```python
"""Do not modify the functions in this file except for where it says
YOUR CODE HERE."""

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from sklearn.preprocessing import PolynomialFeatures


def polynomial_augmentation(X, degree):
    """PART A

    This function takes in the data X and returns the polynomial features
    for X. You should implement constant/bias terms in your augmentations.

    The input shape of X is (100, 1), and the output shape should be
    (100, degree + 1). The first column of X should be the bias column (all
     1s),
    the second should be a copy of X, the third should be X with all its
     values
    squared, etc.
    """

    """YOUR CODE HERE"""
    poly = PolynomialFeatures(degree)
    augmented_matrix = poly.fit_transform(X)
    return augmented_matrix
    """YOUR CODE ENDS"""


def rbf_augmentation(X, means, variances):
    """PART B

    This function takes in the data X and returns the rbf features with
     respect to
    any centers you provide (write this in the function). We recommend
     looking at the
    data and estimating three centers and three corresponding variances.
     Once you have
    those, evaluate X's data with those three rbf functions, and augment
     the matrix with
    those outputs. You should thus have one column for bias, one for the
     original data X,
    and three for the rbf (mean, variance) pairs you come up with (X is 100
     by 5).

    The input shape of X is (100, 1), and the output shape should be (100,
     #(centers) + 2),
    since you want a column of all ones for the bias and one with the
     original data.

    means is a list of 3 means, variances is a list of 3 corresponding
     variances.
    """
    """YOUR CODE HERE"""
```

```python
    n = X.shape[0]
    rbf_matrix = np.copy(X)
    for i in range(len(means)):
        mean = means[i]
        var = variances[i]
        rbf = np.exp(-((X - mean)**2)/(2*var))
        rbf_matrix = np.append(rbf_matrix,rbf, axis=1)
    return rbf_matrix
    """YOUR CODE ENDS"""

def linear_regression(X, y):
    """Return the weights from linear regression.

    X: nxd (d = 1) matrix of data organized in rows
    y: length n vector of labels
    """
    return np.linalg.inv(X.T@X)@X.T@y


def mean_squared_error(y, y_hat):
    """Calculate the mean squared error given truth and predicted labels.

    y: length n vector of true labels
    y_hat: length n vector of predicted labels
    """
    return np.linalg.norm(y - y_hat) ** 2


def plot_compare(X, y, y_hat, figname):
    """Plot both true and predicted values for data.

    X: nxd (d = 1) matrix of data organized in rows
    y: length n vector of labels
    y_hat: same, but the predicted labels
    figname: name of the figure to save as (string))
    """
    # use blue for predicted, red for ground truth
    fig = plt.figure()
    ax1 = fig.add_subplot(111)

    ax1.scatter(X, y, c='r')
    ax1.scatter(X, y_hat, c='b')
    plt.legend(loc='upper left', labels=['true', 'predicted'])
    plt.savefig(figname)


def generate_data():
    """Generate data with noise."""
    X = np.random.rand(100) * 10 - 5
    y = 2.5 * norm.pdf(X, -2, 1.2) - 1.5 * norm.pdf(X, 0, 2) + 3.7 *
     norm.pdf(X, 3, 0.6)
    # inject noise
    y += np.random.rand(100)
    X = np.expand_dims(X, axis=0)
    return X.T, y
```