

Practice Programs

Problem 1: Basic Activation Function Visualizer

Problem Statement: Write a Python program to plot 6 common activation functions (Sigmoid, Tanh, ReLU, Leaky ReLU, ELU, Softplus) on a single figure with subplots. Each plot should display the function curve, its derivative, and annotate key properties like range and formula.

Dataset: No external dataset needed — generate x values using `numpy.linspace(-10, 10, 500)`

Solution Outline:

```
python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-10, 10, 500)

def sigmoid(x): return 1 / (1 + np.exp(-x))
def tanh(x): return np.tanh(x)
def relu(x): return np.maximum(0, x)
def leaky_relu(x, alpha=0.1): return np.where(x > 0, x, alpha * x)
def elu(x, alpha=1.0): return np.where(x > 0, x, alpha * (np.exp(x) - 1))
def softplus(x): return np.log1p(np.exp(x))

functions = [sigmoid, tanh, relu, leaky_relu, elu, softplus]
names = ['Sigmoid', 'Tanh', 'ReLU', 'Leaky ReLU', 'ELU', 'Softplus']

fig, axes = plt.subplots(2, 3, figsize=(15, 10))
for ax, fn, name in zip(axes.flat, functions, names):
    y = fn(x)
    ax.plot(x, y, label=name, color='blue')
    ax.axhline(0, color='k', linewidth=0.5)
    ax.axvline(0, color='k', linewidth=0.5)
    ax.set_title(name)
    ax.legend()
plt.tight_layout()
plt.show()
```

Libraries: numpy, matplotlib **Dataset Link:** [No dataset needed — synthetic data]

Problem 2: Activation Functions on Real Data Distribution

Problem Statement: Load the **Iris dataset**, extract the petal length feature, normalize it, and apply 5 activation functions to it. Plot the original vs transformed distributions using histograms side by side to see how each activation squashes/transforms real data.

Dataset Link: [Iris Dataset — UCI ML Repo](#) or `sklearn.datasets.load_iris()`

Solution Outline:

```
python
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
iris = load_iris()
```

```
X = iris.data[:, 2].reshape(-1, 1) # petal length
```

```
scaler = MinMaxScaler(feature_range=(-1, 1))
```

```
x = scaler.fit_transform(X).flatten()
```

```
activations = {
```

```
    'Sigmoid': lambda x: 1/(1+np.exp(-x)),
```

```
    'Tanh': np.tanh,
```

```
    'ReLU': lambda x: np.maximum(0, x),
```

```
    'ELU': lambda x: np.where(x>0, x, np.exp(x)-1),
```

```

    'Softplus': lambda x: np.log1p(np.exp(x))
}

fig, axes = plt.subplots(2, 3, figsize=(16, 10))
axes[0, 0].hist(x, bins=20, color='gray')
axes[0, 0].set_title('Original (Normalized)')

for ax, (name, fn) in zip(axes.flat[1:], activations.items()):
    ax.hist(fn(x), bins=20)
    ax.set_title(f'After {name}')

plt.tight_layout()
plt.show()

Libraries: sklearn, numpy, matplotlib

```

Problem 3: Effect of Activation Functions on Gradient Flow (Vanishing Gradient)

Problem Statement: Simulate a 10-layer deep neural network using random weights and visualize how gradients shrink (vanish) or explode across layers for Sigmoid, Tanh, and ReLU. Plot gradient magnitude vs layer depth for all three.

Dataset: Synthetic random data `numpy.random.randn`

Solution Outline:

```

python

import numpy as np

import matplotlib.pyplot as plt

```

```

def sigmoid(x): return 1/(1+np.exp(-x))
def sigmoid_grad(x): s = sigmoid(x); return s*(1-s)
def tanh_grad(x): return 1 - np.tanh(x)**2
def relu_grad(x): return (x > 0).astype(float)

layers = 10
grads = {'Sigmoid': [], 'Tanh': [], 'ReLU': []}
grad_fns = {'Sigmoid': sigmoid_grad, 'Tanh': tanh_grad, 'ReLU': relu_grad}

for name, gfn in grad_fns.items():
    g = 1.0
    for _ in range(layers):
        x = np.random.randn(100)
        g *= np.mean(np.abs(gfn(x)))
        grads[name].append(g)

for name, g_list in grads.items():
    plt.plot(range(1, layers+1), g_list, marker='o', label=name)

plt.xlabel('Layer')
plt.ylabel('Gradient Magnitude')
plt.title('Vanishing Gradient Across Layers')
plt.legend()

```

```
plt.show()
```

Libraries: numpy, matplotlib **Dataset Link:** Synthetic — no external data needed

Problem 4: Training a Neural Network with Different Activation Functions on MNIST

Problem Statement: Train a simple 2-layer neural network on the **MNIST handwritten digits dataset** using Sigmoid, ReLU, and Tanh activations separately. Plot training accuracy vs epochs for all three on the same graph to compare convergence speed.

Dataset Link: [MNIST via TensorFlow/Keras](#) — `tf.keras.datasets.mnist.load_data()`

Solution Outline:

```
python
```

```
import tensorflow as tf
```

```
import matplotlib.pyplot as plt
```

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
x_train = x_train.reshape(-1, 784) / 255.0
```

```
histories = {}
```

```
for activation in ['sigmoid', 'relu', 'tanh']:
```

```
    model = tf.keras.Sequential([
```

```
        tf.keras.layers.Dense(128, activation=activation, input_shape=(784,)),
```

```
        tf.keras.layers.Dense(64, activation=activation),
```

```
        tf.keras.layers.Dense(10, activation='softmax')
```

```
    ])
```

```
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
```

```
metrics=['accuracy'])

h = model.fit(x_train, y_train, epochs=10, verbose=0, validation_split=0.1)

histories[activation] = h.history['val_accuracy']


for name, acc in histories.items():

    plt.plot(acc, label=name)

plt.xlabel('Epoch'); plt.ylabel('Validation Accuracy')

plt.title('Activation Function Comparison on MNIST')

plt.legend(); plt.show()
```

Libraries: tensorflow, matplotlib

Problem 5: Decision Boundary Visualization with Different Activations

Problem Statement: Use the **make_moons** or **make_circles** dataset from sklearn and train a neural network classifier using different activation functions. Visualize and compare the decision boundaries produced by each activation function.

Dataset Link: `sklearn.datasets.make_moons(n_samples=500, noise=0.3)`

Solution Outline:

