# Problem Statement : Implement DEAP (Distributed Evolutionary Algorithms) using Python

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


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


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


toolbox = base.Toolbox()


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


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


# Genetic Algorithm parameters
generations = 20


# Run the algorithm
for gen in range(generations):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.1)

    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit

    population = toolbox.select(offspring, k=len(population))


# 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: [0.004203628922240612, 0.0011996933224371602, 0.0001696656135762735]
Best fitness: 1.9138546620442e-05
```

Start coding or generate with AI.