```
import numpy as np

class Perceptron:

    def __init__(self, input_size, learning_rate=0.1):

        self.weights = np.array([0 if i % 2 == 0 else 1 for i in range(input_size)], dtype=float)

        self.bias = 1

        self.learning_rate = learning_rate

    def activation_function(self, x):

        return 1 if x >= 0 else 0

    def predict(self, inputs):

        weighted_sum = np.dot(inputs, self.weights) + self.bias

        return self.activation_function(weighted_sum)

    def train(self, training_data, labels, epochs):

        for _ in range(epochs):

            for inputs, label in zip(training_data, labels):

                prediction = self.predict(inputs)

                error = label - prediction

                self.weights += self.learning_rate * error * inputs

                self.bias += self.learning_rate * error

numbers = [str(i) for i in range(10)]

training_data = np.array([[int(bit) for bit in format(int(c), '08b')] for c in numbers])

labels = np.array([0 if int(c) % 2 == 0 else 1 for c in numbers])

input_size = training_data.shape[1]

perceptron = Perceptron(input_size)

perceptron.train(training_data, labels, epochs=100)

print("Number Predictions:")

for num in numbers:

    binary_input = np.array([int(bit) for bit in format(int(num), '08b')])

    prediction = perceptron.predict(binary_input)

    result = "Odd" if prediction == 1 else "Even"

    print(f"Number: {num} - Prediction: {result}")
```

Output:

```
Number Predictions:

Number: 0 - Prediction: Even

Number: 1 - Prediction: Odd

Number: 2 - Prediction: Even

Number: 3 - Prediction: Odd

Number: 4 - Prediction: Even

Number: 5 - Prediction: Odd

Number: 6 - Prediction: Even

Number: 7 - Prediction: Odd

Number: 8 - Prediction: Even

Number: 9 - Prediction: Odd
```