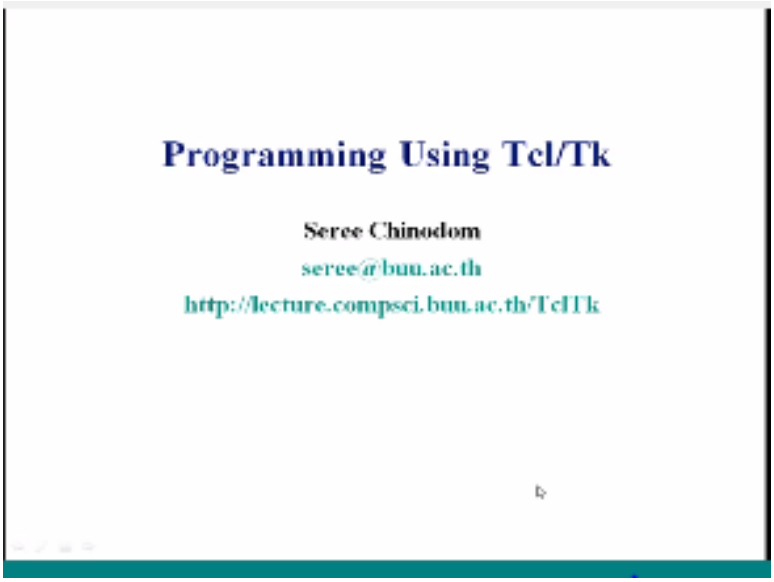
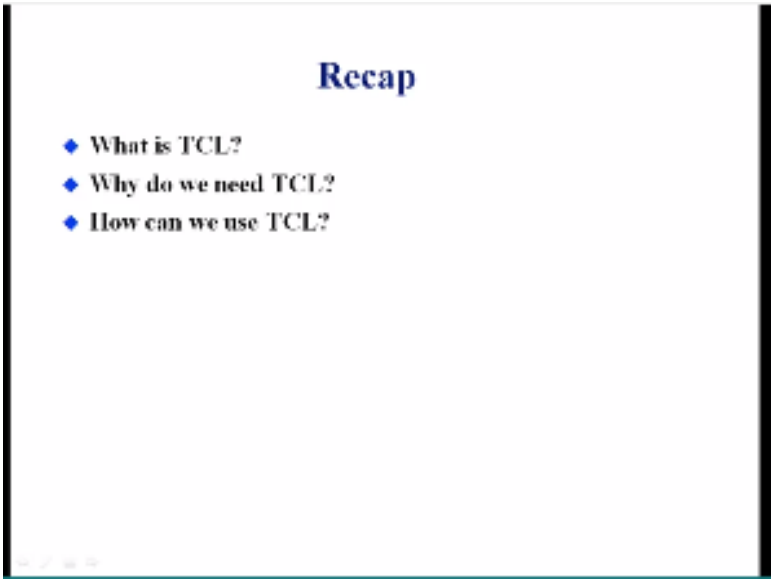
(Refer Slide Time: 00:04)

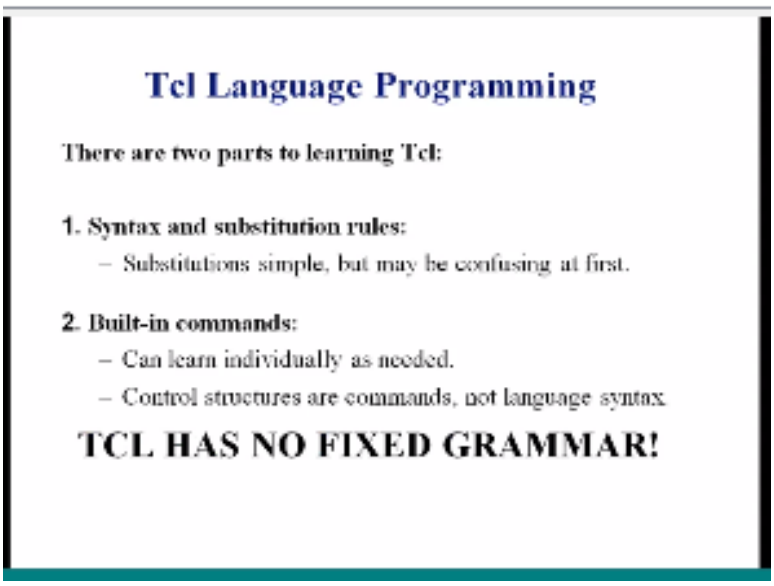


Once again welcome to the Linux programming and scripting + today, we are going to continue with the looking at particular decay from where we left off let me first summarize.
(Refer Slide Time: 00:18)



For you what we talked about in the last lecture we talked about what is Tcl the tool control language essentially it is it was developed by Oscar out in UC Berkeley again it was originally developed as just a scripting language meaning, for just solving some of the difficulties with the existing scripting languages. So again like that goes back to why we need Tcl in the Subspace the

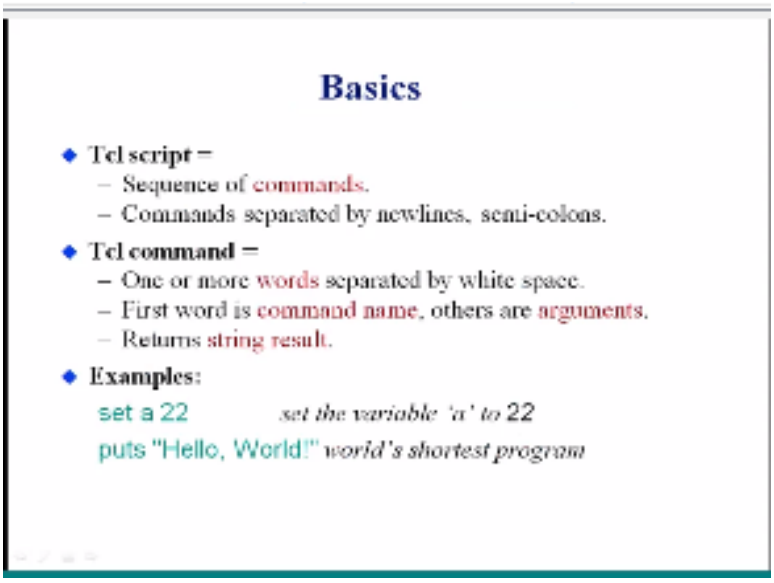
reason is essentially like this is one way to actually use it in applications you know use the one applications as well as extending the various applications.
So, what we mean by that is we saw that in the last lecture that there are two things basically one is on one side or programming is all the debates essentially with very regular data structure and a lot of complex programming and it can take like years to develop and actually, bring it online whereas on the other side we wanted very easy to use some highly interactive quick development capabilities for scripting the former one essentially like there we have very complex we can write very complex programs with very highly developed data structures.
Are languages like C =0 but on the other end want to like just quick scripting that is where the Tcl falls into but one thing to understand is actually the Tcl by nature is it can act it is platform-independent that is how the Tcl is developed so multiple applications can be easily connected through by Tcl as well as you can extend an application or new you can control the ideas or user generated ideas using people and then we also saw like I mean oh can we use Tcl so essentially falls under this bucket as to how do we extend them how do we use the various applications Tcl.
So today Tcl is some very popular just use widely in many applications you name any commercial tool today it is has a Tcl interface and you can use people to interact with the tool as well as extend its capabilities one thing that we will talk about in this Tcl class is also the synopsis tool set which you will be using extensively and synopsis tool set or also the entire tool set is built on Tcl and you can use Tcl scripts to communicate to the tool so this is tools like design compiler.
I see compiler things like that you can easily use Tcl as an extensible language today it is not just limited to that synopsis tools but other commercial tools also use Tcl as their backbone. So today we will be talking about the more of the language constructs of Tcl. So before we go into that I wanted to just give you a brief background on how all the commands is constructed in Tcl.
(Refer Slide Time: 04:22)



