**PB1:- RPC(Factorial)**

**Client:**

import xmlrpc.client

def main():

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

n=int(input("Enter the Number:"))

result = server.calculate\_factorial(n)

print(result)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Server:**

from xmlrpc.server import SimpleXMLRPCServer

def factorial(n):

if n==0 or n==1:

return 1

else:

return n\*factorial(n-1)

server = SimpleXMLRPCServer(('localhost',8000))

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

server.serve\_forever()

**PB2:- RPC(Arithmetic operation)**

**Client:**

import xmlrpc.client

def main():

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

print("Available operations: add, subtract, multiply, divide")

a = int(input("Enter first number: "))

b = int(input("Enter second number: "))

operation = input("Enter operation to perform: ").strip().lower()

result = server.perform\_operation(a, b, operation)

print("Result:", result)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Server:**

from xmlrpc.server import SimpleXMLRPCServer

# Define arithmetic operations

def arithmetic\_operation(a, b, operation):

if operation == 'add':

return a + b

elif operation == 'subtract':

return a - b

elif operation == 'multiply':

return a \* b

elif operation == 'divide':

if b == 0:

return "Error: Division by zero"

return a / b

else:

return "Error: Unsupported operation"

# Set up XML-RPC server

server = SimpleXMLRPCServer(('localhost', 8000))

print("Server is running on port 8000...")

# Register function with the server

server.register\_function(arithmetic\_operation, 'perform\_operation')

# Start the server

server.serve\_forever()

**PB3:- RMI(String Concatenation)**

**Client:**

import Pyro4

uri = input("Enter URI of Server:")

concatenator = Pyro4.Proxy(uri)

str1 = input("Enter the string 1")

str2 = input("Enter the string 2")

result = concatenator.concatenate(str1, str2)

print(result)

**Server:**

import Pyro4

@Pyro4.expose

class stringconcatenator:

def concatenate(self,str1,str2):

return str1+str2

daemon=Pyro4.Daemon()

uri=daemon.register(stringconcatenator)

print(uri)

daemon.requestLoop()

**PB4:- RMI(Palindrome)**

**Client:**

import Pyro4

# Get the URI printed from the server side

uri = input("Enter URI of Server: ")

checker = Pyro4.Proxy(uri)

# Get input string from user

user\_input = input("Enter the string to check for palindrome: ")

# Call remote method

is\_palindrome = checker.is\_palindrome(user\_input)

# Print result

if is\_palindrome:

print("The string is a palindrome.")

else:

print("The string is not a palindrome.")

**Server:**

import Pyro4

@Pyro4.expose

class PalindromeChecker:

def is\_palindrome(self, input\_str):

# Clean the string: remove spaces and convert to lowercase

cleaned = ''.join(c.lower() for c in input\_str if c.isalnum())

return cleaned == cleaned[::-1]

# Register the class as a Pyro object

daemon = Pyro4.Daemon()

uri = daemon.register(PalindromeChecker)

print("URI to use in client:", uri)

# Start the server loop

daemon.requestLoop()

**PB5:- MapReduce(CharCount & WordCount)**

**CharCount:**

from mrjob.job import MRJob

class MRCharCount(MRJob):

def mapper(self, \_, line):

for char in line.strip():

yield char, 1

def reducer(self, char, counts):

yield char, sum(counts)

if \_\_name\_\_ == '\_\_main\_\_':

MRCharCount.run()

**WordCount:**

from mrjob.job import MRJob

import re

WORD\_REGEXP = re.compile(r"[\w']+")

class MRWordCount(MRJob):

def mapper(self, \_, line):

for word in WORD\_REGEXP.findall(line):

yield word.lower(), 1

def reducer(self, word, counts):

yield word, sum(counts)

if \_\_name\_\_ == '\_\_main\_\_':

MRWordCount.run()

**PB6:- MapReduce(SentenceCount)**

**SentenceCount:**

from mrjob.job import MRJob

import re

# Regular expression to detect sentence boundaries

SENTENCE\_END\_REGEX = re.compile(r'[.!?]')

class MRSentenceCount(MRJob):

def mapper(self, \_, line):

# Split the line using sentence-ending punctuation marks

sentences = SENTENCE\_END\_REGEX.split(line)

# Count non-empty sentences (after stripping whitespace)

for sentence in sentences:

if sentence.strip():

yield "Sentence", 1

def reducer(self, key, values):

yield key, sum(values)

if \_\_name\_\_ == '\_\_main\_\_':

MRSentenceCount.run()

**Input:**Hello! My name is Omkar. How are you doing today? I hope everything is going well.

This is an example to test sentence counting. Let's see how many sentences are here!

