In [ ]:

```
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.server_forever()
```

In [ ]:

```
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()
```

In [ ]:

```
import Pyro4

@Pyro.expose
class StringConcatenator():
    def concatenator(self,str1,str2):
        return str1+str2
daemon=Pyro4.Daemon()
uri=daemon.register(StringConcatenator())
print(uri)
daemon.requestLoop()
```

In [ ]:

```
    import Pyro4
    uri=input("Enter the URI")
    concatenator=Pyro4.Proxy(uri)
    str1=input("Enter string 1")
    str2=input("Enter string 2")
    result=concatenator.concatenate(str1,str2)
    print(result)
```

In [ ]:

```
def map_reduce(file_path,target_word):
    word_count=0
    char_count=0
    with open(file_path,'r') as file:
        mapped_data=[(word.lower(),1) for line in file for word in line.strip().split()]
        file.seek(0)
        char_count=sum(len(line) for line in file)


    for word,count in mapped_data:
        if word==target_word.lower():
            word_count+=count
    return word_count,char_count
file_path='test.txt'
target_word='start'
frequency,total_chars=map_reduce(file_path,target_word)
print(f"The word {target_word} appears {frequency} times")
print(f"The character count is {total_chars}")
```

In [6]:

```python
# List of server names
servers = ["Server A", "Server B", "Server C"]

# List of client requests
client_requests = ["Request 1", "Request 2", "Request 3", "Request 4", "Request 5", "Request 6"]

server_index=0

for request in client_requests:
    server=servers[server_index]
    print(f"{request} is handled by {server}")
    server_index=(server_index+1)%len(servers)
```

```
Request 1 is handled by Server A
Request 2 is handled by Server B
Request 3 is handled by Server C
Request 4 is handled by Server A
Request 5 is handled by Server B
Request 6 is handled by Server C
```

In [17]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score,confusion_matrix
iris=load_iris()
X=iris.data
y=iris.target

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)

scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)

k=5
knn=KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_scaled,y_train)

y_pred=knn.predict(X_test_scaled)
acc=accuracy_score(y_test,y_pred)
acc
cnf=confusion_matrix(y_test,y_pred)
cnf
```

Out[17]:

```
array([[10,  0,  0],
       [ 0,  9,  0],
       [ 0,  0, 11]], dtype=int64)
```

In [1]:

```
def map_reduce(file_path):
    with open(file_path,'r') as file:
        mapped=[line.strip().split(',') for line in file]
        mapped=[(int(year),int(temp)) for year,temp in mapped]
    coolest_year=min(mapped,key=lambda x:x[1])
    hottest_year=max(mapped,key=lambda x:x[1])
    return coolest_year,hottest_year
file_path='9th ass.txt'
coolest,hottest=map_reduce(file_path)

print(f"The coolest year {coolest[0]} with temperature {coolest[1]}")
```

```
The coolest year 2016 with temperature 15
```

In [ ]:

```
#pip install tensorflow tensorflow-hub
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt

# Function to load an image
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_image(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    img = img[tf.newaxis, :] #add batch dimension
    return img
# Function to display an image
def show_image(image):
    image = tf.squeeze(image, axis=0) #remove batch dimension
    plt.imshow(image)
    plt.title("Stylized Image")
# Load content and style images
content_image = load_image('content.jpg')
style_image = load_image('style.jpg')
# Load the style transfer model
hub_model = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/2')
# Perform style transfer
stylized_image = hub_model(tf.constant(content_image), tf.constant(style_image))[0]
# Display the images
show_image(stylized_image)
plt.show()
```

In [14]:

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

# Define the evaluation function
def eval_func(individual):
    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()
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)  # Register the evaluation function

# Genetic Algorithm parameters
population = toolbox.population(n=50)
algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.1, ngen=20)

# 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)
```

```
gen     nevals
0       50
1       24
2       30
3       26
4       26
5       28
6       24
7       28
8       30
9       31
10      29
11      27
12      26
13      30
14      37
15      32
16      27
17      26
18      37
19      32
20      23
Best individual: [-0.008843978634866249, 0.0037125257999822223, 0.0008238714637211638]
Best fitness: 9.267757009823837e-05
```

In [15]:

```python
import numpy as np

# Parameters
num_cities = 10
num_ants = 100
num_iterations = 100
alpha = 1         # Influence of pheromone
beta = 2          # Influence of distance (heuristic)
evaporation_rate = 0.1

# Initialize distance matrix (random symmetric distances)
distance_matrix = np.random.randint(1, 100, size=(num_cities, num_cities))
np.fill_diagonal(distance_matrix, 0)
# Make it symmetric (TSP is usually symmetric)
distance_matrix = (distance_matrix + distance_matrix.T) // 2

# Initialize pheromone matrix with small positive values
pheromone_matrix = np.ones((num_cities, num_cities))

best_tour = None
best_tour_length = float('inf')

# ACO algorithm
for iteration in range(num_iterations):
    all_tours = []
    all_lengths = []

    for ant in range(num_ants):
        visited = set()
        current_city = np.random.randint(num_cities)
        tour = [current_city]
        visited.add(current_city)

        while len(visited) < num_cities:
            probabilities = []
            for next_city in range(num_cities):
                if next_city in visited:
                    probabilities.append(0)
                else:
                    pheromone = pheromone_matrix[current_city][next_city] ** alpha
                    heuristic = (1 / distance_matrix[current_city][next_city]) ** beta
                    probabilities.append(pheromone * heuristic)

            probabilities = np.array(probabilities)
            probabilities /= probabilities.sum()
            next_city = np.random.choice(range(num_cities), p=probabilities)
            tour.append(next_city)
            visited.add(next_city)
            current_city = next_city

        # Return to starting city to complete the tour
        tour.append(tour[0])

        # Calculate tour length
        tour_length = sum(
            distance_matrix[tour[i]][tour[i + 1]] for i in range(len(tour) - 1)
        )

        all_tours.append(tour)
        all_lengths.append(tour_length)

        # Update pheromones on the tour
        for i in range(len(tour) - 1):
            from_city = tour[i]
            to_city = tour[i + 1]
            pheromone_matrix[from_city][to_city] += 1 / tour_length
            pheromone_matrix[to_city][from_city] += 1 / tour_length  # symmetric

        # Track the best tour
        if tour_length < best_tour_length:
            best_tour_length = tour_length
            best_tour = tour

    # Evaporate pheromones
    pheromone_matrix *= (1 - evaporation_rate)

# Output the best tour
print("Best tour found:", best_tour)
print("Best tour length:", best_tour_length)
```

```
Best tour found: [3, 7, 5, 9, 0, 1, 6, 8, 4, 2, 3]
Best tour length: 333
```

