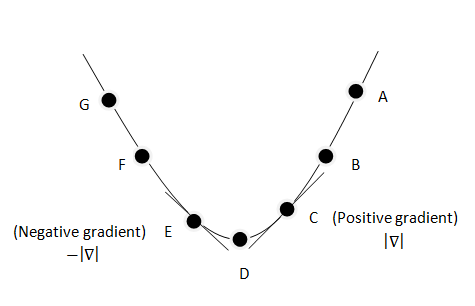
The gradient descent is an iterative approach for minimizing the given function. We start with an initial guess of the solution and we take the gradient of the function at that point. We step the solution in the negative direction of the gradient and we repeat the process. The algorithm will eventually converge where the gradient is zero (which correspond to a local minimum). So your job is to find out the value of theta0 and theta1 that minimization the loss function [for example least squared error]. The term "converges" means you reached in the local minimum and further iteration does not affect the value of parameters i.e. value of theta0 and theta1 remains constant. Lets see an example Note: Assume it is in first quadrant for this explanation.



Lets say you have to minimize a function f(x) [cost function in your case]. For this you need to find out the value of x that minimizes the functional value of f(x). Here is the step by step procedure to find out the value of x using gradient descent method

1. You choose the initial value of x. Lets say it is in point A in the figure.
2. You calculate the gradient of f(x) with respect to x at A.
3. This gives the slope of the function at point A. Since it the function is increasing at A, it will yield a positive value.
4. You subtract this positive value from initial guess of x and update the value of x. i.e. x = x - [Some positive value]. This brings the x more closer to the D [i.e. the minimum] and reduces the functional value of f(x) [from figure]. Lets say after iteration 1, you reach to the point B.
5. At point B, you repeat the same process as mention in step 4 and reach the point C, and finally point D.
6. At point D, since it is local minimum, when you calculate gradient, you get 0 [or very close to 0]. Now you try to update value of x i.e. x = x - [0]. You will get same x [or very closer value to the previous x]. This condition is known as "Convergence". The above steps are for increasing slope but are equally valid for decreasing slope. For example, the gradient at point G results into some negative value. When you update x i.e x = x - [ negative value] = x - [ - some positive value] = x + some positive value. This increases the value of x and it brings x close to the point F [ or close to the minimum].

There are various approaches to solve this gradient descent. As @mattnedrich said, the two basic approaches are

1. Use fixed number of iteration N, for this pseudo code will be
2. iter = 0
3. while (iter < N) {
4. theta0 = theta0 - gradient with respect to theta0
5. theta1 = theta1 - gradient with respect to theta1
6. iter++

}

Gradient descent is one of the approach to minimize the function in Linear regression. There exists direct solution too. Batch processing (also called normal equation) can be used to find out the values of theta0 and theta1 in a single step. If X is the input matrix, y is the output vector and theta be the parameters you want to calculate, then for squared error approach, you can find the value of theta in a single step using this matrix equation

theta = inverse(transpose (X)\*X)\*transpose(X)\*y

But as this contains matrix computation, obviously it is more computationally expensive operation then gradient descent when size of the matrix X is large. I hope this may answer your query. If not then let me know.

For choosing learning rate, the best thing you can do is also plot the cost function and see how it is performing, and always remember these two things:

* if the learning rate is too small you will get slow convergence
* if the learning rate is too large your cost function may not decrease in every iteration and therefore it will not converge

The **stochasticity** comes from updates based on gradient computed for **one** samples (or small batch, which is called mini-bach SGD). It is obviously not a correct gradient of the error function (which is the sum of errors for all the training samples) but one can prove, that under reasonable conditions such process converges to the local optima. The stochastic updates are preferable in many applications due to their simplicity and (in many cases) cheapier computation .

A cost function is something you want to minimize. For example, your cost function might be the sum of squared errors over your training set. Gradient descent is a method for finding the minimum of a function of multiple variables. So you can use gradient descent to minimize your cost function. If your cost is a function of K variables, then the gradient is the length-K vector that defines the direction in which the cost is increasing most rapidly. So in gradient descent, you follow the negative of the gradient to the point where the cost is a minimum. If someone is talking about gradient descent in a machine learning context, the cost function is probably implied (it is the function to which you are applying the gradient descent algorithm).

Gradient descent is a method for finding the minimum/maximum value of some multidimensional function. To keep things simple, imagine looking for the peak of a hill. In this scenario, we're looking for a maximum (altitude) in 3 dimensions (longitude, latitude, altitude). The function is the surface of the hill, with two inputs (longitude, latitude) and one output (altitude).

If you had to use gradient descent, you'd do this:

1. Work out the slope in each direction at your current location
2. Move a small distance in the direction of the steepest positive incline
3. If reached convergence, then stop. Else, go back to step 1.

Convergence means that the result is not going to change significantly if you continue. The above instructions generalize to an arbitrary number of dimensions.

To implement gradient descent in any language, you set up a loop, and implement the step above. It's really the same no matter language you use. Here's a good video about gradient descent with some pseudocode (not that different to Python): <http://youtu.be/5u4G23_OohI?t=26m34s>.

The gradient is nothing but the vector made up of all the derivatives of the loss function with respect to each single parameter.