https://justpaste.it/gxvu5

CI1

# server.py (cell 1)

import xmlrpc.server

import threading

# Function to calculate factorial

def factorial(n):

if n < 0:

raise ValueError("Factorial is not defined for negative numbers.")

if n > 100: # Limit the maximum number to 100

raise ValueError("Input is too large! Try a smaller number.")

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n - 1)

# Function to start the server

def start\_server():

with xmlrpc.server.SimpleXMLRPCServer(('localhost', 8000)) as server:

print("Server is running...")

server.register\_function(factorial, 'factorial')

server.serve\_forever()

# Running the server in a separate thread

server\_thread = threading.Thread(target=start\_server)

server\_thread.daemon = True

server\_thread.start()

# client.py (cell 2)

import xmlrpc.client

# Create a connection to the server

server = xmlrpc.client.ServerProxy('http://localhost:8000')

# Request input from the user (for Jupyter, use input())

number = int(input("Enter an integer to calculate its factorial: "))

# Call the remote procedure (factorial) and print the result

result = server.factorial(number)

print(f"The factorial of {number} is {result}")

CI2

Server.py  
import Pyro5.api

@Pyro5.api.expose

class StringService:

    def concatenate(self, str1, str2):

        return str1 + str2

# Start the Pyro5 Daemon

def main():

    daemon = Pyro5.server.Daemon()

    uri = daemon.register(StringService, "string.service")

    print("Server is running. URI:", uri)

    daemon.requestLoop()

# Uncomment this to run in script mode

if \_\_name\_\_ == "\_\_main\_\_":

    main()

client.py  
import Pyro5.api

# Connect to the server

uri = "PYRO:string.service@localhost:64566"  # Update localhost if needed

try:

    string\_service = Pyro5.api.Proxy(uri)

    # Inputs

    str1 = input("Enter first string: ")

    str2 = input("Enter second string: ")

    # Remote method call

    result = string\_service.concatenate(str1, str2)

    print("Concatenated String:", result)

except Exception as e:

    print("Error:", e)

C:\Users\HP>cd "C:\Users\HP\OneDrive\Desktop\cat"

C:\Users\HP\OneDrive\Desktop\cat>python server.py

Server is running. URI: PYRO:string.service@localhost:64566  
  
copy host number and paste it in client   
python client.py

CI3

#charecter count

text = """Hello world!

This is a test text file for character counting.

Let's count the characters!"""

def char\_mapper(text):

for char in text:

if char.strip():

yield (char, 1)

# Reduce function

def char\_reducer(mapped\_results):

char\_count = {}

for char, count in mapped\_results:

if char in char\_count:

char\_count[char] += count

else:

char\_count[char] = count

return char\_count

# Simulate MapReduce

mapped\_results = list(char\_mapper(text))

char\_counts = char\_reducer(mapped\_results)

# Display results

print("Character Counts:")

for char, count in char\_counts.items():

print(f"'{char}': {count}")

#word count

text = """Hello world!

This is a test text file for word counting.

Let's count the words!"""

def word\_mapper(text):

words = text.lower().split()

for word in words:

word = ''.join(char for char in word if char.isalnum())

if word:

yield (word, 1)

# Reduce function

def word\_reducer(mapped\_results):

word\_count = {}

for word, count in mapped\_results:

if word in word\_count:

word\_count[word] += count

else:

word\_count[word] = count

return word\_count

# Simulate MapReduce

mapped\_results = list(word\_mapper(text))

word\_counts = word\_reducer(mapped\_results)

# Display results

print("Word Counts:")

for word, count in word\_counts.items():

print(f"'{word}': {count}")

CI4

import numpy as np

def fuzzy\_union(A, B):

return np.maximum(A, B)

def fuzzy\_intersection(A, B):

return np.minimum(A, B)

def fuzzy\_complement(A):

return 1 - A

def fuzzy\_difference(A, B):

return np.minimum(A, 1 - B)

def cartesian\_product(A, B):

return np.outer(A, B)

def max\_min\_composition(R, S):

return np.max(np.minimum(R[:, np.newaxis, :], S[np.newaxis, :, :]), axis=2)

# Define two fuzzy sets

A = np.array([0.2, 0.5, 0.8])

B = np.array([0.3, 0.6, 0.7])

# Perform operations

union\_result = fuzzy\_union(A, B)

intersection\_result = fuzzy\_intersection(A, B)

complement\_A = fuzzy\_complement(A)

difference\_A\_B = fuzzy\_difference(A, B)