```
python

from sklearn.datasets import make_moons

from sklearn.neural_network import MLPClassifier

import numpy as np

import matplotlib.pyplot as plt


X, y = make_moons(n_samples=500, noise=0.3, random_state=42)
```

```

activations = ['logistic', 'tanh', 'relu']
xx, yy = np.meshgrid(np.linspace(-3,3,300), np.linspace(-3,3,300))

fig, axes = plt.subplots(1, 3, figsize=(18, 5))
for ax, act in zip(axes, activations):
    clf = MLPClassifier(hidden_layer_sizes=(50,50), activation=act, max_iter=500)
    clf.fit(X, y)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    ax.contourf(xx, yy, Z, alpha=0.4)
    ax.scatter(X[:,0], X[:,1], c=y, edgecolors='k', s=20)
    ax.set_title(f'Activation: {act}')
plt.tight_layout(); plt.show()

Libraries: sklearn, numpy, matplotlib

```

Problem 6: Activation Function Behavior in CNN Feature Maps

Problem Statement: Load a sample image from the **CIFAR-10** dataset, pass it through a single convolutional layer, and apply ReLU, Sigmoid, and Tanh activations. Visualize the resulting feature maps side by side to see how each activation changes the image representation.

Dataset Link: `tf.keras.datasets.cifar10.load_data()`

Solution Outline:

```

python

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

```

```

(x_train, _), _ = tf.keras.datasets.cifar10.load_data()
img = x_train[0:1].astype('float32') / 255.0

conv = tf.keras.layers.Conv2D(8, (3,3), padding='same', use_bias=False)
feature_map = conv(img)

activations = {'ReLU': tf.nn.relu, 'Sigmoid': tf.nn.sigmoid, 'Tanh': tf.nn.tanh}

fig, axes = plt.subplots(3, 8, figsize=(20, 8))
for row, (name, fn) in enumerate(activations.items()):
    out = fn(feature_map)[0]
    for col in range(8):
        axes[row, col].imshow(out[:, :, col], cmap='viridis')
        axes[row, col].axis('off')
    if col == 0:
        axes[row, col].set_ylabel(name, fontsize=12)
plt.suptitle('Feature Maps with Different Activations')
plt.show()

```

Libraries: tensorflow, matplotlib, numpy

Problem 7: Activation Function Output Range & Saturation Analysis

Problem Statement: Generate large-value inputs (ranging from -100 to 100) and analyze **saturation behavior** of Sigmoid, Tanh, Softsign, and Hard Sigmoid. Plot the output range and identify the saturation zones. Annotate dead zones on the

plot.

Dataset: Synthetic `numpy.linspace(-100, 100, 1000)`

Solution Outline:

python

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
x = np.linspace(-100, 100, 1000)
```

```
fns = {
```

```
    'Sigmoid': lambda x: 1/(1+np.exp(-x)),
```

```
    'Tanh': np.tanh,
```

```
    'Softsign': lambda x: x/(1+np.abs(x)),
```

```
    'Hard Sigmoid': lambda x: np.clip(0.2*x+0.5, 0, 1)
```

```
}
```

```
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
```

```
for ax, (name, fn) in zip(axes.flat, fns.items()):
```

```
    y = fn(x)
```

```
    ax.plot(x, y)
```

```
    ax.axhspan(fn(x).max()-0.01, fn(x).max()+0.01, alpha=0.3, color='red',  
label='Saturation')
```

```
    ax.set_title(name)
```

```
    ax.legend()
```

```
plt.tight_layout(); plt.show()
```

Libraries: numpy, matplotlib

Problem 8: Custom Activation Function Benchmarking on Regression

Problem Statement: Using the **California Housing dataset**, train regression models using a neural network with 5 different activation functions (ReLU, Sigmoid, Tanh, SELU, Swish). Compare and plot **MAE (Mean Absolute Error)** vs training epochs for each.

Dataset Link: `sklearn.datasets.fetch_california_housing()` or [StatLib Repository](#)

Solution Outline:

```
python
```

```
from sklearn.datasets import fetch_california_housing
```

```
from sklearn.preprocessing import StandardScaler
```

```
import tensorflow as tf
```

```
import matplotlib.pyplot as plt
```

```
data = fetch_california_housing()
```

```
X, y = data.data, data.target
```

```
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)
```

```
activations = ['relu', 'sigmoid', 'tanh', 'selu', 'swish']
```

```
histories = {}
```

```
for act in activations:
```

```
    model = tf.keras.Sequential([
```

```
        tf.keras.layers.Dense(64, activation=act, input_shape=(X.shape[1],)),
```

```

tf.keras.layers.Dense(32, activation=act),
tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
h = model.fit(X, y, epochs=50, verbose=0, validation_split=0.2)
histories[act] = h.history['val_mae']

for act, mae in histories.items():
    plt.plot(mae, label=act)
plt.xlabel('Epoch'); plt.ylabel('MAE')
plt.title('Regression Performance by Activation')
plt.legend(); plt.show()
Libraries: tensorflow, sklearn, matplotlib

```

Problem 9: Activation Function Heatmap in a Network Layer (Weight × Activation)

Problem Statement: Using the **Wine Quality dataset**, train a neural network, extract weights from the first layer, and create a heatmap of weight × activation_output for ReLU and Sigmoid. This visualizes which neurons are firing and which are dead.

Dataset Link: [Wine Quality Dataset — UCI](https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv) or
pandas.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv', sep=';')

Solution Outline:

```

python

import pandas as pd, numpy as np

```

```
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('winequality-red.csv', sep=';')
X = df.drop('quality', axis=1).values
y = df['quality'].values

for activation in ['relu', 'sigmoid']:
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(16, activation=activation, input_shape=(X.shape[1],)),
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    model.fit(X, y, epochs=20, verbose=0)

    layer_model = tf.keras.Model(inputs=model.input,
    outputs=model.layers[0].output)

    activations_out = layer_model.predict(X[:50])
    sns.heatmap(activations_out, cmap='RdYlGn')
    plt.title(f'Neuron Activations — {activation}')
    plt.show()
```

Libraries: tensorflow, seaborn, pandas, matplotlib

Problem 10: Interactive Activation Function Explorer with Noise Robustness

Problem Statement: Using the **Breast Cancer Wisconsin dataset**, add Gaussian noise to input features, then compare how a neural network with different activation functions handles noisy input. Plot classification accuracy vs noise level ($\sigma = 0$ to 2) for Sigmoid, ReLU, and Tanh.

