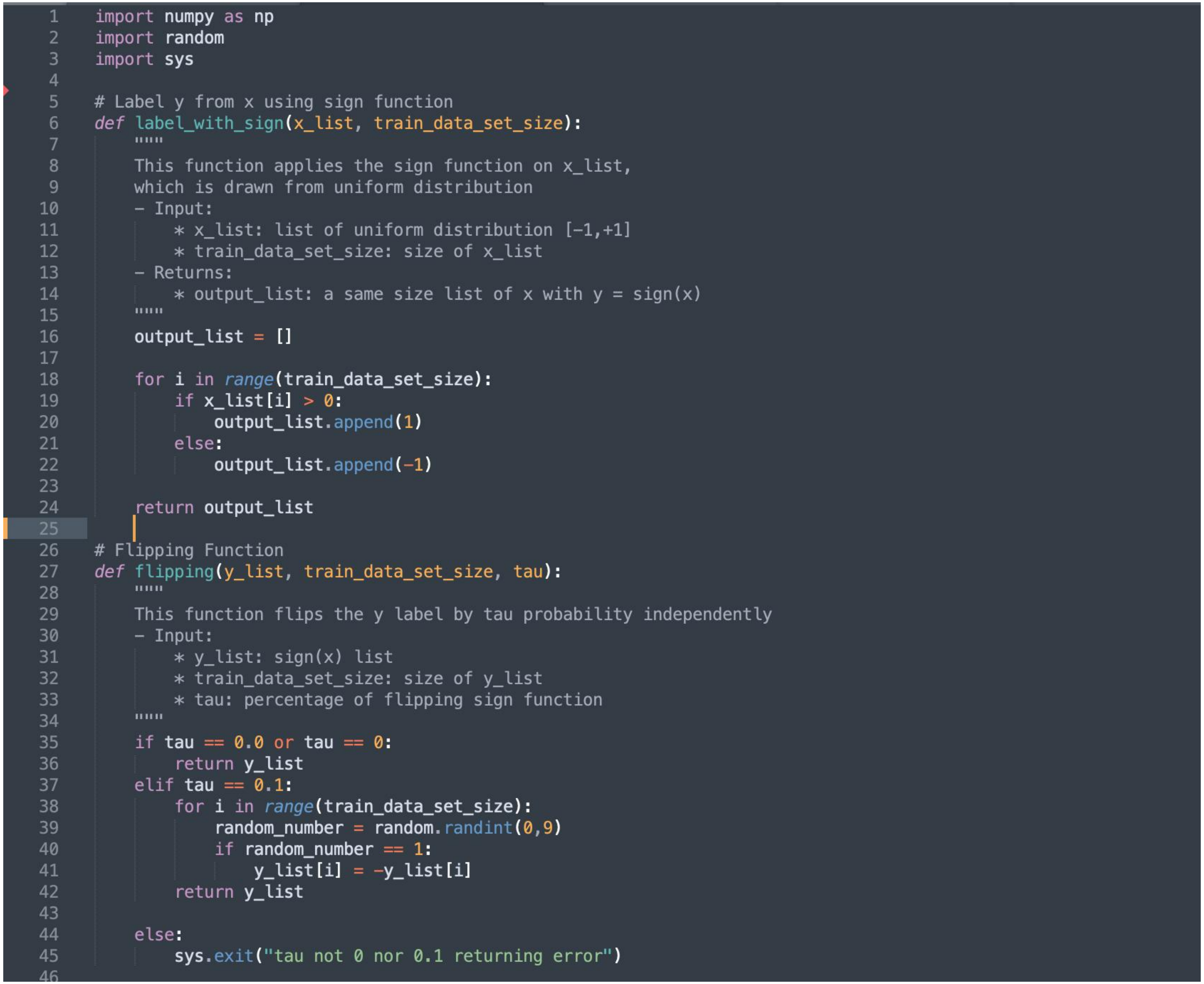
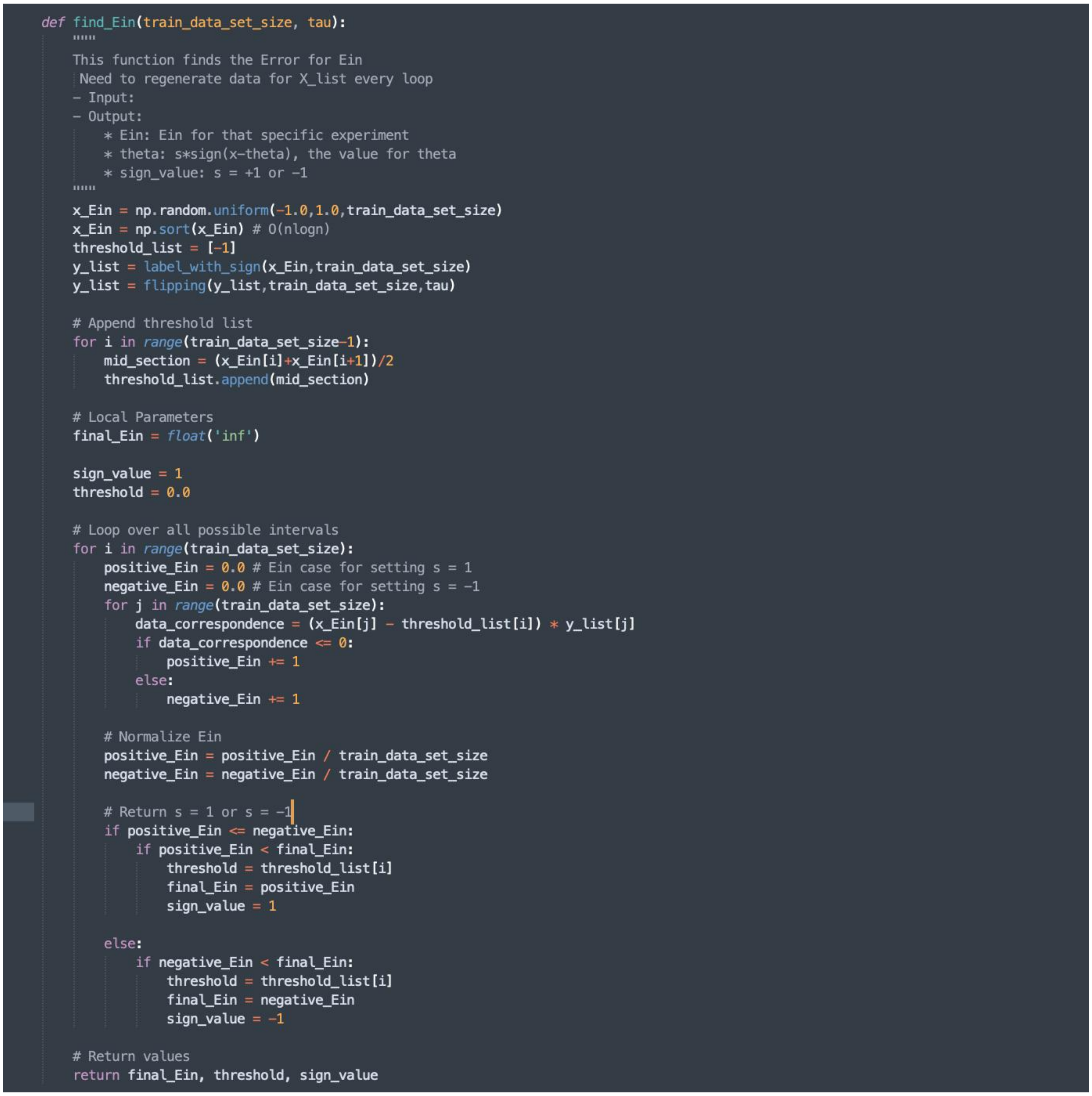
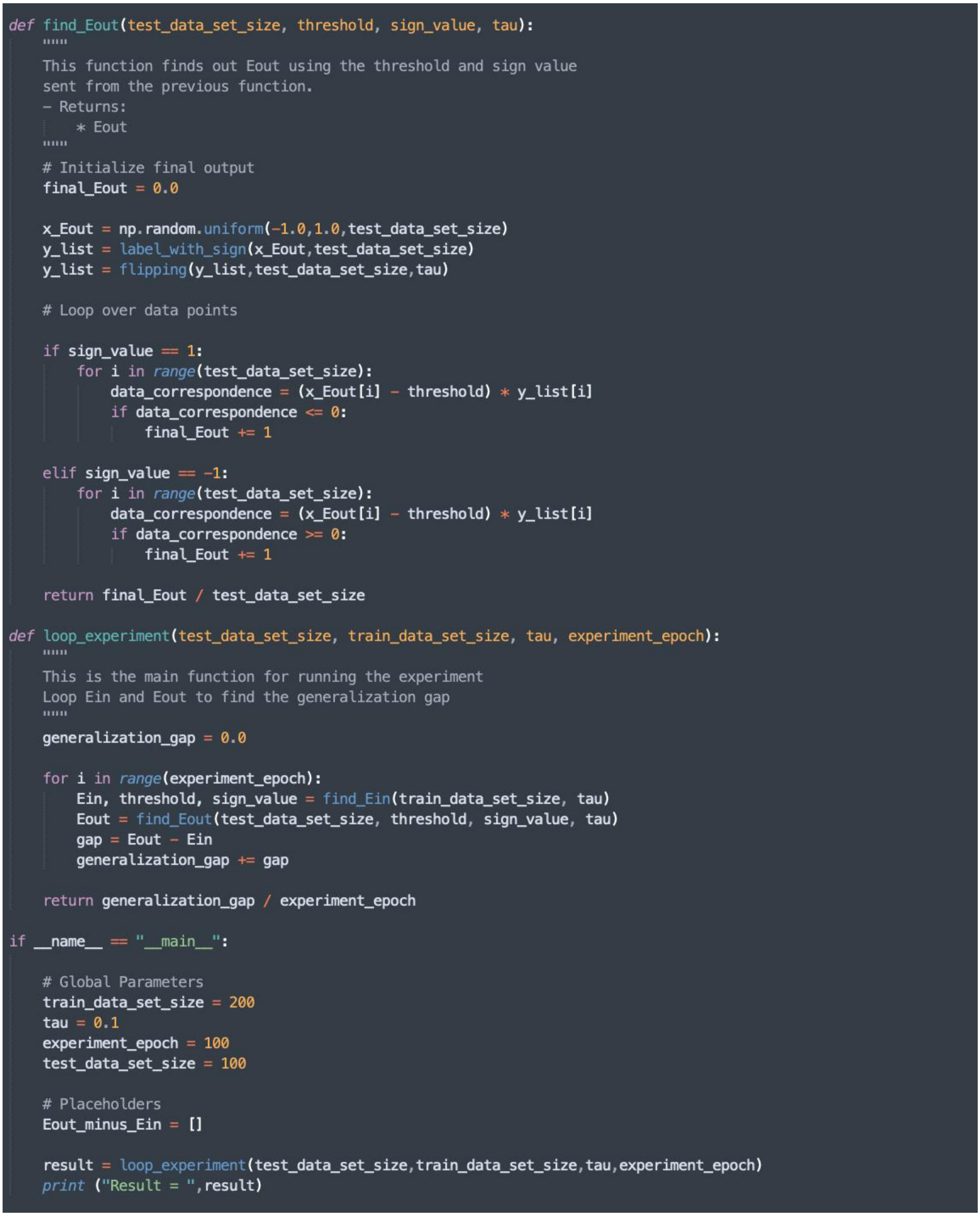


```python
import numpy as np
import random
import sys

# Label y from x using sign function
def label_with_sign(x_list, train_data_set_size):
    """
    This function applies the sign function on x_list,
    which is drawn from uniform distribution
    - Input:
        * x_list: list of uniform distribution [-1,+1]
        * train_data_set_size: size of x_list
    - Returns:
        * output_list: a same size list of x with y = sign(x)
    """
    output_list = []

    for i in range(train_data_set_size):
        if x_list[i] > 0:
            output_list.append(1)
        else:
            output_list.append(-1)

    return output_list

# Flipping Function
def flipping(y_list, train_data_set_size, tau):
    """
    This function flips the y label by tau probability independently
    - Input:
        * y_list: sign(x) list
        * train_data_set_size: size of y_list
        * tau: percentage of flipping sign function
    """
    if tau == 0.0 or tau == 0:
        return y_list
    elif tau == 0.1:
        for i in range(train_data_set_size):
            random_number = random.randint(0,9)
            if random_number == 1:
                y_list[i] = -y_list[i]
        return y_list

    else:
        sys.exit("tau not 0 nor 0.1 returning error")
```



```python
def find_Ein(train_data_set_size, tau):
    """
    This function finds the Error for Ein
     Need to regenerate data for X_list every loop
    - Input:
    - Output:
        * Ein: Ein for that specific experiment
        * theta: s*sign(x-theta), the value for theta
        * sign_value: s = +1 or -1
    """
    x_Ein = np.random.uniform(-1.0,1.0,train_data_set_size)
    x_Ein = np.sort(x_Ein) # O(nlogn)
    threshold_list = [-1]
    y_list = label_with_sign(x_Ein,train_data_set_size)
    y_list = flipping(y_list,train_data_set_size,tau)

    # Append threshold list
    for i in range(train_data_set_size-1):
        mid_section = (x_Ein[i]+x_Ein[i+1])/2
        threshold_list.append(mid_section)

    # Local Parameters
    final_Ein = float('inf')

    sign_value = 1
    threshold = 0.0

    # Loop over all possible intervals
    for i in range(train_data_set_size):
        positive_Ein = 0.0 # Ein case for setting s = 1
        negative_Ein = 0.0 # Ein case for setting s = -1
        for j in range(train_data_set_size):
            data_correspondence = (x_Ein[j] - threshold_list[i]) * y_list[j]