# Display results

print("Union:", union\_result)

print("Intersection:", intersection\_result)

print("Complement of A:", complement\_A)

print("Difference A - B:", difference\_A\_B)

# Define two fuzzy sets for Cartesian product

C = np.array([0.1, 0.4, 0.9])

D = np.array([0.2, 0.5, 0.8])

# Create fuzzy relations using Cartesian product

R = cartesian\_product(A, B)

S = cartesian\_product(C, D)

# Perform max-min composition

composition\_result = max\_min\_composition(R, S)

# Display results

print("Fuzzy Relation R (Cartesian product of A and B):")

print(R)

print("Fuzzy Relation S (Cartesian product of C and D):")

print(S)

print("Max-Min Composition of R and S:")

print(composition\_result)

CI5

import numpy as np

from sklearn.neural\_network import MLPRegressor

from sklearn.model\_selection import train\_test\_split

class GeneticAlgorithm:

def \_\_init\_\_(self, population\_size, mutation\_rate, generations):

self.population\_size = population\_size

self.mutation\_rate = mutation\_rate

self.generations = generations

def create\_population(self):

return np.random.rand(self.population\_size, 3) # Example: 3 parameters

def fitness(self, individual):

# Example: Train a neural network and return a fitness score

X = np.random.rand(100, 3) # Example input data

y = np.random.rand(100) # Example output data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

model = MLPRegressor(hidden\_layer\_sizes=(10,), max\_iter=1000)

model.fit(X\_train, y\_train)

return model.score(X\_test, y\_test) # Return R^2 score as fitness

def mutate(self, individual):

for i in range(len(individual)):

if np.random.rand() < self.mutation\_rate:

individual[i] += np.random.normal(0, 0.1) # Example mutation

return individual

def crossover(self, parent1, parent2):

return np.array([(p1 + p2) / 2 for p1, p2 in zip(parent1, parent2)]) # Simple average crossover

def run(self):

population = self.create\_population()

for generation in range(self.generations):

fitness\_scores = np.array([self.fitness(ind) for ind in population])

sorted\_indices = np.argsort(fitness\_scores)[::-1]

population = population[sorted\_indices] # Select the best individuals

# Create new population

new\_population = []

for i in range(0, self.population\_size, 2):

parent1, parent2 = population[i], population[i + 1]

child1 = self.crossover(parent1, parent2)

child2 = self.crossover(parent2, parent1)

new\_population.append(self.mutate(child1))

new\_population.append(self.mutate(child2))

population = np.array(new\_population)

best\_individual = population[0]

return best\_individual

if \_\_name\_\_ == "\_\_main\_\_": # Corrected if condition

ga = GeneticAlgorithm(population\_size=20, mutation\_rate=0.1, generations=50)

best\_params = ga.run()

print("Best parameters found:", best\_params)

CI6

import numpy as np

import matplotlib.pyplot as plt

# Define the fitness function

def fitness\_function(x):

return x \* np.sin(10 \* np.pi \* x) + 1.0

# Initialize population

def initialize\_population(size):

return np.random.rand(size)

# Clone selection

def select\_top(population, fitness, n\_selected):

indices = np.argsort(fitness)[-n\_selected:]

return population[indices]

# Hypermutation (inversely proportional to fitness)

def hypermutate(clones, fitness, mutation\_rate=0.1):

max\_fit = np.max(fitness)

mutated = []

for i, clone in enumerate(clones):

rate = mutation\_rate \* (1 - fitness[i] / max\_fit)

new\_clone = clone + np.random.normal(0, rate)

new\_clone = np.clip(new\_clone, 0, 1) # Ensure within [0, 1]

mutated.append(new\_clone)

return np.array(mutated)

# Clonal Selection Algorithm

def clonal\_selection\_algorithm(pop\_size=50, generations=100, n\_selected=10, n\_clones=5):

population = initialize\_population(pop\_size)

best\_scores = []

for gen in range(generations):

fitness = fitness\_function(population)

best\_scores.append(np.max(fitness))

selected = select\_top(population, fitness, n\_selected)

clones = np.repeat(selected, n\_clones)

clone\_fitness = fitness\_function(clones)

mutated\_clones = hypermutate(clones, clone\_fitness)

mutated\_fitness = fitness\_function(mutated\_clones)

best\_from\_clones\_indices = np.argsort(mutated\_fitness)[-n\_selected:]

best\_new = mutated\_clones[best\_from\_clones\_indices]

population = np.concatenate([population, best\_new])

population\_fitness = fitness\_function(population)

top\_indices = np.argsort(population\_fitness)[-pop\_size:]

population = population[top\_indices]

best\_solution = population[np.argmax(fitness\_function(population))]

return best\_solution, best\_scores

# Run the algorithm

best, scores = clonal\_selection\_algorithm()

print("Best solution found:", best)

print("Best fitness:", fitness\_function(best))