Dataset Link: `sklearn.datasets.load_breast_cancer()` or [UCI Breast Cancer Dataset](#)

Solution Outline:

```
python
```

```
from sklearn.datasets import load_breast_cancer
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
```

```
import tensorflow as tf, numpy as np, matplotlib.pyplot as plt
```

```
data = load_breast_cancer()
```

```
X, y = data.data, data.target
```

```
X = StandardScaler().fit_transform(X)
```

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

```
noise_levels = np.linspace(0, 2, 10)
```

```
results = {act: [] for act in ['sigmoid', 'relu', 'tanh']}
```

```
for act in results:
```

```
    model = tf.keras.Sequential([
```

```
        tf.keras.layers.Dense(32, activation=act, input_shape=(X.shape[1],)),
```

```
        tf.keras.layers.Dense(1, activation='sigmoid')
```

```

])

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

model.fit(X_train, y_train, epochs=30, verbose=0)

for sigma in noise_levels:

    X_noisy = X_test + np.random.normal(0, sigma, X_test.shape)

    _, acc = model.evaluate(X_noisy, y_test, verbose=0)

    results[act].append(acc)

for act, accs in results.items():

    plt.plot(noise_levels, accs, marker='o', label=act)

plt.xlabel('Noise Level ( $\sigma$ )'); plt.ylabel('Accuracy')

plt.title('Robustness to Noise by Activation Function')

plt.legend(); plt.show()

```

Libraries: tensorflow, sklearn, numpy, matplotlib

Based on Assignment 2 of NN

1. Handwritten Digit Classification Using TensorFlow

Problem Statement:

Train a neural network to classify handwritten digits (0–9) using the offline MNIST dataset. The program should show forward propagation, loss calculation, and backpropagation during training.

```

import tensorflow as tf
from tensorflow.keras.datasets import mnist

```

```

from tensorflow.keras.utils import to_categorical

# Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize and reshape
x_train = x_train.reshape(-1, 784) / 255.0
x_test = x_test.reshape(-1, 784) / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Define model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

# Compile model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train model
history = model.fit(x_train, y_train, epochs=5, batch_size=32)

# Evaluate
loss, accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy:", accuracy)

```

2. Binary Classification of Breast Cancer Dataset Using PyTorch

Problem Statement:

Train a neural network to classify whether a tumor is malignant or benign using the offline Breast Cancer dataset from scikit-learn. The training process should demonstrate forward propagation and backpropagation.

```

import torch
import torch.nn as nn
import torch.optim as optim

```

```

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load dataset
cancer = load_breast_cancer()
X = cancer.data
Y = cancer.target

# Normalize data
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

# Convert to tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)

# Define neural network
class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(30, 16)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(16, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)    # Forward propagation
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x

model = NeuralNet()
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

```

```

# Training loop
for epoch in range(100):
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    optimizer.zero_grad()
    loss.backward()    # Backpropagation
    optimizer.step()

    if (epoch + 1) % 10 == 0:
        print(f"Epoch [{epoch+1}/100], Loss: {loss.item():.4f}")

# Test accuracy
with torch.no_grad():
    predicted = model(X_test)
    predicted = (predicted > 0.5).float()
    accuracy = (predicted.eq(y_test).sum() / y_test.shape[0]).item()
    print("Accuracy:", accuracy)

```

3. Iris Flower Classification Using TensorFlow

Problem Statement:

Develop a neural network to classify iris flowers into three species using the offline Iris dataset. The program should illustrate the steps of forward propagation and backpropagation.

```

import tensorflow as tf
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.utils import to_categorical

# Load dataset
iris = load_iris()
X = iris.data
Y = iris.target

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

```

```

# Convert output to categorical
Y = to_categorical(Y, 3)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

# Model definition
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu', input_shape=(4,)),
    tf.keras.layers.Dense(8, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compile model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train model
model.fit(X_train, y_train, epochs=50, batch_size=8)

# Evaluate
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Accuracy:", accuracy)

```

4. House Price Prediction Using PyTorch

Problem Statement:

Train a neural network to predict house prices using the offline California Housing dataset. The program should emphasize forward propagation and backpropagation during regression training.

```

import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

```

```

# Load dataset
housing = fetch_california_housing()
X = housing.data
Y = housing.target

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

# Convert to tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
y_test = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)

# Define network
class HousePriceNN(nn.Module):
    def __init__(self):
        super(HousePriceNN, self).__init__()
        self.fc1 = nn.Linear(8, 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x

model = HousePriceNN()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training
for epoch in range(200):
    prediction = model(X_train) # Forward propagation
    loss = criterion(prediction, y_train)

```

```

optimizer.zero_grad()
loss.backward()      # Backpropagation
optimizer.step()

if (epoch + 1) % 20 == 0:
    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")

# Testing
with torch.no_grad():
    test_pred = model(X_test)
    test_loss = criterion(test_pred, y_test)
    print("Test Loss:", test_loss.item())

```

5. Fashion Item Classification Using TensorFlow

Problem Statement:

Build a neural network to classify fashion items such as shirts, shoes, and bags using the offline Fashion-MNIST dataset. The code should demonstrate forward propagation and backpropagation during model training.

```

import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical

# Load dataset
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Preprocess
x_train = x_train.reshape(-1, 784) / 255.0
x_test = x_test.reshape(-1, 784) / 255.0

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Neural network
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

```

```

# Compile
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Training
model.fit(x_train, y_train, epochs=5, batch_size=64)

# Evaluation
loss, accuracy = model.evaluate(x_test, y_test)
print("Accuracy:", accuracy)

```

Based on Assignment 3 of NN

1. Problem Statement: Implement fuzzy union, intersection, difference, and complement operations for two fuzzy sets representing customer satisfaction levels.