In [16]:

```
import numpy as np

# Define the objective function (fitness function)
def objective_function(x):
    return np.sum(x**2)

# Clonal Selection Algorithm
def clonal_selection_algorithm(num_antibodies, num_dimensions, search_space, num_generations, num_clones, clone_factor, r
    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))
```

```
Best Solution: [-2.11647270e-04  2.17840302e-03  5.50043947e-05]
Objective Value: 4.793259782767753 6e-06
```

In [ ]:

```
from xmlrpc.server import SimpleXMLRPCServer

# Arithmetic operation functions
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Error: Division by zero"
    return a / b

def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Start XML-RPC server
server = SimpleXMLRPCServer(("localhost", 8000))
print("Server is running on port 8000...")

# Register functions
server.register_function(add, "add")
server.register_function(subtract, "subtract")
server.register_function(multiply, "multiply")
server.register_function(divide, "divide")
server.register_function(factorial, "factorial")

server.serve_forever()
```

In [ ]:

```
import xmlrpc.client

def main():
    server = xmlrpc.client.ServerProxy("http://localhost:8000")
    print("Choose operation:")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Factorial")

    choice = int(input("Enter your choice (1-5): "))

    if choice in [1, 2, 3, 4]:
        a = int(input("Enter first number: "))
        b = int(input("Enter second number: "))

        if choice == 1:
            result = server.add(a, b)
        elif choice == 2:
            result = server.subtract(a, b)
        elif choice == 3:
            result = server.multiply(a, b)
        elif choice == 4:
            result = server.divide(a, b)

    elif choice == 5:
        n = int(input("Enter a number: "))
        result = server.factorial(n)
    else:
        result = "Invalid choice!"

    print("Result:", result)

if __name__ == "__main__":
    main()
```

In [ ]:

```python
import Pyro4

@Pyro4.expose
class PalindromeChecker():
    def is_palindrome(self, text):
        cleaned = text.lower().replace(" ", "")  # Optional: remove spaces and lowercase
        return cleaned == cleaned[::-1]

daemon = Pyro4.Daemon()
uri = daemon.register(PalindromeChecker)
print("Server is ready. URI is:", uri)
daemon.requestLoop()
```

In [ ]:

```python
import Pyro4

uri = input("Enter the URI shown by the server: ")
checker = Pyro4.Proxy(uri)

user_input = input("Enter a string to check if it's a palindrome: ")
result = checker.is_palindrome(user_input)

if result:
    print("Yes, it's a palindrome!")
else:
    print("No, it's not a palindrome.")
```

In [1]:

```python
def map_reduce(file_path, target_word):
    word_count = 0
    char_count = 0
    sentence_count = 0

    with open(file_path, 'r') as file:
        # Count words (map phase)
        mapped_data = [(word.lower(), 1) for line in file for word in line.strip().split()]

        # Reset file pointer to read again
        file.seek(0)

        # Count characters and sentences
        for line in file:
            char_count += len(line)
            sentence_count += line.count('.') + line.count('!') + line.count('?')

    # Count occurrences of the target word (reduce phase)
    for word, count in mapped_data:
        if word == target_word.lower():
            word_count += count

    return word_count, char_count, sentence_count

# Example usage
file_path = 'test.txt'
target_word = 'start'
frequency, total_chars, total_sentences = map_reduce(file_path, target_word)

print(f"The word '{target_word}' appears {frequency} times.")
print(f"The character count is {total_chars}.")
print(f"The number of sentences is {total_sentences}.")
```

```
The word 'start' appears 2 times.
The character count is 950.
The number of sentences is 1.
```

In [2]:

```python
def map_reduce(file_path, target_word):
    word_count = 0
    char_count = 0
    unique_words = set()

    with open(file_path, 'r') as file:
        mapped_data = []
        for line in file:
            words = line.strip().split()
            for word in words:
                word_lower = word.lower()
                mapped_data.append((word_lower, 1))
                unique_words.add(word_lower)

        # Reset file pointer and calculate character count
        file.seek(0)
        char_count = sum(len(line) for line in file)

    # Count the occurrences of the target word
    for word, count in mapped_data:
        if word == target_word.lower():
            word_count += count

    return word_count, char_count, len(unique_words)

# Example usage
file_path = 'test.txt'
target_word = 'start'
frequency, total_chars, unique_word_count = map_reduce(file_path, target_word)

print(f"The word '{target_word}' appears {frequency} times.")
print(f"The character count is {total_chars}.")
print(f"The number of unique words is {unique_word_count}.")
```

```
The word 'start' appears 2 times.
The character count is 950.
The number of unique words is 100.
```

In [ ]: