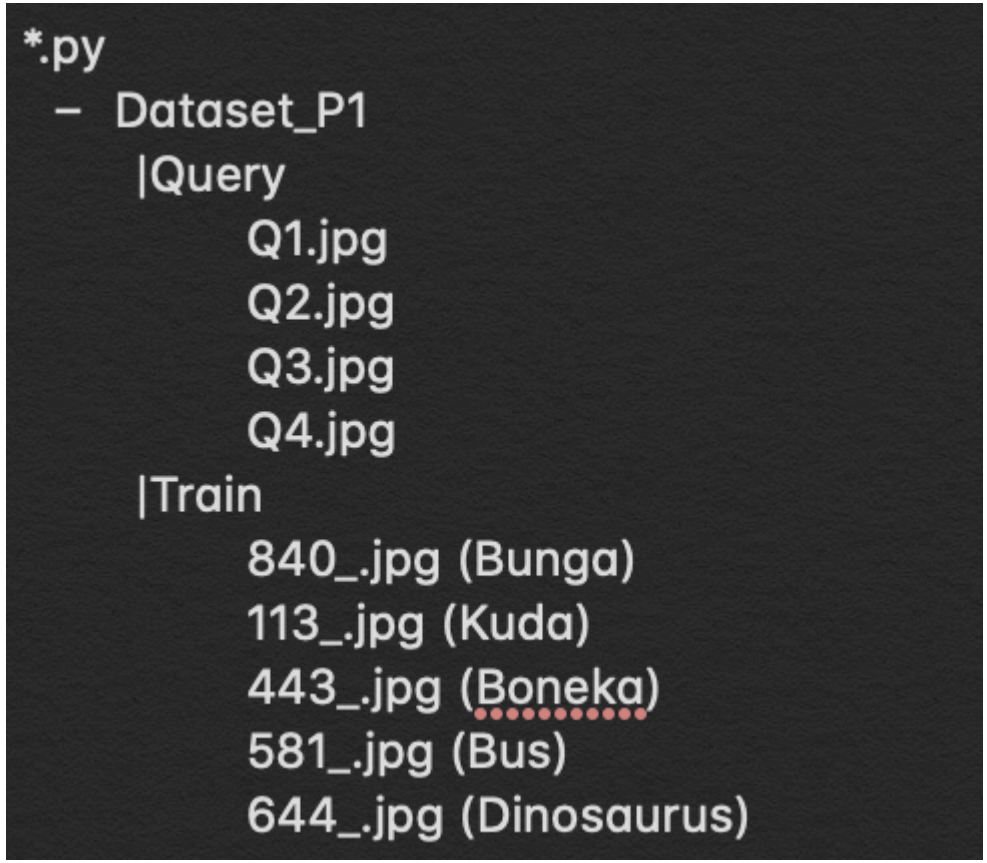
Nomor 1A

Preparing the Library

cv2 - library for image manipulation
os - library for listing directory and its content
numpy - library for modifying python array content
scipy - library used for clustering
sklearn - library used for scaling data and classify data
matplotlib - library used for plotting
joblib - to save to pickle format

In [1]:
```
import cv2
import os
import numpy as np
import joblib
from scipy.cluster.vq import *
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import cosine_similarity
from matplotlib import pyplot as plt
from PIL import Image
```

Image location map



Get Image Location

In [2]:
```
train_path = "COMP7116_Computer Vision_REGULER & GLOBAL_UAS - Dataset/Dataset_P1/train"
image_train_name = os.listdir(train_path)

train_image_path_list = []
for image_train in image_train_name:
    train_image_path_list.append(train_path + '/' + image_train)
```

Get Image Descriptor

We use SIFT to extract the Descriptor

In [3]:
```
sift = cv2.xfeatures2d.SIFT_create()
descriptor_list_train = []
for train_image_path in train_image_path_list:
    img_train = cv2.imread(train_image_path)
    img_train_gray = cv2.cvtColor(img_train, cv2.COLOR_BGR2GRAY)
    _, des_train = sift.detectAndCompute(img_train_gray, None)
    descriptor_list_train.append(des_train)
```

Stack the Descriptor

We should already get n-amount of ndarray. But, we need to group it by their similarity, and this only need 1 ndarray, so we could just stack our previously discovered descriptors of all training images into 1.

In [4]:
```
stacked_descriptor_train = descriptor_list_train[0]
for descriptor_train in descriptor_list_train[1:]:
    stacked_descriptor_train = np.vstack((stacked_descriptor_train, descriptor_train))
```

Cluster The Descriptor

Aftrer stacking the descriptor into 1 big ndarray, now it needs to be divided into some groups based on their similarities. This can be done using the help of clustering method using kmeans. After divided those arrays into some groups, now it's the time where we distributed which descriptors goes to which cluster group. This was done using vq (Vector Quantization). Simply, Vector Quantization here assigns a code book to each observation. Each observation is compared with the centroids in the code book and assigned the code of the closest centroid.

In [5]:
```
#stacked_descriptor_train = np.float32(stacked_descriptor_train)
centroids_train, _ = kmeans(stacked_descriptor_train, 5, 1)
im_features_train = np.zeros((len(train_image_path_list), len(centroids_train)), "float32")
for i in range(0, len(train_image_path_list)):
    words_train, _ = vq(descriptor_list_train[i], centroids_train)
    for w in words_train:
        im_features_train[i][w] += 1
```

Normalize the Histogram

It is best advised to normalized the data distribution for better result.

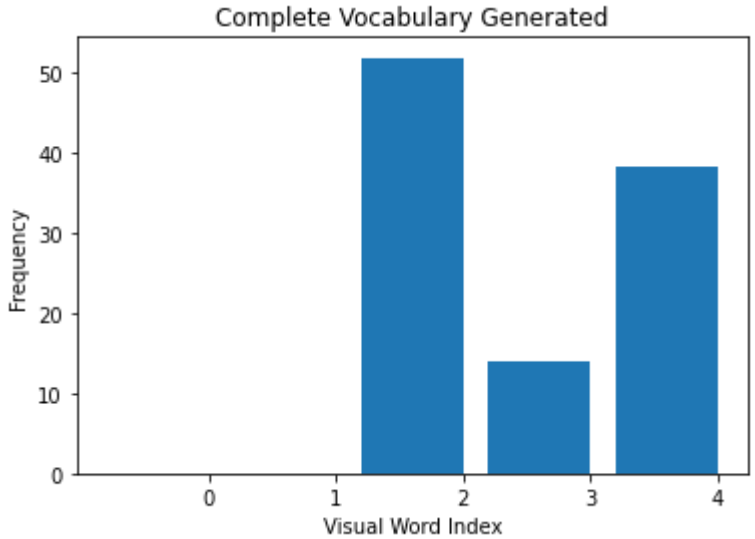
In [6]:
```
nbr_occurences = np.sum( (im_features_train > 0) * 1, axis = 0)
idf = np.array(np.log((1.0*len(train_image_path_list)+1) / (1.0*nbr_occurences + 1)), 'float32')

im_features_train = im_features_train*idf
im_features_train = preprocessing.normalize(im_features_train, norm='l2')
```

Histogram Plotting

In [8]:
```
histo = plt.figure()
joblib.dump(histo, open("histo_train.pkl", 'wb'))
x_scalar_train = np.arange(5)
y_scalar_train = np.array([abs(np.sum(im_features_train[:, h], dtype=np.float32)) for h in range(5)])

plt.bar(x_scalar_train, y_scalar_train)
plt.xlabel("Visual Word Index")
plt.ylabel("Frequency")
plt.title("Complete Vocabulary Generated")
plt.xticks(x_scalar_train + 0.4, x_scalar_train)
plt.show()
```



Nomor 1B

Just repeat the step above, but we don't need to cluster

Get Image Location

In [9]:
```
query_path = "COMP7116_Computer Vision_REGULER & GLOBAL_UAS - Dataset/Dataset_P1/Query"
image_query_name = os.listdir(query_path)

query_image_path_list = []
for image_query in image_query_name:
    query_image_path_list.append(query_path + '/' + image_query)
```

Get Image Descriptor

In [10]:
```
sift = cv2.xfeatures2d.SIFT_create()
descriptor_list_query = []
for query_image_path in query_image_path_list:
    img_query = cv2.imread(query_image_path)
    img_query_gray = cv2.cvtColor(img_query, cv2.COLOR_BGR2GRAY)
    _, des_query = sift.detectAndCompute(img_query_gray, None)
    descriptor_list_query.append(des_query)
```

