```
import numpy as np

# Define unit step function
def unitStep(v):
    return 1 if v >= 0 else 0

# Design Perceptron Model
def perceptronModel(x, w, b):
    v = np.dot(w, x) + b
    return unitStep(v)

# Perceptron training function with loop for epochs
def train_perceptron(X, y, w, b, learning_rate=0.01, epochs=10):
    for epoch in range(epochs):
        for i in range(len(X)):
            y_pred = perceptronModel(X[i], w, b)
            error = y[i] - y_pred
            w += learning_rate * error * X[i]
            b += learning_rate * error
    return w, b

# Logic functions
def NOT_logicFunction(x, wNOT, bNOT):
    return perceptronModel(x, wNOT, bNOT)

def AND_logicFunction(x, wAND, bAND):
    return perceptronModel(x, wAND, bAND)

def OR_logicFunction(x, wOR, bOR):
    return perceptronModel(x, wOR, bOR)

# XOR logic using perceptrons
def XOR_logicFunction(x, epochs=10):
    wNOT = -1
```

```
    bNOT = 0.5
    x_not = np.array([0, 1])
    y_not = np.array([1, 0])
    wNOT, bNOT = train_perceptron(x_not, y_not, wNOT, bNOT, epochs=epochs)

    wAND = np.array([1, 1])
    bAND = -1.5
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y_and = np.array([0, 0, 0, 1])
    wAND, bAND = train_perceptron(X, y_and, wAND, bAND, epochs=epochs)

    wOR = np.array([1, 1])
    bOR = -0.5
    y_or = np.array([0, 1, 1, 1])
    wOR, bOR = train_perceptron(X, y_or, wOR, bOR, epochs=epochs)

    # XOR logic computation
    y_and_result = AND_logicFunction(x, wAND, bAND)
    y_or_result = OR_logicFunction(x, wOR, bOR)
    y_not_result = NOT_logicFunction(y_and_result, wNOT, bNOT)

    final_input = np.array([y_or_result, y_not_result])
    final_output = AND_logicFunction(final_input, wAND, bAND)

    return final_output

# Input data for XOR
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
Y = np.array([0, 1, 1, 0])

# Output results for XOR using a loop
for i in range(len(X)):
    print("XOR({}, {}) = {}".format(X[i][0], X[i][1], XOR_logicFunction(X[i])))
```

MODIFICATIONS:

1. **Epochs Loop in train_perceptron:** Now the number of epochs is flexible and can be controlled with a parameter, making it easier to adjust training.

2.**Training Logic in XOR:** The XOR logic now uses a loop to iterate over training data, handling weights and biases more systematically.

3. **Loop for Output Testing:** Instead of manually printing each XOR result, a loop is used to print the XOR output for all inputs.

4. **Loop for Epochs in train_perceptron:** The perceptron training now runs for multiple epochs, improving learning stability.

5. **Automatic XOR Output Loop:** The code now uses a loop to automatically test the XOR logic for all input combinations, making it cleaner and more efficient.

6. **More Flexible Training:** The epochs parameter is now adjustable when calling XOR_logicFunction, allowing control over training intensity.

OUTPUT:

XOR(0, 0) = 0

XOR(0, 1) = 1

XOR(1, 0) = 1

XOR(1, 1) = 0