So there are two parts of learning Tcl one is the syntax and substitution books we saw this one the substitution fell simple but may be confusing at first. So we will deal with this and then the second one is all this built-in commands the this can be learnt individually as needed and then the control structures are actually command and it is not language syntax. So essentially, like in Tcl even the control structures we will see all this commands. So we do not create anything differently than any other command what that means you say like I mean you if you write at the mall or anything and a specialized application.

They also fall all under the similar treatment as any other command basic reading again if-then-else which has no built-in commands. So again we call this as not control structure but just built-in commands. So we will see like I mean how Tcl program looks like or what is the structure of a ticket program and then, we will go into more details one thing to note is Tcl has no fixed form so essentially it is all like there is no like a lexical convention as to how you have happened.

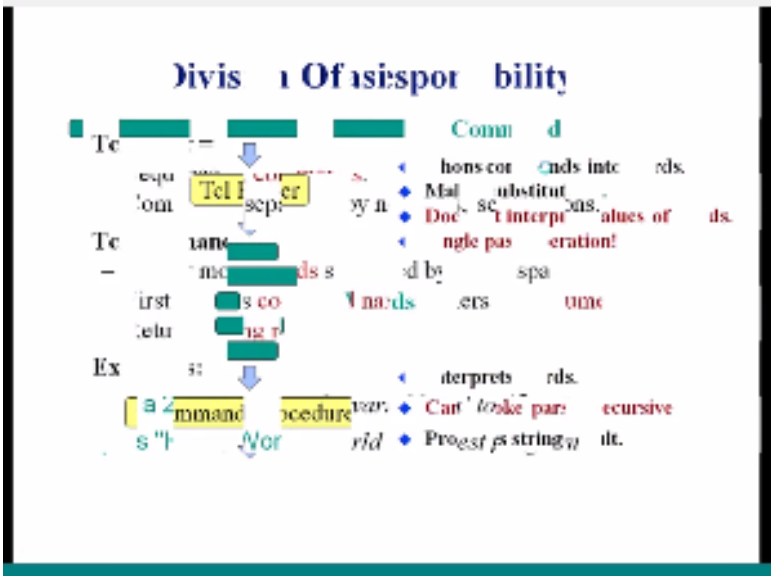
(Refer Slide Time: 05:51)



So again going back to basics Tcl script is just a sequence of command, the commands are separated by new lines; and then essentially a typical command itself like me. So now we look at the script is comprised of commands and the command termination. For blue lines or; now the Tcl command itself each command and have one or more words separated by just white space. So the white space has meaning in Tcl and then always it is taken as the first word as the command name and all others are arguments and the command always results in a string.

So those are the things that you need to understand and you need to keep in mind and whenever we talk about any commands just think about these two basic concepts. 1. A Tcl script is nothing but sequence of commands and then commands are separated by either new line or;. So these are the only command terminations. So if you do not terminate it with a new line it is it still continues and set mode so we will see like some examples as to how this plays out in the in the real world because, there are these concepts are also like confusing with respect to like a language like Perl.

(Refer Slide Time: 07:26)



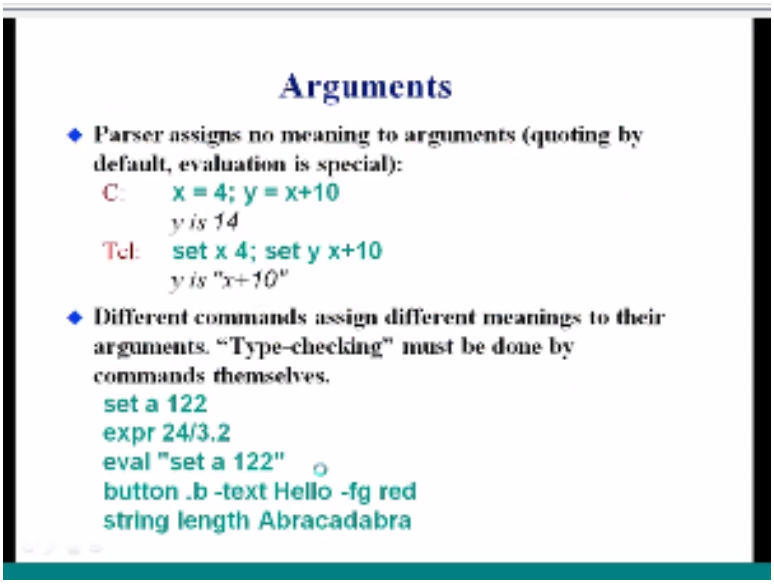
Where only the; was the, main the main terminator of the commands and Tcl command again it is one or more words separated by white space. Again that is another key difference and very important in other languages you do not need to separate by white space and not even the ladies characters have different means and that is used as a parsing mechanism but in Tcl it is it is very simple basically like the white space is treated as a word separator and then the command.
Actually has this number of words in the first word is always demanding and then the remaining ones are all parts. And it also returns a string value so here, is one example this command is set A and 22 this is the same as A = 22 in Perl. But, notice that there is no space here, we do not need to have any space between we can actually say the $ way. That is more make sense so $ in is a variable and then the assignment is assigned into a Terminator as; whereas here there is no $ here, and 22 in there is no termination; and then.
Another very short Perl program whoops hello world. So here notice that actually like whoops is the command name it has only one argument right now which is inside this Court hello words. So it just comments all of them and then just outputs that so I think so far so good so let us move on.
(Refer Slide Time: 09:39)

So in terms of how it commands are parsed basically there is a division of responsibilities inside the Tcl parser. The Tcl parser has two components one is the Tcl parser itself, and then the other one is a common procedure. So they go from the concepts that I talked about earlier. So understand that this line is the vertical command essentially it is just a string of characters that are coming one by one the Tcl parts that all it does is it chops the commands in the words. So this is one long command it basically like takes it and then it basically chops it into first word, second word, third word, fourth word and fifth word.