A = {'Service': 0.8, 'Quality': 0.6, 'Price': 0.7, 'Delivery': 0.5}

B = {'Service': 0.7, 'Quality': 0.9, 'Price': 0.4, 'Delivery': 0.8}

union = {}

intersection = {}

difference = {}

complement_A = {}

for key in A:

 union[key] = max(A[key], B[key])

 intersection[key] = min(A[key], B[key])

 difference[key] = min(A[key], 1 - B[key])

 complement_A[key] = 1 - A[key]

print("Set A:", A)

print("Set B:", B)

print("Union:", union)

print("Intersection:", intersection)

print("Difference A-B:", difference)

print("Complement of A:", complement_A)

2. Problem Statement: Implement fuzzy membership functions to classify rainfall intensity into Low, Medium, and High categories.

```
import numpy as np
import matplotlib.pyplot as plt

rainfall = np.arange(0, 101, 1)

low = np.maximum(0, np.minimum(1, (50 - rainfall) / 50))
medium = np.maximum(0, np.minimum((rainfall - 20) / 30, (80 - rainfall) / 30))
high = np.maximum(0, np.minimum(1, (rainfall - 50) / 50))

plt.plot(rainfall, low, label='Low')
plt.plot(rainfall, medium, label='Medium')
plt.plot(rainfall, high, label='High')

plt.xlabel('Rainfall (mm)')
plt.ylabel('Membership Value')
plt.title('Fuzzy Membership Functions for Rainfall')
plt.legend()
plt.grid(True)
plt.show()
```

3. Problem Statement: Implement fuzzy relation composition for a transportation system showing the relationship between cities and transport facilities.

```
import numpy as np

R1 = np.array([
    [0.8, 0.6, 0.4],
    [0.7, 0.9, 0.5],
    [0.6, 0.8, 0.7]
])

R2 = np.array([
    [0.9, 0.5, 0.4],
    [0.6, 0.8, 0.7],
    [0.5, 0.7, 0.9]
```

```

])
rows = R1.shape[0]
cols = R2.shape[1]
composition = np.zeros((rows, cols))

for i in range(rows):
    for j in range(cols):
        composition[i][j] = max(
            min(R1[i][0], R2[0][j]),
            min(R1[i][1], R2[1][j]),
            min(R1[i][2], R2[2][j])
        )

print("Relation Matrix R1:")
print(R1)

print("\nRelation Matrix R2:")
print(R2)

print("\nMax-Min Composition:")
print(composition)

```

4. Problem Statement: Implement fuzzy inference for a fan speed controller based on temperature and humidity values.

```

temperature = 32
humidity = 65

temp_hot = 0.8
temp_warm = 0.2

humidity_high = 0.7
humidity_medium = 0.3

rule1 = min(temp_hot, humidity_high)
rule2 = min(temp_warm, humidity_medium)

fan_speed_high = rule1
fan_speed_medium = rule2

```

```
print("Temperature:", temperature)
print("Humidity:", humidity)

print("\nFan Speed Memberships:")
print("High:", fan_speed_high)
print("Medium:", fan_speed_medium)
```

5. **Problem Statement:** Implement fuzzy decision-making to select the best laptop based on performance, battery life, and price.

```
laptops = {
    'Laptop_A': {'performance': 0.9, 'battery': 0.7, 'price': 0.6},
    'Laptop_B': {'performance': 0.8, 'battery': 0.9, 'price': 0.7},
    'Laptop_C': {'performance': 0.7, 'battery': 0.8, 'price': 0.9}
}

scores = {}

for laptop, values in laptops.items():
    score = min(values['performance'], values['battery'], values['price'])
    scores[laptop] = score

best_laptop = max(scores, key=scores.get)

print("Laptop Scores:")
for laptop, score in scores.items():
    print(laptop, ":", score)

print("\nBest Laptop:", best_laptop)
```

Based on Assignment 4 of NN

1. **Problem Statement:** Design and implement a fuzzy inference system to determine washing machine speed based on cloth dirtiness and load weight.

```
import numpy as np
```

```

# Inputs
dirtiness = 70
load_weight = 6

# Membership functions for dirtiness
low_dirt = max(0, min(1, (50 - dirtiness) / 50))
medium_dirt = max(0, min((dirtiness - 30) / 20, (80 - dirtiness) / 30))
high_dirt = max(0, min(1, (dirtiness - 60) / 40))

# Membership functions for load weight
light_load = max(0, min(1, (5 - load_weight) / 5))
heavy_load = max(0, min(1, (load_weight - 3) / 7))

# Rules
slow_speed = max(min(low_dirt, light_load), min(medium_dirt, light_load))
medium_speed = min(medium_dirt, heavy_load)
fast_speed = max(min(high_dirt, light_load), min(high_dirt, heavy_load))

# Defuzzification
speed = (slow_speed * 30 + medium_speed * 60 + fast_speed * 90) / (
    slow_speed + medium_speed + fast_speed
)

print("Washing Machine Speed:", round(speed, 2))

```

-
2. **Problem Statement:** Design and implement a fuzzy inference system to decide restaurant tip amount based on food quality and service quality.

```

# Inputs
food_quality = 8
service_quality = 6

# Membership functions
poor_food = max(0, min(1, (5 - food_quality) / 5))
good_food = max(0, min((food_quality - 3) / 3, (9 - food_quality) / 3))
excellent_food = max(0, min(1, (food_quality - 7) / 3))

poor_service = max(0, min(1, (5 - service_quality) / 5))
good_service = max(0, min((service_quality - 3) / 3, (9 - service_quality) / 3))
excellent_service = max(0, min(1, (service_quality - 7) / 3))

```

```

# Rules
low_tip = max(poor_food, poor_service)
medium_tip = max(min(good_food, good_service), min(excellent_food,
good_service))
high_tip = max(excellent_food, excellent_service)

# Defuzzification
tip = (low_tip * 5 + medium_tip * 15 + high_tip * 25) / (
    low_tip + medium_tip + high_tip
)

print("Suggested Tip Percentage:", round(tip, 2), "%")

