```
function J = computeCost(X, y, theta)
%COMPUTECOST Compute cost for linear regression
%   J = COMPUTECOST(X, y, theta) computes the cost of using theta as the
%   parameter for linear regression to fit the data points in X and y

% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = 0;

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the cost of a particular choice of theta
%               You should set J to the cost.

h=X*theta; % funcion de hipotesis
e=h.-y; % error de cada prediccion
e_cuadrado=e.^2; % cuadrado de los errores
suma_e_cuadrado=sum(e_cuadrado); % suma de los cuadrados

J=1/(2*m)*suma_e_cuadrado % Cost Function


% =========================================================================

end

```

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha,
num_iters)
%GRADIENTDESCENT Performs gradient descent to learn theta
%   theta = GRADIENTDESCENT(X, y, theta, alpha, num_iters) updates theta by
%   taking num_iters gradient steps with learning rate alpha

% Initialize some useful values
m = length(y); % number of training examples
J_history = zeros(num_iters, 1);

for iter = 1:num_iters

    % ====================== YOUR CODE HERE ======================
    % Instructions: Perform a single gradient step on the parameter vector
    %               theta.
    %
    % Hint: While debugging, it can be useful to print out the values
    %       of the cost function (computeCost) and gradient here.
    %

    h=X*theta; % funcion de hipotesis
    e=h.-y; % error de cada prediccion

    X_1 = X(:, 1); % permite la actualizacion simultánea
    X_2 = X(:, 2);

    theta(1, 1) = theta(1, 1) - (alpha * (1/m) * e' * X_1);
    theta(2, 1) = theta(2, 1) - (alpha * (1/m) * e' * X_2);

    % ============================================================

    % Save the cost J in every iteration
    J_history(iter) = computeCost(X, y, theta);

end

end
```