# Plot the convergence

plt.plot(scores)

plt.title("Fitness Over Generations")

plt.xlabel("Generation")

plt.ylabel("Best Fitness")

plt.grid(True)

plt.show()

DC7

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Simulate a dataset: 0 = No Damage, 1 = Damage

X, y = make\_classification(n\_samples=500, n\_features=10, n\_informative=6,

n\_redundant=2, n\_classes=2, random\_state=42)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

from sklearn.base import BaseEstimator, ClassifierMixin

class ArtificialImmuneClassifier(BaseEstimator, ClassifierMixin):

def \_\_init\_\_(self, n\_clones=5, mutation\_rate=0.1, n\_generations=10):

self.n\_clones = n\_clones

self.mutation\_rate = mutation\_rate

self.n\_generations = n\_generations

self.memory\_cells = []

def \_affinity(self, antibody, antigen):

return -np.linalg.norm(antibody['vector'] - antigen)

def fit(self, X, y):

self.memory\_cells = [{'vector': x, 'label': label} for x, label in zip(X, y)]

for gen in range(self.n\_generations):

new\_memory = []

for antigen, label in zip(X, y):

# Select best match (BMU)

bmu = max(self.memory\_cells, key=lambda ab: self.\_affinity(ab, antigen))

clones = [dict(bmu) for \_ in range(self.n\_clones)]

# Hypermutation

for clone in clones:

noise = np.random.normal(0, self.mutation\_rate, size=antigen.shape)

clone['vector'] += noise

# Evaluate clones

best\_clone = max(clones, key=lambda ab: -np.linalg.norm(ab['vector'] - antigen))

# Add if improved match

if self.\_affinity(best\_clone, antigen) > self.\_affinity(bmu, antigen):

new\_memory.append({'vector': best\_clone['vector'], 'label': label})

else:

new\_memory.append(bmu)

self.memory\_cells = new\_memory

return self

def predict(self, X):

preds = []

for x in X:

bmu = max(self.memory\_cells, key=lambda ab: -np.linalg.norm(ab['vector'] - x))

preds.append(bmu['label'])

return np.array(preds)

model = ArtificialImmuneClassifier(n\_clones=10, mutation\_rate=0.05, n\_generations=20)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

acc = accuracy\_score(y\_test, y\_pred)

print("Test Accuracy:", acc)

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

import seaborn as sns

plt.figure(figsize=(5, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Damage', 'Damage'], yticklabels=['No Damage', 'Damage'])

plt.title('Structural Damage Classification')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

DC8

pip install deap

import random

from deap import base, creator, tools, algorithms

import matplotlib.pyplot as plt

# Define fitness and individual types

creator.create("FitnessMin", base.Fitness, weights=(-1.0,)) # Minimize

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

# Initialize toolbox

toolbox = base.Toolbox()

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

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

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

# Objective function: f(x) = x^2

def eval\_func(individual):

x = individual[0]

return (x\*\*2,)

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)

# Parameters

population = toolbox.population(n=50)

NGEN = 30

CXPB = 0.5

MUTPB = 0.2

# Store best fitness per generation

best\_fit = []

for gen in range(NGEN):

offspring = algorithms.varAnd(population, toolbox, cxpb=CXPB, mutpb=MUTPB)

fits = list(map(toolbox.evaluate, offspring))

for fit, ind in zip(fits, offspring):

ind.fitness.values = fit

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

best\_ind = tools.selBest(population, 1)[0]

best\_fit.append(best\_ind.fitness.values[0])

print(f"Gen {gen+1}: Best fitness = {best\_ind.fitness.values[0]:.4f}")

print("\nBest solution found:", best\_ind)

plt.plot(best\_fit, label='Best Fitness')

plt.xlabel('Generation')

plt.ylabel('Fitness')

plt.title('Genetic Algorithm Optimization (x²)')

plt.grid()

plt.legend()

plt.show()

