In [ ]:

```
#program2
#1BM22AI035
#DEVELOP AND IMPLEMENT A PROGRAM TO EXECUTE THE PERCEPTRON LEARNING ALGORI
#TO TRAIN A SINGLE LAYER PERCEPTRON FOR BINARY CLASSIFICSTION TEST
#CREATE A ROBUST ALGORITHM THAT REFINES THE MODELS WEIGHT ITERATIVELY RESU
#IN A PROFFICIENT SINGLE LAYERED PERCEPTRON CAPABLE OF EFFICIENTLY HANDLIN
```

In [3]:

```
import numpy as np
class Perceptron:
    def __init__(self, learning_rate=0.01, n_iter=1000):
        self.learning_rate = learning_rate
        self.n_iter = n_iter
        self.weights = None
        self.bias = None
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0
        for _ in range(self.n_iter):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self._activation_function(linear_output)
                update = self.learning_rate * (y[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update
    def _activation_function(self, x):
        return np.where(x >= 0, 1, 0)
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        y_predicted = self._activation_function(linear_output)
        return y_predicted
if __name__ == "__main__":
    from sklearn.datasets import make_blobs
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score
    X, y = make_blobs(n_samples=100, centers=2, random_state=42, cluster_s
    y = np.where(y == 0, 0, 1)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
    perceptron = Perceptron(learning_rate=0.1, n_iter=1000)
    perceptron.fit(X_train, y_train)
    predictions = perceptron.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

In [ ]:

```

```