```

3. **Problem Statement:** Design and implement a fuzzy inference system to control room heater level based on room temperature and outside temperature.

```

# Inputs
room_temp = 18
outside_temp = 10

# Membership functions
cold_room = max(0, min(1, (22 - room_temp) / 10))
warm_room = max(0, min((room_temp - 15) / 5, (30 - room_temp) / 10))

cold_outside = max(0, min(1, (20 - outside_temp) / 20))
mild_outside = max(0, min((outside_temp - 10) / 10, (30 - outside_temp) / 10))

# Rules
low_heat = warm_room
medium_heat = min(cold_room, mild_outside)
high_heat = min(cold_room, cold_outside)

# Defuzzification
heater = (low_heat * 20 + medium_heat * 50 + high_heat * 90) / (
    low_heat + medium_heat + high_heat
)

print("Heater Level:", round(heater, 2))

```

-
4. **Problem Statement:** Design and implement a fuzzy inference system to estimate student performance based on attendance and assignment completion.

```
# Inputs
attendance = 82
assignment = 75

# Membership functions
low_attendance = max(0, min(1, (60 - attendance) / 60))
good_attendance = max(0, min((attendance - 50) / 20, (100 - attendance) / 30))

low_assignment = max(0, min(1, (60 - assignment) / 60))
good_assignment = max(0, min((assignment - 50) / 20, (100 - assignment) / 30))

# Rules
poor_performance = max(low_attendance, low_assignment)
average_performance = min(good_attendance, low_assignment)
excellent_performance = min(good_attendance, good_assignment)

# Defuzzification
performance = (
    poor_performance * 40 +
    average_performance * 70 +
    excellent_performance * 95
) / (
    poor_performance +
    average_performance +
    excellent_performance
)

print("Student Performance Score:", round(performance, 2))
```

5. **Problem Statement:** Design and implement a fuzzy inference system to determine traffic signal duration based on traffic density and waiting time.

```
# Inputs
traffic_density = 80
waiting_time = 50
```

```

# Membership functions
low_density = max(0, min(1, (50 - traffic_density) / 50))
high_density = max(0, min(1, (traffic_density - 30) / 70))

short_wait = max(0, min(1, (30 - waiting_time) / 30))
long_wait = max(0, min(1, (waiting_time - 20) / 40))

# Rules
short_signal = min(low_density, short_wait)
medium_signal = max(min(high_density, short_wait), min(low_density,
long_wait))
long_signal = min(high_density, long_wait)

# Defuzzification
signal_duration = (
    short_signal * 20 +
    medium_signal * 45 +
    long_signal * 70
) / (
    short_signal +
    medium_signal +
    long_signal
)

print("Traffic Signal Duration:", round(signal_duration, 2), "seconds")

```

Based on Assignment 1 of Cloud Computing

Prerequisites

Install KVM and related tools:

```

sudo apt update
sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients virtinst
bridge-utils -y

```

```
sudo systemctl enable --now libvirtd
```

Verify KVM installation:

```
virsh list --all
```

Create a folder for all scripts:

```
mkdir ~/kvm-cloud-practicals  
cd ~/kvm-cloud-practicals
```

1. Create Two Ubuntu VMs (Private Cloud)

Create the script:

```
nano create_private_cloud.sh
```

Paste:

```
#!/bin/bash
```

```
virt-install --name cloud-node1 \  
--ram 2048 --vcpus 2 \  
--disk path=/var/lib/libvirt/images/cloud-node1.qcow2,size=15 \  
--os-variant ubuntu22.04 \  
--cdrom /home/$USER/Downloads/ubuntu-22.04.iso \  
--network network=default \  
--graphics vnc --noautoconsole
```

```
virt-install --name cloud-node2 \  
--ram 2048 --vcpus 2 \  
--disk path=/var/lib/libvirt/images/cloud-node2.qcow2,size=15 \  
--os-variant ubuntu22.04 \  
--cdrom /home/$USER/Downloads/ubuntu-22.04.iso \  
--network network=default \  
--graphics vnc --noautoconsole
```

Save and run:

```
chmod +x create_private_cloud.sh
./create_private_cloud.sh
```

Check VMs:

```
virsh list --all
```

2. Install Apache Web Server Inside a VM

Start the VM:

```
virsh start cloud-node1
```

Find the IP address:

```
virsh domifaddr cloud-node1
```

SSH into the VM:

```
ssh username@VM_IP
```

Inside the VM, run:

```
sudo apt update
sudo apt install apache2 -y
echo "<h1>Cloud Server Running</h1>" | sudo tee /var/www/html/index.html
sudo systemctl start apache2
```

Open a browser and visit:

```
http://VM_IP
```

3. Create a Snapshot (Backup)

Start the VM if not already running:

```
virsh start cloud-node1
```

Create snapshot:

```
virsh snapshot-create-as cloud-node1 backup1 "Snapshot before changes"
```

View snapshots:

```
virsh snapshot-list cloud-node1
```

Restore snapshot:

```
virsh snapshot-revert cloud-node1 backup1
```

4. Clone a VM to Simulate Scaling

Shutdown the original VM:

```
virsh shutdown cloud-node1
```

Clone the VM:

```
virt-clone \  
--original cloud-node1 \  
--name cloud-node1-copy \  
--file /var/lib/libvirt/images/cloud-node1-copy.qcow2
```

Start the copied VM:

```
virsh start cloud-node1-copy
```

Verify:

```
virsh list --all
```

5. Configure Load Balancing with Nginx

Create a third VM named loadbalancer-vm.

