Lab Assignment: 01

———————————786———————————

Name: Muhammad Sherjeel Akhtar

Roll No: 20p-0101

Subject: Artificial Intelligence Lab

Submitted To Respected
Ma’am: Hurmat Hidayat

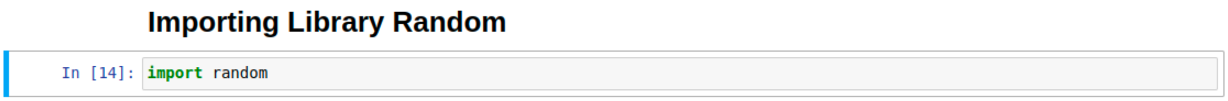
Section: BCS-6C

Importing Library Random:

Code:

```
import random
```

Visually:



Importing Library Random

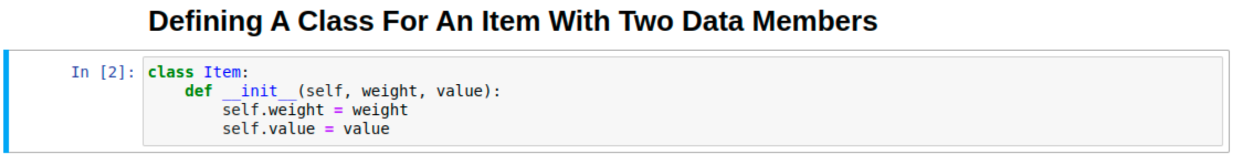
```
In [14]: import random
```

Defining A Class For An Item With Two Data Members:

Code:

```
class Item:
    def __init__(self, weight, value):
        self.weight = weight
        self.value = value
```

Visually:



Defining A Class For An Item With Two Data Members

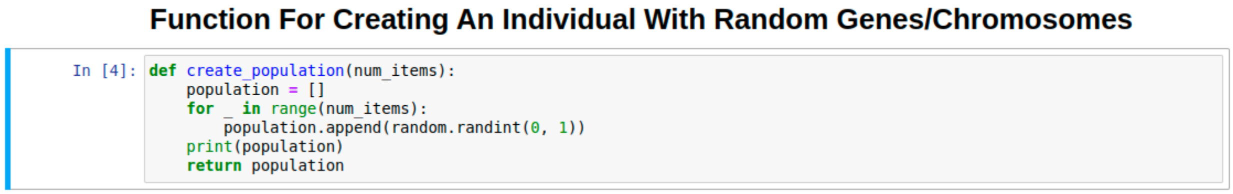
```
In [2]: class Item:
            def __init__(self, weight, value):
                self.weight = weight
                self.value = value
```

Function For Creating An Individual With Random Genes/Chromosomes:

Code:

```
def create_population(num_items):
    population = []
    for _ in range(num_items):
        population.append(random.randint(0, 1))
    print(population)
    return population
```

Visually:



Function For Creating An Individual With Random Genes/Chromosomes

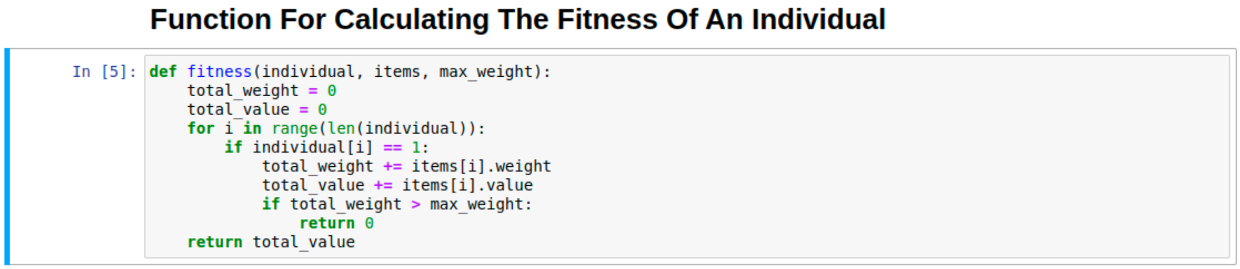
```
In [4]: def create_population(num_items):
            population = []
            for _ in range(num_items):
                population.append(random.randint(0, 1))
            print(population)
            return population
```

Function For Calculating The Fitness Of An Individual:

Code:

```
def fitness(individual, items, max_weight):
    total_weight = 0
    total_value = 0
    for i in range(len(individual)):
        if individual[i] == 1:
            total_weight += items[i].weight
            total_value += items[i].value
            if total_weight > max_weight:
                return 0
    return total_value
```

Visually:



Function For Calculating The Fitness Of An Individual

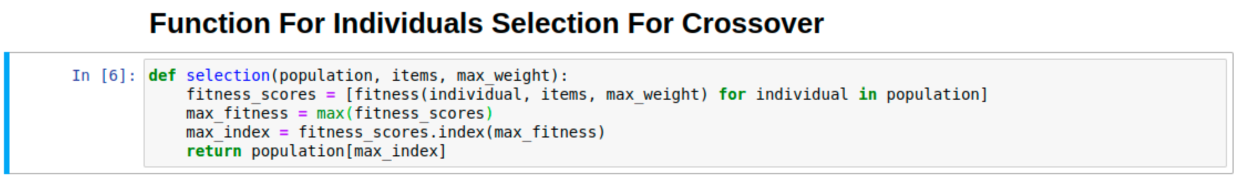
```
In [5]: def fitness(individual, items, max_weight):
            total_weight = 0
            total_value = 0
            for i in range(len(individual)):
                if individual[i] == 1:
                    total_weight += items[i].weight
                    total_value += items[i].value
                    if total_weight > max_weight:
                        return 0
            return total_value
```

Function For Individuals Selection For Crossover:

Code:

```
def selection(population, items, max_weight):
    fitness_scores = [fitness(individual, items, max_weight) for individual in population]
    max_fitness = max(fitness_scores)
    max_index = fitness_scores.index(max_fitness)
    return population[max_index]
```

Visually:



Function For Individuals Selection For Crossover

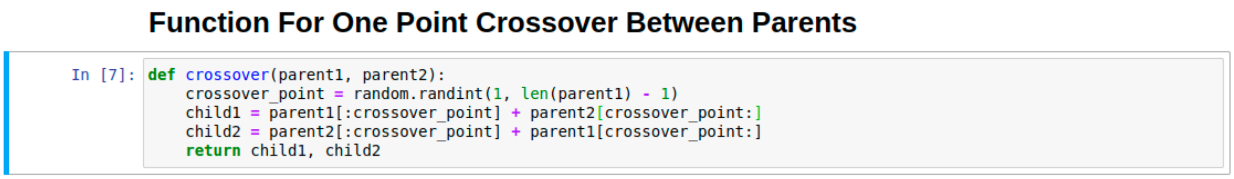
```
In [6]: def selection(population, items, max_weight):
            fitness_scores = [fitness(individual, items, max_weight) for individual in population]
            max_fitness = max(fitness_scores)
            max_index = fitness_scores.index(max_fitness)
            return population[max_index]
```

Function For One Point Crossover Between Parents:

Code:

```
def crossover(parent1, parent2):
    crossover_point = random.randint(1, len(parent1) - 1)
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2
```

Visually:



Function For One Point Crossover Between Parents

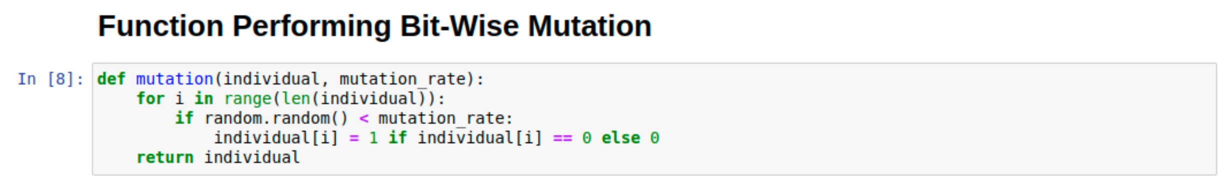
```
In [7]: def crossover(parent1, parent2):
            crossover_point = random.randint(1, len(parent1) - 1)
            child1 = parent1[:crossover_point] + parent2[crossover_point:]
            child2 = parent2[:crossover_point] + parent1[crossover_point:]
            return child1, child2
```

Function Performing Bit-Wise Mutation:

Code:

```
def mutation(individual, mutation_rate):
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            individual[i] = 1 if individual[i] == 0 else 0
    return individual
```

Visually:



Function Performing Bit-Wise Mutation

```
In [8]: def mutation(individual, mutation_rate):
            for i in range(len(individual)):
                if random.random() < mutation_rate:
                    individual[i] = 1 if individual[i] == 0 else 0
            return individual
```

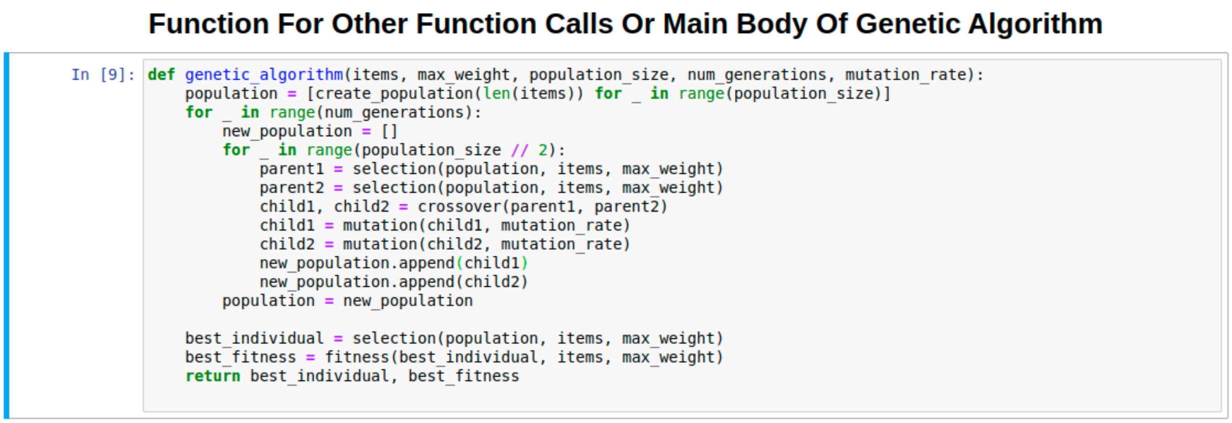
Function For Other Function Calls Or Main Body Of Genetic Algorithm:

Code:

```
def genetic_algorithm(items, max_weight, population_size, num_generations, mutation_rate):
    population = [create_population(len(items)) for _ in range(population_size)]
    for _ in range(num_generations):
        new_population = []
        for _ in range(population_size // 2):
            parent1 = selection(population, items, max_weight)
            parent2 = selection(population, items, max_weight)
            child1, child2 = crossover(parent1, parent2)
            child1 = mutation(child1, mutation_rate)
            child2 = mutation(child2, mutation_rate)
            new_population.append(child1)
            new_population.append(child2)
        population = new_population

    best_individual = selection(population, items, max_weight)
    best_fitness = fitness(best_individual, items, max_weight)
    return best_individual, best_fitness
```

Visually:



Function For Other Function Calls Or Main Body Of Genetic Algorithm

```
In [9]: def genetic_algorithm(items, max_weight, population_size, num_generations, mutation_rate):
            population = [create_population(len(items)) for _ in range(population_size)]
            for _ in range(num_generations):
                new_population = []
                for _ in range(population_size // 2):
                    parent1 = selection(population, items, max_weight)
                    parent2 = selection(population, items, max_weight)
                    child1, child2 = crossover(parent1, parent2)
                    child1 = mutation(child1, mutation_rate)
                    child2 = mutation(child2, mutation_rate)
                    new_population.append(child1)
                    new_population.append(child2)
                population = new_population

            best_individual = selection(population, items, max_weight)
            best_fitness = fitness(best_individual, items, max_weight)
            return best_individual, best_fitness
```

Execution And Visualization:

Code:

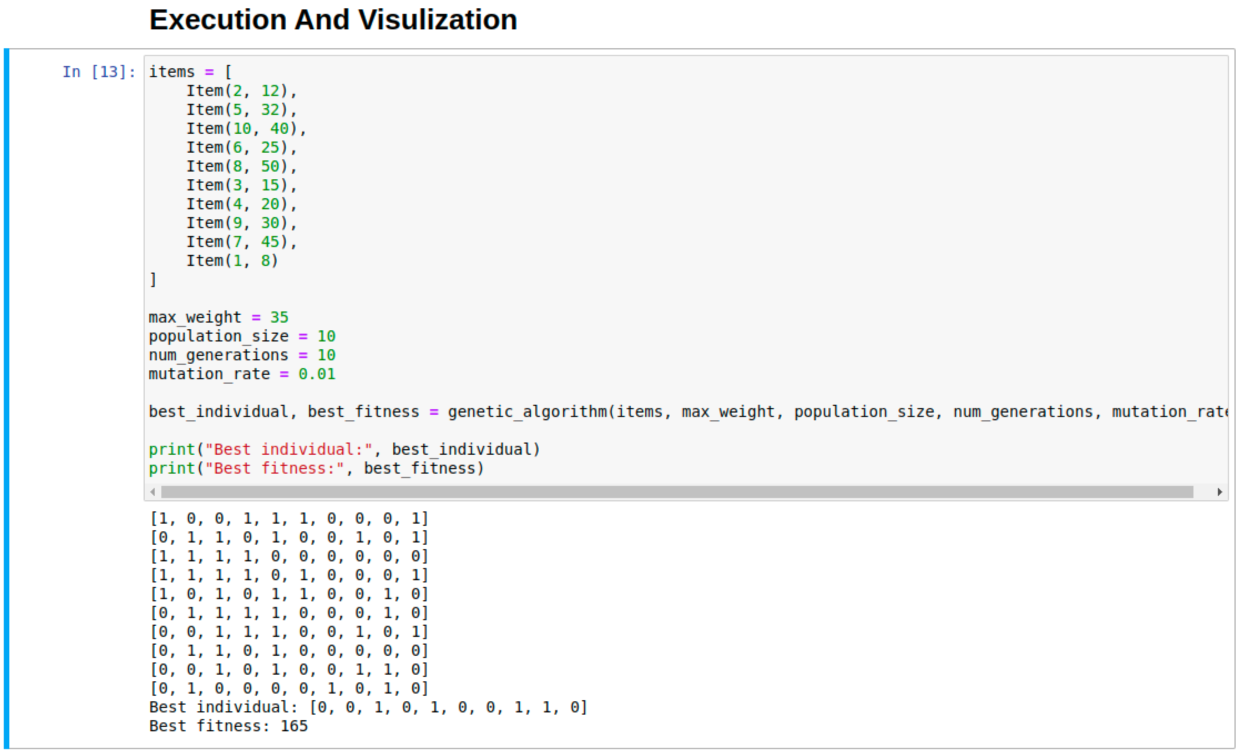
```
items = [
    Item(2, 12),
    Item(5, 32),
    Item(10, 40),
    Item(6, 25),
    Item(8, 50),
    Item(3, 15),
    Item(4, 20),
    Item(9, 30),
    Item(7, 45),
    Item(1, 8)
]

max_weight = 35
population_size = 10
num_generations = 10
mutation_rate = 0.01

best_individual, best_fitness = genetic_algorithm(items, max_weight, population_size, num_generations, mutation_rate)

print("Best individual:", best_individual)
print("Best fitness:", best_fitness)
```

Visually:



Execution And Visulization

In [13]:

```
items = [
    Item(2, 12),
    Item(5, 32),
    Item(10, 40),
    Item(6, 25),
    Item(8, 50),
    Item(3, 15),
    Item(4, 20),
    Item(9, 30),
    Item(7, 45),
    Item(1, 8)
]

max_weight = 35
population_size = 10
num_generations = 10
mutation_rate = 0.01

best_individual, best_fitness = genetic_algorithm(items, max_weight, population_size, num_generations, mutation_rat

print("Best individual:", best_individual)
print("Best fitness:", best_fitness)
```

```
[1, 0, 0, 1, 1, 1, 0, 0, 0, 1]
[0, 1, 1, 0, 1, 0, 0, 1, 0, 1]
[1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 1, 0, 1, 0, 0, 0, 1]
[1, 0, 1, 0, 1, 1, 0, 0, 1, 0]
[0, 1, 1, 1, 1, 0, 0, 0, 1, 0]
[0, 0, 1, 1, 1, 0, 0, 1, 0, 1]
[0, 1, 1, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 1, 0, 0, 1, 1, 0]
[0, 1, 0, 0, 0, 0, 1, 0, 1, 0]
Best individual: [0, 0, 1, 0, 1, 0, 0, 1, 1, 0]
Best fitness: 165
```

FIN…!