Lab 5: Simulated Annealing algorithm

General Algorithm:

```
def annealing (init_sol, temp, cooling, final_temp)
    current ← init_sol
    new ← current
    while temp > final_temp:
        neighbor = getNeighbor (current)
        if current cost < neighbor
        new = getNeighbor (current)
        if deltaE ← Cost(new) - Cost(current)
        if deltaE < 0:
            current = new
        else if random() < e^(-deltaE/temp):
            current ← new
        temp *= cooling
    return current
```

→ Utility functions:

```
def objective-function (x)
    // Define any mathematical objective
    // function
    return x^4 + 5 + sin(x 5^4 * π * x)

def cost (sol)
    return objective-function (sol)
```

→ Initialization code:

initial_sol =

def getNeighbor(sol):
    return ~~current~~ sol + random(-1, 1)
Adding a random perturbation to solution

→ Initialization Code:

initial_solution = random(-10, 10)
# Initial solution b/w -10, 10
initial_temperature = 10
final_temperature = 0.1
cooling_factor = 0.99
print(annealing(initial_solution, initial_temp, cooling, final_temperature))

→ Code:

import math
import random

class Annealing:
    def __init__(self):
        self.initial_sol = random.uniform(-10, 10)
        self.temp = 10
        self.cooling = 0.99
        self.final = 0.01
        self.annealing()

    def cost(self, x):
        return x**4 + 5 * math.sin(5*math.pi*x)

    def getNeighbors(self, sol):
        return sol + random.uniform(-1, 1)

```
def annealing(self):
    current = self.initial_sol
    new = current ; best = current
    while self.temp > self.final:
        new = self.getNeighbors(current)
        dE = self.cost(new) - self.cost(current)
        print(
        if dE<0 or random.random()
                < math.exp(-dE/self.temp
            current = new
        if self.cost(new) < self.cost(best)
            best = new
        self.temp *= self.cooling
    print(f"Best sol: {best:.4f}")

c = Annealing()
```

Output:

Find Solution:

-0.0977

22/10/24