In [11]:

```
%matplotlib inline
import numpy as np
from sklearn.datasets import make_blobs, load_iris
from matplotlib import pyplot as plt
from scipy import misc
import random
```

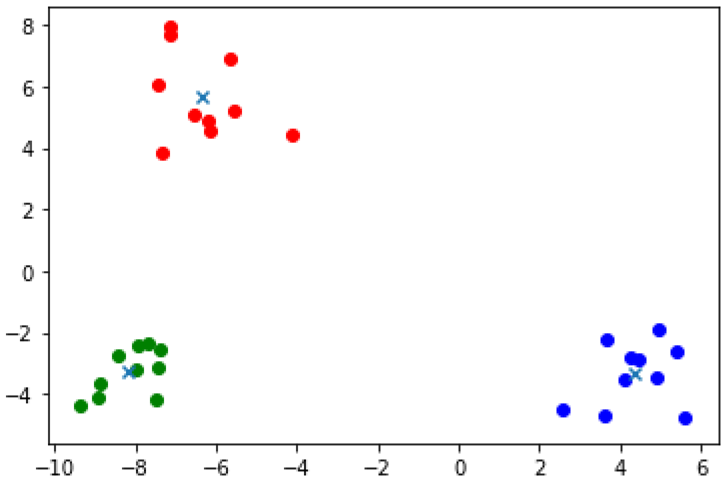
In [66]:

```
def kmeans(X,K,max_iter=1000):
    """
    Perform k-means on the dataset X.

    Parameters
    ----------
    X : ndarray of shape (n,d)
        Data are given row-wise
    K : int
        Number of clusters
    max_iter : int
        Maximum number of iterations to perform

    Returns
    -------
    means : ndarray of shape (K,d)
        the K recovered cluster means given row-wise
    groups : dict (of length K) of lists
        each list gives the indices of points in the cluster
    """
    random.seed(671)
    # initialize the means as randomly selected points in the data
    ### your code here
    n = X.shape[0]
    d = X.shape[1]
    initial_index = random.sample(range(n), K)
    means = np.zeros((K,d))
    for i in range(K):
        means[i][:] = X[initial_index[i]][:]

    # initialize the groups. a disctionary is recommended.
    ### your code here
    groups = {new_list: [] for new_list in range(K)}

    # iterate
    for i in range(max_iter):
        # assign each point to a group
        for j in range(X.shape[0]):
            groups[np.argmin(((means-X[j,:])**2).sum(axis=1))].append(j)

        # calculate new means
        ### your code here
        new_means = np.zeros((K,d))
        for k in range(K):
            index = groups[k]
            mean_temp = np.zeros((1,d))
            for jj in range(len(index)):
                mean_temp[0][:] = mean_temp[0][:] + X[index[jj]][:]
            for dd in range(d):
                mean_temp[0][dd] = mean_temp[0][dd]/len(index)

            new_means[k][:] = mean_temp

        # see if we have converged
        if np.allclose(means,new_means):
            print("Converged after {} iterations!".format(i))
            return means, groups
        else:
            means = new_means
            groups = {i:[] for i in range(K)}
    print("Failed to converge after {} iterations...".format(max_iter))
```

In [43]:

```
# toy example
X,y = make_blobs(n_samples=30,n_features=2,centers=3)
means, groups = kmeans(X,3)
colorstring='rgb'
for k in range(3):
    plt.scatter(X[groups[k],0],X[groups[k],1],c=colorstring[k])
plt.scatter(means[:,0],means[:,1],marker="x")
plt.show()
```

```
Converged after 1 iterations!
```



In [44]:

```
# iris dataset example
iris = load_iris()
X = iris.data
Y = iris.target
means, groups = kmeans(X,3)
conf = np.empty((3,3))
for i in range(3):
    for j in range(3):
        conf[i,j] = (Y[groups[j]]==i).sum()
print("Confusion Matrix")
print(conf)
```

```
Converged after 4 iterations!
Confusion Matrix
[[50.  0.  0.]
 [ 0.  2. 48.]
 [ 0. 36. 14.]]
```