Now so mainly it so it also makes some substitution this part of it we will come to it does not interpret any values of words. So it does not assign anything, and it is a single parse operation. So every command basically the command you know that it is separated by either a new line or a; so it gets all these things that that forms this command. And then when it goes through the Tcl parser first it just breaks it into multiple words based on just the whitespace character and the same blank.

So it based on the whitespace character it just splits it into all these words and then it just sends these things to the command procedure which essentially interprets the words of has commands. Whether it is there, and then once the first one actually look at the command and the remaining on some arguments it takes in and then it processes and then it produces a string result. And then that result is all this focused. Now let us look at all these other things basically like how this entire process works.

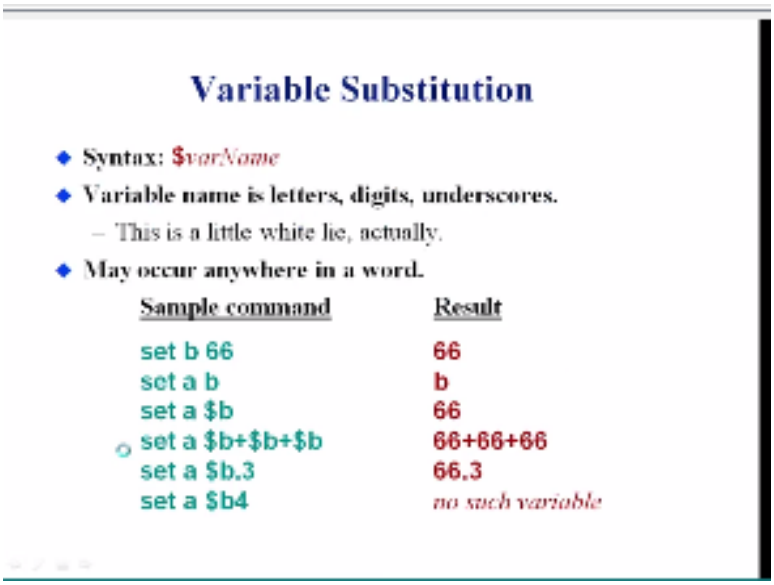
So once again this to recap the resolution of responsibilities in type optical parser the parser of the true parser basically once it gets the command which is separated by the newline or ; it parses and it actually separates the whole command into multiple words. This chopping is based on just that blank space basically, that we say and it also makes them such it substitutions which we will see what it is it does not interpret value of the words and it is a single pass operation.
Then once it chops up the works it sends it to the command procedure which interprets the words it can invoke the partial recursively meaning. You have something that another, and it can actually like then go back to the parser and expand that and then gets it and finally it produces a string result which is shown here as another string.
(Refer Slide Time: 12:49)



So now let us talk about some of the things out of all there are arguments, which are after this one all the remaining ones are the arguments right. So the passer assigns no meaning to this argument so here you can say like okay in the in C program X = 4 y = to X + 10 then Y is actually 14 because this Y is evaluated and then basically like the value is kept. Whereas integral if you say set X 4 and set Y X + 10 the X left end is just retaining basically just get that thing and the different commands.

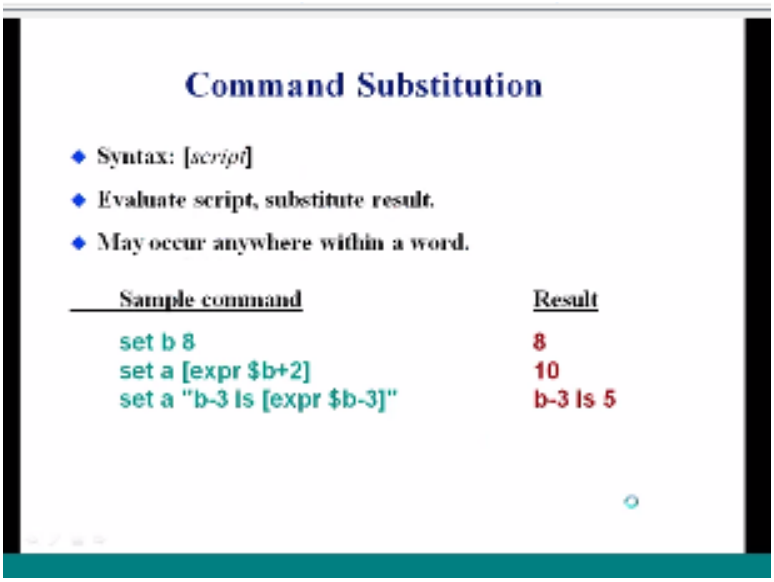
Assign different meanings to their arguments the type checking must be done by the, and them. Again we will we will discuss this in more details but, the commands themselves has an onus to see whether they is of the same type and if they are of compatible types and execute it does not go back to the language to provide any kind of guidance as to what this so here we have a set of commands first one set a 112 then we have another , and called XPR which actually evaluates an

expression and you can see actually that there is a fixed point number and the floating point number.
So now how does this get interpreted, so now the third one is basically instant how value has a string as an argument. And in this example the button. B it has many arguments also so button . B has my - text as an argument hello it is an argument if G is an argument and it cannot. And now another , and for string which is an argument of length and then another kind of a string but we do not know what it is but it is just another word because the string has two arguments now this one has two arguments these are actually three arguments basically, and then you have this has only one argument this one has one two three four arguments and this one has this towards and there are all of different types.
So this is a string of one type here, there are string arguments and all these are selecting an argument here, there is another string this is a floating point and the fixed point this is a fixed point so again how to distinguish and how to make sure that there are all consistent is left to the particular , and that is that is getting both sides. So we will see like some of this examples as how the man shillings with certain arguments.
(Refer Slide Time: 16:23)



