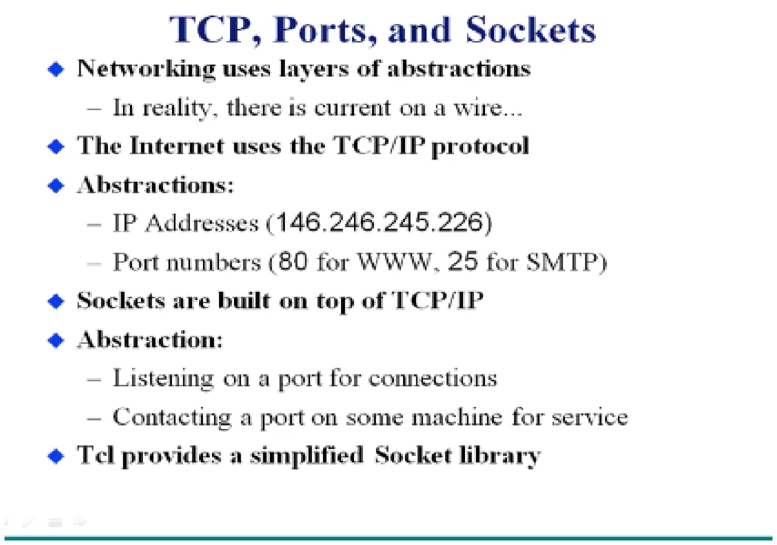
Hi everyone again welcome to this lecture of the LPS course programming class, we will be looking at the TCL programming we will continue with the TCL programming today. I will go through some of the some more advanced topics on the TCL before that, I just wanted to mention a quick recap of what we covered in the last class the last lecture we continued our string substitution.

I mean the strings and lists from various data structures we started looking at the procedures and as you know the procedure for have like three main sections actually if you are counting the number of arguments there are five and then the first, it starts with the keyword proc then followed by the name of the procedure then some set of terms then it goes into a body essentially actually the for the four main parts and then when you and most of the vertical scripts are written with these procedures.

(Refer Slide Time: 01:34)

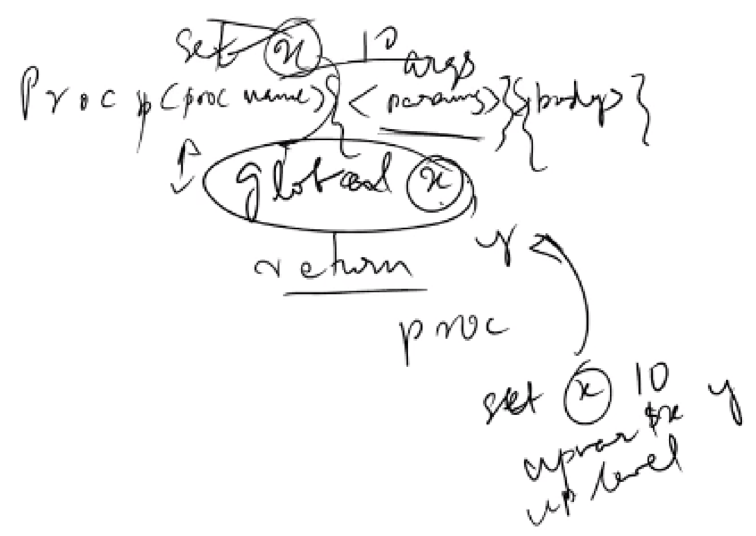


## TCP, Ports, and Sockets

- **Networking uses layers of abstractions**
  - In reality, there is current on a wire...
- **The Internet uses the TCP/IP protocol**
- **Abstractions:**
  - IP Addresses (146.246.245.226)
  - Port numbers (80 for WWW, 25 for SMTP)
- **Sockets are built on top of TCP/IP**
- **Abstraction:**
  - Listening on a port for connections
  - Contacting a port on some machine for service
- **Tcl provides a simplified Socket library**

So it is basically started that is it proc.

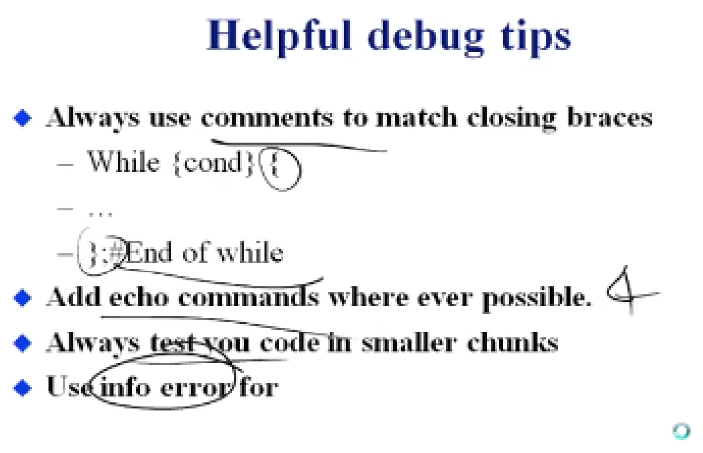
(Refer Slide Time: 01:45)



