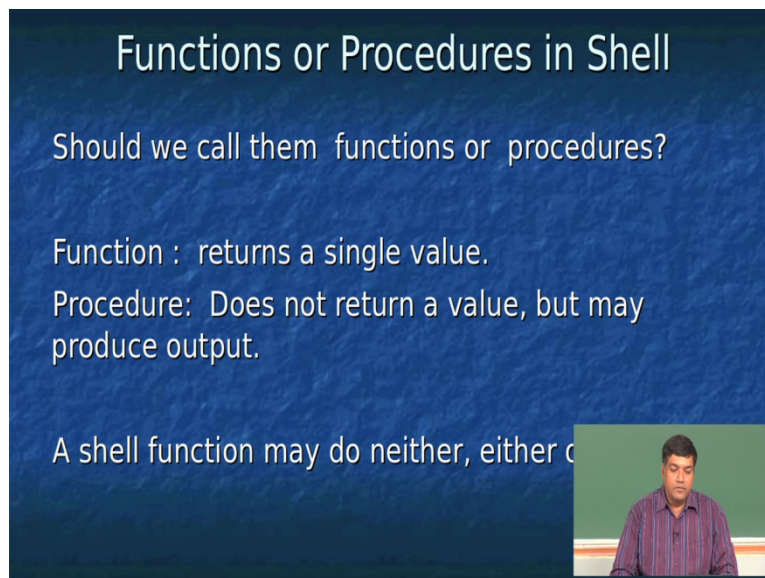


Information Security 3
Mod Sri M J Shankar Raman,
Consultant of Computer Science and Engineering
Indian Institute of Technology Madras
Module 33
Shell Functions

Hi there in the last session we saw about three examples of some small shell scripts we also saw that its not only writing shell script, we should also do unit testing of the shell scripts and we should ensure that our code is very logical. Now this tells you the importance of writing commands in the code also. So were you able to debug the last shell script? If you have not please try to debug and try to identify why the problems? Now we will move on to the next topic which is how to write functions within a shell script?

(Refer Slide Time: 01:00)



Functions or Procedures in Shell

Should we call them functions or procedures?

Function : returns a single value.

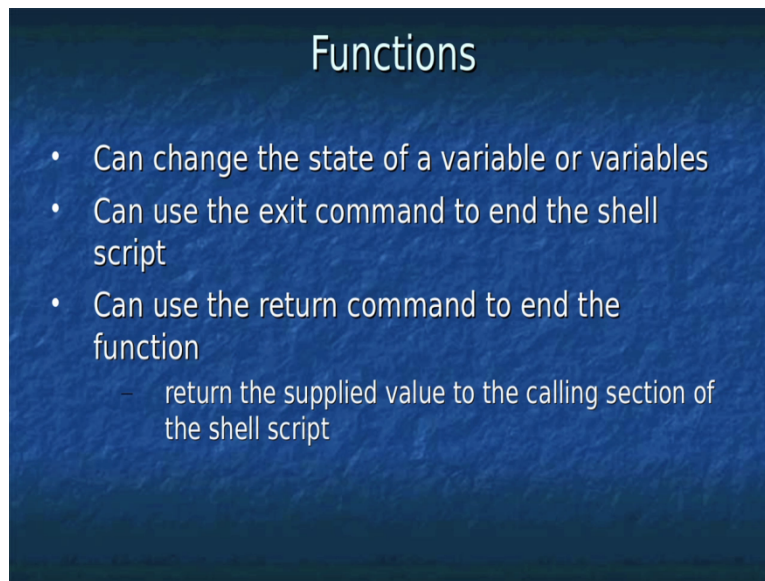
Procedure: Does not return a value, but may produce output.

A shell function may do neither, either or both.

The slide features a dark blue background with white text. In the bottom right corner, there is a small inset video showing a man in a striped shirt speaking.

