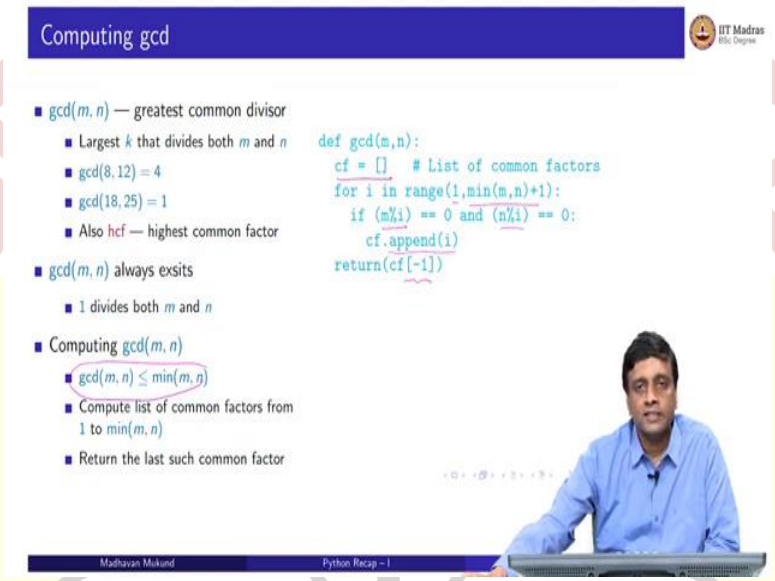
IIT Madras

ONLINE DEGREE

Programming Data Structures and Algorithms using Python
Professor Madhavan Mukund
Python Recap - 1

Let us begin with a quick recap of Python. So, I am not going to go into great detail but through some examples we will just revisit the basic syntax and just understand what we need to remember.

(Refer Slide Time: 0:21)



So, my first running example is going to be the greatest common desire is a problem gcd. So, remember that the gcd which is also sometimes called this hcf is the largest common factor or the greatest common divisor of m and n. So, you want to take all the numbers which divide m, all the numbers which divide n and find the largest k that divides both of them.

So, for instance if you take 8 and 12, then the gcd is 4, because 4 divides 8, 4 divides 12, 4 times 3 and 4 times 2, so there is no larger factor which divides both 8 and 12. On the other hand if you see 18 and 25, both of them have factors, so 18 is actually 9 times 2, so it is 3, 3 and 2 are the factors of 18 and 5 times 5, but there are no common factors other than 1.

So, the gcd of 18 and 25, the largest number that divides both 18 and 25 is 1. So, 1 divide everything as we know and that means that 1 is always available as a factor of both m and n. So, this is good because gcd of m and n is always defined, we do not have to worry about whether gcd exists or not. In the worst case if they have no common factors, like in the earlier case of 18 and 25, they have no common factors, then gcd will be 1.

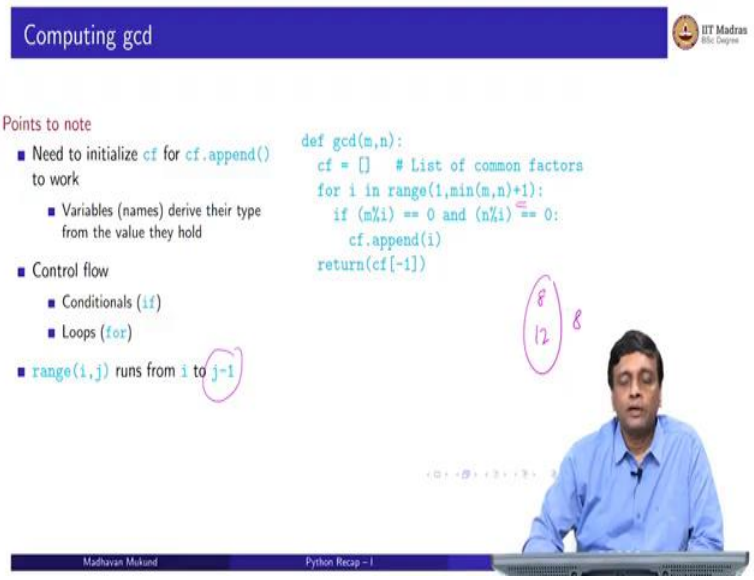
So, our question now is how to compute this gcd. So, first of all any factor of m is going to be smaller than n, the factors of m run from 1 to n, similarly any factor of n is going to run from 1 to n and if something has to be a factor of both m and n, it has to be smaller than the minimum, so the gcd is always going to be smaller than the minimum of the two numbers because it has to divide both numbers.