And then you are with proc name followed by pants and then followed by the body as you know like I mean you need a blank space here this you have can put a parenthesis and this is in parenthesis . I mean sorry the braces the double bass is saying is not it so and then the return value typically is can specify the return at the end of the procedure or if you do not have anything return the last evaluated function ,that output will become the output of the proc and then we also look at how do we specify the four patterns are the arguments what if we want to do it with the we have like mini number of patterns a variable.
List variable length parameters then we can use the keyword ARGS which kind of it is stores in a list of all the values so every time you call the proc the proc the number of arguments can be different and it will take it we also looked at some of the scope related issues which is why when you use the variable how do we determine the scope, which is inside the proc and if you are setting it outside like for set X 10 then now if you want the mix to be used inside the proc what will it be so to get this X into the proc.
We use this command called GLOB which is essentially the I mean actually the GLOB global the global command actually gets this value and global X basically confirms that actually the X that is defined here can that value can be used inside as low again point notice the global X command itself is not setting any value to the value as to the variable X that means to be set outside its only make sure that whatever is it off side can be read inside and then we have like a couple of other things.

We want to communicate a particular value of the variable that means that sailing you want to edit this one basically before let us find this and then say let us say the proc you are saying like

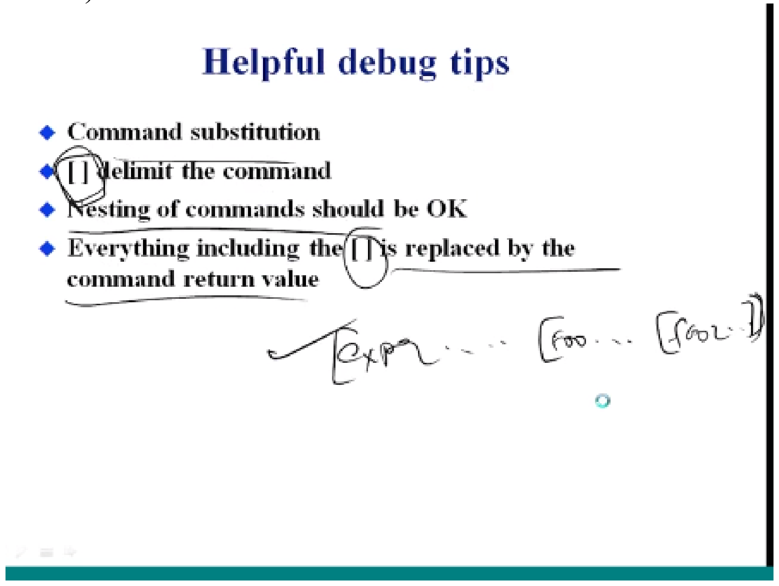
set X 10 and then this X wants to be visible outside the proc the way to communicate is through in using the upward and then though you can specify saying that my upward X is actually or $ X is whichever value in the previous one say here it is actually Y then small so this 10 whatever the changes that brought back to the Y after it is excluding procedure.
Same thing like I mean we also have more proc level.
(Refer Slide Time: 05:41)



## Helpful debug tips

- Always use comments to match closing braces
  - While {cond} {
  - …
  - }; #End of while
- Add echo commands where ever possible.
- Always test you code in smaller chunks
- Use info error for

(Refer Slide Time: 05:43)

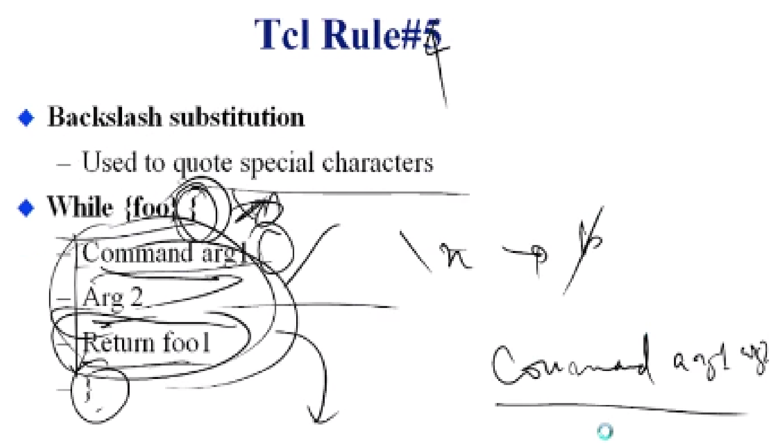


## Helpful debug tips

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

[expr … [foo … [foo2 …]]

(Refer Slide Time: 05:44)