Now we come to this variable substitution there are two main topics in Tcl is that you would not understand and once you understood those two concepts I think like, like will be much easier the 1. Is the variable substitution. 2. Will be the , and substitution. And this is something that we will talk about in a later slide not today right now let us talk about the variable substitution so the

variable substitution the syntax is $ followed by the variable name actually the $ is what gives the Tcl to see that okay this is available and I will substitute this in single one.
And the variable name can be letters digits and underscores it is a, it is a basically like a miniature white line the main reason is like sometimes you do not put the digits at the first one but you can we will see how it is so and then the other thing is basically like this $ can occur anywhere in the world in the world in Word so now let us look at some of those commands and what happens. So here let us say we are setting B to 66 now when we sail except A to B the result is this B what is that this is another just value it is not the same variable in order for getting this b-66.
What we need to put the key variable substitution syntax which is the $ so now in this command set a $ B now we get a s 66 now what happens when we say like set a $ B + number B + $ P notice that even though like I mean we our intention is like 60 6-4 6-2 6-4 66 you know and evaluate that we do not have any commands for valuation so it is just going to assign exactly the same variable which is it substitutes every $ the occurrence of every $ B by 66. So there will be A string which is 66 specific statistics this is not the same as um like 196.
So please note that try 2198 it does not put it that way only light has the 66 4-6 6-4 66. So now what happens when you sail accept a is $ B. 3 so this. Actually it is also applied. So here, you get 66.3. Now what happens when we say like set $1 B for is do the same as 6, 6, and 4. Actually in this case it just says that there is no variable because now the four becomes part of the variable name because you can see that activated mean these little bits and underscores so it treats $ before as a single variable and it finds that actually that is never defined so it just complains about no such way.
(Refer Slide Time: 20:38)

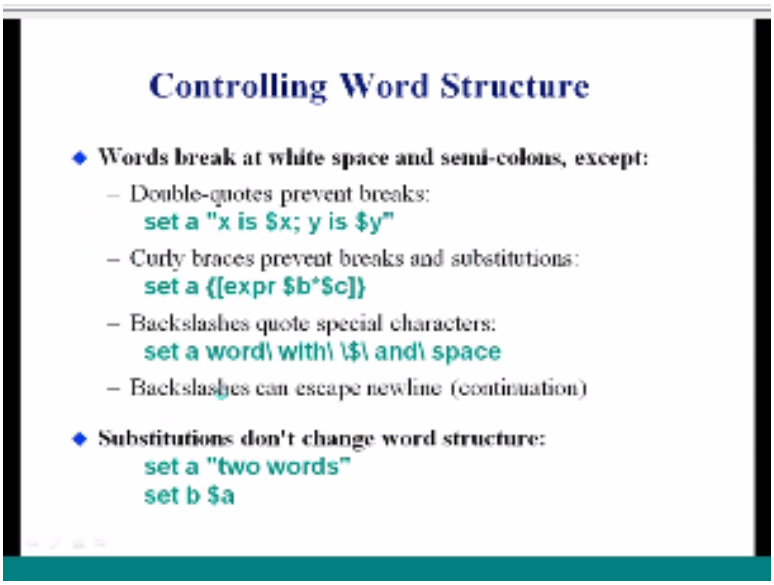


Now the second aspect of it that I talked about which is the command substitution. So for the variable substitution the T character is the $ now for the command substitution the key characters are this [ ]. So any script which is embedded within a [ ] it substitutes that our interprets that as a command and then executes that number. So essentially it is evaluating the script and then substitutes the results.

And it may occur anywhere with enable so let us look at some example here, so here, set B is 8 so B is set to value 8 so that is the result here and then we say like set a as expression $ B + 2 so now when you say like the [ ]. And then we give another function name which is or a command name it is EXPR here, so that EXPR is now neatly executed because it is kept under [ ]. So now it executes this command. So when it executes the command its knows that these 8 so it substitutes 8 here.

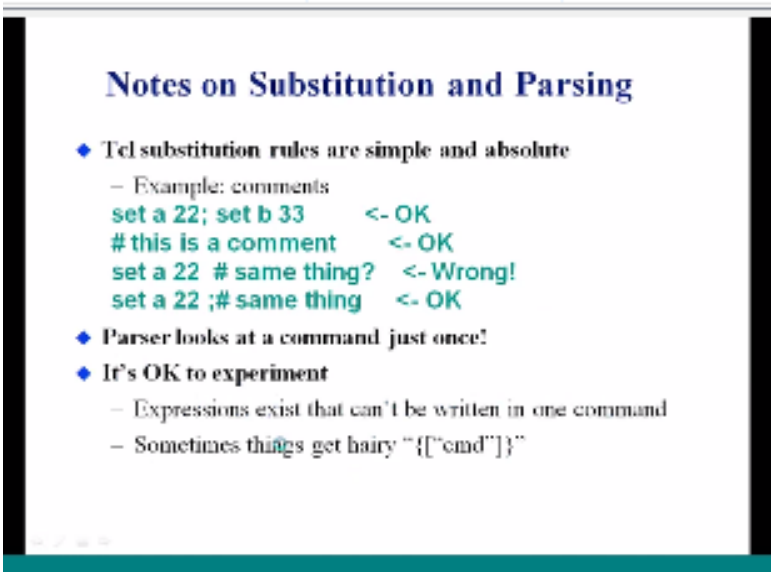
And then it actually adds this also so there, is a variable substitution and now a command substitution as well. So now it evaluates 8 + 2 circum them so the answer is 10 and then that it descends to it so here, this is the distance. Now what happens? when we say like set A, B - 3 is expression B - 3 notice that there is no $ in front of B so B is taken as just another just name and this whole thing is a string. So it is basically like it puts B - 3 as a string and then it also uses the space ease as a string. And then when it comes to the [ ] again it substitutes the command essentially so, that is $ B is 8 - 3 is 5 and a B is it as B - A is 5 as the string that is stored in a okay.

(Refer Slide Time: 23:19)



Now the words break it white space and; except there are few rules on this 1. Is the double quotes the double quotes prevent breaks because the entire double quote is treated as this one string and then it is taken as one steps. So for example in this one in this example set A $ X is I mean X is $ X and Y is $ Y this entire thing is assigned to a as noticed in the piggy signal example. Only the $ X is substituted $ Y is substituted but this entire string is disappeared to A. So the words do not break here essentially it is just one continuous form now the second case. Where, it will not break is the curly braces essentially the curly braces themselves is considered as one unit wherever you are using curly braces. It is used it is considered as just one word so anything between the curly braces is one word. I want you to understand this concept because we will revisit this over and over again later on so that is another one. The third one is the \ which code special characters. So here, set a word \ empty with \ and \ $ in \ and space. This entire thing is treated as this one and then the \ also can escape new line. So when we say like I mean when the end like a 1, 2 ,3 ,4 and is \ and then =5, 6, 7, 8 this is just considered as one row with 1, 2, 3, 4, 5, 6 and 8. With no break here or because of the \, \ is just escaping the new line. So it is a, continuation now one more thing is the substitutions do not change the word structure some here, as an example we set this AAS to words and then when we say like set B as $ a it is not going to change the word structure itself. So this is as a treat as one word it had still be assigned to B so there is no nothing there.

