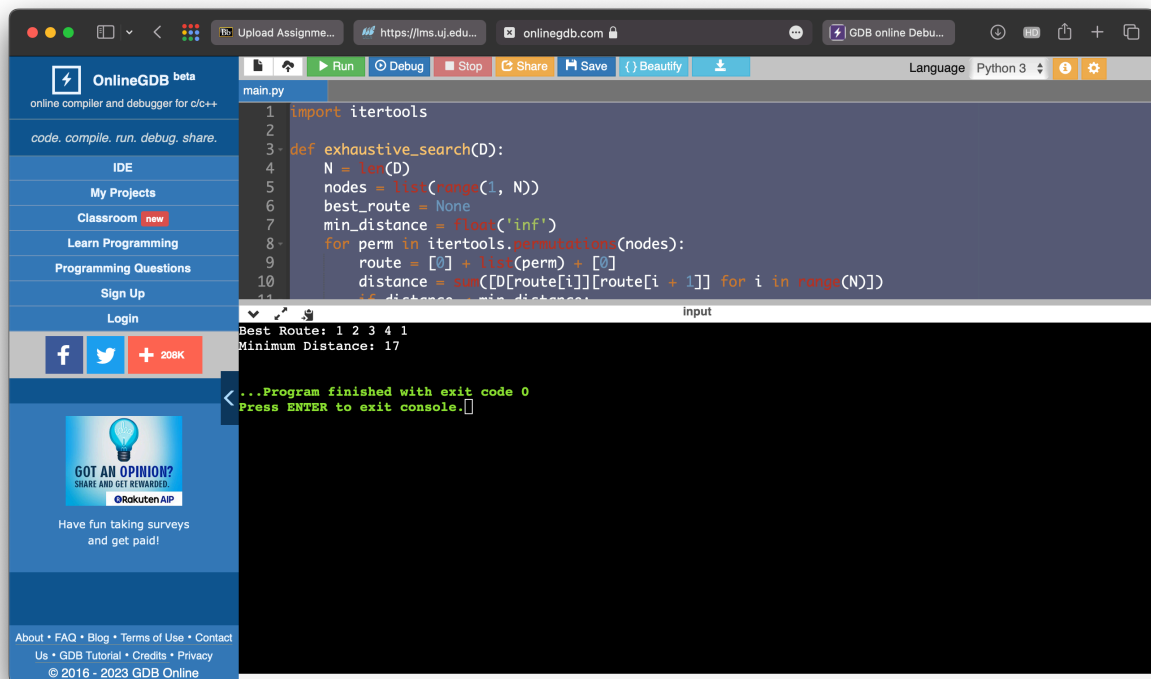
```
            min_distance = distance
```

```
    print("Best Route:", ' '.join([str(node + 1) for node in best_route]))
```

```
    print("Minimum Distance:", min_distance)
```

```
if __name__ == '__main__':  
    D = [  
        [0, 2, 8, 5],  
        [2, 0, 3, 4],  
        [8, 3, 0, 7],  
        [5, 4, 7, 0]  
    ]  
    exhaustive_search(D)
```

Output:



The screenshot shows the OnlineGDB web interface. The left sidebar contains navigation links: IDE, My Projects, Classroom (marked 'new'), Learn Programming, Programming Questions, Sign Up, and Login. The main area displays a Python script named 'main.py' with the following code:

```
1 import itertools  
2  
3 def exhaustive_search(D):  
4     N = len(D)  
5     nodes = list(range(1, N))  
6     best_route = None  
7     min_distance = float('inf')  
8     for perm in itertools.permutations(nodes):  
9         route = [0] + list(perm) + [0]  
10        distance = sum([D[route[i]][route[i + 1]] for i in range(N)])  
11        if distance < min_distance:  
12            min_distance = distance  
13            best_route = route
```

The output console shows the results of the program execution:

```
Best Route: 1 2 3 4 1  
Minimum Distance: 17  
...Program finished with exit code 0  
Press ENTER to exit console.
```

the best path: 1 2 3 4 1

minimum distance: 17