Stack the Descriptor

In [11]:
```
stacked_descriptor_query = descriptor_list_query[0]
for descriptor_query in descriptor_list_query[1:]:
    stacked_descriptor_query = np.vstack((stacked_descriptor_query, descriptor_query))
```

Extract the Features

In [12]:
```
im_features_query = np.zeros((len(query_image_path_list), 5), "float32")
for i in range(0, len(query_image_path_list)):
    words_query, _ = vq(descriptor_list_query[i], centroids_train)
    for w in words_query:
        im_features_query[i][w] += 1
```

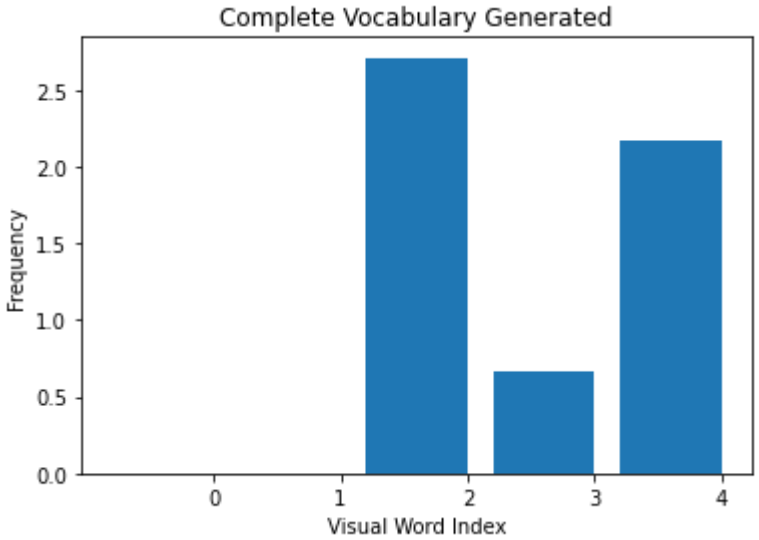
Normalize the Data

In [13]:
```
im_features_query = im_features_query*idf
im_features_query = preprocessing.normalize(im_features_query, norm='l2')
```

Plot the Histogram

In [14]:
```
x_scalar_query = np.arange(5)
y_scalar_query = np.array([abs(np.sum(im_features_query[:, h], dtype=np.float32)) for h in range(5)])

plt.bar(x_scalar_query, y_scalar_query)
plt.xlabel("Visual Word Index")
plt.ylabel("Frequency")
plt.title("Complete Vocabulary Generated")
plt.xticks(x_scalar_query + 0.4, x_scalar_query)
plt.show()
```



Nomor 1C

Count the Cosine Similarity and Sort it

In [17]:

```
import math
def cosine_similarity(v1, v2):
    "compute cosine similarity of v1 to v2: (v1 dot v2)/{||v1||*||v2||)"
    sumxx, sumxy, sumyy = 0, 0, 0
    for i in range(len(v1)):
        x = v1[i]; y = v2[i]
        sumxx += x*x
        sumyy += y*y
        sumxy += x*y
    return sumxy/math.sqrt(sumxx*sumyy)

def sort_result(score):
    n = len(score)
    for i in range(n):
        for j in range(0, n-i-1):
            if score[j][0] > score[j+1][0]:
                score[j], score[j+1] = score[j+1], score[j]
    return score

for i, query in enumerate(im_features_query):
    result_list = []
    for j in range(len(train_image_path_list)):
        result_list.append((cosine_similarity(im_features_train[j], query), j))

    result_list = sort_result(result_list)
    result_list.reverse()
    img_query = cv2.imread(query_image_path_list[i], cv2.COLOR_BGR2RGB)
    plt.figure(i+1)
    plt.subplot(4,5,1)
    plt.imshow(img_query)
    plt.xticks([])
    plt.yticks([])

    for i in range(15):
        similarity = round(result_list[i][0]*100, 2)
        index = result_list[i][1]
        img = cv2.imread(train_image_path_list[index], cv2.COLOR_BGR2RGB)
        plt.subplot(4,5, i+1+5)
        plt.imshow(img)
        plt.xticks([])
        plt.yticks([])

    print()
```

```
    plt.show()
```