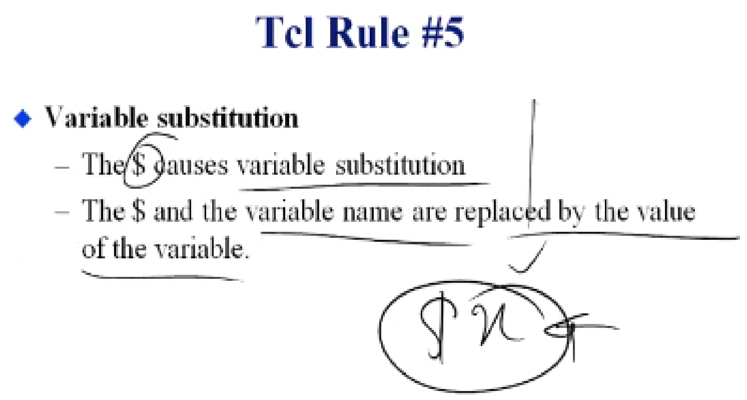


## Tcl Rule#5

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

\n → b
Command arg1 arg

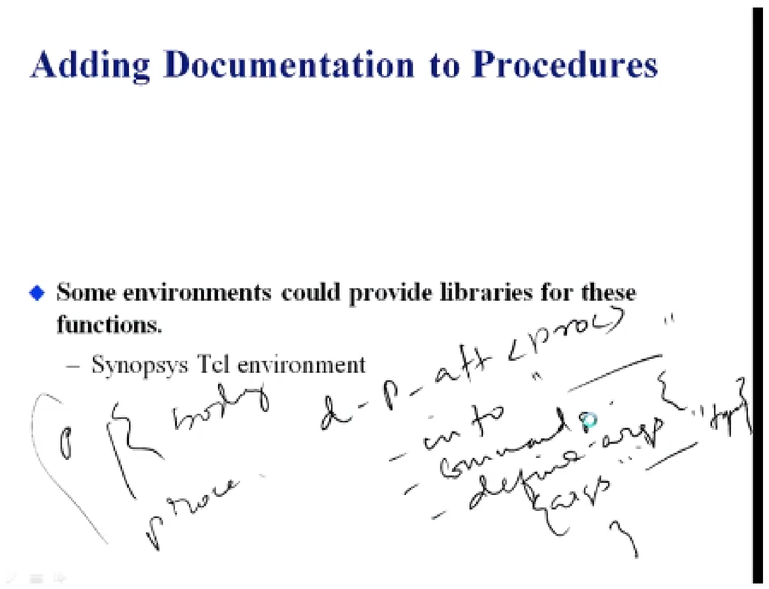
(Refer Slide Time: 05:45)



## Tcl Rule #5

- **Variable substitution**
  - The $ causes variable substitution
  - The $ and the variable name are replaced by the value of the variable.

(Refer Slide Time: 05:49)

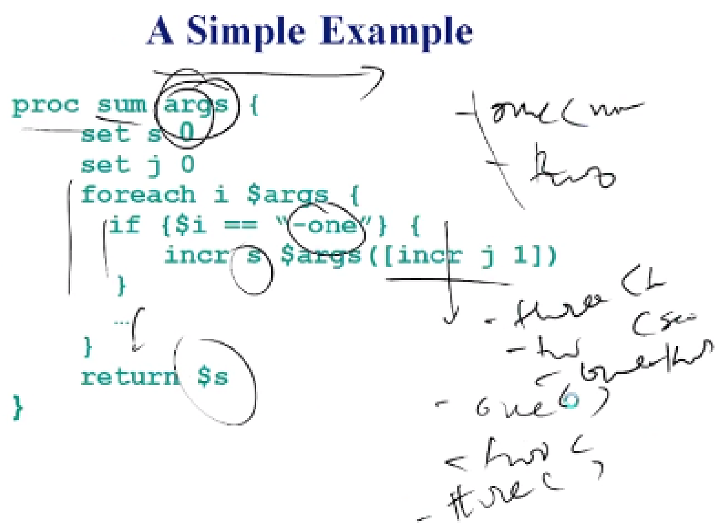


## Adding Documentation to Procedures

- Some environments could provide libraries for these functions.
  - Synopsys Tcl environment

d-p-att &lt;proc&gt;
- info
- command p
- define-args
{args
}
proc

(Refer Slide Time: 05:51)

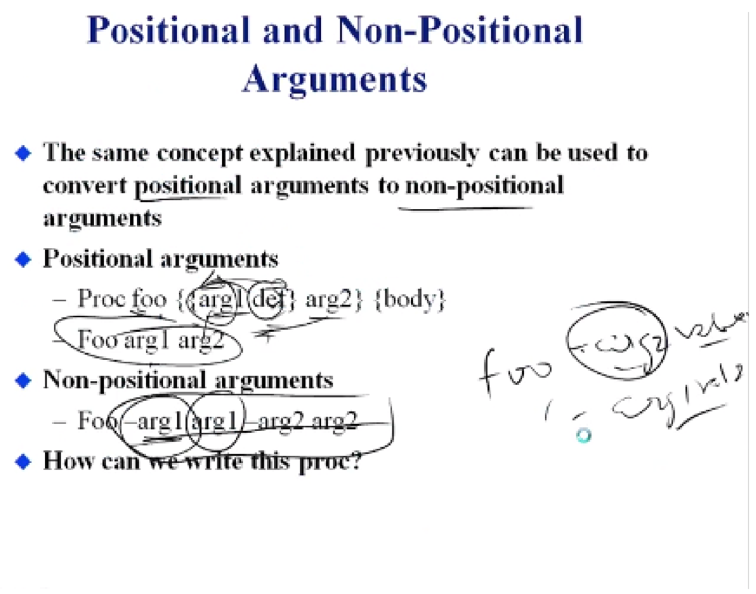


## A Simple Example

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

-one
-two
-three
-two
-one
-two
-three

(Refer Slide Time: 05:51)

