import random  
  
# Step 1: Initialize the algorithm parameters in NSGA-II.  
population\_size = 1000  
max\_iterations = 500  
crossover\_probability = 0.5  
mutation\_probability = 0.5  
  
  
# Define the chromosome coding method and other necessary parameters  
  
class Individual:  
 def \_\_init\_\_(self, chromosome):  
 self.chromosome = chromosome  
 self.fitness = None  
  
  
def initialize\_population():  
 # Generate the initial population  
 population = []  
 for \_ in range(population\_size):  
 # Create a new individual  
 individual = create\_individual()  
 population.append(individual)  
 return population  
  
  
def create\_individual():  
  
  
# Create a new individual (chromosome) based on the coding method  
# Return the individual  
  
def evaluate\_fitness(individual):  
  
  
# Calculate the fitness value for an individual based on the problem domain  
# Return the fitness value  
  
def crossover(parent1, parent2):  
  
  
# Perform crossover between two parents to generate offspring  
# Return the offspring  
  
def mutate(individual):  
  
  
# Perform mutation on an individual  
# Return the mutated individual  
  
def selection(population):  
  
  
# Perform selection to create the mating pool for reproduction  
# Return the selected individuals  
  
def non\_dominated\_sorting(population):  
  
  
# Perform non-dominated sorting on the population  
# Assign non-dominance ordinal values to individuals  
# Return the sorted population  
  
def calculate\_crowding\_distance(front):  
  
  
# Calculate crowding distance for individuals in a front  
# Update crowding distance values for the individuals in the front  
  
# Main code:  
population = initialize\_population()  
  
for iteration in range(max\_iterations):  
 # Evaluate fitness of the population  
 for individual in population:  
 fitness = evaluate\_fitness(individual)  
 individual.fitness = fitness  
  
 # Perform non-dominated sorting  
 non\_dominated\_population = non\_dominated\_sorting(population)  
  
 # Perform crowding distance calculation  
 for front in non\_dominated\_population:  
 calculate\_crowding\_distance(front)  
  
 # Perform selection, crossover, and mutation  
 offspring = []  
 while len(offspring) < population\_size:  
 parent1, parent2 = selection(population)  
 child = crossover(parent1, parent2)  
 mutated\_child = mutate(child)  
 offspring.append(mutated\_child)  
  
 # Update the population with the offspring  
 population = offspring  
  
# Save results or perform further analysis