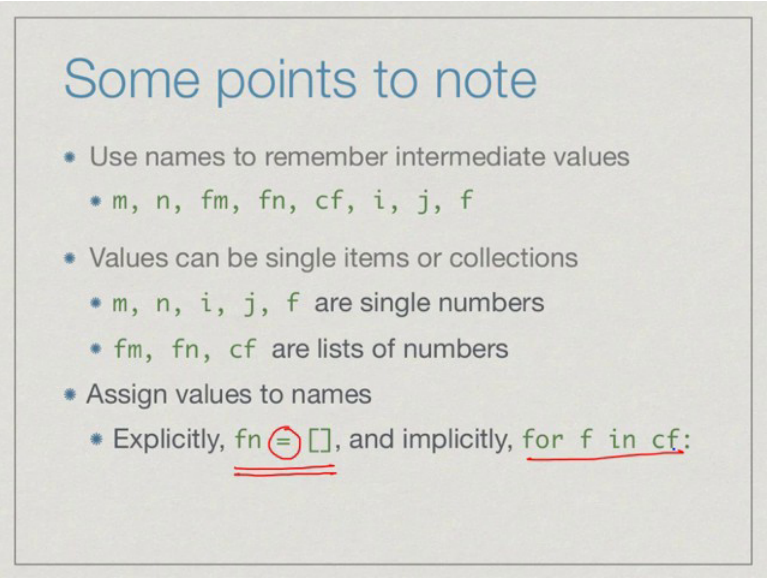
the factors in cf. So, these are all ways of keeping track of intermediate values. The second point to note is that a value can be a single item.

For example, m n are numbers, similarly i, j and f at each step are numbers. So, these will be single values or they could be collections. So, there are lists. So fm is a list, fn is a list. So, it is a single name denoting a collection of values in this case a list a sequence has a first position and next position and a last position. These are list of numbers.

One can imagine the other collections and we will see them as we go along. So, collections are important, because it would be very difficult to write a program if we had to keep producing a name for every factor of m separately. We need a name collectively for all the factors of m regardless of how big m is. These names can denote can be denote single values or collections of values. And a collection of values with the particular structure is precisely what we call data structure. So, these are more generally called data structures. So, in this case the data structure we have is a list.

(Refer slide Time: 23:56)

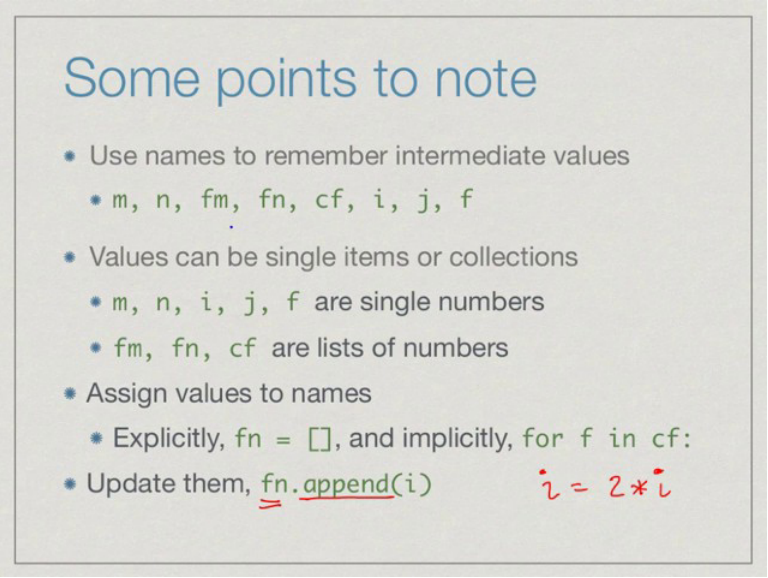


What can we do with these names and values well one thing is we can assigned a value to a name. So, for instance when we write fn is equal to the empty list we are explicitly setting the value of fn to be the empty list. This tells two things this says the value is

empyt list, so it is also tells python the fn denotes the lists these are the two steps going on here as we see.

And the other part is that when we write something like for each f in the list cf, which is implicitly saying that take every value in cf and assign it one by one to the values f to the name f. Right though they do not have this equality sign explicitly implicitly this is assigning the new values for f as we step the list cf right. So, the main thing that we do in a python program is to assign values to names.

(Refer slide Time: 24:37)

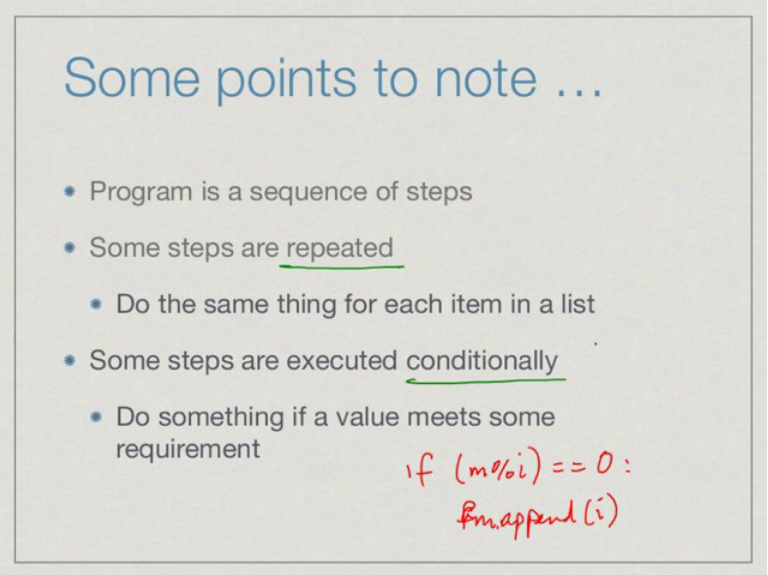


And having assigned a value we can then modify the value. For instance every time we find a new factor of n we do not want to through any old factor we want to take the existing list fm and we want to add it. So, this function append for instance modifies the value of the name fn to a new name which takes the old name and sticks an i at the end of it.

More generally you could have a number and we could want a replaces by two times a number. So, we might have something like i is equal to two times i. So, star stands for multiplication this does not mean that i is equals to two times i arithmetically because; obviously, unless i is 0 i cannot be equal to two times itself. What is means is that take

the current value of i, multiply it by two and assign it to i. So, we will see this as we go along, but assignment can either assign a completely new value or you could update the value using the old value. So, here we taking the old value of the function of the list fn and we are appending a value it would getting a new value of fn.

(Refer slide Time: 25:49)



The other part that we are need to note is how we execute steps. So, we said at the beginning of today's lecture a program is a sequence of steps. But we do not just execute the sequence of steps from beginning to end blindly. Sometimes we have to do the same thing again and again. For instance we have to check for every possible factor from 1 to m if it divides m and then put it in the list. So, some steps are repeated we do something, for examples here for each item in a list.

And some steps are executed only if the value that we are looking at meets particular conditions. When we say something like if m percent i is 0, if the remainder of m divided by a is 0 then append. So, the step append i to fm the factors of m this happens only if i matches the condition that it is a factor of m. So, we have repeated steps where same thing done again and again. And they have conditionals steps something which is done only if a particular condition holds.

So, we will stop here. These examples should show you that programs are not very different from what we know intuitively, it is only a question of writing them down correctly, and making sure that we keep track of all the intermediate values and steps that we need as we long, so that we do not lose things. We will look at this example in more detail as we go long, and try to find other ways of writing it, and examine other features, but essentially this is a good way of illustrating programming.