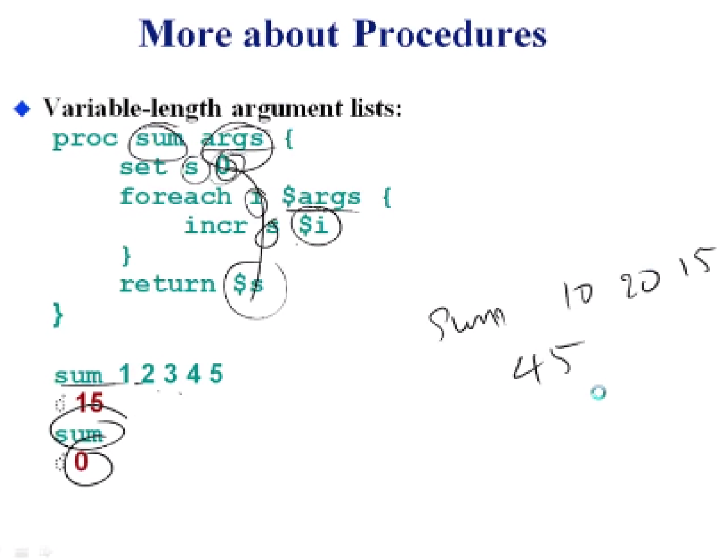


## Positional and Non-Positional Arguments

- 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 val

(Refer Slide Time: 05:52)



## More about Procedures

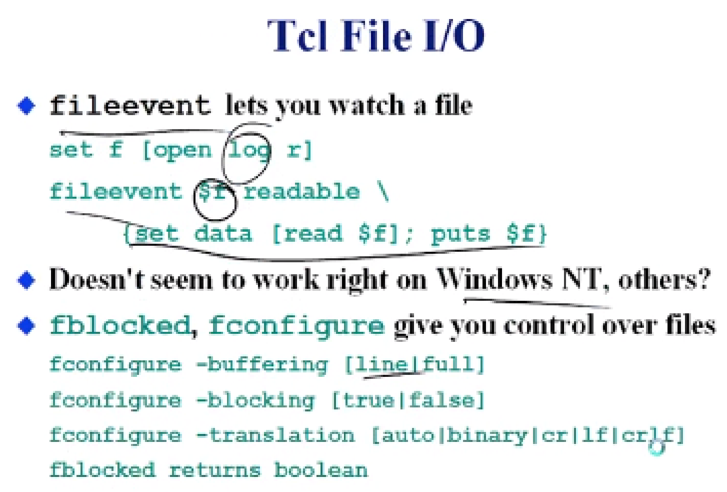
- Variable-length argument lists:

```
proc sum args {
     set s 0
     foreach i $args {
          incr s $i
     }
     return $s
}

sum 1 2 3 4 5
15
sum
0
```

sum 10 20 15
45

(Refer Slide Time: 05:52)



## Tcl File I/O

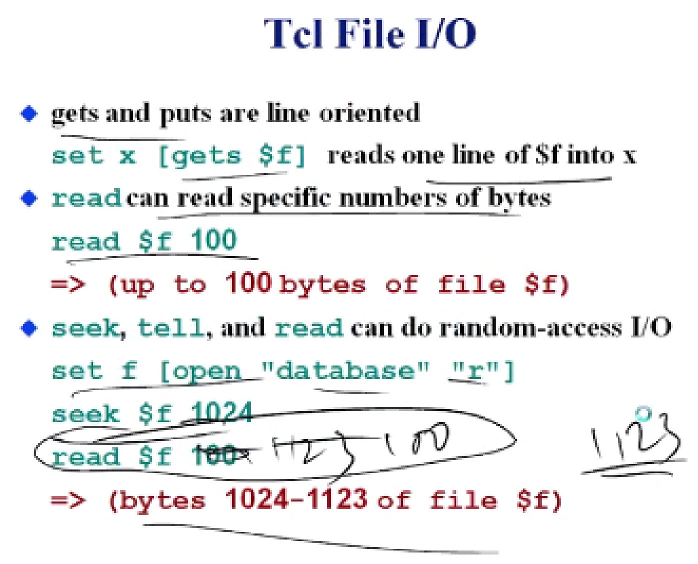
- `fileevent` lets you watch a file

```
set f [open log r]
fileevent $f readable \
    {set data [read $f]; puts $f}
```

- Doesn't seem to work right on Windows NT, others?
- `fblocked`, `fconfigure` give you control over files

```
fconfigure -buffering [line|full]
fconfigure -blocking [true|false]
fconfigure -translation [auto|binary|cr|lf|crlf]
fblocked returns boolean
```

(Refer Slide Time: 05:53)



