Perceptron(X, Y, rho, max_iter): given data from two linearly seperable classes ω_1 and ω_2, trains a linear classifier to distinguish the two classes.

This implementation uses the reward and punishment variant of the Perceptron Algorithm and assumes bias. That is, it will automatically augment the input dataset to accomodate a bias term.

Args:

- X: the entire training data from both classes (must be of shape $\#\text{samples} \times \#\text{features}$).
- Y: labels for each instance of training data (must be of shape $\#\text{samples} \times 1$ and labels must be 1s and 2s for the first and the second class respectively).
- ρ: learning rate of the algorithm. Defaults to 0.5.

Returns:

- w: a 3×1 vector containing the trained weights (w_1, w_2) and the bias (w_3) of the network.

```
function w = perceptron(X, Y, rho, max_iter)
    if nargin<4
        if nargin < 3
            % learning rate
            rho = 0.5;
        end
        % maximum number of allowed iterations, if it takes longer than
        % this, assume the algorithm cannot converge
        max_iter = 1000;
    end
    % augment the training data with a vector of "1"s
    X_aug = [X, ones([size(X,1), 1])];
    % set initial weights to zero (as per the instructions of the problem
    % statement)
    w = zeros([3, 1]);
    % convergence flag
    converged = false;
    % the perceptron algorithm main loop
    iteration = 1;
    while iteration < max_iter
        % index of the input of the current iteration
        input_idx = mod(iteration, size(X_aug,1)) + 1;
        % calculate output of perceptron
        output = w'*X_aug(input_idx,:)';
        if Y(input_idx) == 1 && output <= 0
            % misclassified sample from class 1
            w = w + rho*X_aug(input_idx,:)';
        elseif Y(input_idx) == 2 && output >= 0
            % misclassified sample from class 2
            w = w - rho*X_aug(input_idx,:)';
        end
        % check for convergence
        % because the perceptron algorithm guarantees perfect convergence after
        % finitely many steps given a linearly separable problem, we will check
```

```
        % for convergence every time after seeing the entire dataset
        if mod(iteration, size(X_aug,1)) == 0
            iteration
            w
            converged = true;
            for i=1:size(X_aug,1)
                output = w'*X_aug(i,:)';
                if (Y(i) == 1 && output <= 0) || (Y(i) == 2 && output >= 0)
                    converged = false;
                    break;
                end
            end
        end
        if converged
            break;
        end
        iteration = iteration + 1;
    end
    fprintf('Converged in %d steps.\n', iteration);
    disp('Final weights:');
    w
end
```