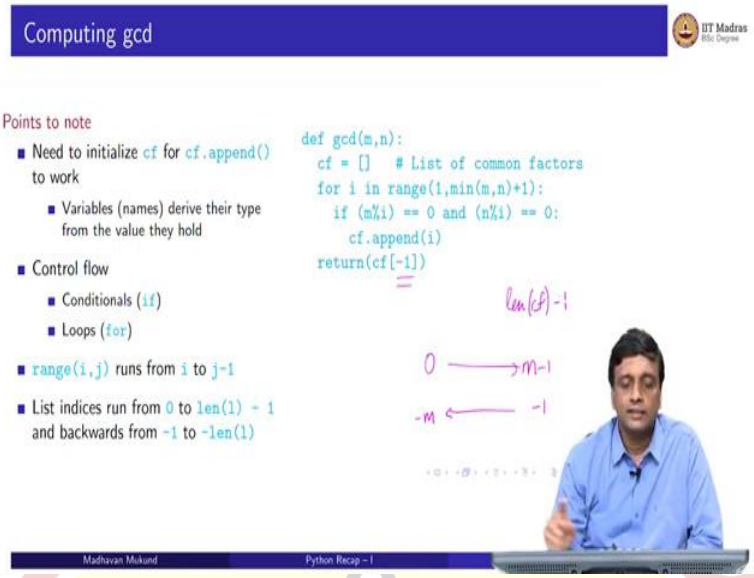
So, if a number is bigger than m, it cannot divide m, if it is bigger than n, it cannot divide n. So, the most obvious way is to run through all the numbers from 1 to the minimum of m and n and check if a number divides both m and n, so look for all the common factors, collect these in a list and return the last element in the list.

So, here is the code for that, which we will look at in a minute in more detail. So, you start by setting this list of common factors to be empty and now you run i from 1 to this minimum plus 1 and for each i you check whether it divides m, so when it divides m it means that this remainder operator percent, the remainder of m divided by i is 0, that is there is no remainder.

So, it divides m. It also divides n and if this is the case, then you add it to this list of common factors and finally at the end you are interested in the biggest one of these. So, these are being added in sequence from 1 and increasing order, so the largest 1 will be the last one and that will be at index minus 1. So, let us just look at this code a little more carefully to understand all the nuances in that.

(Refer Slide Time: 3:26)





The first thing is that we need to do this initialization here, we cannot just start off with cf inside the body of the loop because we are applying this append function, so the append function applies to any name which is holding a list, but in Python as you know, there is no way to announce to the Python interpreter that cf is a list without actually setting a value.

So, the only way that Python can associate types with names or variables is to look at the value that that variable holds. So, a value a variable which has not been initialized has no type and if it has no type, then we do not know what operations are legal. For example, is append legal or not, is plus legal or not?

So, that is why we have to first announce to Python that we have an empty list. so that when we start adding things to the list using append it is legal. So, this is the first thing to note about this code. Then of course, we are using two basic elements of Python control flow, so the assignment statement is this one which assigns an expression or a value in general to a name, but we need to run through a sequence of statements in some order or may be repeated.

So, we have conditional statement, so this statement is executed provided the condition that I divides both m and n is true and overall this statement is repeated a certain number of times and that is what for if, so we have loops and we have conditionals. The other thing is that python has this general principle that whenever you take a sequence of numbers…

So, in particular the range function is something that generates a sequence of numbers from a lower limit to an upper limit, in this case separated by plus 1, you can of course modify it to get a range in which you skip elements, but the important thing to remember is it always

stops before the last element and that is why we have to, if we want to look for factors from one to the minimum of m and n, the range function has to have this extra plus 1.