## Tcl File I/O

- gets and puts are line oriented

`set x [gets $f]` reads one line of $f into x

- `read` can read specific numbers of bytes

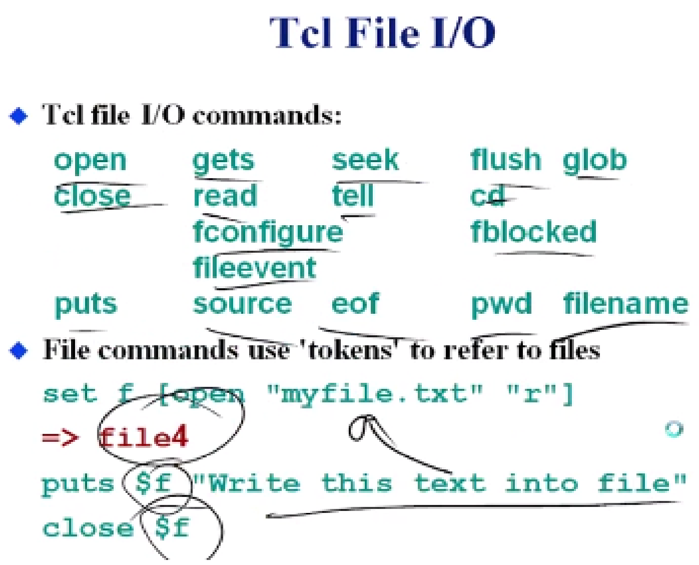
```
read $f 100
=> (up to 100 bytes of file $f)
```

- `seek`, `tell`, and `read` can do random-access I/O

```
set f [open "database" "r"]
seek $f 1024
read $f ~~100~~ 1123 100
=> (bytes 1024-1123 of file $f)
```

1123

(Refer Slide Time: 05:54)



## Tcl File I/O

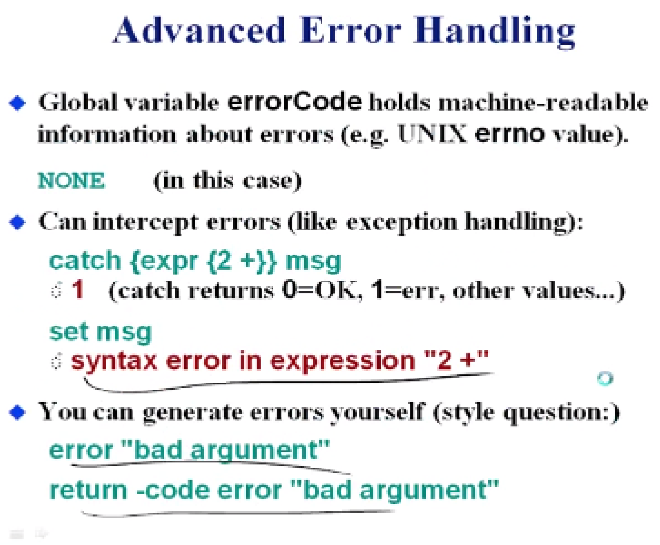
- Tcl file I/O commands:

| | | | | |
|---|---|---|---|---|
| open | gets | seek | flush | glob |
| close | read | tell | cd | |
| | fconfigure | | fblocked | |
| | fileevent | | | |
| puts | source | eof | pwd | filename |

- File commands use 'tokens' to refer to files

```
set f [open "myfile.txt" "r"]
=> file4
puts $f "Write this text into file"
close $f
```

(Refer Slide Time: 05:55)



## Advanced Error Handling

- Global variable errorCode holds machine-readable information about errors (e.g. UNIX errno value).

  NONE (in this case)

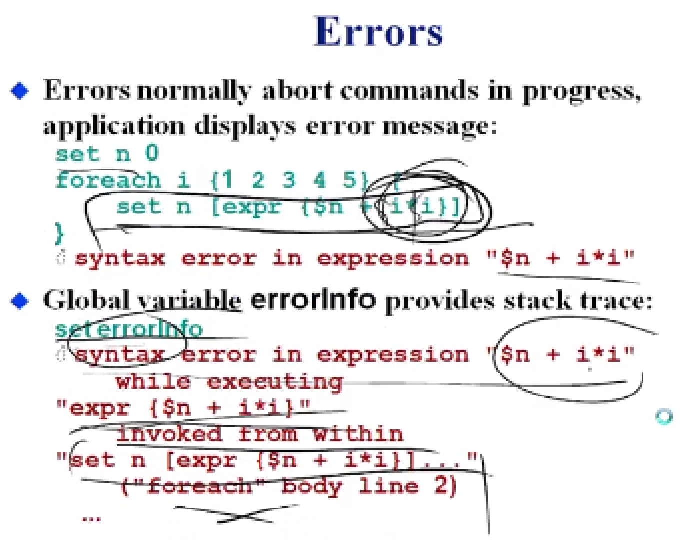
- Can intercept errors (like exception handling):

```
catch {expr {2 +}} msg
⇒ 1  (catch returns 0=OK, 1=err, other values...)
set msg
⇒ syntax error in expression "2 +"
```

- You can generate errors yourself (style question:)

```
error "bad argument"
return -code error "bad argument"
```

(Refer Slide Time: 05:56)



