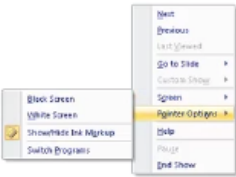


(Refer Slide Time: 00:01)

More about Procedures

◆ Variable-length argument lists:

```
proc sum args {  
  set s 0  
  foreach i $args {  
    incr s $i  
  }  
  return $s  
}  
  
sum 1 2 3 4 5  
⇒ 15  
sum  
⇒ 0
```



Okay, I saw one thing that I want to emphasize that we talked about the previous slide was also regarding this variable argument list variable ink document list, so once you want to actually supply an argument list with the variable number of arguments then we can use this standard our keyword this keyword actually describes this is already setting the language for a variable length of arguments.

So it is read as a list essentially and then essentially now you can go through the list to understand of each of the arguments and then parse them separately so here do the example that before previously, so here the proc name is sum which actually returns the sum of set of integers and then we do not specify like all the integers and separated with this specify args as the keyword.

So inside this product we set this to zero and then we can actually parse this args as a list so for each I dollar args we just increment s with whatever the value of dollar I so and then we return that the sum basic key as so here a simple example, sum 1 2 3 4 5 the answer will be 15 because every time is intimated by that much so it is 1 + 2 + 3+4+5 if you say sum 10 20 15 then you the answer will be 45 , and then if you get this done without any arguments then it returns the zero itself.

And that is fine in zero here so this kind of using a variable length is very useful because now we can extend this concept to many other things so let us see one way to extend this concept.

(Refer Slide Time: 02: 34)

Positional and Non-Positional Arguments

- ◆ The same concept explained previously can be used to convert positional arguments to non-positional arguments
- ◆ Positional arguments
 - Proc foo {{arg1}}{def} arg2} {body}
 - Foo arg1 arg2
- ◆ Non-positional arguments
 - Foo -arg1 arg1 -arg2 arg2
- ◆ How can we write this proc?

foo -arg2 value
- arg1 value

So oftentimes you see in procedures you do not want the positional arguments and you want non positional arguments what do I mean by that so a positional argument for example, for this Proc means foo we have an argument one argument two and it needs to be exactly specified when we call this procedure if you miss it then it would not be proper for example, in this one exactly like. I purposely wrote this way so I hope you remember this format acting or you specify a pawn argument and another value inside the places that is actually the default if you do not specify then that is what it takes the value of so here say like I mean R 1 R 2 you need to specify exactly in this order otherwise the this function would say, so now let us look at the non position like one person argument we can say like I mean the same thing like in any order only thing is we depend with arg one - arg one and then give one. And the -r 2 to give the -r2 to you can write like foo R - R - and then something some value and there -r1 value 2 this is also possible because now it does not matter where you specific values in your beginning with to the arg it knows like what exactly to put in so how do we write these kind of procedure ,this is one of the curious examples, essentially what you will be seeing more and more in real world problems.
(Refer Slide Time: 04:44)

A Simple Example

```

proc sum args {
  set s 0
  set j 0
  foreach i $args {
    if ($i == "-one") {
      incr s $args([incr j 1])
    }
  }
  return $s
}

```

Handwritten notes:
 - one (no) / two
 - three (1) / two (2) / one (3)
 - three (1) / two (2) / one (3)

So the way to do this kind of procedure is as follows, so here a simple example, the same home procedure some with the keyword arguments now this is as you know it is a list and this list is essentially like now be like this basically that is arg one up to all these things are there so. I am assuming that you will be specifying like - 1 and the first number and then - 2 second number things like that now you can actually inside.

The program you can look for those keywords and then you can mark those things exactly as is here it is a simple increment of s with whatever the next value is but this could be a complex program right here and then once you write this program here and then same thing for each of these dollar I values like if it is two then do something three then do something think, things like that once you have that and then you return actually now you do not have to specify like.

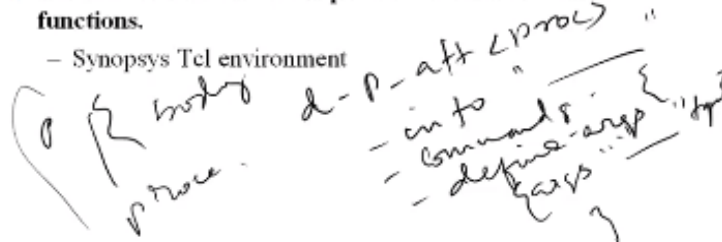
I mean if you have it three numbers you can specify in any order - 1 and then first number - 2 second number - 3 third number this is one way to specify or you can just say like - three the first number - 2 second number and then - 1 the third number so basically like I am in this kind of a method actually the procedure call itself or the way that you specify the arguments does not matter you can get actually.

