**1. What is DEAP?**

**DEAP** = **Distributed Evolutionary Algorithms in Python**  
It’s a **Python library** for building **evolutionary algorithms**, **genetic programming**, and **other optimization techniques** easily.

✅ Features:

* Easy to define custom problems.
* Powerful genetic operators (mutation, crossover, selection).
* Support for **parallelism** (speeding up evaluation) using **SCOOP**.

**🚀 2. Problem Statement in Your Code**

You are solving this:

**Maximize** the function:

f(x)=x2f(x) = x^2f(x)=x2

where x∈[−10,10]x \in [-10, 10]x∈[−10,10].

Meaning: find the xxx that makes x2x^2x2 largest (maximum value).

Clearly, at x=10x = 10x=10 or x=−10x = -10x=−10, f(x)=100f(x) = 100f(x)=100 — that’s the global maximum.

**🛠 3. Step-by-Step Theory of Your Code**

**STEP 1: Problem Definition (Maximization)**

python

CopyEdit

creator.create("FitnessMax", base.Fitness, weights=(1.0,))

creator.create("Individual", list, fitness=creator.FitnessMax)

* **creator** dynamically creates two classes:
  + FitnessMax: Defines that **fitness must be maximized** (weights=(+1.0,)).
  + Individual: An **individual solution**, basically a list with one number xxx.

🔵 **Fitness** = how "good" a solution is.  
🔵 **Individual** = one possible solution (just a single float value here).

**STEP 2: Initialize Toolbox (Factories)**

python

CopyEdit

toolbox = base.Toolbox()

toolbox is like a **factory** that builds individuals, populations, applies operations.

**You register functions:**

**Attribute Generator:**

python

CopyEdit

toolbox.register("attr\_float", random.uniform, -10, 10)

* Randomly generate **1 float** between −10-10−10 and 101010.

**Structure Initializers:**

python

CopyEdit

toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr\_float, n=1)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

* **individual** = list of 1 float.
* **population** = list of individuals.

🔵 **Population** = group of candidate solutions.

**STEP 3: Evaluation Function**

python

CopyEdit

def evalFunction(individual):

x = individual[0]

return x\*\*2, # COMMA here makes it a tuple!

* Takes an individual (list with one float).
* Returns its **fitness** = x2x^2x2.

🔵 **Important:** In DEAP, evaluation **must return a tuple** like (fitness\_value,).

**STEP 4: Register Genetic Operators**

python

CopyEdit

toolbox.register("mate", tools.cxBlend, alpha=0.5)

toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=2, indpb=0.2)

toolbox.register("select", tools.selTournament, tournsize=3)

**These control how individuals "evolve" over generations:**

| **Operator** | **What It Does** | **Your Setting** |
| --- | --- | --- |
| Crossover (mate) | Blend two parents randomly to create offspring | cxBlend, mixing with α=0.5 |
| Mutation (mutate) | Add random Gaussian noise to genes | Mean=0, σ=2, mutate with 20% probability |
| Selection (select) | Select the best individuals to survive | Tournament between 3 random candidates |

✅ These mimic **natural evolution**: recombination + random changes + survival of fittest.

**STEP 5: Parallel Evaluation**

python

CopyEdit

toolbox.register("map", futures.map)

Normally, map applies a function serially.  
Here, you tell DEAP to use **SCOOP** to evaluate individuals **in parallel across multiple CPU cores**!

**SCOOP** lets you distribute work automatically to maximize speedup.

**STEP 6: Main Function: Running the Evolution**

python

CopyEdit

pop = toolbox.population(n=50)

hof = tools.HallOfFame(1)

* Create a population of 50 individuals.
* HallOfFame saves the **best individual found**.

**Statistics Setup:**

python

CopyEdit

stats = tools.Statistics(lambda ind: ind.fitness.values)

stats.register("avg", np.mean)

stats.register("min", np.min)

stats.register("max", np.max)

* Track how the population's fitness improves over time.

**Evolution Loop:**

python

CopyEdit

pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2,

ngen=20, stats=stats, halloffame=hof,

verbose=True)

* Run **20 generations** (ngen=20).
* Each generation:
  + 50% chance for **crossover** (cxpb=0.5).
  + 20% chance for **mutation** (mutpb=0.2).
  + Select individuals for next generation.

✅ eaSimple = simple evolutionary loop.

**Print Best Result:**

python

CopyEdit

print("\nBest individual:", hof[0])

print("Fitness value:", hof[0].fitness.values[0])

* After evolution finishes, it shows the **best found xxx** and corresponding x2x^2x2.

**🧬 4. Full Evolutionary Algorithm Cycle (Simple Overview)**

| **Step** | **Action** |
| --- | --- |
| 1. Initialize | Random individuals (x between -10 and 10). |
| 2. Evaluate | Calculate fitness x2x^2x2. |
| 3. Selection | Choose individuals for mating. |
| 4. Crossover | Exchange genes between individuals. |
| 5. Mutation | Add random noise. |
| 6. Replacement | Create new generation. |
| 7. Repeat | For 20 generations. |
| 8. Result | Best individual and fitness found. |

**📈 Behavior of the Population:**

* Initially: Random guesses for xxx.
* Over generations:
  + Bad individuals die.
  + Good individuals reproduce.
  + Random mutations add exploration.
* Eventually: The population **converges** near x=10x = 10x=10 or x=−10x = -10x=−10.

✅ Global maximum reached.

**⚡ Parallelism with SCOOP**

Normally:

* You evaluate each individual **one by one** (slow if large population).

Using SCOOP:

* Each CPU core evaluates one individual **in parallel**.
* Much faster for big problems.
* You don't have to manually manage multiprocessing.

✅ Just toolbox.register("map", futures.map) automatically makes DEAP distribut

╔════════════════════════════════╗

║ START ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Initialize Population (x ∈ [-10, 10]) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Evaluate Fitness f(x) = x² ║

║ (using parallel SCOOP map) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ FOR generation = 1 to 20 ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Selection (Tournament) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Crossover (cxBlend, 50%) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Mutation (Gaussian, 20%) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Evaluate Fitness of Offspring ║

║ (again in parallel) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Replace Population ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Record Stats (avg, min, max) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ END FOR Loop (after 20 gens) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Extract Best Individual (HoF) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ Print Best x and Fitness f(x) ║

╚════════════════════════════════╝

↓

╔════════════════════════════════╗

║ END ║

╚════════════════════════════════╝