(Refer Slide Time: 26:25)



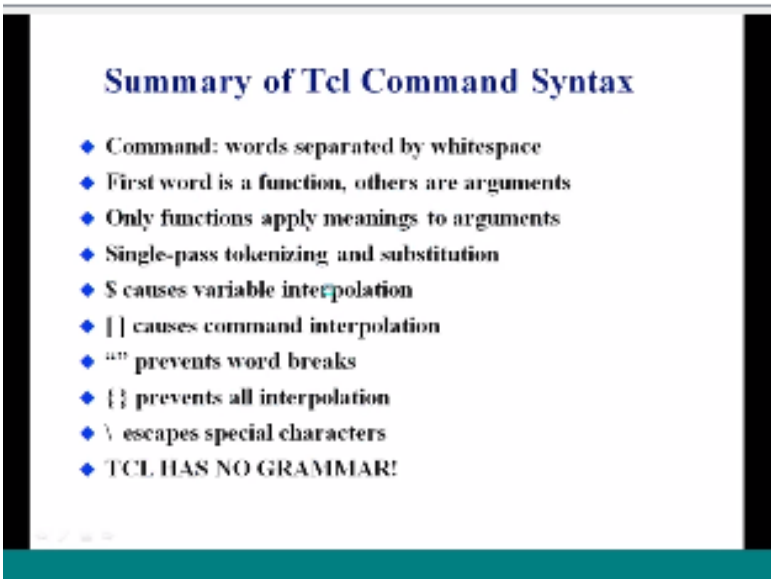
Now let us look at some of the typical substitution rules so and parsing one thing is the key to such as you can move or early simple and absolute. So here, are some of the examples. So here, we do set a 22;set B 33 this is okay because we have A ; so that me knows that actually this is one command this is M now we have a command, commend is always like defeated by the national symbol and here, ash this is a comment this is also okay because this happens in a new line and with the first character as the - now when we sail accept A 22 ash same thing this is the wrong one long usage.

Because, there is no break and basically now you can say that actually a is having like two or more words and all these are arguments but the set A has can set can take only like two arguments we see. So this is really not a valid syntax and it comes with an arrow to correct this one we need to include a ; between there we are now breaking this into two commands and then the second command this happens to be just a comment, comment okay. So the bottom line is essentially the parser looked at the commands only once this does not treat okay.

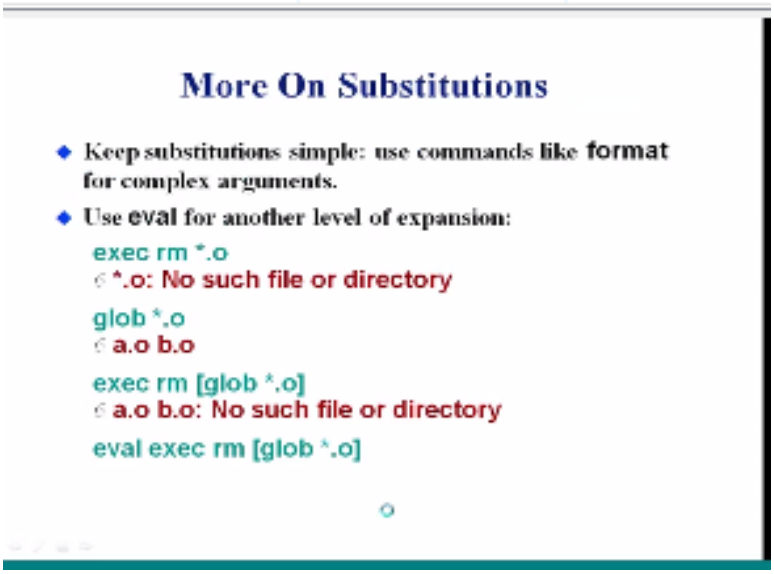
Now that there is a command starting so maybe the formal command and if its treated as a comment it does not do that and so as a result what happens is so we cannot write expressions in this single command we have to use multiple commands samples we will talk about one or one such example mm slightly later on but I will I would like to give you and motivate you on the pump oh and one thing to notice actually like that that particular kind of firm version I have never been written as a single command or we cannot find a way to write a single.

Sometimes things get hairy basically again where we put this curly braces with square brackets so the curly brace essentially is a way to that we saw earlier that is to combine words into one

here, it prevents the breaks and then look at the square bracket within basic you so that is to evaluate the text session so it is a command substitution.
(Refer Slide Time: 29:45)



So now we will go into the command context so, the Tcl essentially like I mean what we learn all the commands are words separated by white space. So as it shown here and the first word is a function and all the others are arguments only functions apply meaning both of them and it is always a single parse to the mountain for fusion and then the substitution can take two forms 1. Is a variable substitution they are the ones of emancipation the $ was this variable contains a subscription or variable interpolation and then the [ ]. For the command substitution or command installation to prevent the word breaks.
We can use three main things or you can also extend it to the four one, the first one is using the chords that prevents the word breaks and then we can use the curly braces that also prevents all the interpolations and then the \ escapes all the special characters. So including any word breaks so, this is another way that we can avoid the word break. And then another key thing that we are always talking about is both Tcl has no grammar what more.
(Refer Slide Time: 31:23)