            if data_correspondence <= 0:
                positive_Ein += 1
            else:
                negative_Ein += 1

        # Normalize Ein
        positive_Ein = positive_Ein / train_data_set_size
        negative_Ein = negative_Ein / train_data_set_size

        # Return s = 1 or s = -1
        if positive_Ein <= negative_Ein:
            if positive_Ein < final_Ein:
                threshold = threshold_list[i]
                final_Ein = positive_Ein
                sign_value = 1

        else:
            if negative_Ein < final_Ein:
                threshold = threshold_list[i]
                final_Ein = negative_Ein
                sign_value = -1

    # Return values
    return final_Ein, threshold, sign_value
```



```python
def find_Eout(test_data_set_size, threshold, sign_value, tau):
    """
    This function finds out Eout using the threshold and sign value
    sent from the previous function.
    - Returns:
        * Eout
    """
    # Initialize final output
    final_Eout = 0.0

    x_Eout = np.random.uniform(-1.0,1.0,test_data_set_size)
    y_list = label_with_sign(x_Eout,test_data_set_size)
    y_list = flipping(y_list,test_data_set_size,tau)

    # Loop over data points

    if sign_value == 1:
        for i in range(test_data_set_size):
            data_correspondence = (x_Eout[i] - threshold) * y_list[i]
            if data_correspondence <= 0:
                final_Eout += 1

    elif sign_value == -1:
        for i in range(test_data_set_size):
            data_correspondence = (x_Eout[i] - threshold) * y_list[i]
            if data_correspondence >= 0:
                final_Eout += 1

    return final_Eout / test_data_set_size

def loop_experiment(test_data_set_size, train_data_set_size, tau, experiment_epoch):
    """
    This is the main function for running the experiment
    Loop Ein and Eout to find the generalization gap
    """
    generalization_gap = 0.0

    for i in range(experiment_epoch):
        Ein, threshold, sign_value = find_Ein(train_data_set_size, tau)
        Eout = find_Eout(test_data_set_size, threshold, sign_value, tau)
        gap = Eout - Ein
        generalization_gap += gap

    return generalization_gap / experiment_epoch

if __name__ == "__main__":

    # Global Parameters
    train_data_set_size = 200
    tau = 0.1
    experiment_epoch = 100
    test_data_set_size = 100

    # Placeholders
    Eout_minus_Ein = []

    result = loop_experiment(test_data_set_size,train_data_set_size,tau,experiment_epoch)
    print ("Result = ",result)
```