The results with any kind of order at sum, so that is mainly like I mean one of the things with this positional and non positional so you can use this array concept to convert the positional arguments into non positional arguments.

(Refer Slide Time: 06:59)

Adding Documentation to Procedures

- ◆ Using similar concept, you can add documentation to procedures
 - Help commands
 - Manual pages
- ◆ Some environments could provide libraries for these functions.
 - Synopsys Tcl environment



And one more thing is here instead of using if you can also use the switch command. I hope you remember the switch command the switch command is essentially your also like doing similar kind of thing. Where you can use like if it is equal into multiple if come at so another application for these kind of this args concept is you can use this to also extend the procedure to add documentation for example the help commands and also the manual pages you can easily extend and add those commands into the into your TCL procedures.

So again I think these are like very useful and a helpful way to do this way yeah some TCL environments provide like libraries that already contain these extensions to add the documentation the help commands for example in the synopsis TCL environment you get like couple of things one is the parse arg arguments and also procedure or proc attributes, so once you set the protest boots that pretty much you can use it for describing your help and some kind of pages and then the proc that that one is also like fairly simple to use.

So essentially the way to use will be just give like there are a few fields that you need to describe on the procedural attributes which can be considered as attributes and then that is what will be used to process or to define that attributes so mainly like, I mean you can specify the procedure the it is the command it is also called like define attributes and you can specify your particular name followed by - info and then that info will have so basically like the define procedure attributes.

And then it will specify you can specify the procedure name and then you can specify - info and then you can specify a string with actually contains the information to display and then you can

also like do some command groups and these command groups can be the arguments that you are defining and then here you can also define on you then this one will be like. I mean what kind of arguments who have that is arg and then basically like I mean what if the help command that will to show and then what kind of type of argument to this and then this you can describe for all of them and then once you have this is probably the group , so once you define this whole thing this can be used as the documentation, so once anyone like I mean wants to write the right and the help procedure or any type of arguments or any type of help procedure . Or the final pages they can use this command to describe that and then it will be it will be together with the procedure usually you have the procedure you write the procedure basically body and then at the end you will add the procedure attributes and then once you have the attributes then now the TCL interpreter can actually take the whole thing and then parse it and then if you are just calling the procedure - L it will display the help information from the different pockets attributes. So I think this is this is another way like I mean another e-learning thing so I want to also like give you the basic rules in TCL. (Refer Slide Time: 12:23)

Tcl Rules - #1

- ◆ White space separates command name and each arguments
- ◆ Newlines and semicolons separate commands

Handwritten diagram illustrating Tcl command syntax. It shows 'command' followed by 'arg 1' and 'arg 2' separated by spaces. Below, it shows 'Command' followed by a list of arguments 'arg 1', 'arg 2', and 'arg n' separated by semicolons on the same line, and another 'Command' followed by a list of arguments 'arg 1', 'arg 2', and 'arg n' separated by newlines. A large curly brace groups the second set of arguments.

And then basically like I mean this will give you a good framework when you start doing your typical cram so rule number one this we saw earlier whitespace separates command name and each of the arguments ,so when we do the commands essentially like it is command then followed by whitespace then r1 space r 2 etc. So this is one thing that I want you to understand and then the other thing is also like. I mean new lines and the semicolon separate problems so you have command one it is all of these arguments and then command to will be in the next line or you can say command one and

then ; ended command 2, so now the question is if one of these arguments itself has a blank space then how do we or a new line how do we how do you write that so in that case essentially like.

I mean we need to use the grouping so the grouping is essentially through the curly braces or through the braces so now let us see like ,I mean how that is done.

(Refer Slide Time: 14:28)

Tcl Rule #2

- ◆ Grouping
- ◆ {} – Prevents all substitutions by Tcl Interpreter.
- ◆ Groups all characters until matching } is found.
- ◆ Will the following work?

```
while { [expr $i + $j] <=10} {body}
```

Why? Why not?

So that comes to the TCL rule number two the rule number two is on grouping so number one is this curly braces this prevents all substitutions by TCL interpreter so these two are very important events substitutions and TCL interpreter, so as the you recall the way that TCL is done is we have the typical interpreter and then followed by parcel.

So all the commands first go through the detail interpreter it is interpreted into like multiple words and then that is sent to the parser and then the parser actually understands what the command is and then what is needed for that command. So one thing to notice is essentially like a move all this once you start the group with the brace it moves all characters until a matching another brace is found.

And it is also like brilliant also like now let us look at this particular command while we started the brace and you know like the while has two arguments. This is argument one argument to the one is usually a condition when the mark 2 is the body. So this condition we say basically wishing move dollar I plus dollar J less than 10 then do this while this is less than 10 keep doing this body do you think this program will work A why or why not. So here, you look at this one basically there is a command expect here.