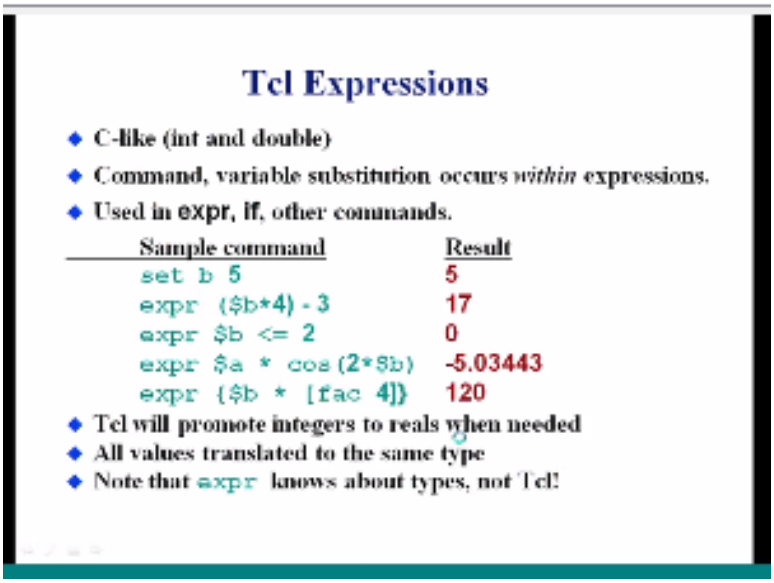


So now let us look at some more substitutions. You so here, we can use eval for another level of expansion so and then we can also use command of format for complex arguments. So exec RM star. O. Basically this exudes this particular command again this itself is a command that takes on two arguments and then the first argument being another command books executed. And here, we see it basically language start off oh no such file. Because, there is no meaning to the staff. O. And then if you say globe starter go that gives the two files which are in the directory this A. O and B motto.

Now if you say basically exact RM and then execute this command also using the command substitution [ ] yeah it comes up with that and then that it saves the no such file or directory because it is treated as just one, one full word. So try to find the file name called a V Rho. O is b. o and it comes up with no such file so in order to delete these two files in this directory we need to use this eval function. So when we say eval then we say exec RM glob $ start or oh this works correctly and essentially.

We can now remove these funds so eval is used as another level of expansion and there are other ways of writing this also on warm this is the test see using like this one sign. You can also split this whole thing into multiple lines and then and also like it is you could say in things anyway this is one challenge I am going to give you how to get this without using the eval but using multiple lines. So try that out we can discuss in the next class.

(Refer Slide Time: 34:41)



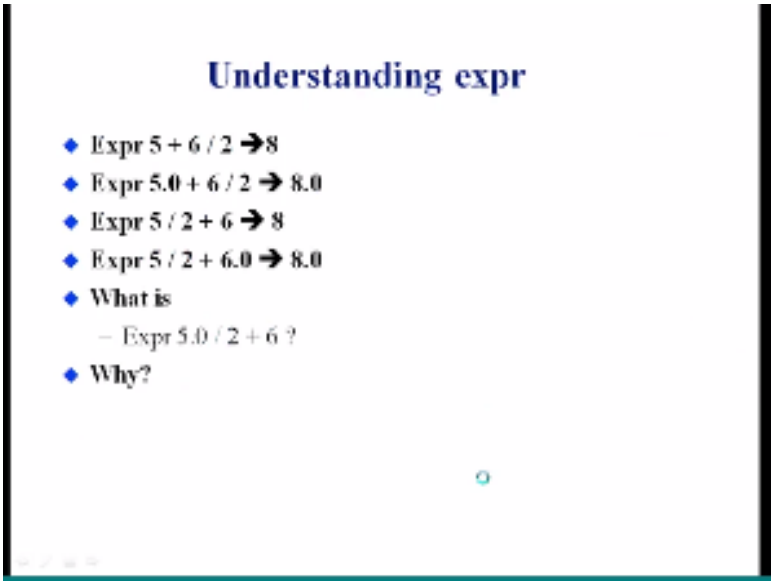
Now I want to talk about the Tcl expressions basically this small see like int and double we can think of this as integer and floating-point so we will see like how this works the command variable substitutions all occur within the expression itself and it is used in expr if and other commands so what do we mean by the, the command and the variable substitutions and how are these exhibitions for done so here is some sample comments set b5 so the result is 5 now we have this command called extr $ b times 4 – 3.

So the answer is 17 because it is a 5 this gets a substituted with 5 so it is a way of substitution so that is 20 - 3 is 17 now you can also say like I mean some logical operation with BB then I throw too and since it is not it returns the false value which is actually 0 in the Tcl 0 is always false and anything other than 0 it is always true. You just keep that in mind now here we also say like I mean from path function basically to $10 be multiplied by A here, you can see that actually it is the floating point number it automatically converts that importing point number.

Because one of them happens to be the floating point number and usually it is it calculates as any, any data type until the type changes then the type changes it gives that the new type meaning and then the president is essentially like I mean Peter has the lowest residence on bone the floating point has higher business in the expr statement. So if you put a floating file up friend everything is completed a supporting bubble if you put the floating point towards the end then until that time is computed as an integer and then two at the end it is turning the point.

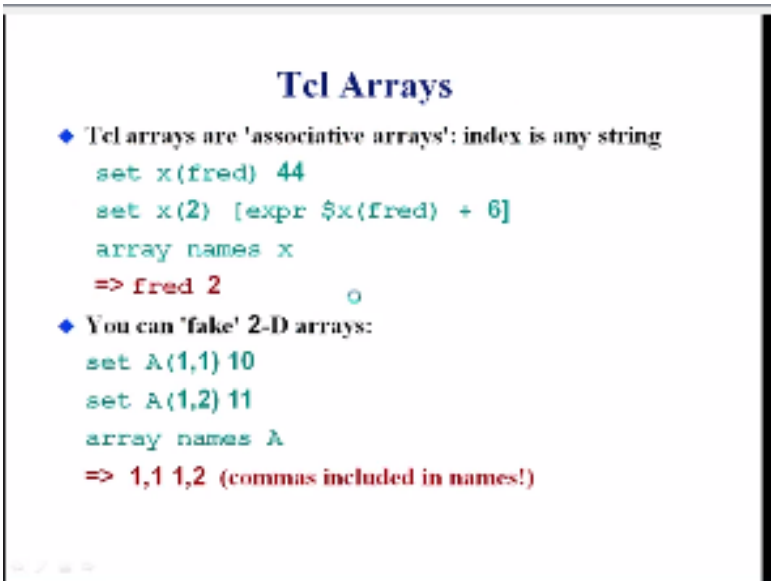
We will see some examples as to how this will affect, now the other one is expr talk B times facts for now here, you notice that actually like among the boom of Robotics. So this is um command

substituted basically if it computes this track for and then it replaces that it is at four and then times $5 B so here this also will prosecuted so it is 5 times 4 factorial which becomes 120.
So the bottom line is Tcl will promote integers to rears then needed all the values translated to one of the same types and then here also like you can notice that actually the extr command knows about the tags and the Tcl does not Tcl is still it is just a collection of words. So imagine rights so it is get the collection of words and still move this go and today a lot of programs actually run with this.
(Refer Slide Time: 38:43)



