In [27]:
```
import numpy as np
import matplotlib.pyplot as plt
#from perceptron import Perceptron
from matplotlib.colors import ListedColormap

class Perceptron(object):
   def __init__(self, rate = 0.01, niter = 10):
      self.rate = rate
      self.niter = niter

   def fit(self, X, y):
      """Fit training data
      X : Training vectors, X.shape : [#samples, #features]
      y : Target values, y.shape : [#samples]
      """

      # weights
      self.weight = np.zeros(1 + X.shape[1])

      # Number of misclassifications
      self.errors = []  # Number of misclassifications

      for i in range(self.niter):
         err = 0
         for xi, target in zip(X, y):
            delta_w = self.rate * (target - self.predict(xi))
            self.weight[1:] += delta_w * xi
            self.weight[0] += delta_w
            err += int(delta_w != 0.0)
         self.errors.append(err)
      return self

   def net_input(self, X):
      """Calculate net input"""
      return np.dot(X, self.weight[1:]) + self.weight[0]

   def predict(self, X):
      """Return class label after unit step"""
      return np.where(self.net_input(X) >= 0.0, 1, -1)
```

In [28]:
```
# We will use the pandas library to load the Iris data set into a DataFrame ob
ject:

import pandas as pd
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/ir
is/iris.data', header=None)

df.tail()
```

Out[28]:

	0	1	2	3	4
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

In [29]:
```
df.iloc[145:150, 0:5]
```

Out[29]:

	0	1	2	3	4
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

In [30]:
```
# We extract the first 100 class labels that correspond to the 50 Iris-Setosa
 and 50 Iris-Versicolor flowers, respectively:
y = df.iloc[0:100, 4].values
y
```

Out[30]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor'],
      dtype=object)
```

In [31]:
```
# We want to convert the class  labels into the two integer: label1(Versicolo
r) and label-1(Setosa)
y = np.where(y == 'Iris-setosa', -1, 1)
y
```

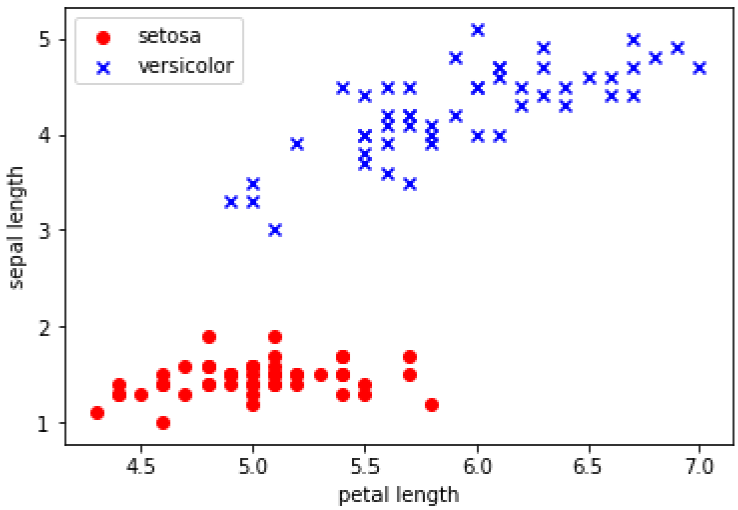
Out[31]:
```
array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1])
```

In [32]:
```
# let extract the first feature column(sepal length) and the third feature col
umn(pedal length)
X = df.iloc[0:100, [0, 2]].values
X
```

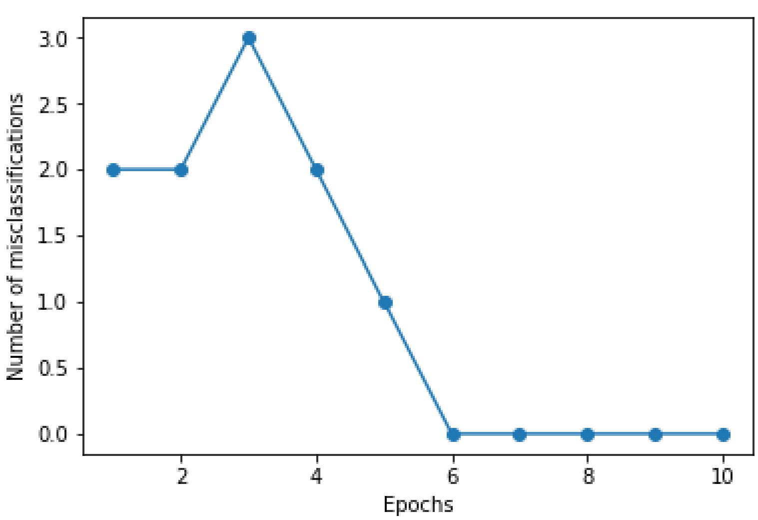
```
Out[32]:  array([[5.1, 1.4],
                 [4.9, 1.4],
                 [4.7, 1.3],
                 [4.6, 1.5],
                 [5. , 1.4],
                 [5.4, 1.7],
                 [4.6, 1.4],
                 [5. , 1.5],
                 [4.4, 1.4],
                 [4.9, 1.5],
                 [5.4, 1.5],
                 [4.8, 1.6],
                 [4.8, 1.4],
                 [4.3, 1.1],
                 [5.8, 1.2],
                 [5.7, 1.5],
                 [5.4, 1.3],
                 [5.1, 1.4],
                 [5.7, 1.7],
                 [5.1, 1.5],
                 [5.4, 1.7],
                 [5.1, 1.5],
                 [4.6, 1. ],
                 [5.1, 1.7],
                 [4.8, 1.9],
                 [5. , 1.6],
                 [5. , 1.6],
                 [5.2, 1.5],
                 [5.2, 1.4],
                 [4.7, 1.6],
                 [4.8, 1.6],
                 [5.4, 1.5],
                 [5.2, 1.5],
                 [5.5, 1.4],
                 [4.9, 1.5],
                 [5. , 1.2],
                 [5.5, 1.3],
                 [4.9, 1.5],
                 [4.4, 1.3],
                 [5.1, 1.5],
                 [5. , 1.3],
                 [4.5, 1.3],
                 [4.4, 1.3],
                 [5. , 1.6],
                 [5.1, 1.9],
                 [4.8, 1.4],
                 [5.1, 1.6],
                 [4.6, 1.4],
                 [5.3, 1.5],
                 [5. , 1.4],
                 [7. , 4.7],
                 [6.4, 4.5],
                 [6.9, 4.9],
                 [5.5, 4. ],
                 [6.5, 4.6],
                 [5.7, 4.5],
                 [6.3, 4.7],
```

```
       [4.9, 3.3],
       [6.6, 4.6],
       [5.2, 3.9],
       [5. , 3.5],
       [5.9, 4.2],
       [6. , 4. ],
       [6.1, 4.7],
       [5.6, 3.6],
       [6.7, 4.4],
       [5.6, 4.5],
       [5.8, 4.1],
       [6.2, 4.5],
       [5.6, 3.9],
       [5.9, 4.8],
       [6.1, 4. ],
       [6.3, 4.9],
       [6.1, 4.7],
       [6.4, 4.3],
       [6.6, 4.4],
       [6.8, 4.8],
       [6.7, 5. ],
       [6. , 4.5],
       [5.7, 3.5],
       [5.5, 3.8],
       [5.5, 3.7],
       [5.8, 3.9],
       [6. , 5.1],
       [5.4, 4.5],
       [6. , 4.5],
       [6.7, 4.7],
       [6.3, 4.4],
       [5.6, 4.1],
       [5.5, 4. ],
       [5.5, 4.4],
       [6.1, 4.6],
       [5.8, 4. ],
       [5. , 3.3],
       [5.6, 4.2],
       [5.7, 4.2],
       [5.7, 4.2],
       [6.2, 4.3],
       [5.1, 3. ],
       [5.7, 4.1]])
```

In [33]:
```
#visualization via a two dimensional scatter plot
plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x', label='versi
color')
plt.xlabel('petal length')
plt.ylabel('sepal length')
plt.legend(loc='upper left')
plt.show()
```



In [34]:
```
# let train our perceptron algoriym on the Iris data subset
# we plot the misclassification error to check the convergence for finding the
dicision boundary
pn = Perceptron(0.1, 10)
pn.fit(X, y)
plt.plot(range(1, len(pn.errors) + 1), pn.errors, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of misclassifications')
plt.show()
# We can see the plot of the misclassification errors versus the number of epo
chs as shown below
# Our perceptron converged after the 6th epoch or iteration
```

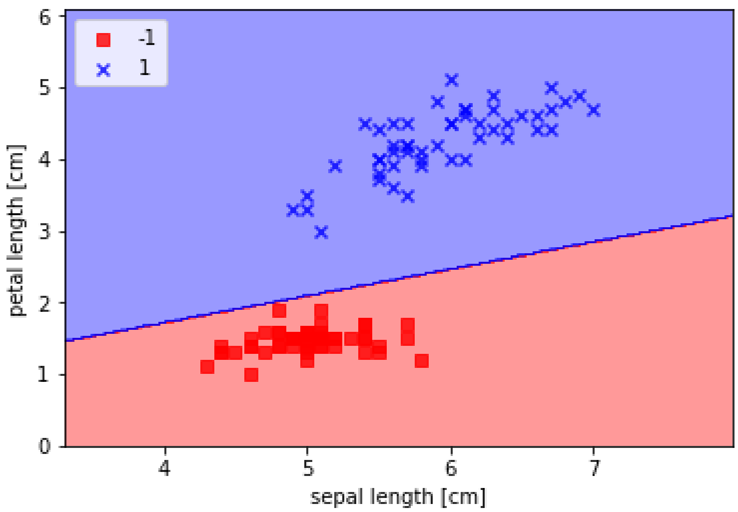


In [35]:
```
#We are able to classify the training samples above 90 percent accurate(will t
ry the perfect accurate)
```

In [36]:
```
# let define a number of colors and markers
# we create a color map from the list of colors
# we will find the the minimum amd maximum values for the two features
# we will use them as vectors to create a pair of grid arrays xx1 and xx2

def plot_decision_regions(X, y, classifier, resolution=0.02):
   # setup marker generator and color map
   markers = ('s', 'x', 'o', '^', 'v')
   colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
   cmap = ListedColormap(colors[:len(np.unique(y))])

   # plot the decision surface
   x1_min, x1_max = X[:,  0].min() - 1, X[:, 0].max() + 1
   x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
   xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
   np.arange(x2_min, x2_max, resolution))
   Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
   Z = Z.reshape(xx1.shape)
   plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
   plt.xlim(xx1.min(), xx1.max())
   plt.ylim(xx2.min(), xx2.max())

   # plot class samples
   for idx, cl in enumerate(np.unique(y)):
      plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
      alpha=0.8, c=cmap(idx),
      marker=markers[idx], label=cl)
```

In [37]:

```
# reshape the predicted c;ass labels Z into a grid with the same xx1 and xx2 d
imensions
# let draw a contour plot
plot_decision_regions(X, y, classifier=pn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



In [ ]:

```
# Our decision boundary was able to classify all flowers samples in the iris t
raining subset above 90% accurate.
```