And since we are using the curly brace this prevents all the comments of situation so that means that once this is passed to the while you know while, while is passed into the mystical interpreter

it does not return anything it just returns the poor thing. So this will not work you need to actually like do this separately and then put it inside. So that is the one of the rules so now let us go to you rule number 3.
(Refer Slide Time: 17:00)

Tcl Rule #3

- ◆ “ “ used for weak grouping
- ◆ Substitutions allowed by Tcl interpreter
- ◆ Groups all characters until another “ is found
- ◆ No nesting of quotes.

Rule number three is essentially using the codes for weak grouping here substitutions are allowed by the TCL interpreter it loops all characters. Until a matching code is found one thing to notice there is no next thing of course so you cannot say like code and start something like. I mean \$ X + then you got another start and then for y and then in this, this does not work because up to here it is taken as one and then this coat and coat is taken and that is wrong. So it always left with nothing. So the nesting does not work in these kinds of rules so this is another key thing that I want you to keep in mind. Now let us look at the next rule.
(Refer Slide Time: 18:05)

Tcl Rule#5

- ◆ **Backslash substitution**
 - Used to quote special characters
- ◆ **While {foo} {**
 - Command arg1 \
 - Arg 2
 - Return foo1
 - }

The next rule is the \ substitution this is used to coat special characters essentially. So here, you can see one interesting program while few. We have a command are 1 or 2 returned and here, you see that basically like the new line is actually escaped with this, this \ so \ and then you go. So what do you think this will happen? So this when it executes this command this, this \ and the newline is replaced with Mr. blank space so this is as good as saying command are one array exact.

So that is how this will work and then you can see that actually like or then we start the curly braces we do not use the escape newline essentially the, the reason for this is once you start the, the braces the command interpreter groups everything include inside that inside the two braces or basically like a building until the matching bases form. So that means that basically these are automatically grouped into a single argument. So we do not have to really do all that and then here the tricky part is basically like the command interpreter me group everything and then sends it to the parser and the parser first of all understands the command.

And basically like that if we know that it has two arguments completed command and then it also sends this thing back to the interpreter which is basically a written form to return value. So this is the back slash substitution now the rule number five is the, the variable substitution we saw this also like in the first lecture of second lecture actually the dollar causes the variable Substitution and here the dollar and the variable name are replaced by the value of variable. So whenever you say like dollar X and then it, it finds what is the value of this X and then it replaces dollar X with the actual value.

(Refer Slide Time: 20:59)

Tcl Rule #6

- ◆ Command substitution
- ◆ [] delimit the command
- ◆ Nesting of commands should be OK
- ◆ Everything including the [] is replaced by the command return value

And then finally the rule numbers six in the commands education rule here, essentially. We use the [] to delimit the command one thing to note in command substitution is nesting of commands are okay and then everything including the [] is replaced by the command return value. So this is another Keating key learning from this one. So you can use this kind of method to actually do the command substitution and the open-ended decent. So in summary essentially like you need to understand these six rules so in this module actually.

We went through quite a bit of TCL mainly we started describing the procedures I think you have the basic understanding of the procedure there. We define the keyword proc and then we used the procedure name followed by the list of arguments and then, then the procedure body. So there are three parables, parable for the proc or command itself and then once the procedure are defined. Then they just behave like built-in commands for example you can say like sub 1 3 will return value 2 and again or stop one of searching.

So this is the sub 1 is the procedure name and then the 3 is the argument here, and then sub 1 of 3 the terms to does it basically like subtract 3 from whatever I mean 1 from the target value then the other thing that. We also saw was arguments can have default values for example here decrement X and then Y, Y is the another argument for usually like x1 of Y but the default value of that Y is 1. So if you do not specify only like went 5 then the answer will be 4. Say decrement sorry Tcr 5 for then the answer is 1.