So now let us look at some more examples to understand expr in more. So the first one is fairly simple now you get an answer 5 + 6 by 2, so 6 by 2 is evaluated this is 3 and then this. We added 2 pi we get the 8 now the next one is 5 point 0 + 6 by 2 again since this is a real number everything is converted into real and then begin an answer of 8.0 which is a real number. Now it gets slightly interesting so now the next one is 5/2 + 6 so it evaluates 5 divided 2 which is actually 2.5 but since this is an integer leave these two out which is just 2 and then when it adds to 6 it is and eight now this expression again.
So if you look at it basically it is gain five pi divided by two which is 2.5 + 6.0 but since then it computes that it is still integer data type so it does not use this + sacred adds and then that becomes eight point 0 now the interesting thing is instead of putting 6.0 I put 5.0 below to the sticks now you look at this actually this is already real number. So the answer also should be real

which is now 2.5 now 6.0 then you give you 8. 5. So look at these two expressions they are exactly identical but those two have two different answers.
So this is another caveat that you have to always adhere to in writing Tcl programs you know the data type actually changes as it goes so if you have like the variable substitution all over and at some point it turns into like floating point you get a different answer than just a number if it is just our integer okay.
(Refer Slide Time: 41:19)



You so now let us talk about the Tcl arrays in general and since we are using string for more words and mix of etc the Tcl arrays always associate one is the index. Can be anything that you want it may not be just a integer number can be anything that you want for example here step X Fred so the red representation is X followed by boom index here the index is called Fred and we set that to 44 now. We say set X to is this expression which is $ $ X friend so here, the command is available is substituted to 44 and then the , and substitution happens because of [ ].
So this expression is evaluated and you get 50 as the answer and that is a thank you x2 so now if you want to print out the name. So here, the arrays are basically like visible meaning and value so to print out the name we just say array names and the array name is X then it prints these two that is basically instead and two. So using this principle we can actually fake two-dimensional arrays we do not need to have construct the two terms nary but we can construct a single dimensional array and then take it as a 2-dimensional how do we do it here is an example.

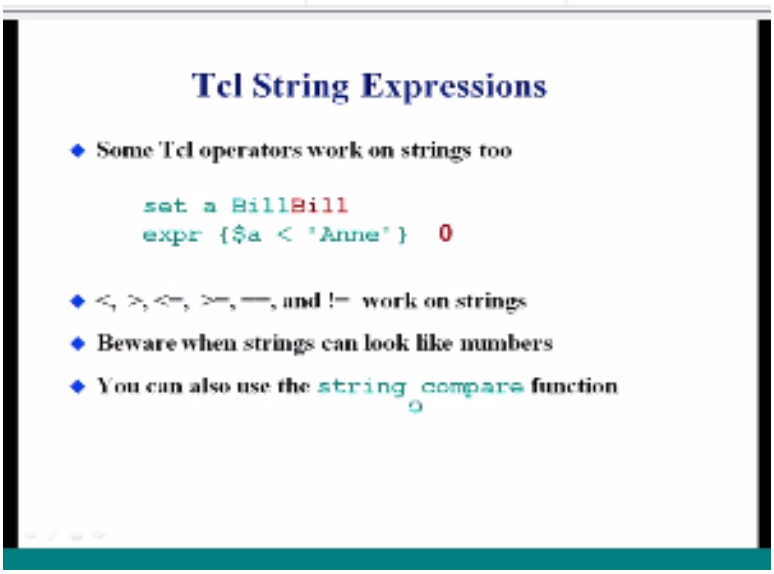
So here, we say set 8 1 , 1 as 10 so it appears as if like I mean a two-dimensional array with 1 2 3 4 and then we will go 1, 2, 3, 4 and then we are making this 1, 1 of 10 and then 1, 2 and then but in reality this is just one name so 1 , 1 is a name that has value of 10 and 1 , 2 is another name that has a value of 11. So if you print out the array names a then it prints out together which is 1.1 .1, 1 and. So now you see that actually like, s can be included in means.

So now let us look into the one of the things that we saw earlier in more detail this is the expression a $ a x + 2 star $ B 2 multiplied by gamma P so here actually this is evaluated all these things are basically like voila a and B are substituted so whatever this value that they get substituted and then the expr is called after that basically so it is substituted and then A. X here, is called and then expr called after one. And say the thing I do not know the formula T that goes back it gets evaluated then it goes back to expression.

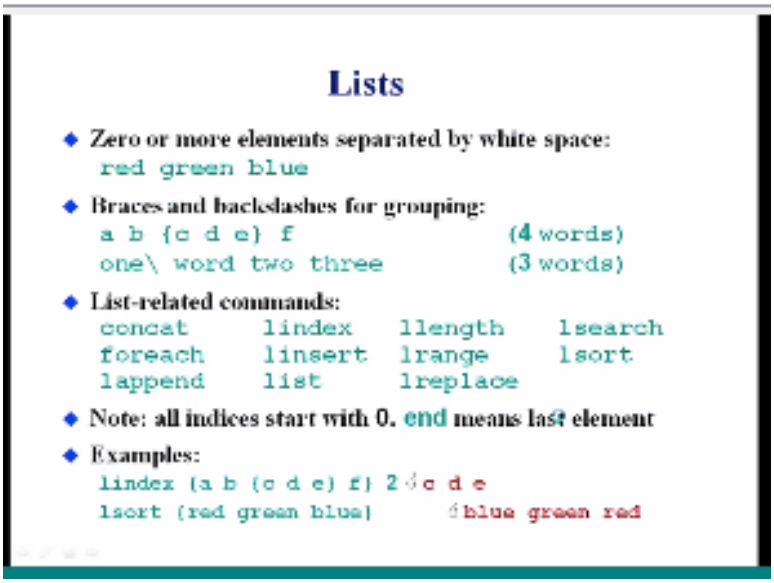
And then you get this answer you so the difference here, is in this one the $ end of A, B substituted by scanner before the expr is called whereas in this expression .Where we are saying expr notice that this curly braces and $ B star fact for essentially here the doll be substituted expr itself the reason is because of this and you should have the single goal expression. So it basic in substitutes this so you get again 5 into B get because substituted so this is evaluated in time 120.

So the expressions can get substituted more than once so here, one example set B is \ $ A and then set a is 4 then expr $ B times 2 now you can see that actually begets was substituted with this a and then it looks at the A and actually the is also needs to be substituted so it is actually 4 and 2 people.

