```python
import numpy as np

class KMeans():
    # This function initializes the KMeans class
    def __init__(self, k = 3, num_iter = 1000, order = 2):
        # Set a seed for easy debugging and evaluation
        np.random.seed(42)

        # This variable defines how many clusters to create
        # default is 3
        self.k = k

        # This variable defines how many iterations to recompute centroids
        # default is 1000
        self.num_iter = num_iter

        # This variable stores the coordinates of centroids
        self.centers = None

        # This variable defines whether it's K-Means or K-Medians
        # an order of 2 uses Euclidean distance for means
        # an order of 1 uses Manhattan distance for medians
        # default is 2
        if order == 1 or order == 2:
            self.order = order
        else:
            raise Exception("Unknown Order")

    # This function fits the model with input data (training)
    def fit(self, X):
        # m, n represent the number of rows (observations)
        # and columns (positions in each coordinate)
        m, n = X.shape

        # self.centers are a 2d-array of
        # (number of clusters, number of dimensions of our input data)
        self.centers = np.zeros((self.k, n))

        # self.cluster_idx represents the cluster index for each observation
        # which is a 1d-array of (number of observations)
        self.cluster_idx = np.zeros(m)

        ##### TODO 1 ######
        #
        # Task: initialize self.centers
        #
        # Instruction:
        # For each dimension (feature) in X, use the 10th percentile and
        # the 90th percentile to form a uniform distribution. Then, we will initialize
        # the values of each center by randomly selecting values from the distributions.
        #
        # Note:
        # This method is by no means the best initialization method. However, we would
        # like you to follow our guidelines in this HW. We will ask you to discuss some better
        # initializaiton methods in the notebook.
        #
        # Hint:
        # 1. np.random.uniform(), np.percentile() might be useful
        # 2. make sure to look over its parameters if you're not sure
        ####################
        for i in range(n):
            lower, upper = np.percentile(X[:,i], [10, 90])
            self.centers[:,i] = np.random.uniform(lower, upper, self.k)
        ##### END TODO 1 #####

        for i in range(self.num_iter):
            # new_centers are a 2d-array of
            # (number of clusters, number of dimensions of our input data)
            new_centers = np.zeros((self.k, n))

            ##### TODO 2 ######
            #
            # Task: calculate the distance and create cluster index for each observation
            #
            # Instruction:
```

```
            # You should calculate the distance between each observation and each centroid
            # using specified self.order. Then, you should derive the cluster index for
            # each observation based on the minimum distance between an observation and
            # each of the centers.
            #
            # Hint:
            # 1. np.linalg.norm() might be useful, along with parameter axis, ord
            # for that function
            # 2. You can transpose an array using .T at the end
            # 3. np.argmin() might be useful along with parameter axis in finding
            # the desired cluster index of all observations
            #
            # IMPORTANT:
            # Copy-paste this part of your implemented code
            # to the predict function, and return cluster_idx in that function
            ####################
            distances = np.array([np.linalg.norm(X - center, ord=self.order, axis=1) for center in self.centers])
            cluster_idx = np.argmin(distances.T, axis=1)
            ##### END TODO 2 #####

            ##### TODO 3 ######
            #
            # Task: calculate the coordinates of new_centers based on cluster_idx
            #
            # Instruction:
            # You should assign the coordinates of the new_center by calculating
            # mean/median of the coordinates of observations belonging to the same
            # cluster.
            #
            # Hint:
            # 1. np.mean(), np.median() with axis might be helpful
            ####################
            for idx in range(self.k):
                cluster_coordinates = X[cluster_idx == idx]
                if self.order == 2:
                    cluster_center = np.mean(cluster_coordinates, axis=0)
                elif self.order == 1:
                    cluster_center = np.median(cluster_coordinates, axis=0)
                new_centers[idx, :] = cluster_center
            ##### END TODO 3 #####

            ##### TODO 4 ######
            #
            # Task: determine early stop and update centers and cluster_idx
            #
            # Instructions:
            # You should stop tranining as long as cluster index for all
            # observations is the same as the previous iteration
            # Hint:
            # 1. .all() might be helpful
            ####################
            if (cluster_idx == self.cluster_idx).all():
                print(f"Early Stopped at Iteration {i}")
                return self
            self.centers = new_centers
            self.cluster_idx = cluster_idx
            ##### END TODO 4 #####
        return self


    # This function makes predictions with input data
    # Copy-paste your code from TODO 2 and return cluster_idx
    def predict(self, X):
        distances = np.array([np.linalg.norm(X - center, ord=self.order, axis=1) for center in self.centers])
        return np.argmin(distances.T, axis=1)
```