Start all VMs:

```
virsh start cloud-node1
virsh start cloud-node2
virsh start loadbalancer-vm
```

Find the IP addresses:

```
virsh domifaddr cloud-node1
virsh domifaddr cloud-node2
virsh domifaddr loadbalancer-vm
```

SSH into loadbalancer-vm:

```
ssh username@LOADBALANCER_IP
```

Install Nginx:

```
sudo apt update
sudo apt install nginx -y
```

Edit Nginx configuration:

```
sudo nano /etc/nginx/sites-available/default
```

Replace with:

```
upstream backend {
    server CLOUD_NODE1_IP;
    server CLOUD_NODE2_IP;
}

server {
    listen 80;

    location / {
```

```
    proxy_pass http://backend;
  }
}
```

Restart Nginx:

```
sudo systemctl restart nginx
```

Open browser:

```
http://LOADBALANCER_IP
```

6. Transfer a File Between Two VMs

Create a sample file on the host:

```
echo "Cloud Computing Practical" > sample.txt
```

Transfer file:

```
scp sample.txt username@CLOUD_NODE2_IP:/home/username/
```

Verify inside VM:

```
ssh username@CLOUD_NODE2_IP
ls
```

7. Monitor Resource Usage of VMs

Run:

```
virsh list --name | while read vm
do
  echo "VM Name: $vm"
  virsh dominfo $vm
done
```

This displays:

- CPU count
 - RAM allocated
 - VM state
-

8. Automatically Start VMs on Host Boot

Enable auto-start:

```
virsh autostart cloud-node1  
virsh autostart cloud-node2
```

Verify:

```
virsh dominfo cloud-node1 | grep Autostart
```

Expected output:

Autostart: enable

9. Create Shared Storage Between Host and VM

Create a shared folder:

```
mkdir ~/shared-cloud
```

Add a test file:

```
echo "Cloud Storage File" > ~/shared-cloud/info.txt
```

Attach to VM:

```
virsh attach-disk cloud-node1 ~/shared-cloud vdb --targetbus virtio
```

Inside VM, check:

```
lsblk
```

10. Shut Down All VMs Together

Create a script:

```
nano shutdown_all.sh
```

Paste:

```
#!/bin/bash
```

```
for vm in cloud-node1 cloud-node2 cloud-node1-copy loadbalancer-vm
do
    virsh shutdown $vm
done
```

Run:

```
chmod +x shutdown_all.sh
./shutdown_all.sh
```

Verify:

```
virsh list --all
```

The VMs should show shut off.

Based on Assignment 2 of Cloud Computing

1. Student Attendance Management System

Problem Statement:

Develop a web application on Google App Engine to store and display student attendance.

app.yaml

runtime: python39

entrypoint: gunicorn -b :\$PORT main:app

main.py

```
from flask import Flask, request
```

```
app = Flask(__name__)
```

```
attendance = []
```

```
@app.route('/')  
def home():
```

```
    output = "<h2>Student Attendance</h2>"
```

```
    output += """
```

```
    <form method='post' action='/add'>
```

```
    Name: <input type='text' name='name'><br>
```

```
    Roll No: <input type='text' name='roll'><br>
```

```
    Status: <input type='text' name='status'><br>
```

```
    <input type='submit' value='Add'>
```

```
    </form><hr>
```

```
    """
```

```
    for a in attendance:
```

```
        output += f"{a['name']} - {a['roll']} - {a['status']}<br>"
```

```
    return output
```

```
@app.route('/add', methods=['POST'])
```

```
def add():
```

```
    attendance.append({
```

```
        'name': request.form['name'],
```

```
        'roll': request.form['roll'],
```

```
        'status': request.form['status']
```

```
    })
```

```
    return home()
```

```
if __name__ == '__main__':  
    app.run()
```

Run:

```
dev_appserver.py app.yaml
```

2. Library Management System

Problem Statement:

Create a cloud-based library system to add and display book records.

app.yaml

```
runtime: python39  
entrypoint: gunicorn -b :$PORT main:app
```

main.py

```
from flask import Flask, request
```

```
app = Flask(__name__)
```

```
books = []
```

```
@app.route('/')  
def index():
```

```
    html = """
```

```
    <h2>Library Management</h2>
```

```
    <form method='post' action='/add'>
```

```
    Book Name: <input type='text' name='book'><br>
```

```
    Author: <input type='text' name='author'><br>
```

```
    <input type='submit' value='Add Book'>
```

```
    </form><hr>
```

```
    """
```

```
    for b in books:
```

```
        html += f"{b['book']} by {b['author']}<br>"
```

```
    return html
```

```
@app.route('/add', methods=['POST'])
```

```
def add():
```

```
    books.append({
```

```
        'book': request.form['book'],
```

```

        'author': request.form['author']
    })
    return index()

if __name__ == '__main__':
    app.run()

```

Run:

```
dev_appserver.py app.yaml
```

3. Employee Record Management System

Problem Statement:

Develop an application to store employee details such as ID, name, and department.

app.yaml

```
runtime: python39
entrypoint: gunicorn -b :$PORT main:app
```

main.py

```
from flask import Flask, request
```

```
app = Flask(__name__)
```

```
employees = []
```

```
@app.route('/')
def home():
```

```
    data = """
```

```
    <h2>Employee Record Management</h2>
```

```
    <form method='post' action='/add'>
```

```
    Employee ID: <input type='text' name='id'><br>
```

```
    Name: <input type='text' name='name'><br>
```

```
    Department: <input type='text' name='dept'><br>
```

```
    <input type='submit' value='Save'>
```

```
    </form><hr>
```

```
    """
```

```
    for e in employees:
```

```
        data += f"{e['id']} - {e['name']} - {e['dept']}<br>"
```

```
return data
```

```
@app.route('/add', methods=['POST'])
def add():
    employees.append({
        'id': request.form['id'],
        'name': request.form['name'],
        'dept': request.form['dept']
    })
    return home()
```

```
if __name__ == '__main__':
    app.run()
```

Run:

```
dev_appserver.py app.yaml
```

4. Student Result Processing System

Problem Statement:

Create a web application that accepts marks of three subjects and calculates percentage and grade.

app.yaml

```
runtime: python39
entrypoint: gunicorn -b :$PORT main:app
```

main.py

```
from flask import Flask, request
```

```
app = Flask(__name__)
```

```
@app.route('/')
def form():
    return """
    <h2>Student Result Processing</h2>
    <form method='post' action='/result'>
    Subject 1: <input type='number' name='s1'><br>
    Subject 2: <input type='number' name='s2'><br>
    Subject 3: <input type='number' name='s3'><br>
    """
```

```

<input type='submit' value='Calculate'>
</form>
"""

@app.route('/result', methods=['POST'])
def result():
    s1 = int(request.form['s1'])
    s2 = int(request.form['s2'])
    s3 = int(request.form['s3'])

    percent = (s1 + s2 + s3) / 3

    if percent >= 75:
        grade = 'A'
    elif percent >= 60:
        grade = 'B'
    elif percent >= 40:
        grade = 'C'
    else:
        grade = 'Fail'

    return f"Percentage = {percent}%<br>Grade = {grade}"

if __name__ == '__main__':
    app.run()

