Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```python
import numpy as np
import pandas as pd
from random import seed, random


class Perceptron:
    __weights = np.zeros((1, 3))
    __epoch = 0
    __learning_rate = 0.5

    def __init__(self, inputs: int):
        seed(1)
        self.__weights = np.array([round(random(), 2) for i in range(0, inputs + 1)])
        #self.__weights = np.array([0.01, 0.08, 0.08])
        print(f'Initial Weights: {self.__weights[0]} {self.__weights[1:]}')

    def __default_step_function(self, inputs):
        product = self.__weights * inputs
        return 1 if product.sum() > 0 else 0

    def __default_update_parameters(self, inputs, t, y):
        mult = self.__learning_rate * (t - y)
        changes = np.array(inputs) * mult
        self.__weights += changes

    def set_hyper_parameters(self, learning_rate=0.5):
        self.__learning_rate = learning_rate

    def train(self, df: pd.DataFrame, targets, epochs=None):
        if len(df) != len(targets):
            raise Exception("Data and Targets don't have same number of rows.")

        converged = False
        while (epochs is None and not converged) or (epochs is not None and epochs != self.__epoch):

            print('\n**************************************\n')
            self.__epoch += 1
            print(f'Epoch #{self.__epoch}')

            correct_predictions = 0
            for i in df.index:
                inputs = df.loc[i].to_list()
                print(f'\ninputs: {inputs}')
                print(f'weights: {self.__weights[0]} {self.__weights[1:]}')

                inputs.insert(0, 1)
                y = self.__default_step_function(inputs)
                print(f'y={y}\tt={targets[i]} ==> ', end='')

                if y == targets[i]:
                    print('correct')
                    correct_predictions += 1
                else:
                    print('incorrect')
                    # update weights
                    self.__default_update_parameters(inputs, targets[i], y)

            converged = correct_predictions == len(df)

        print(f'\n\nSolution took {self.__epoch} epochs.')
        print(f'Final weights: {self.__weights[0]} {self.__weights[1:]}')

    def classify(self, inputs):
        print(f'\ninputs: {inputs}')
        print(f'weights: {self.__weights[0]} {self.__weights[1:]}')
        ins = inputs.copy()
        ins.insert(0, 1)
        y = self.__default_step_function(ins)
        print(f'Prediction: {y}')
        return y

```