Mr. Smith is a good teacher. He teaches well. What time is it?

Wow! That's amazing. Can you believe it?

**PB7:- MapReduce(UniqueWordCount)**

**UniqueWordCount:**

from mrjob.job import MRJob

import re

WORD\_RE = re.compile(r"[\w']+")

class MRUniqueWordCount(MRJob):

def mapper(self, \_, line):

for word in WORD\_RE.findall(line):

yield word.lower(), None # emit each word with dummy value

def reducer\_init(self):

self.count = 0

def reducer(self, word, \_):

self.count += 1 # count each unique word

def reducer\_final(self):

yield "Unique Word Count", self.count

def steps(self):

return [self.mr(mapper=self.mapper,

reducer\_init=self.reducer\_init,

reducer=self.reducer,

reducer\_final=self.reducer\_final)]

if \_\_name\_\_ == '\_\_main\_\_':

MRUniqueWordCount.run()

**Input:**Hello world! Hello again. This is a test file.

World of code is vast. Code again and again!

**PB8:- Load Balancing Algorithm**

import random

class Server:

def \_\_init\_\_(self, name):

self.name = name

self.active\_connections = 0

class LoadBalancer:

def \_\_init\_\_(self, servers):

self.servers = [Server(s) for s in servers]

self.rr\_index = 0

def random\_selection(self):

return random.choice(self.servers)

def round\_robin(self):

server = self.servers[self.rr\_index % len(self.servers)]

self.rr\_index += 1

return server

def least\_connections(self):

return min(self.servers, key=lambda s: s.active\_connections)

def assign\_request(self,strategy):

if strategy == "random":

server = self.random\_selection()

elif strategy == "round\_robin":

server = self.round\_robin()

elif strategy == "least\_connections":

server = self.least\_connections()

else:

raise ValueError("Invalid Strategy")

server.active\_connections += 1

print(f"{strategy.capitalize():<15} - Request -> {server.name} (Active: {server.active\_connections})")

server.active\_connections -= 1

def simulate\_client\_requests(load\_balancer, num\_requests):

strategies = ["random", "round\_robin", "least\_connections"]

for strategy in strategies:

print(f"\n---{strategy.upper()} STRATEGY ---")

for i in range(num\_requests):

load\_balancer.assign\_request(strategy)

servers = ["server A", "Server B", "Server C"]

load\_balancer = LoadBalancer(servers)

simulate\_client\_requests(load\_balancer, 5)

**PB9:- Clonal Selection Algorithm**

import numpy as np

def objective\_function(x):

return np.sum(x\*\*2)

def clonal\_selection\_algorithm(num\_antibodies, num\_dimensions, search\_space, num\_generations, num\_clones, clone\_factor, mutation\_rate):

antibodies = np.random.uniform(search\_space[:, 0], search\_space[:, 1], size=(num\_antibodies, num\_dimensions))

for generation in range(num\_generations):

fitness = np.array([objective\_function(antibody) for antibody in antibodies])

clones = np.repeat(antibodies, np.round(num\_clones \* (1 / (1 + fitness \* clone\_factor))).astype(int), axis=0)

mutation\_mask = np.random.rand(\*clones.shape) < mutation\_rate

mutation\_amounts = np.random.uniform(-0.5, 0.5, size=clones.shape) \* (search\_space[:, 1] - search\_space[:, 0])

mutated\_clones = np.clip(clones + mutation\_mask \* mutation\_amounts, search\_space[:, 0], search\_space[:, 1])

combined\_population = np.vstack((antibodies, mutated\_clones))

fitness\_combined = np.array([objective\_function(antibody) for antibody in combined\_population])

antibodies = combined\_population[np.argsort(fitness\_combined)][:num\_antibodies]

return antibodies[0]

best\_solution = clonal\_selection\_algorithm(50, 3, np.array([[-5,5]]\* 3), 100, 10, 0.1, 0.1)

print("Best Solution:", best\_solution)

print("Objective Value:",objective\_function(best\_solution))

**PB11:- Artificial immune pattern recognition**

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

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

from sklearn import datasets

iris = datasets.load\_iris()

X = iris.data

y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=42)

class AISClassifier:

def \_\_init\_\_(self, n\_neighbors=10):

self.knn = KNeighborsClassifier(n\_neighbors=n\_neighbors)

def fit(self, X\_train, y\_train):

self.knn.fit(X\_train, y\_train)

def predict(self, X\_test):

return self.knn.predict(X\_test)

ais\_clf = AISClassifier(n\_neighbors=10)

ais\_clf.fit(X\_train, y\_train)

y\_pred = ais\_clf.predict(X\_test)

