Let's assume you want to fill up a library of books and not just stuff them in there, but you want to be able to easily find them again when you need them.

So, you decide that if the person that wants to read a book knows the title of the book and the exact title to boot, then that's all it should take. With the title, the person, with the aid of the librarian, should be able to find the book easily and quickly.

So, how can you do that? Well, obviously you can keep some kind of list of where you put each book, but then you have the same problem as searching the library, you need to search the list. Granted, the list would be smaller and easier to search, but still you don't want to search sequentially from one end of the library (or list) to the other.

You want something that, with the title of the book, can give you the right spot at once, so all you have to do is just stroll over to the right shelf, and pick up the book.

But how can that be done? Well, with a bit of forethought when you fill up the library and a lot of work when you fill up the library.

Instead of just starting to fill up the library from one end to the other, you devise a clever little method. You take the title of the book, run it through a small computer program, which spits out a shelf number and a slot number on that shelf. This is where you place the book.

The beauty of this program is that later on, when a person comes back in to read the book, you feed the title through the program once more, and get back the same shelf number and slot number that you were originally given, and this is where the book is located.

The program, as others have already mentioned, is called a hash algorithm or hash computation and usually works by taking the data fed into it (the title of the book in this case) and calculates a number from it.

For simplicity, let's say that it just converts each letter and symbol into a number and sums them all up. In reality, it's a lot more complicated than that, but let's leave it at that for now.

The beauty of such an algorithm is that if you feed the same input into it again and again, it will keep spitting out the same number each time.

Ok, so that's basically how a hash table works.

Technical stuff follows.

First, there's the size of the number. Usually, the output of such a hash algorithm is inside a range of some large number, typically much larger than the space you have in your table. For instance, let's say that we have room for exactly one million books in the library. The output of the hash calculation could be in the range of 0 to one billion which is a lot higher.

So, what do we do? We use something called modulus calculation, which basically says that if you counted to the number you wanted (i.e. the one billion number) but wanted to stay inside a much smaller range, each time you hit the limit of that smaller range you started back at 0, but you have to keep track of how far in the big sequence you've come.

Say that the output of the hash algorithm is in the range of 0 to 20 and you get the value 17 from a particular title. If the size of the library is only 7 books, you count 1, 2, 3, 4, 5, 6, and when you get to 7, you start back at 0. Since we need to count 17 times, we have 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, and the final number is 3.

Of course modulus calculation isn't done like that, it's done with division and a remainder. The remainder of dividing 17 by 7 is 3 (7 goes 2 times into 17 at 14 and the difference between 17 and 14 is 3).

Thus, you put the book in slot number 3.

This leads to the next problem. Collisions. Since the algorithm has no way to space out the books so that they fill the library exactly (or the hash table if you will), it will invariably end up calculating a number that has been used before. In the library sense, when you get to the shelf and the slot number you wish to put a book in, there's already a book there.

Various collision handling methods exist, including running the data into yet another calculation to get another spot in the table, or simply to find a space close to the one you were given (i.e. right next to the previous book). This would mean that you have some digging to do when you try to find the book later, but it's still better than simply starting at one end of the library.

Finally, at some point, you might want to put more books into the library than the library allows. In other words, you need to build a bigger library. Since the exact spot in the library was calculated using the exact and current size of the library, it goes to follow that if you resize the library you might end up having to find new spots for all the books since the calculation done to find their spots has changed.

I hope this explanation was a bit more down to earth than buckets and functions :)