DC9

**HotelImpl.java**

import java.rmi.server.UnicastRemoteObject;   
import java.rmi.RemoteException;   
import java.util.\*;   
   
public class HotelImpl extends UnicastRemoteObject implements Hotel {   
    private Map<String, Integer> bookings;   
    private int availableRooms;   
   
    public HotelImpl(int totalRooms) throws RemoteException {   
        super();   
        bookings = new HashMap<>();   
        availableRooms = totalRooms;   
    }   
   
    public synchronized String bookRoom(String guestName) throws   
RemoteException {   
        if (bookings.containsKey(guestName)) {   
            return "Guest already has a booking.";   
        }   
        if (availableRooms > 0) {   
            bookings.put(guestName, bookings.size() + 1);   
            availableRooms--;   
            return "Room booked for " + guestName;   
        }   
        return "No rooms available.";   
    }   
   
    public synchronized String cancelBooking(String guestName) throws   
RemoteException {   
        if (bookings.containsKey(guestName)) {   
            bookings.remove(guestName);   
            availableRooms++;   
            return "Booking cancelled for " + guestName;   
        }   
        return "No booking found for " + guestName;   
    }   
} 

**Hotel.java**  
  
  
import java.rmi.Remote;   
import java.rmi.RemoteException;   
   
public interface Hotel extends Remote {   
    String bookRoom(String guestName) throws RemoteException;   
    String cancelBooking(String guestName) throws RemoteException;   
}  
  
  
**HotelServer.java**  
  
import java.rmi.Naming;   
import java.rmi.registry.LocateRegistry;   
   
public class HotelServer {   
    public static void main(String[] args) {   
        try {   
            LocateRegistry.createRegistry(1099); // Start RMI registry   
            HotelImpl hotel = new HotelImpl(5); // 5 rooms total   
            Naming.rebind("HotelService", hotel);   
            System.out.println("Hotel Server is ready...");   
        } catch (Exception e) {   
            System.out.println("Server Error: " + e);   
        }   
    }   
}   
  
  
**HotelClient.java**  
  
  
import java.rmi.Naming;   
import java.util.Scanner;   
   
public class HotelClient {   
    public static void main(String[] args) {   
        try {   
            Hotel hotel = (Hotel)   
Naming.lookup("rmi://localhost/HotelService");   
            Scanner sc = new Scanner(System.in);   
            while (true) {   
                System.out.println("\n1. Book Room\n2. Cancel Booking\n3.   
Exit");   
                System.out.print("Choice: ");   
                int choice = sc.nextInt();   
                sc.nextLine(); // Consume newline   
   
                if (choice == 1) {   
                    System.out.print("Enter Guest Name: ");   
                    String name = sc.nextLine();   
                    System.out.println(hotel.bookRoom(name));   
                } else if (choice == 2) {   
                    System.out.print("Enter Guest Name to Cancel: ");   
                    String name = sc.nextLine();   
                    System.out.println(hotel.cancelBooking(name));   
                } else {   
                    break;   
                }   
            }   
            sc.close();   
        } catch (Exception e) {   
            System.out.println("Client Error: " + e);   
        }   
    }   
}   
  
**# to Run the program**  
install Java for windows   
  
javac \*.java  
javac --version  
  
start rmiregistry   
java HotelServer  
  
java HotelClient

DC10

Input.txt  
dt,temp,col3,col4,col5,col6,col7

2010-06-01,35.2,a,b,c,d,e

2011-07-01,38.5,a,b,c,d,e

2010-12-01,10.5,a,b,c,d,e

2011-01-01,5.0,a,b,c,d,e

2012-08-15,40.1,a,b,c,d,e

mapper.py  
import sys

for line in sys.stdin:

    line = line.strip()

    if line.startswith("dt"):

        continue

    parts = line.split(",")

    if len(parts) != 7:

        continue

    date = parts[0]

    year = date.split("-")[0]

    try:

        temperature = float(parts[1])

        print(f"{year}\t{temperature}")

    except ValueError:

        continue

reducer.py

import sys

year\_temps = {}

for line in sys.stdin:

    year, temp = line.strip().split("\t")

    temp = float(temp)

    if year not in year\_temps:

        year\_temps[year] = {"min": temp, "max": temp}

    else:

        year\_temps[year]["min"] = min(year\_temps[year]["min"], temp)

        year\_temps[year]["max"] = max(year\_temps[year]["max"], temp)

if not year\_temps:

    print("No data to process.")

    sys.exit(1)

coolest\_year = min(year\_temps.items(), key=lambda x: x[1]["min"])

hottest\_year = max(year\_temps.items(), key=lambda x: x[1]["max"])

print(f"Coolest Year: {coolest\_year[0]}, Min Temp: {coolest\_year[1]['min']}")

print(f"Hottest Year: {hottest\_year[0]}, Max Temp: {hottest\_year[1]['max']}")

In Command Prompt Type  
cd "C:\Users\HP\OneDrive\Desktop\cat"  
python mapper.py < input.txt | python reducer.py