```
import numpy as np
import pandas as pd
import random as rd
from random import randint
import matplotlib.pyplot as plt
```

```
item_number = np.arange(1,11)
weight = np.random.randint(1, 15, size = 10)
value = np.random.randint(10, 750, size = 10)
knapsack_threshold = 35    #Maximum weight that the bag of thief can hold
print('The list is as follows:')
print('Item No.   Weight   Value')
for i in range(item_number.shape[0]):
    print(f'{item_number[i]}\t    {weight[i]}\t    {value[i]}\n')
```

```
The list is as follows:
Item No.   Weight   Value
1           8       581

2           12      11

3           9       231

4           9       238

5           14      183

6           12      202

7           11      182

8           10      25

9           5       513

10          5       395
```

```
solutions_per_pop = 8
pop_size = (solutions_per_pop, item_number.shape[0])
print('Population size = {}'.format(pop_size))
initial_population = np.random.randint(2, size = pop_size)
initial_population = initial_population.astype(int)
num_generations = 50
print(f'Initial population: \n{initial_population}')
```

```
Population size = (8, 10)
Initial population:
[[1 1 0 0 0 0 1 1 0 0]
 [1 0 0 1 1 1 1 0 1 1]
 [1 1 0 0 0 1 1 0 1 1]
 [1 0 0 1 1 1 1 0 1 0]
 [0 0 0 0 1 0 0 1 1 0]
 [0 0 1 0 0 1 0 0 1 1]
```

```
 [1 0 1 0 1 0 1 1 1 0]
```

```
def cal_fitness(weight, value, population, threshold):
    fitness = np.empty(population.shape[0])
    for i in range(population.shape[0]):
        S1 = np.sum(population[i] * value)
        S2 = np.sum(population[i] * weight)
        if S2 <= threshold:
            fitness[i] = S1
        else :
            fitness[i] = 0
    return fitness.astype(int)
```

```
def selection(fitness, num_parents, population):
    fitness = list(fitness)
    parents = np.empty((num_parents, population.shape[1]))
    for i in range(num_parents):
        max_fitness_idx = np.where(fitness == np.max(fitness))
        parents[i,:] = population[max_fitness_idx[0][0], :]
        fitness[max_fitness_idx[0][0]] = -999999
    return parents
```

```
def crossover(parents, num_offsprings):
    offsprings = np.empty((num_offsprings, parents.shape[1]))
    crossover_point = int(parents.shape[1]/2)
    crossover_rate = 0.8
    i=0
    while (parents.shape[0] < num_offsprings):
        parent1_index = i%parents.shape[0]
        parent2_index = (i+1)%parents.shape[0]
        x = rd.random()
        if x > crossover_rate:
            continue
        parent1_index = i%parents.shape[0]
        parent2_index = (i+1)%parents.shape[0]
        offsprings[i,0:crossover_point] = parents[parent1_index,0:crossover_point]
        offsprings[i,crossover_point:] = parents[parent2_index,crossover_point:]
        i=+1
    return offsprings
```

```
def mutation(offsprings):
    mutants = np.empty((offsprings.shape))
    mutation_rate = 0.4
    for i in range(mutants.shape[0]):
        random_value = rd.random()
        mutants[i,:] = offsprings[i,:]
        if random_value > mutation_rate:
            continue
        int_random_value = randint(0,offsprings.shape[1]-1)
        if mutants[i,int_random_value] == 0 :
            mutants[i,int_random_value] = 1
        else :
            mutants[i,int_random_value] = 0
```

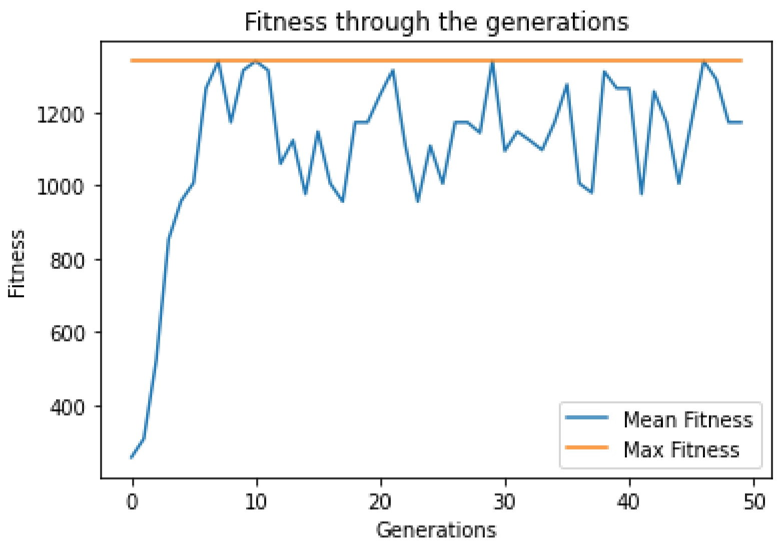
```
    return mutants
```

```
def optimize(weight, value, population, pop_size, num_generations, threshold):
    parameters, fitness_history = [], []
    num_parents = int(pop_size[0]/2)
    num_offsprings = pop_size[0] - num_parents
    for i in range(num_generations):
        fitness = cal_fitness(weight, value, population, threshold)
        fitness_history.append(fitness)
        parents = selection(fitness, num_parents, population)
        offsprings = crossover(parents, num_offsprings)
        mutants = mutation(offsprings)
        population[0:parents.shape[0], :] = parents
        population[parents.shape[0]:, :] = mutants

    print('Last generation: \n{}\n'.format(population))
    fitness_last_gen = cal_fitness(weight, value, population, threshold)
    print('Fitness of the last generation: \n{}\n'.format(fitness_last_gen))
    max_fitness = np.where(fitness_last_gen == np.max(fitness_last_gen))
    parameters.append(population[max_fitness[0][0],:])
    return parameters, fitness_history
```

```
parameters, fitness_history = optimize(weight, value, initial_population, pop_size,
print('The optimized parameters for the given inputs are: \n{}'.format(parameters))
selected_items = item_number * parameters
print('\nSelected items that will maximize the knapsack without breaking it:')
for i in range(selected_items.shape[1]):
  if selected_items[0][i] != 0:
     print('{}\n'.format(selected_items[0][i]))
```

```
Last generation:
[[0 0 1 0 0 1 0 0 1 1]
 [0 0 1 0 0 1 0 0 1 1]
 [0 0 1 0 0 1 0 0 1 1]
 [0 0 1 0 0 1 0 0 1 1]
 [0 0 1 0 0 1 0 0 1 1]
 [0 0 1 0 0 1 0 0 0 1]
 [0 1 1 0 0 1 0 0 1 1]
 [0 0 1 0 0 0 0 0 1 1]]

Fitness of the last generation:
[1341 1341 1341 1341 1341  828    0 1139]

The optimized parameters for the given inputs are:
[array([0, 0, 1, 0, 0, 1, 0, 0, 1, 1])]

Selected items that will maximize the knapsack without breaking it:
3

6

9

10
```

```
fitness_history_mean = [np.mean(fitness) for fitness in fitness_history]
fitness_history_max = [np.max(fitness) for fitness in fitness_history]
plt.plot(list(range(num_generations)), fitness_history_mean, label = 'Mean Fitness'
plt.plot(list(range(num_generations)), fitness_history_max, label = 'Max Fitness')
plt.legend()
plt.title('Fitness through the generations')
plt.xlabel('Generations')
plt.ylabel('Fitness')
plt.show()
print(np.asarray(fitness_history).shape)
```




```
(50, 8)
```