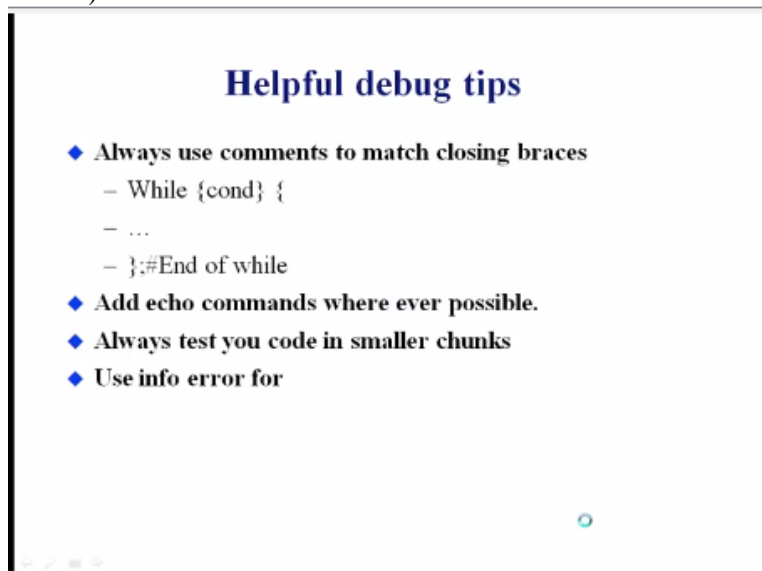
So because like the default value for the second wall paper is always 1 in this particular truck. So this way like you can define multiple default conditions. So this is the first thing that we saw here, we saw like a quick example of the problem and how we can it will work or not work and then we started defining the scope because we need to set the scope and then. If you use the

variable inside a procedure you need to have explicitly the scope from outside. So the way to give the scope to the variable is by using this for keyword global. And then the global needs to be specified in psychopath ego in order to get the transfer from outside into, into the procedure one thing to note is that if you do not set this X and you are just using the global X it still does not work you need to the, the global itself does not set anything for X so that is another key thing so you need to have that variable set outside. Then you can use global to transfer that value into the procedure. So then essentially so that that is what is shown here is not fitting for your though yet by specifying like \$ warning does not mean that basically people get transferred you need to make sure that it is defensible. And then you can use that inside this and then there are two other semantics until Allah Akbar and level they can they help you to some the, the names by a reference. So here, one big example is implement one name up or one, one name and this is war, so what you can do is basically like now and then the and then we all actually accept the war to \$ war + 1. So now what happens is actually like and you can exit communicate this back to another place there this war is getting pod and then this one here, denotes the level basically up to there you can actually like send to send this variable. So here, war is only sent to the procedure that is outside of this and then if you want it to be even higher then you need to increment this one to two and things like that. So that is the level naming essentially and then up level is the similar kind of other thing only for used for setting scripts from one level to the next level. (Refer Slide Time: 26:53)

Positional and Non-Positional Arguments

- ◆ The same concept explained previously can be used to convert positional arguments to non-positional arguments
- ◆ Positional arguments
 - Proc foo {{arg1 def} arg2} {body}
 - Foo arg1 arg2
- ◆ Non-positional arguments
 - Foo -arg1 arg1 -arg2 arg2
- ◆ How can we write this proc?

Brandon we saw this variable-length procedure and then we also noted like how? We can use this for the both you know defining the positional arguments as well as non positional arguments. So how do we do this, this is something that we saw and in this simple example we will actually go into details so it and then, then we also like had some simple purpose to give all those things. And we call like how we can talk movies videos and then. We also covered some a little bit of error handling as to what kind of what is the error info and things like that essentially. So we can use this error info to and then we can keep track of what is going on this again. So with that I am also like I want to give you like some coding guidelines as to what you can do. (Refer Slide Time: 28:19)



So here, a couple of things that you want to make sure one is always used comments to match clothing bases. So because the braces are quite important, So like and you close any base make sure that you put that comment out there. So that whoever is reading they know that they are to close it comments the other.

One is need to add you can add the echo commands they are ever possible. So this also helps to provide some feedback and that by that people will understand what you are trying to do and what is actually printed out in those sometimes in the commands then make sure that you test your code in small chunks and then.

Always use this error rate for both is what we saw in the debugging procedure and then finally we also saw like the six rules for effective TCL coding the rule number one is to understand the concept of whitespace with separate command name and all the arguments and then the new lines and the call ; separate commands. So this is the very simple one this understands this one

and then the rule number 2 is that grouping essentially the curly braces which prevents all the substitutions by TCL interpreter.

So here the key thing is also this TCL interpreter that I mentioned and then for the start grouping essentially use the quotes or sound grouping this actually substitutes all the variables or essentially an but there is no nesting of the codes also the rule number five is to use the \ substitution so how that works that is the keeping here, you can review.

This one and then actually so this is the rule number four which should be before then rule number five is the variable substitution which is by using the \$ here, the variable is whenever we specify the variable name with a dollar that variable name is replaced by the value available.

And then finally the last rule is for the financial situation it is using the square brackets and the square brackets delimit the command here. We can nest multiple commands essentially with means that you can start like there is experimenter and then start another one cool and then to online then close all of them.

So this is allowed this is this thing is allowed so I think this basically as most of the basic typical programming is all covered in these modules we will look at some of the advanced concepts from T, T in the next lecture and then after that we will start looking into TK as the next one okay thank you very much bye.