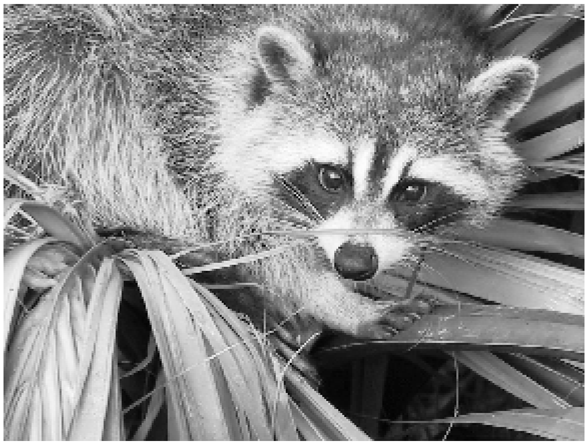
In [45]:

```
# save data for future use
iris_array = np.empty((X.shape[0],5))
iris_array[:,:-1] = X
iris_array[:,-1] = Y
np.savetxt("irisdata.txt",iris_array, fmt='%.1f')
```

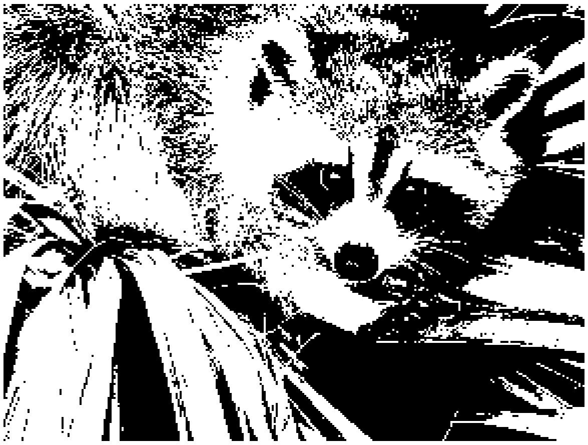
In [57]:

```
# compression example

# load image
f = misc.face(gray=True)
print("Original")
plt.imshow(f,cmap=plt.cm.Greys_r)
plt.axis('off')
plt.show()

# quantize and plot

### your code here.
f = misc.face(gray=True)

K_range = [2,4,6]
for K in K_range:

    print("Quantized k = {}".format(K))

    # turn the pixels to (1, 768*1024) vector
    f_flat = f.flatten()
    f_vert = f_flat.reshape(-1, 1)

    # run kmeans
    means, groups = kmeans(f_vert,K)
    for k in range(means.shape[0]):
        list_temp = groups[k]
        for i in range(len(list_temp)):
            f_vert[list_temp[i]] = means[k][0]

    # convert back to (768,1024)
    f_quant = f_vert.reshape((768, 1024))

    # plot compressed picture
    plt.imshow(f_quant,cmap=plt.cm.Greys_r)
    plt.axis('off')
    plt.show()
```

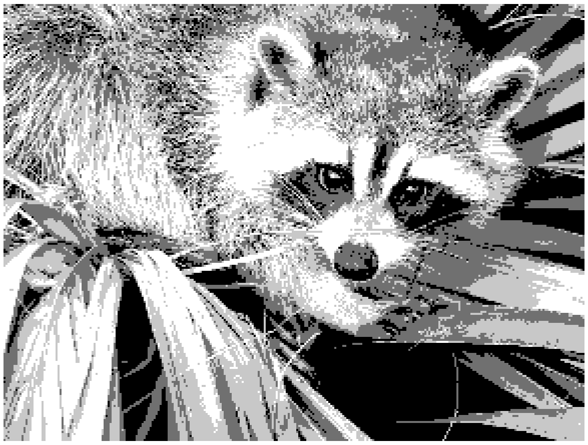
```
Original
```



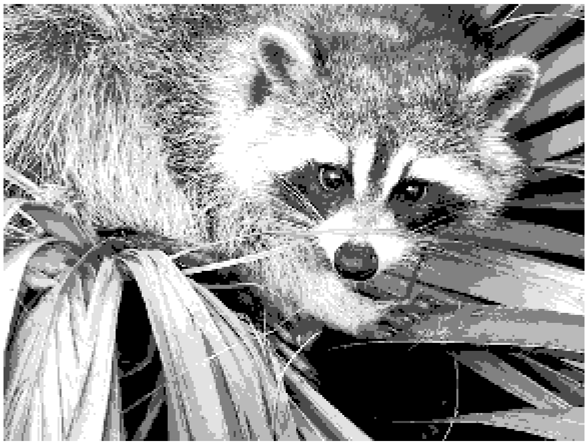
```
Quantized k = 2
Converged after 8 iterations!
```



```
Quantized k = 4
Converged after 12 iterations!
```



```
Quantized k = 6
Converged after 7 iterations!
```



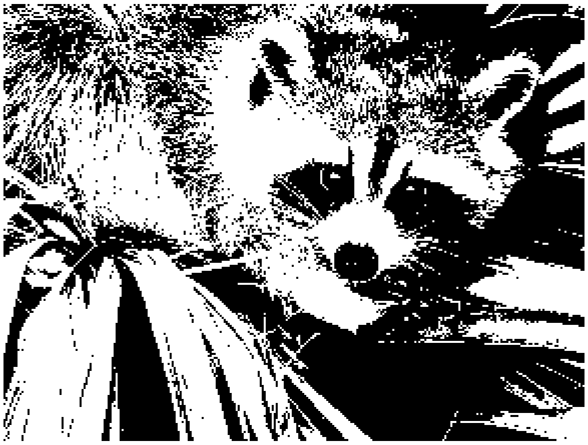
We can still see the difference between compressed photos and the original one. As the K increased, the compressed photo gets more and more clear.

In [67]:

```
def kmedians(X,K,max_iter=1000):


    random.seed(671)
    # initialize the means as randomly selected points in the data
    ### your code here
    n = X.shape[0]
    d = X.shape[1]
    initial_index = random.sample(range(n), K)
    medians = np.zeros((K,d))
    for i in range(K):
        medians[i][:] = X[initial_index[i]][:]

    # initialize the groups. a disctionary is recommended.
    ### your code here
    groups = {new_list: [] for new_list in range(K)}

    # iterate
    for i in range(max_iter):
        # assign each point to a group
        for j in range(X.shape[0]):
            groups[np.argmin((np.absolute(medians-X[j,:])).sum(axis=1))].appen
d(j)

        # calculate new means
        ### your code here
        new_medians = np.zeros((K,d))
        for k in range(K):
            index = groups[k]
            new_medians[k][:] = np.median(X[index][:])

        # see if we have converged
        if np.allclose(medians,new_means):
            print("Converged after {} iterations!".format(i))
            return medians, groups
        else:
            medians = new_medians
            groups = {i:[] for i in range(K)}
    print("Failed to converge after {} iterations...".format(max_iter))
```

In [62]:
```
K_range = [2]
for K in K_range:

    print("Quantized k = {} K_medians".format(K))

    # turn the pixels to (1, 768*1024) vector
    f_flat = f.flatten()
    f_vert = f_flat.reshape(-1, 1)

    # run kmeans
    means, groups = kmedians(f_vert,K)
    for k in range(means.shape[0]):
        list_temp = groups[k]
        for i in range(len(list_temp)):
            f_vert[list_temp[i]] = means[k][0]

    # convert back to (768,1024)
    f_quant = f_vert.reshape((768, 1024))

    # plot compressed picture
    plt.imshow(f_quant,cmap=plt.cm.Greys_r)
    plt.axis('off')
    plt.show()
```

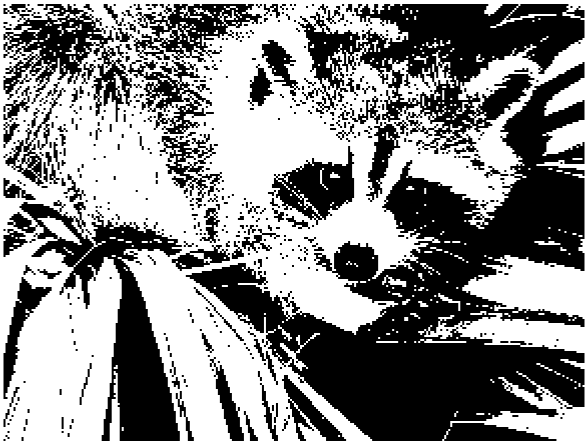
```
Quantized k = 2
Converged after 7 iterations!
```



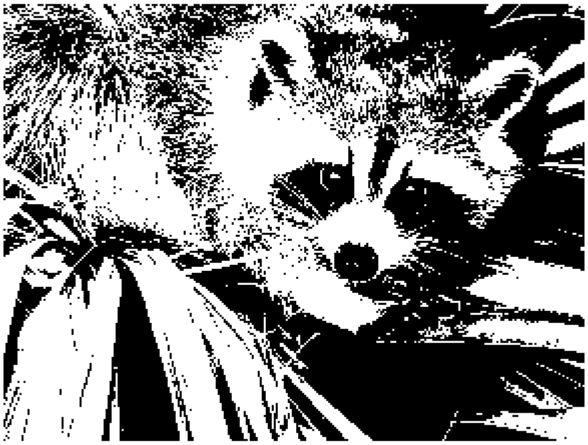
In [68]:

```
f_flat = f.flatten()
f_vert = f_flat.reshape(-1, 1)
means, groups_means = kmeans(f_vert,2)
list_temp = []
for k in range(means.shape[0]):
    list_temp = groups_means[k]
    for i in range(len(list_temp)):
        f_vert[list_temp[i]] = means[k][0]

# convert back to (768,1024)
f_quant_means = f_vert.reshape((768, 1024))

# plot compressed picture
plt.imshow(f_quant_means,cmap=plt.cm.Greys_r)
plt.axis('off')
plt.show()


f_flat = f.flatten()
f_vert = f_flat.reshape(-1, 1)
medians, groups_medians = kmedians(f_vert,2)

list_temp = []
for k in range(medians.shape[0]):
    list_temp = groups_medians[k]
    for i in range(len(list_temp)):
        f_vert[list_temp[i]] = medians[k][0]

# convert back to (768,1024)
f_quant_medians = f_vert.reshape((768, 1024))

# plot compressed picture
plt.imshow(f_quant_medians,cmap=plt.cm.Greys_r)
plt.axis('off')
plt.show()

np.array_equal(f_quant_means, f_quant_medians)
```

```
Converged after 8 iterations!
```



```
Converged after 10 iterations!
```



```
Out[68]: False
```

```
In [64]: f_quant_means
```

```
Out[64]: array([[156, 156, 156, ..., 156, 156, 156],
                [ 65,  65, 156, ..., 156, 156, 156],
                [ 65,  65,  65, ..., 156, 156, 156],
                ...,
                [ 65,  65, 156, ..., 156, 156, 156],
                [ 65,  65, 156, ..., 156, 156, 156],
                [ 65,  65, 156, ..., 156, 156, 156]], dtype=uint8)
```

```
In [65]: f_quant_medians
```

```
Out[65]: array([[155, 155, 155, ..., 155, 155, 155],
                [ 71,  71, 155, ..., 155, 155, 155],
                [ 71,  71,  71, ..., 155, 155, 155],
                ...,
                [ 71,  71, 155, ..., 155, 155, 155],
                [ 71,  71, 155, ..., 155, 155, 155],
                [ 71,  71, 155, ..., 155, 155, 155]], dtype=uint8)
```

The image is NOT exactly the same, but they do look very similar(qualitatively the same).