## Errors

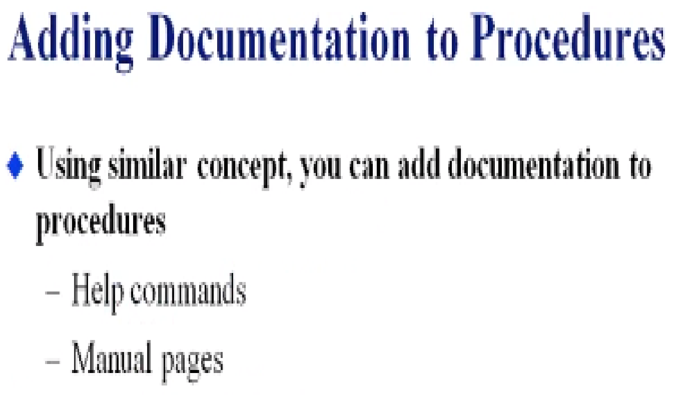
- Errors normally abort commands in progress, application displays error message:

```
set n 0
foreach i {1 2 3 4 5} {
    set n [expr {$n + i*i}]
}
syntax error in expression "$n + i*i"
```

- Global variable errorInfo provides stack trace:

```
set errorInfo
syntax error in expression "$n + i*i"
    while executing
"expr {$n + i*i}"
    invoked from within
"set n [expr {$n + i*i}]..."
    ("foreach" body line 2)
  ...
```

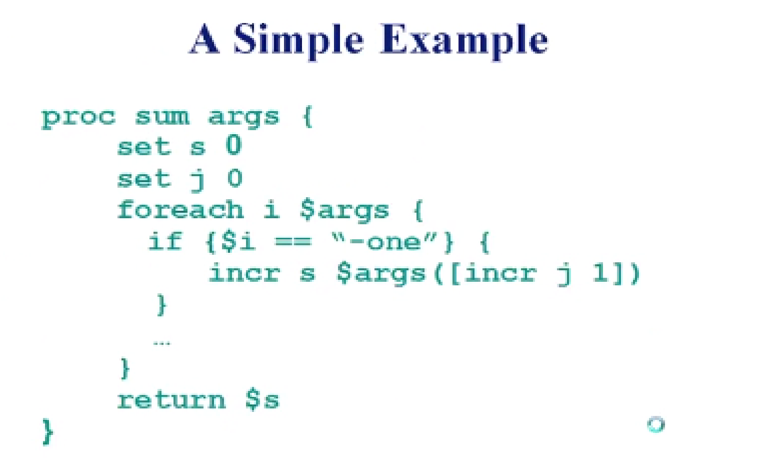
(Refer Slide Time: 05:56)



## Adding Documentation to Procedures

- Using similar concept, you can add documentation to procedures
  - Help commands
  - Manual pages

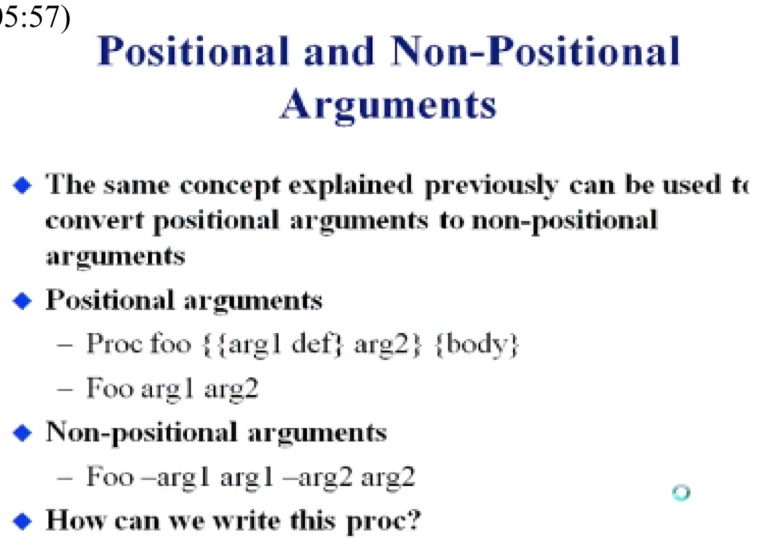
(Refer Slide Time: 05:57)