(Refer Slide Time: 46:58)

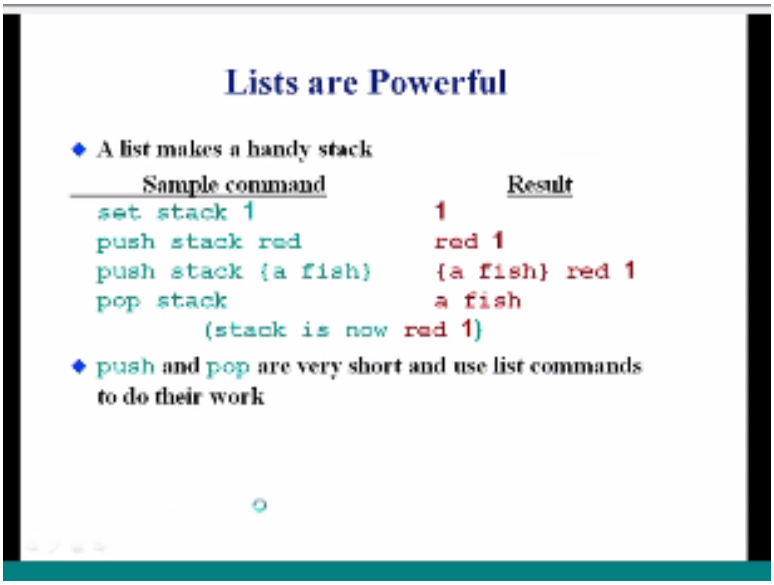


And some Tcl operators can also work on strings for example is bill, bill expr $ a < and is going to be 0. And then the left and > ,< or = > =, = and then not = you can all work on strings only thing is you have to caution yourself when the strings can also look like numbers in general the Tcl is always like generating with strings. So when it is generating a number with MVS assistant, and you can also use the string compare function to compare any strings.
(Refer Slide Time: 47:58)

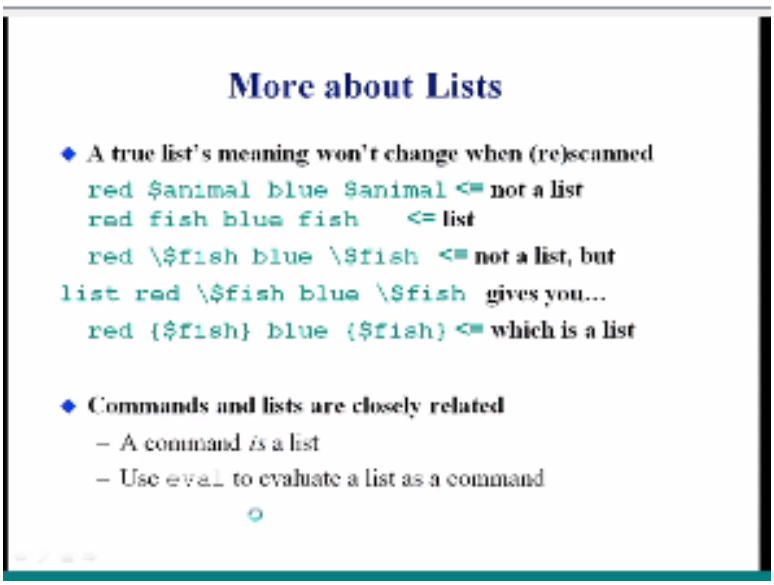


Now let us talk about the list the lists are nothing but 0 or more elements separated by which the thing in each component is the 0. Let me not be just one in this little and here, red, green, and blue this is a list essentially. Because it has a white space in the curly braces and the back flashes are for grouping this we saw earlier. So here, a, b, c, d, e, f and the CDE is included in the curly braces so this is treated as this one. So there are actually four words which are a, b, c, d, e and then f.
Then we can also like use escape characters for hooping so here, one escape and then space and then word this is one so this is 1, 2, and 3. Only three words then we have some list related comments which are all these things the concoct l index a length L search for each L insert L range l thought L append list and LT place. All these things are related commands, and one thing to notice all the indices will start with a 0 the end means the last elements.
So here, use this function called L index, L index of this array with two that the second one is actually post on 0 ,1, 2 and 3. So the two is this big CDE expression which is which will and then

when, we do that L sort of red green and blue it does an alphabetic sort. So to put this as one put this as two.
(Refer Slide Time: 50:08)



And you the list actually are very powerful syntax Tcl we will learn more about lists and there are some specific comments. That it also allows a good, to light the synopsis and tools. So a sample command will be set stack is one so that means that the stack gets warm. Now when we say like push stack red the red is pushed ahead of one for one is not the end and then that is pushed and then it is a push stack of fish with the curly braces that gets presidents essentially at the pearly basis and then it puts the curly braces fish at the first element then, then followed by one and if you pop the stack.
Then it comes up as just a fish because it is only like the first element is popped and then the remaining ones are kept inside so now the stack has red one but a fish is popped out and then the push and pop are very short and useless commands to do their work essential you, you now.
(Refer Slide Time: 51:44)



Some more interesting facts about lists the true list meaning would not change when it is reset you. Now let us look at this example where the first $ animal glue in $ animal this is not a list because there are this here which may not be substituted. Now fish blue fish red fish blue fish this is the list now red escaped fish bloom escaped fish this is not a list but if you say just list read this one this gives you basically like red and then it puts this as a um one collection and now this can be a list since that is a one.

So one thing in that commands and lists are closely related that command is actually a list and use eval to evaluate the list of lists as a comment. So in summary today we are learning more and more about Tcl two things I want you to take away here from this lecture are this variable substitution and the command substitution the variable substitution is using $ as unavailable now as the marker for the wherever you find like $ on variable name it get substitute the realign the commands themselves are its identified the command with the [ ] essentially.

So then it, it substitutes these actual steps whenever it finds as for markets and then evaluate sound better. And then one other key thing that we saw a horrible boom Tcl recognized on the words essentially we know that actually the word distinguished by very daybreak concealing so they break with wide spaces or ; but there are some exceptions to the rule one is the double quote spoons breaks curly braces prevents breaks and substitutions as well as \ form and code to the special character.

And thus eliminate the word separation, so word array, so this is another key thing that we saw, and then in the commands actually the first world all overconfident model of arguments. And

only the function of line means to the arguments and everything is done in single pass tokenizing and substitution. So something to understand in expr is except around basic scan from left to right it uses the same arithmetic evaluation techniques only thing notice essentially warm over it encounters any argument with you for floating point from that point onwards.
Everything is computed in 4.0 so this can pass some serious headaches a tech program and I want you to be aware of this when you write your own ticket products and then we also learnt about lists and how to use those so we will revisit again in the next lecture so I think that is pretty much it and very much thank you for listening.