Now there is confusion within whether we should call them as functions or we should call them as procedures? Formally in computer science a function is designed to be y is equal to $f(x)$ which means for every value of x you have a defined value of y . So in that sense a function returns a single value whereas a procedure does not return any value but may produce some sort of output, ok? The reason why we have termed it as shell functions or shell procedures or whatever it is and whatever you feel you can call them but we will in this course we will call them as functions and a shell function may do neither , either or both, so let us call them as functions for the time being.

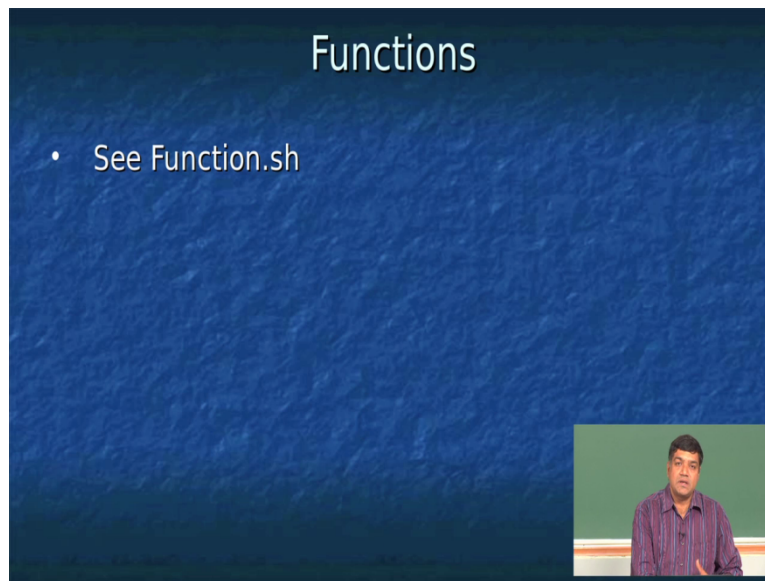
(Refer Slide Time: 02:00)



So what are all the things that a function a shell function can do? The first thing that a shell function can do is it can change the state of a variable or variables. Now this is true in any shell script,ok? So if you have a shell variable you can either add something to the variable or you can display the variable etc. Similarly a function can exit from a shell and third most importantly a function can also return values back and these values the returned values can be used by the calling section of the shell script. Now why is it necessary to have functions? It is always better that a long shell script are a very big shell script is divided into modules.

When I say divided into modules I donot mean that you take a thousand line shellscrip and every ten lines you make it as a function. Every function should do a particular job or must have a single responsibility. if you write a shell script a very large shell script which runs to thousands of lines of code its better you modularize the shell script and make it in the smaller components such that the person who is reading the shell script or is trying to use the shell script understands what you want to do. This is essentially one of the reason why you have a function in a shell script.

(Refer Slide Time: 03:45)



Now let us directly dive into an example and we will see the various ways of returning from a function. And how to use a return value of functions and so on. We will see this using some examples. So let us see this example function.sh

(Refer Slide Time: 04:08)

A terminal window with a yellow background. The code is as follows:

```
1 #!/bin/bash
2
3 add_two_numbers_ver1() {
4   if [ $# -ne 2 ]
5   then
6     echo "Usage - $0 Integer1 Integer2"
7     echo " Example: $0 5 7"
8     echo " will print "
9     echo " Sum of 5 and 7 is 12 "
10    exit 1
11  fi
12  echo "Sum of $1 and $2 is `expr $1 + $2`"
13
14 }
15
16 num1=0
17 num2=0
18 num3=0
19
20 add_two_numbers_ver1 5 5
21
22 add_two_numbers_ver2() {
23   if [ $# -ne 2 ]
```

So we will have two or three versions of the functions and we will be using the same function same example that we used in the last session. In the last session we were trying to add two integers. What we will do now is? We will the same shell script we will try to make it as a function. So a function definition in shell script consists of the function name ok this is the

function name, the function name that I have given is add underscore 2 underscore numbers underscore ver1 that is essentially I am calling this as version 1. And after this you close, you open and close the brace, curved brace, ok? And then you start with a curly brace. Now this is similar to the way you write a function and see in the sense that you use the same curly braces here, ok? And after this you can write your logic so if you remember the example that we had seen in the last class we had first tried to see whether the number of parameters ok that are passed to the function is correct ok? That is what we are checking at this line. And then if the parameters are not ok at an example of how to write a function.

