In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

perceptron class

In [2]:
```
class Perceptron:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
        self.n_iterations = n_iterations

    def fit(self, X, y):
        self.weights = np.zeros(1 + X.shape[1])

        for _ in range(self.n_iterations):
            for xi, target in zip(X, y):
                update = self.learning_rate * (target - self.predict(xi))
                self.weights[1:] += update * xi
                self.weights[0] += update
        return self

    def net_input(self, X):
        return np.dot(X, self.weights[1:]) + self.weights[0]

    def predict(self, X):
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

In [3]:
```
# Define your own data
X = np.array([[2, 2], [4, 4], [4, 0], [3, 2], [8, 4], [8, 0]])
y = np.array([1, 1, -1, 1, -1, -1])  # Target labels for the data
```

In [4]:
```
# Create a perceptron instance
perceptron = Perceptron()
```

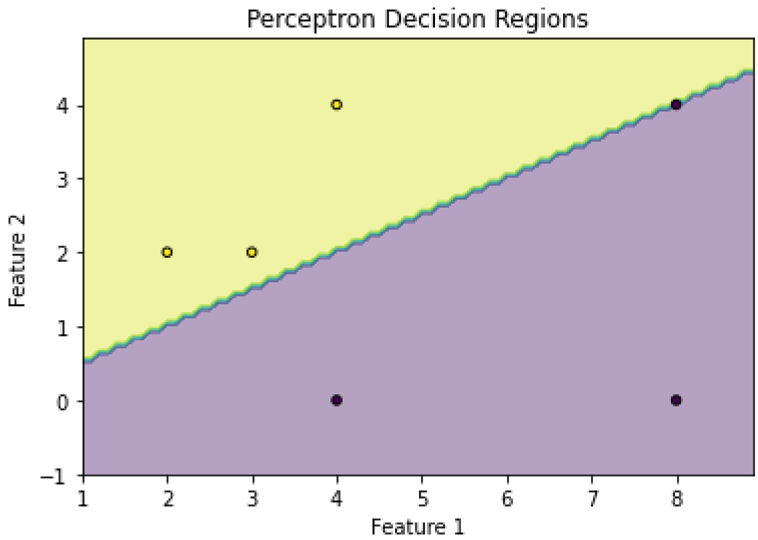
In [5]:
```
# Fit the perceptron to the data
perceptron.fit(X, y)
```

Out[5]:
```
<__main__.Perceptron at 0x2564ca5b5b0>
```

In [6]:

```
# Plot the decision boundary
def plot_decision_boundary(X, y, classifier):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                         np.arange(y_min, y_max, 0.1))
    Z = classifier.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Perceptron Decision Regions')
    plt.show()

plot_decision_boundary(X, y, perceptron)
```



In [ ]: