Implement Perceptron Model for Basic Logical Functions like Logical AND, Logical NOT and Logical OR.

```
class Perceptron:
    def __init__(self, input_size, learning_rate=1, epochs=10):
        self.weights = [0.0 for _ in range(input_size)]
        self.bias = 0.0
        self.lr = learning_rate
        self.epochs = epochs

    def activation(self, x):
        return 1 if x >= 0 else 0

    def predict(self, inputs):
        summation = sum(w * x for w, x in zip(self.weights, inputs)) + self.bias
        return self.activation(summation)

    def train(self, training_data):
        for epoch in range(self.epochs):
            for inputs, expected in training_data:
                prediction = self.predict(inputs)
                error = expected - prediction
                # Update weights and bias
                for i in range(len(self.weights)):
                    self.weights[i] += self.lr * error * inputs[i]
                self.bias += self.lr * error


# AND gate training data
and_data = [
    ([0, 0], 0),
    ([0, 1], 0),
    ([1, 0], 0),
    ([1, 1], 1)
]

print("Training Perceptron for AND Gate")
and_perceptron = Perceptron(input_size=2)
and_perceptron.train(and_data)

# Test AND gate
print("AND Gate Results:")
for inputs, _ in and_data:
    print(f"{inputs} => {and_perceptron.predict(inputs)}")

# OR gate training data
or_data = [
    ([0, 0], 0),
    ([0, 1], 1),
    ([1, 0], 1),
    ([1, 1], 1)
]

print("\nTraining Perceptron for OR Gate")
or_perceptron = Perceptron(input_size=2)
or_perceptron.train(or_data)

# Test OR gate
print("OR Gate Results:")
for inputs, _ in or_data:
    print(f"{inputs} => {or_perceptron.predict(inputs)}")

# NOT gate training data
not_data = [
    ([0], 1),
    ([1], 0)
]

print("\nTraining Perceptron for NOT Gate")
not_perceptron = Perceptron(input_size=1)
not_perceptron.train(not_data)

# Test NOT gate
print("NOT Gate Results:")
for inputs, _ in not_data:
    print(f"{inputs} => {not_perceptron.predict(inputs)}")
```