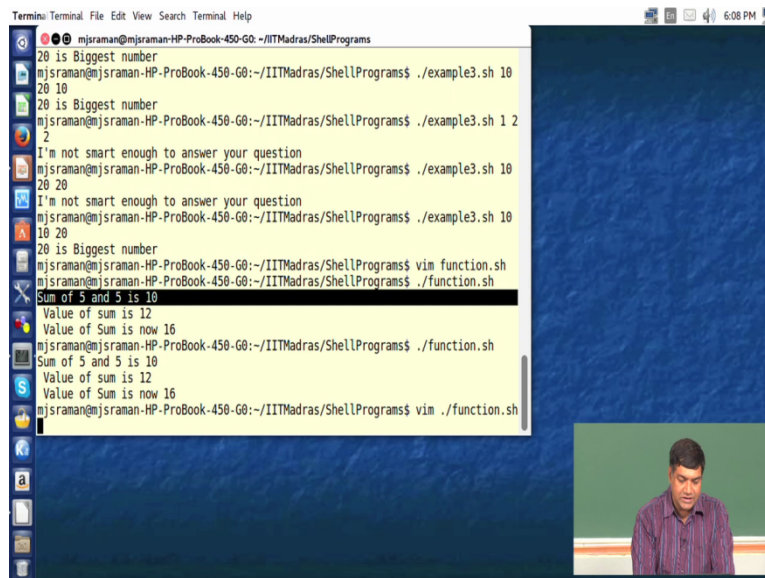
So in this case if you look at this at line number 3 we actually define a function every function has a function name. Unfortunately on a shell script you do not pass parameters within this curved braces what we actually do is we pass parameters as we usually pass the parameters in shell that is something like this. You give the command name and then after that give a space and then pass the parameters. The same convention would be followed for the functions also when you are calling the function. This is the function definition and in this definition of a function what we are doing is we are initially trying to see whether the number of input parameters is two. If it is not two then we give the usage example and if it is two then we calculate the value of the function, ok? The parameters that have passed and then printed, now this is known as a function definition and in line number 20 we have something known as the function call.

If you see the names it will be the same add underscore two underscore numbers underscore ver1 and next to it I am passing two parameters that is 5 and 5 which essentially means when this function when the shell scripts comes to line number 20 and it executes this line what it does, it calls this function so when it calls this function the first line that would be executed is it tries to check whether the number of parameters that you have passed is actually 2. If the input parameters is 2 then it actually calculates the value and prints this so let us see the first version whether it works correctly and if you look at number 1 number 2 number 3 these are used for the next other two versions of the function so let us not worry about line number 16, 17, and 18 currently what we will be worried about is line number 3 to 14 and then line number 20. Let us see how this works.

So what I expect is that first it should print some of dollar 1 plus and dollar 2 which is nothing but sum of 5 plus 5 is 10.

So what happens is that this is known as the function call where I use the function name and along with the function name I passed the parameters if you remember in the previous session instead of calling this as add underscore two underscore numbers underscore ver 1 we had call it as example two dot sh and in the same way we have passed the parameters in the same way. So the way you pass the parameters through the function call is similar to the way you pass parameters for invoking the shell scripts.

(Refer Slide Time: 08:14)