```

5. Road Project Monitoring System

Problem Statement:

Develop a road project monitoring application to store project name, contractor name, and project status. This is especially relevant for infrastructure and government contract monitoring.

app.yaml

runtime: python39

entrypoint: gunicorn -b :\$PORT main:app

main.py

from flask import Flask, request

app = Flask(__name__)

```

projects = []

@app.route('/')
def home():
    html = """
    <h2>Road Project Monitoring System</h2>
    <form method='post' action='/add'>
    Project Name: <input type='text' name='project'><br>
    Contractor Name: <input type='text' name='contractor'><br>
    Status: <input type='text' name='status'><br>
    <input type='submit' value='Add Project'>
    </form><hr>
    """
    for p in projects:
        html += f"{p['project']} - {p['contractor']} - {p['status']}<br>"
    return html

@app.route('/add', methods=['POST'])
def add():
    projects.append({
        'project': request.form['project'],
        'contractor': request.form['contractor'],
        'status': request.form['status']
    })
    return home()

if __name__ == '__main__':
    app.run()

```

Run:

```
dev_appserver.py app.yaml
```

Based on Assignment 3 of Cloud Computing

Before running:

- Download and add the CloudSim library to your Java project.
- Import cloudsim-3.0.3.jar.

- Create each program in a separate Java file.
-

1. Priority-Based Cloudlet Scheduling

Problem Statement

Create a CloudSim program where cloudlets are scheduled according to priority. A lower priority value means the cloudlet executes first.

```
import org.cloudbus.cloudsim.*;
import org.cloudbus.cloudsim.core.CloudSim;

import java.util.*;

public class PriorityScheduling {

    public static void main(String[] args) {
        try {
            int numUser = 1;
            Calendar calendar = Calendar.getInstance();
            boolean traceFlag = false;

            CloudSim.init(numUser, calendar, traceFlag);

            Datacenter datacenter = createDatacenter("Datacenter_1");
            DatacenterBroker broker = new DatacenterBroker("Broker");
            int brokerId = broker.getId();

            Vm vm = new Vm(0, brokerId, 1000, 1, 512, 1000, 10000, "Xen", new
CloudletSchedulerTimeShared());
            broker.submitVmList(Arrays.asList(vm));

            List<Cloudlet> cloudletList = new ArrayList<>();

            cloudletList.add(createCloudlet(0, brokerId, 4000));
            cloudletList.add(createCloudlet(1, brokerId, 1000));
            cloudletList.add(createCloudlet(2, brokerId, 2000));

            // Custom Priority Scheduling
            Collections.sort(cloudletList,
```

```

Comparator.comparingLong(Cloudlet::getCloudletLength));

    broker.submitCloudletList(cloudletList);

    CloudSim.startSimulation();
    CloudSim.stopSimulation();

    List<Cloudlet> result = broker.getCloudletReceivedList();

    System.out.println("Priority Scheduling Output:");
    for (Cloudlet c : result) {
        System.out.println("Cloudlet " + c.getCloudletId() + " finished");
    }

} catch (Exception e) {
    e.printStackTrace();
}
}

private static Cloudlet createCloudlet(int id, int brokerId, long length) {
    UtilizationModel utilizationModel = new UtilizationModelFull();

    Cloudlet cloudlet = new Cloudlet(id, length, 1, 300, 300,
        utilizationModel, utilizationModel, utilizationModel);
    cloudlet.setUserId(brokerId);
    return cloudlet;
}

private static Datacenter createDatacenter(String name) throws Exception {
    List<Host> hostList = new ArrayList<>();

    List<Pe> peList = new ArrayList<>();
    peList.add(new Pe(0, new PeProvisionerSimple(1000)));

    hostList.add(new Host(0,
        new RamProvisionerSimple(2048),
        new BwProvisionerSimple(10000),
        1000000,
        peList,
        new VmSchedulerTimeShared(peList)));

    DatacenterCharacteristics characteristics = new DatacenterCharacteristics(

```

```

        "x86", "Linux", "Xen", hostList, 10.0, 3.0, 0.05, 0.001, 0.0);

    return new Datacenter(name, characteristics,
        new VmAllocationPolicySimple(hostList), new LinkedList<>(), 0);
}
}

```

2. Shortest Job First (SJF) Scheduling

Problem Statement

Create a CloudSim program where the smallest cloudlet is executed first.

```

Collections.sort(cloudletList, new Comparator<Cloudlet>() {
    @Override
    public int compare(Cloudlet c1, Cloudlet c2) {
        return Long.compare(c1.getCloudletLength(), c2.getCloudletLength());
    }
});

```

Full scheduling section:

```

List<Cloudlet> cloudletList = new ArrayList<>();
cloudletList.add(createCloudlet(0, brokerId, 6000));
cloudletList.add(createCloudlet(1, brokerId, 2000));
cloudletList.add(createCloudlet(2, brokerId, 4000));

Collections.sort(cloudletList, (c1, c2) ->
    Long.compare(c1.getCloudletLength(), c2.getCloudletLength()));

broker.submitCloudletList(cloudletList);

```

Expected Output

Execution Order: Cloudlet 1 -> Cloudlet 2 -> Cloudlet 0

3. Longest Job First (LJF) Scheduling

Problem Statement

Create a CloudSim simulation where the cloudlet with maximum length is executed first.

```
import java.util.*;

public class LongestJobFirst {
    public static void main(String[] args) {

        List<Integer> jobs = Arrays.asList(1000, 5000, 3000, 7000);

        jobs.sort(Collections.reverseOrder());

        System.out.println("LJF Scheduling Order:");

        for (int i = 0; i < jobs.size(); i++) {
            System.out.println("Job Length: " + jobs.get(i));
        }
    }
}
```

Expected Output

```
LJF Scheduling Order:
Job Length: 7000
Job Length: 5000
Job Length: 3000
Job Length: 1000
```