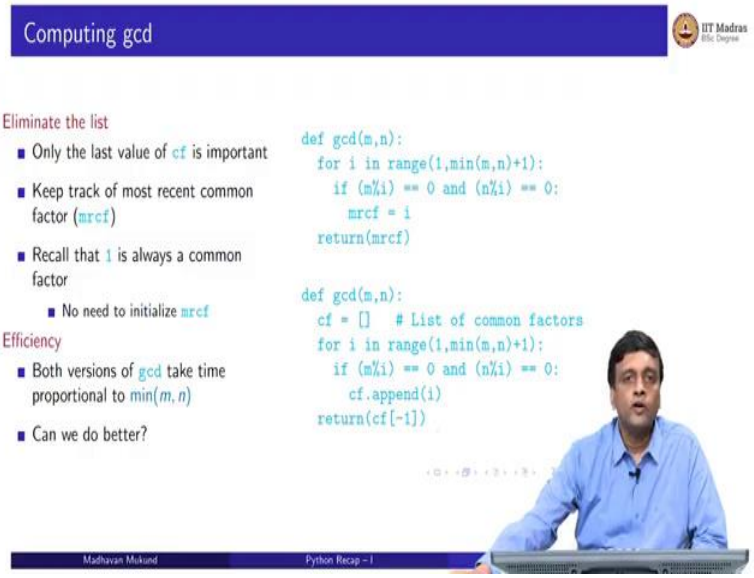
Otherwise, the last factor will not be considered. So, for instance supposing we were looking at 8 and 12, then the minimum of 8 and 12 will be 8, and we need to check whether 8 is a factor or not otherwise we do not get the correct gcd. And if we stop at the minimum in the range function, then it will only go up to 7, so this is why we have to put that plus 1.

So, these are all minor points which I am sure that you are familiar with, but it is worth repeating, so that you do not make silly mistakes when you are writing your code. The other thing is that lists are indexed starting from 0 as usual, but Python has this interesting option of counting backwards, so you go forwards from 0 to n minus 1, let me say m minus 1.

So, supposing the list is m, length m, then the indices run forwards from 0 to m minus 1, but they also run backwards from minus 1 to minus m. So, this way it is much more convenient to talk about the last element of the list as having index minus 1, which is what we are doing here rather than saying it is the length of cf, so this is the same we could have said length of cf minus 1.

This would have given us the same position in the list, but obviously writing minus one alone is much less cumbersome than writing length of cf minus 1. So, these are some interesting things which we have captured in this very simple function, which captures gcd in a very naive way.

(Refer Slide Time: 6:54)



So, one of the things we can realize is that we really do not need all the common factors, we are only looking for the largest common factor, in particular we are only looking for the last element of this list that we are constructing and if you are only looking for the last element of the list that we are constructing there is no point in remembering all the earlier items in the list. So, we can actually eliminate the list right and just keep track incrementally of the most recent common factor.

Now, remember we are computing common factors in ascending order, we are starting with 1, 2, 3, 4, so every common factor we find will be the latest candidate we have for the greatest common divisor, because once we have found it all the earlier ones are now superseded, so here we have a different version of the same code where we eliminate the list, so we have the same for, and we have the same if.

But now whenever we find a common factor, we just update this name mrcf - most recent common factor to be 1. So, we have not initialized this. Is this a problem? Will we reach this return statement and return a value which has never been defined? Recall that 1 is always a divisor, so 1 will be always a common factor, so even though we do not initialize mrcf here, like we did for the list.

When we first encounter 1 we are going to get a value here. Now, the difference between this loop and the earlier loop is that when we first encountered a number in the earlier loop, we had to append it to the list, so we had to apply some operation to the value we were storing, namely the list, so we needed to know that it was a list, that is why we have to initialize it.

Here we are not appending, we are merely assigning, so the very first time we find a common factor we are assigning it to the name, so name gets a value and the next time we will update that value we are not going to operate, we do not have to put a plus or a minus or a times or something, so Python does not really care what the type of that name is as long as the name is updated and because one is always a common divisor we will definitely return a valid value.

So, how long do these two functions take? This is going to be one of the items that we are going to look at in detail in this course, namely how efficient are the algorithms or the pieces of code that we write. So, as you can see it has a for loop which runs from 1 to the minimum of m and n, so this is going to take time proportional to minimum of m and n because that is how many times this loop is going to execute.

Whether we do it with a list or whether we do it in this mrcf version or whether we do it with a list. So, although we have changed the data structure in one case we are keeping track of only the last common factor we found and in one case we are keeping track of all the common factors we found; fundamentally the time taken does not change both are proportional to the minimum of m and n. So, we will see later that we can do much better than this, but for now let us move on to another example.