```
import random
import string

# Note: If you want to run this code to test your solution, you will need to install SciKit Learn and Numpy
# This can be done with: pip install scikit-learn numpy
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss

########################
##  Helper Functions ##
########################


def fit(X, y, phi, lmbd):
    '''
    Fits a logistic regression model on the data (X, y), using the parameters
    phi and lmbd. Returns the learned regression coefficients theta

    Parameters
    ----------
    X: np.array
        2-D feature matrix
    y: np.array
        1-D array of targets
    phi: np.array
        Feature mask used to select subset of features
        e.g., if d=3, phi= [True, False, True] would select the first and last features
    lmbd: float
        Regularization penalty weight

    Returns
    ----------
    theta: sklearn.LogisticRegression model
        Trained logistic regression model, with properties .coef_ and .intercept_
    '''
    if lmbd == 0: # no regularization
        theta = LogisticRegression(penalty='none')
    else:
        # C is inverse of lambda
        theta = LogisticRegression(penalty='l2', C=1/lmbd)

    # Assuming phi is feature mask
    theta.fit(X[:, phi], y)
    # cofficients of model can be access by theta.coef_ and theta.intercept_
    return theta


def predict(X, phi, theta):
    '''
    Returns predictions for given model theta on phi(X).

    Parameters
    ----------
    X: np.array
        2-D feature matrix
```

```
    phi: np.array
        Feature mask used to select subset of features
        e.g., if d=3, phi= [True, False, True] would select the first and last features
    theta: sklearn.LogisticRegression
        Trained logistic regression model, with properties .coef_ and .intercept_

    Returns
    ----------
    np.array
        1-D array of predictions of model theta on X for subset of features
        indicated by feature mask phi
    '''
    return theta.predict(X[:, phi])


def logloss(y, y_hat):
    '''
    Returns logistic loss between targets y and predictions y_hat.

    Parameters
    ----------
    y: np.array
        1-D array of ground truth targets
    y_hat: np.array
        1-D array of predictions

    Returns
    ----------
    float
        Logistic loss between targets y and predictions y_hat
    '''
    return log_loss(y, y_hat)


####################
##  Example Data  ##
####################


# if you would like to test your code for problem 1.3, you can use this data
# and set of possible lambda/phis
N = 100
d = 5
X = np.random.normal(size=(N, d))
y = np.random.choice([0, 1], size=N)
lmbds = [0, 0.1, 1, 10]
phis = [np.random.choice(a=[True, False], size=d) for i in range(3)]
# make sure at least one feature from each feature mask is selected
for phi in phis:
    phi[0] = True


####################
##  Problem 1.1  ##
####################
```

```
def sweep_hyperparameters(X_train, y_train, X_val, y_val, lmbds, phis):
    '''
    Finds the best settings of lambda and phi, and trains a logistic regression model with these parameters.

    You need to decide what other variables to pass as input arguments. (i.e., what split(s) of data to use)

    Parameters
    ----------
    X_train: np.array
        2-D matrix of training data features
    y_train: np.array
        1-D array of train targets
    X_val: np.array
        2-D matrix of validation data features
    y_val: np.array
        1-D array of validation targets
    lmbds: list
        List of possible settings of regularization term lambda to consider.
    phis: list
        List of feature masks to consider.
        Note: a feature mask is a boolean array (e.g., [True, False, True]) that
        specifies what features to use vs ignore.

    Returns
    ----------
    best_lmbd: float
        chosen regularization parameter lambda
    best_phi: np.array
        chosen feature mask (i.e., feature subset) phi
    best_theta: sklearn.LogisticRegression model
        Logistic regresion model trained using best_lmbd and best_phi
    '''

    loss_tracker = 999999999999
    best_phi = 0
    best_lmbd = 0
    for lamb in lmbds:
        for phi in phis:
            theta = fit(X_train, y_train, phi, lamb)
            pred = predict(X_val, phi, theta)
            loss = logloss(y_val,pred)
            if loss < loss_tracker:
                    loss_tracker = loss
                    best_phi = phi
                    best_lmbd = lamb
                    best_theta = theta

    return best_lmbd, best_phi, best_theta


####################
##  Problem 1.2  ##
####################


def evaluate_model(theta, phi, X_test, y_test):
    '''
```

```
    Evaluates a trained logistic regression model.

    You need to decide what other variables to pass as input arguments. (i.e., what split(s) of data to use)

    Parameters
    ----------
    theta: sklearn.LogisticRegression
        rained logistic regression model
    phi: np.array
        Feature mask used to select subset of features
    X_test: np.array
        2-D matrix of test data features
    y_test: np.array
        1-D array of test targets

    Returns
    ----------
    loss: float
        loss of trained model on some partition of the data (you need to decide what partition)
    '''
    # TODO: your code goes here
    pred = predict(X_test, phi, theta)
    loss = logloss(y_test,pred)
    return loss

####################
##  Problem 1.3  ##
####################


def train_and_eval_model(X, y, lmbds, phis):
    '''
    Uses a feature matrix X and labels y to train and evaluate
    a logistic regression model.

    Sweeps over all combinations of lambda and phi to choose the
    best setting.

    Parameters
    ----------
    X: np.array
        2-D feature matrix
    y: np.array
        1-D array of targets
    lmbds: list
        List of possible settings of regularization term lambda to consider.
    phis: list
        List of feature masks to consider.
        Note: a feature mask is a boolean array (e.g., [True, False, True]) that
        specifies what features to use vs ignore.
    '''
    # 1. Split data into groups
    n_samples = len(X)
    n_train = int(n_samples * 0.5)
    n_val = int(n_samples * 0.25)
    X_train, y_train = X[:n_train], y[:n_train]
    X_val, y_val = X[n_train:n_train+n_val], y[n_train:n_train+n_val]
```

```
X_test, y_test = X[n_train+n_val:], y[n_train+n_val:]

# 2. Choose best hyperparameters and train model w/ these parameters

loss_tracker = 999999999999
best_phi = 0
best_lmbd = 0
for lamb in lmbds:
    for phi in phis:
        theta = fit(X_train, y_train, phi, lamb)
        pred = predict(X_val, phi, theta)
        loss = logloss(y_val,pred)
        if loss < loss_tracker:
                loss_tracker = loss
                best_phi = phi
                best_lmbd = lamb
                best_theta = theta


# 3. Produce estimate of how model will perform on unseen data
pred = predict(X_test, phi, theta)
loss = logloss(y_test,pred)
return loss
```