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

print("Accuracy:",accuracy)

**PB12:- DEAP**

!pip install deap

import random

from deap import base, creator, tools, algorithms

def eval\_func(individual):

return sum(x \*\* 2 for x in individual),

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

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

toolbox = base.Toolbox()

toolbox.register("attr\_float", random.uniform, -5.0, 5.0)

toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr\_float, n=3)

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

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)

toolbox.register("evaluate", eval\_func)

population = toolbox.population(n=50)

algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.1, ngen=20)

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)

**PB13:- Ant Colony optimization(TSP)**

import numpy as np

num\_cities = 10

num\_ants = 100

num\_iterations = 1000

alpha = 1

beta = 2

evaporation\_rate = 0.1

pheromone\_matrix = np.ones((num\_cities, num\_cities))

distance\_matrix = np.random.randint(1, 100, size=(num\_cities, num\_cities))

distance\_matrix = (distance\_matrix + distance\_matrix.T) / 2

np.fill\_diagonal(distance\_matrix, np.inf)

# ACO algorithm

best\_tour = None

best\_tour\_length = np.inf

for iteration in range(num\_iterations):

all\_tours = []

all\_lengths = []

for ant in range(num\_ants):

tour = []

visited = set()

current\_city = np.random.randint(0, num\_cities)

tour.append(current\_city)

visited.add(current\_city)

for \_ in range(num\_cities - 1):

probabilities = []

for city in range(num\_cities):

if city not in visited:

pheromone = pheromone\_matrix[current\_city][city] \*\* alpha

visibility = (1.0 / distance\_matrix[current\_city][city]) \*\* beta

probabilities.append(pheromone \* visibility)

else:

probabilities.append(0)

probabilities = np.array(probabilities)

if probabilities.sum() == 0:

break # Safeguard: no valid moves

probabilities /= probabilities.sum()

next\_city = np.random.choice(num\_cities, p=probabilities)

tour.append(next\_city)

visited.add(next\_city)

current\_city = next\_city

all\_tours.append(tour)

length = sum(distance\_matrix[tour[i]][tour[i+1]] for i in range(len(tour)-1))

all\_lengths.append(length)

if length < best\_tour\_length:

best\_tour\_length = length

best\_tour = tour

# Pheromone evaporation

pheromone\_matrix \*= (1 - evaporation\_rate)

# Pheromone update

for tour, length in zip(all\_tours, all\_lengths):

for i in range(len(tour) - 1):

pheromone\_matrix[tour[i]][tour[i+1]] += 1.0 / length

pheromone\_matrix[tour[i+1]][tour[i]] += 1.0 / length # symmetric update

print("Best tour:", best\_tour)

print("Best tour length:", best\_tour\_length)

**PB14:- RPC(Factorial)**

**Client:**

import xmlrpc.client

def main():

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

n=int(input("Enter the Number:"))

result = server.calculate\_factorial(n)

print(result)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Server:**

from xmlrpc.server import SimpleXMLRPCServer

def factorial(n):

if n==0 or n==1:

return 1

else:

return n\*factorial(n-1)

server = SimpleXMLRPCServer(('localhost',8000))

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

server.serve\_forever()

**PB15:- MapReduce(CharCount & WordCount)**

**CharCount:**

from mrjob.job import MRJob

class MRCharCount(MRJob):

def mapper(self, \_, line):

for char in line.strip():

yield char, 1

def reducer(self, char, counts):

yield char, sum(counts)

if \_\_name\_\_ == '\_\_main\_\_':

MRCharCount.run()

**WordCount:**

from mrjob.job import MRJob

import re

WORD\_REGEXP = re.compile(r"[\w']+")

class MRWordCount(MRJob):

def mapper(self, \_, line):

for word in WORD\_REGEXP.findall(line):

yield word.lower(), 1

def reducer(self, word, counts):

yield word, sum(counts)

if \_\_name\_\_ == '\_\_main\_\_':

MRWordCount.run()

**PB16:- RMI(String Concatenation)**

**Client:**

import Pyro4

uri = input("Enter URI of Server:")

concatenator = Pyro4.Proxy(uri)

str1 = input("Enter the string 1")

str2 = input("Enter the string 2")

result = concatenator.concatenate(str1, str2)

print(result)

**Server:**

import Pyro4

@Pyro4.expose

class stringconcatenator:

def concatenate(self,str1,str2):

return str1+str2

daemon=Pyro4.Daemon()

uri=daemon.register(stringconcatenator)

print(uri)

daemon.requestLoop()