## 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
}
```

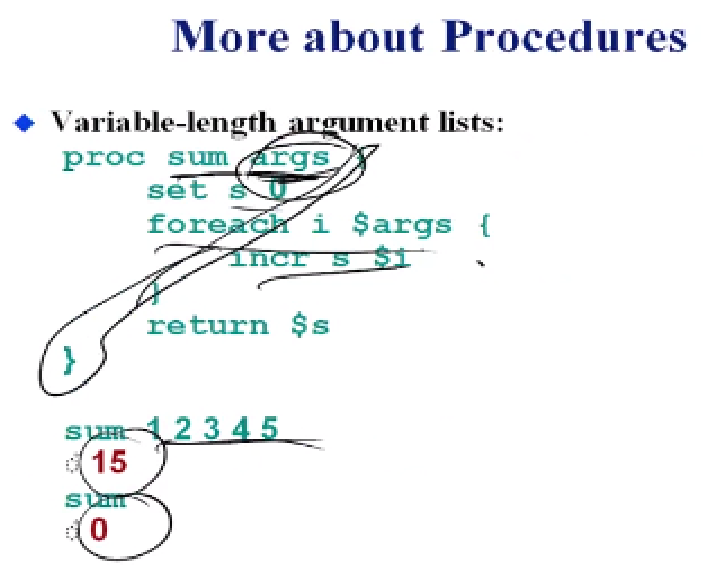
(Refer Slide Time: 05:57)



## Positional and Non-Positional Arguments

- The same concept explained previously can be used tc 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?

(Refer Slide Time: 05:59)



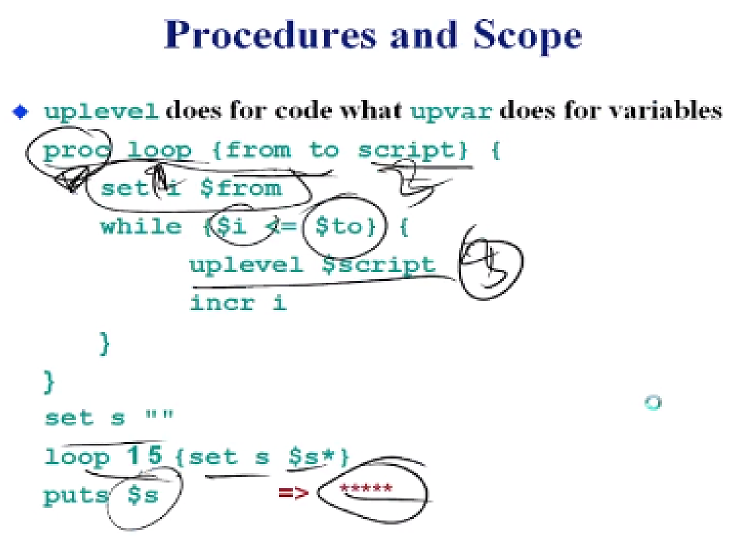
## More about Procedures

- 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
```

(Refer Slide Time: 06:00)



## Procedures and Scope

- uplevel does for code what upvar does for variables

```
proc loop {from to script} {
    set i $from
    while {$i <= $to} {
        uplevel $script
        incr i
    }
}
set s ""
loop 1 5 {set s $s*}
puts $s                =>  *****
```

Yeah so the up level we saw that basically it is for to communicate a script back to the top level that this is up level so we thought that then we also saw from above the main rules of TCL. I mentioned like six rules you six rules of TCL. I hope you remember all those things would be but just in case like I mean so the mean to think the variable substituted this is $2 and then the command substation this is using the square brackets.

So these are two main rules and then other than that the rule about the interpreter versus the parser is important interpreter takes in a command and then basically splits it into words based on blank space and you know like the blank space for words inside determine and then this is file sent to the parcel the parser passes this command and again it knows like how many variables need to be there or the given now command and then it processes according and this can be like

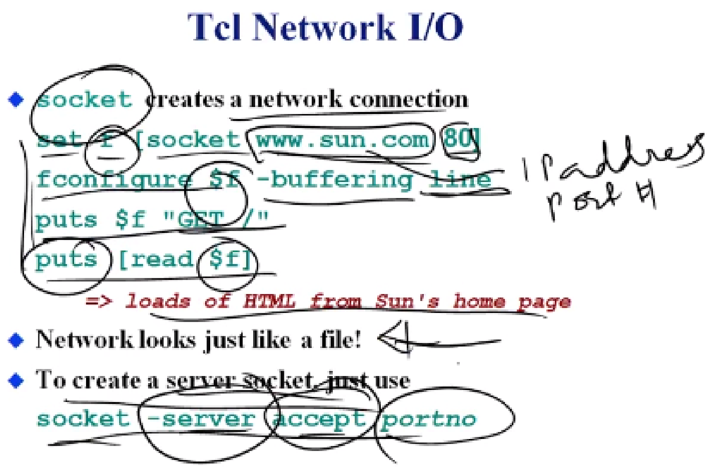
recursively called meaning like the command again once again it sends the thing if it has a another command substitution maybe you can send it back to the interpreter and then it can it again like goes through so the recursion is possible in the commands so the key thing is okay.
So then the blank space and then for separating out the commands we need to use the semicolon or a new line so these are all the these are variable various rules that we saw and then the remaining rules are pretty much exceptions to these rules meaning like if you want to group the stock grouping in the hard grouping the hard grouping with the braces the braces and then the stock opening is through coats and the difference between this and this is any kind of special characters can substitute.
That inside this whereas here it is not substituted and then it is TCL you so we end through all these different things basically in the last lecture. I hope it is a write down like a medial much more capable using TCL and then probably like I mean you can use for TCL for programming your various assignments and a couple of other things also that we mentioned was so how to actually use some of these commands to provide the documentation or health commands things like that within the park itself and then also like.
I mean we can we can there are some guidelines for making sure that the programs are correct from the head book we talked about all those things so today we will be talking about the TCP port sockets basically like how to proceed with TCL and then how do we communicate between various servers using TCL and then finally I have one example that we will kind of will go through it there are some concepts that we learnt in that example we will see how those concepts.
How these rules the six rules that I mentioned now also like explain where, so we will see like I mean how those four things play out and how does it work our restrictive work in a big comment so let us look at first of all some of the networking items, so we saw in the very first lectures and saw the whole networking works we know that the late there are layers of abstraction we defined the seven layers form or defining the network in a PCP scenario.
It could be just five layers but essentially and then the lowest layer is simply the five the physical layer which is essentially it is a wire that this that communicates between I mean our sense the real packets of real data between two computers and then we also saw the how the Internet is using this TCP/IP protocol which is shown here and then we saw in fact even the IP addresses basically idea.
This is the lawn values essentially it is like a 32-bit number divided into like various tell essentially, so and then we also knew that actually how to code all these values what is the subnet versus the main network address and then there are also like port numbers in BO

decoding usually we use the port number 80 for any of the web traffic and then 25 is used for the mail traffic so these are all like physical ports opening in various servers and then we can use this to actually build pipes through which we can communicate and traditionally these pipes are known as sockets.

So socket build on top of some BPPV we actually saw some of the fastest in earlier versions as how we can use UNIX file system so the sockets essentially the main function of the socket is it listens on a port for any connections and then it contacting port on sum machine or service so those are the two things that the sockets ,can do one is if it is in the current machine it listens for the port or any connections that is being itself established and if you are communicating to another machine.

Where your machine is more like a client and the other businesses services then it basically port that machine the ones the socket is established it pulls that machine for service ,so these are the two main things for what you may want to look into and then the TCL already has a built in simplified and socket library ,so let us look at how TCL works in this scenario.

(Refer Slide Time: 14:11)



## Tcl Network I/O

- `socket` creates a network connection

```
set f [socket www.sun.com 80]
fconfigure $f -buffering line
puts $f "GET /"
puts [read $f]
```

=> *loads of HTML from Sun's home page*

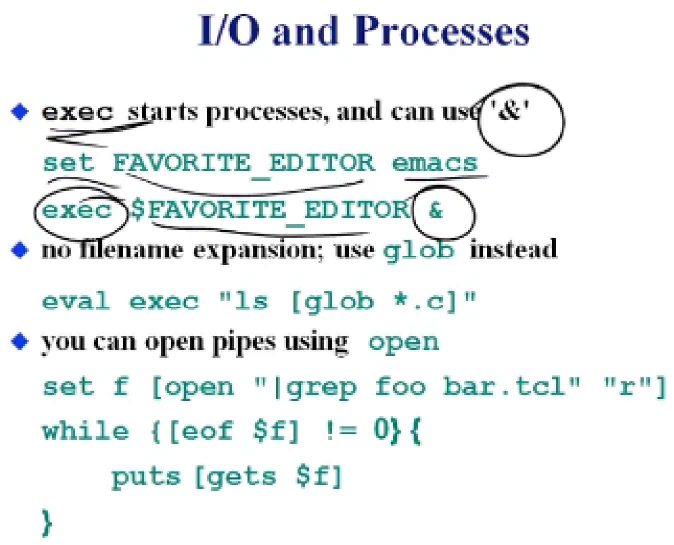
- Network looks just like a file!
- To create a server socket, just use

```
socket -server accept portno
```

IP address
port #

So TCL has a built in procedure all the socket and which creates this little connection here there is a quick example, set F socket and then the address and then the port number so now tell me like I mean how many arguments the socket demands have, I said there are only two here which is one is the IP address and then the other one is this port number actually there is also an this additional argument that you cannot satisfy with you except essentially and then when you specify the except actually turn a particular machine into a server and so essentially like.

I mean this means that actually the server socket so your machine is now going to act like a server and it will it will service the requests from other times , so let us continue this one for the other one is like the Fconfigure whatever this socket that we got and then basically like we put some buffering on it and then the buffering is for line so every line the first line when you get it like.
I mean the person we saw was so that the buffer and then the subsequent lines basically would be this line is so in the buffer right and then basically we just use get and then basically we just read the socket and then write outside into this, so now if you if you run this code essentially like having this load of HTML from sun's home page, now anyway the server the socket is basically like socket - server except and then which port number that you want to open.
This socket 80 again we talked about this in the unit vector the Inter network just looks like a large file so that is why you can actually work these kinds of things.
(Refer Slide Time: 16:59)



## I/O and Processes

- exec starts processes, and can use '&'

```
set FAVORITE_EDITOR emacs
exec $FAVORITE_EDITOR &
```

- no filename expansion; use glob instead

```
eval exec "ls [glob *.c]"
```

- you can open pipes using open

```
set f [open "|grep foo bar.tcl" "r"]
while {[eof $f] != 0}{
    puts [gets $f]
}
```

So in order to execute certain processes we can have like two things one is we can use an exec command which is shown here and then we can also use the & so here we are starting exec the basically setting the favorability exec and then executing a favorite editor with & so this actually opens up a new window with a new exec, and it would not obstruct the main process that is unbelievable and then the filename expansion itself like I mean we do not have to do it we can use the glob command to do it.
So here a quick example is from LS glob start up see that gives you all the C programs essentially and then if you want to open pipes.