RECURSION 04

Alright guys you had seen iterative version this version where we use the loop to find something is called the iterative version, the iterative version that is we are repeating something again and again this version is what you called as iterative version, so we have seen the iterative version of finding the factorial in the previous programming screen cast, i have told you to observe the solution you could find some patters, i hope you would have found some patterns, using that patterns we will solve the factorial problem here in yet another technique what we call as recursion, so recursion is nothing but a technique in which the programming languages allow a function to call itself again, it may sound confusing at the beginning but as you go it will be rally very easy so see as i had said if you want to compute the factorial of four, one into two the product is two ok then the next time when you approach three the product till now is two, two into three you compute is six, the product till now is six, six into four is twenty four that is how you got the answer right? and observe that you need four factorial you were making use of the values of three factorial that is one into two into three the values is six is being used to compute four factorial and you need three factorial you were using the values of two factorial so there is a dependency between these values so when you need larger values you are requiring a smaller value to compute the larger value so there were so dependency so the n factorial dependent on n minus one factorial this was the general pattern you could observe from the solutions mathematically also there is yet another definition of this is one definition of n factorial so one into two into three into till n another definition of n factorial is n into n minus one factorial so for n minus one factorial what would you do? N into n minus one into n minus two factorial so ultimately if you keep expanding where would you stop? As i had said zero factorial is one so that is where we stop, if you hit zero you will say i have reached the n so one is the answer here so till now whatever has been multiplied multiply one to it that is nothing but by multiplying one answer won't change so we have reached the answer, so zero marks the end of this procedure here that is here we are depending on a smaller value for a larger value competition this process stops at the point zero right? so that place where we stop we call as the base case or the anchor case i would write it here probably base case here i will use the green colour because this appears ok so this is the multicolour command basically this appear in green colour it would be easier i guess base case or the anchor case, we can call it this is nothing but point where your recursion, recursion is nothing but your dependency on something of the same type that is factorial of n is a computation it depends on factorial of n minus one, how is that computtered? The same way. Again you apply the same formula n minus one into factorial of n minus two so you are basically doing the same thing but every time you do it again you are doing it on a smaller number so then you multiply one into two into three up to hundred if you multiply up to ten then use it to find eleven then use it to find twelve you are basically break it into smaller things and solving it so when you do that, that particular process where the bigger thing was depending on something similar smaller instance of it a bigger value of n depended on the similar kind of thing of a smaller value right? the similar factorial computation is similar three factorial depended on two factorial, two is a smaller value compare to three so something like it n depends on n minus one, n minus one intern depends on n minus two this chain of dependency how this is being that this process is called as recursion. A function calling itself with a smaller instance is what you call as recursion. So if it keeps calling at some point it has to stop calling here, we have to say that ok we have hit the answer, we have arrived at the answer we have to say that. This particular point what you call as base case or the anchor case point where the recursion stops so this is what you call as the anchor case so let me modify this functionality this particular thing is not being used here this is the iterative version iterative version now let me start off with the recursive version recursive version as i had said that factorial of n can be represented as n into factorial of n minus one this is yet another way of defining the factorial function mathematically, so this particular way we are going to use here so where will it start? Factorial of zero is equal to one what if negative number occurs? They won't occur because here we are taking care of it, if negative numbers have been given as an input we don't even call the functionality we are calling it only on zero or greater than zero only so here we may not worry about handling negative numbers because in the actual calling part we had handled it so we are not worrying here ok this thing we are going to say so first the first step in recursion is always putting the base case base case, here base case was factorial of zero is one so i should say if we have passed the number as n, n is equal to zero i should return the product as such right so let me bring use this statement the product is initially set to one right? this is not needed actually i can even remove this i can directly say return one so i should return one so i had return one that is factorial of zero is one that is why id n is zero i returned one else what should i do? I should return n into n minus one factor n into n minus one factorial so i should call it as factorial of n minus one ok let me give you illustration before we actually run the code, so if i call factorial of three let suppose let start with a smaller value so that it is easier you can extend it for any value even factorial of hundred anything you can extend that's not an issue ok so how it begins is it will first check if n is zero three it is not equal to zero so it will come to the else part it will see three into factorial of two, so value of factorial of three depends on the value of factorial of two, similar instance but the value inside is smaller ok so now this particular thing has to be called ok factorial of two gets called so again see factorial of two so two is passed here n is equal to zero no two is not equal to zero so its comes here it will say two into sorry two into factorial of one ok so now factorial of one gets called ok what is factorial of one gets called, so one is passed here if one equal equal to zero so false come back to else return one into factorial of n minus one so it will compute one into sorry factorial of one minus one is zero ok so now ok factorial of zero, zero is passed here n equal to zero, zero equal to zero its true so it will return one so this will return one so now your calculation will go in the upward direction ok let me show you the upward calculation ok so one is returned so it goes one level up, one level up so it will calculate one into one what is the answer one so that particular value is passed to one level up ok so two into what is the value being passed there one so that is being calculated so two into one is calculated two into one this one there is a passed from one level below for it there is passed from one level below so the computation breaks down into smaller numbers and once you get the number where you get the where you know the answers you trace back till you get the larger answer that is how we are working here ok so two into one is two ok let me now say let me now say so two has been passed here already it was having three, three into two, three into two ok what is the answer? Six so this thing has found it so it will get the answer has six, so this answer will be passed on to the main function value we have called this so when you call factorial of three it will keep splitting the problems into smaller ones until it knows the answer and once you know the answer trace in the backward direction till you get the answer for the larger problem this is how recursion works ok this is the Woking of recursion let us start a new console and let us run it ok? before that i have to save this i will save ok let the console start ok so it has started so let me run the file enter a positive number the same negativity check all these are the same so i am not giving a negative number so let me give back as positive number let me say five, factorial of five is one twenty so in a similar fashion it will break down five will break down to four, four will break down to three, three to two, two to one, one to zero for zero you know the answer now use that answer trace in the backward direction and you will get the answer as one twenty. So this is how the recursive programme work, i hope you have understand the concept here or it's not a difficult thing it's a easier one all that you need is some practice that's it take some examples you try thinking in this fashion, try breaking into smaller pieces then once you get the answer try constructing back the actual answer. So i hope you have been remembering the movie clip that has been shown at the beginning of the video, the person would ask answer for something to a person before him, the thing keeps on propagating till someone who knows the answer, once you get the answer you propagate in the backward direction to the person who has asked the question, in this case what is the factorial of three? That is the question asked by your main function that is question asked by your sir, so you cannot compute it directly so you try splitting it into a smaller instance so that is something analogue us to asking the person before him whether he knows so whether this thing can be calculated it checks even this cannot be calculated so again split this is can be calculated no so again split can this be calculated? Yes, calculate use the answer and propagate in the backward direction, i guess now you understand why we had shown the movie clip and also how the concept of recursion works, this is really a very powerful concept in computer science, it will make it institutive things be easier to code that is something will be intuitive for us we will have in mind that this is how things has to be done. But how you would translate it into a code? It may be difficult if only iteration value for rescue recursion is handy in many situation you would come across as you keep learning some advance stuff even in some stuff what you had learned till now you can have a recursive version for some problems please note that for this we had an iterative version as well as a recursive version so for every recursive version there is an equalent iterative version too, so you can use anything that is convenient for you and in some places recursion is highly intuitive and easy on minds so once such application will see in the next video. Thanks for watching, have a nice day.