4. Round Robin Scheduling

Problem Statement

Simulate Round Robin scheduling in CloudSim where each cloudlet gets equal CPU time.

```
import java.util.*;

public class RoundRobinScheduling {
```

```

public static void main(String[] args) {

    int[] burstTime = {10, 5, 8};
    int quantum = 2;

    int[] remaining = burstTime.clone();
    int time = 0;

    System.out.println("Round Robin Execution:");

    boolean done;

    do {
        done = true;

        for (int i = 0; i < remaining.length; i++) {
            if (remaining[i] > 0) {
                done = false;

                if (remaining[i] > quantum) {
                    time += quantum;
                    remaining[i] -= quantum;
                    System.out.println("Task " + i + " executed till " + time);
                } else {
                    time += remaining[i];
                    System.out.println("Task " + i + " completed at " + time);
                    remaining[i] = 0;
                }
            }
        }
    } while (!done);
}

```

Expected Output

```

Task 0 executed till 2
Task 1 executed till 4
Task 2 executed till 6
...

```

5. Min-Min Scheduling Algorithm

Problem Statement

Implement the Min-Min scheduling algorithm where the task with minimum execution time is assigned first.

```
import java.util.*;

public class MinMinScheduling {

    public static void main(String[] args) {

        int[] tasks = {12, 5, 20, 7};

        List<Integer> taskList = new ArrayList<>();

        for (int t : tasks) {
            taskList.add(t);
        }

        while (!taskList.isEmpty()) {
            int min = Collections.min(taskList);
            System.out.println("Executing Task with time: " + min);
            taskList.remove(Integer.valueOf(min));
        }
    }
}
```

Expected Output

```
Executing Task with time: 5
Executing Task with time: 7
Executing Task with time: 12
Executing Task with time: 20
```

Based on Assignment 4 of Cloud Computing

1. Display a Welcome Message

Problem Statement

Write an Apex program to display the message "Welcome to Apex Programming".

Solution

```
public class WelcomeMessage {  
    public static void showMessage() {  
        System.debug('Welcome to Apex Programming');  
    }  
}
```

Output

Welcome to Apex Programming

2. Add Two Numbers

Problem Statement

Write an Apex program to add two numbers and display the result.

Solution

```
public class AdditionProgram {  
    public static void addNumbers() {  
        Integer a = 10;  
        Integer b = 20;  
        Integer sum = a + b;  
  
        System.debug('Sum = ' + sum);  
    }  
}
```

Output

Sum = 30

3. Find Whether a Number is Even or Odd

Problem Statement

Write an Apex program to check whether a number is even or odd.

Solution

```
public class EvenOddProgram {  
    public static void checkNumber() {  
        Integer num = 15;  
  
        if(num % 2 == 0) {  
            System.debug(num + ' is Even');  
        } else {  
            System.debug(num + ' is Odd');  
        }  
    }  
}
```

Output

15 is Odd

4. Find the Largest of Two Numbers

Problem Statement

Write an Apex program to find the larger number between two integers.

Solution

```
public class LargestNumber {  
    public static void findLargest() {  
        Integer a = 45;  
        Integer b = 28;
```

```
    if(a > b) {  
        System.debug(a + ' is larger');  
    } else {  
        System.debug(b + ' is larger');  
    }  
}  
}
```

Output

45 is larger

5. Print Numbers from 1 to 10

Problem Statement

Write an Apex program to print numbers from 1 to 10 using a loop.

Solution

```
public class PrintNumbers {  
    public static void showNumbers() {  
        for(Integer i = 1; i <= 10; i++) {  
            System.debug(i);  
        }  
    }  
}
```

Output

1
2
3
4
5
6
7
8
9
10

6. Find the Factorial of a Number

Problem Statement

Write an Apex program to find the factorial of a number.

Solution

```
public class FactorialProgram {  
    public static void calculateFactorial() {  
        Integer num = 5;  
        Integer fact = 1;  
  
        for(Integer i = 1; i <= num; i++) {  
            fact = fact * i;  
        }  
  
        System.debug('Factorial = ' + fact);  
    }  
}
```

Output

Factorial = 120

7. Reverse a String

Problem Statement

Write an Apex program to reverse a given string.

Solution

```
public class ReverseString {  
    public static void reverseText() {  
        String str = 'Apex';  
        String rev = "";  
  
        for(Integer i = str.length() - 1; i >= 0; i--) {  
            rev += str.substring(i, i + 1);  
        }  
    }  
}
```

```
    }  
    System.debug('Reversed String = ' + rev);  
  }  
}
```

Output

Reversed String = xepA

8. Count the Number of Characters in a String

Problem Statement

Write an Apex program to count the number of characters in a string.

Solution

```
public class CharacterCount {  
    public static void countCharacters() {  
        String text = 'Salesforce';  
        Integer count = text.length();  
  
        System.debug('Number of characters = ' + count);  
    }  
}
```

Output

Number of characters = 10

9. Store and Display Student Names Using List

Problem Statement

Write an Apex program to store student names in a list and display them.

Solution

```
public class StudentListProgram {
```

```
public static void displayStudents() {  
    List<String> students = new List<String>();  
  
    students.add('Rahul');  
    students.add('Priya');  
    students.add('Amit');  
  
    for(String name : students) {  
        System.debug(name);  
    }  
}
```

Output

Rahul
Priya
Amit

10. Find the Sum of Elements in a List

Problem Statement

Write an Apex program to find the sum of all numbers stored in a list.

Solution

```
public class SumOfList {  
    public static void calculateSum() {  
        List<Integer> numbers = new List<Integer>{10, 20, 30, 40};  
        Integer sum = 0;  
  
        for(Integer num : numbers) {  
            sum += num;  
        }  
  
        System.debug('Total Sum = ' + sum);  
    }  
}
```

Output

Total Sum = 100

How to Run These Programs in Salesforce Developer Console

1. Open Salesforce Developer Console.
2. Click **File** → **New** → **Apex Class**.
3. Enter the class name.
4. Paste the code.
5. Save the class.
6. Open **Debug** → **Open Execute Anonymous Window**.
7. Call the method, for example: `WelcomeMessage.showMessage();`
8. Click **Execute** and check the debug log for output.