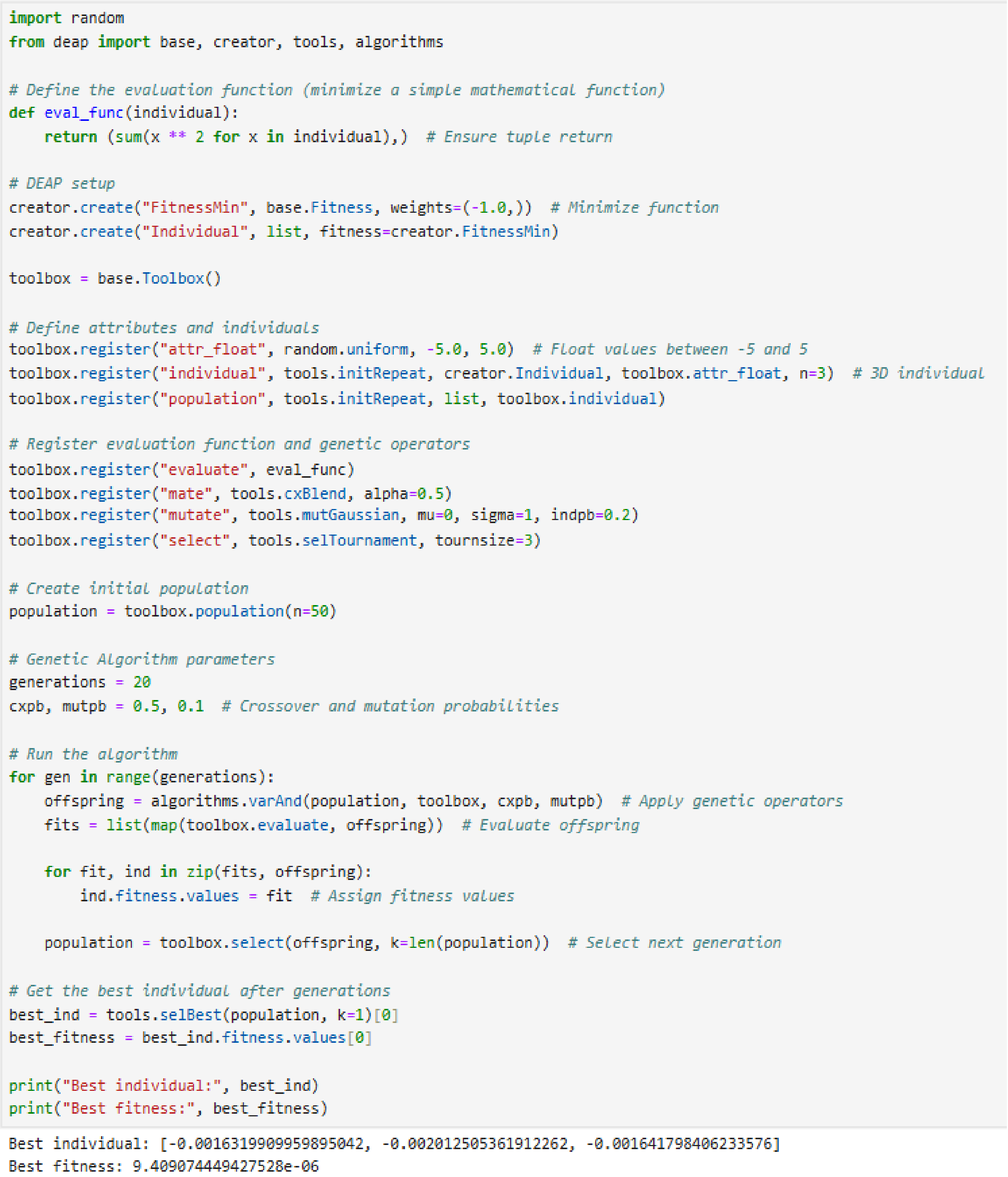
ASSIGNMENT No: 08



```python
import random
from deap import base, creator, tools, algorithms

# Define the evaluation function (minimize a simple mathematical function)
def eval_func(individual):
    return (sum(x ** 2 for x in individual),)  # Ensure tuple return

# DEAP setup
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))  # Minimize function
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

# Define attributes and individuals
toolbox.register("attr_float", random.uniform, -5.0, 5.0)  # Float values between -5 and 5
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_float, n=3)  # 3D individual
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Register evaluation function and genetic operators
toolbox.register("evaluate", eval_func)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)

# Create initial population
population = toolbox.population(n=50)

# Genetic Algorithm parameters
generations = 20
cxpb, mutpb = 0.5, 0.1  # Crossover and mutation probabilities

# Run the algorithm
for gen in range(generations):
    offspring = algorithms.varAnd(population, toolbox, cxpb, mutpb)  # Apply genetic operators
    fits = list(map(toolbox.evaluate, offspring))  # Evaluate offspring

    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit  # Assign fitness values

    population = toolbox.select(offspring, k=len(population))  # Select next generation

# Get the best individual after generations
best_ind = tools.selBest(population, k=1)[0]
best_fitness = best_ind.fitness.values[0]

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

```
Best individual: [-0.0016319909959895042, -0.002012505361912262, -0.001641798406233576]
Best fitness: 9.409074449427528e-06
```