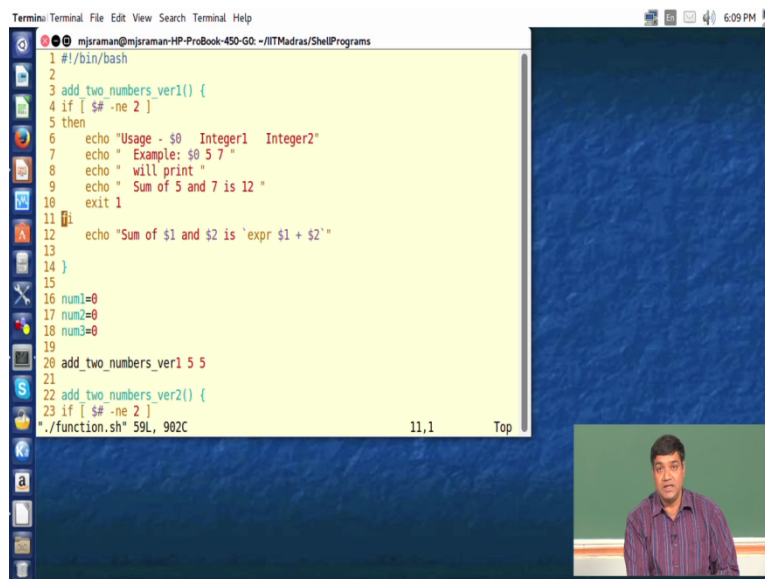
The screenshot shows a terminal window with the following content:

```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
20 is Biggest number
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./example3.sh 10
20 10
20 is Biggest number
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./example3.sh 1 2
2
I'm not smart enough to answer your question
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./example3.sh 10
20 20
I'm not smart enough to answer your question
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./example3.sh 10
10 20
20 is Biggest number
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim function.sh
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./function.sh
Sum of 5 and 5 is 10
Value of sum is 12
Value of Sum is now 16
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./function.sh
Sum of 5 and 5 is 10
Value of sum is 12
Value of Sum is now 16
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim ./function.sh
```

In the bottom right corner, there is a small video inset showing a man with dark hair, wearing a purple and white striped shirt, looking down.

So let us see what happens we will execute upto line 20 of course this is going to execute all the other lines also but we will see what happens when we come to line 20. So line 20 should print if the value is the sum of 5 plus 5 is 10. Let us look at this and if you see it prints the sum of 5 plus 5 is 10.

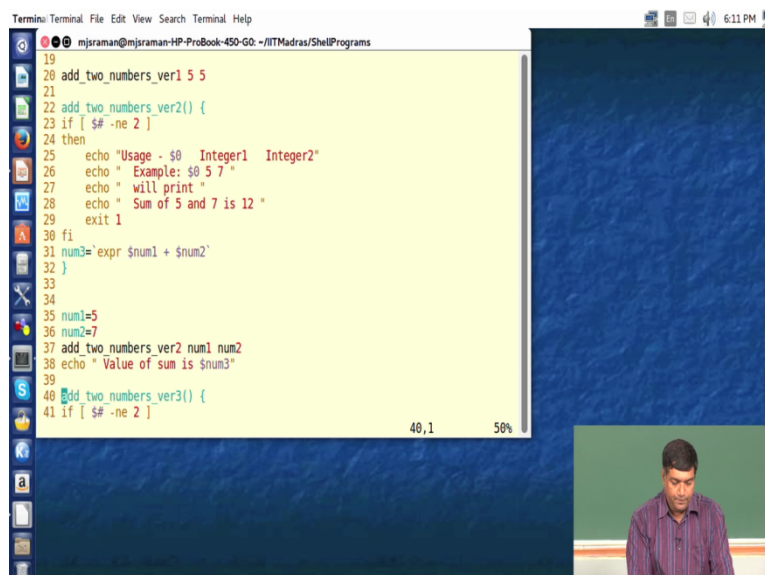
(Refer Slide Time: 08:40)



```
1 #!/bin/bash
2
3 add_two_numbers_ver1() {
4     if [ $# -ne 2 ]
5     then
6         echo "Usage - $0 Integer1 Integer2"
7         echo "Example: $0 5 7"
8         echo "will print"
9         echo "Sum of 5 and 7 is 12"
10        exit 1
11    fi
12    echo "Sum of $1 and $2 is `expr $1 + $2`"
13}
14
15num1=0
16num2=0
17num3=0
18
19add_two_numbers_ver1 5 5
20
21
22add_two_numbers_ver2() {
23    if [ $# -ne 2 ]
24    then
25        echo "Usage - $0 Integer1 Integer2"
26        echo "Example: $0 5 7"
27        echo "will print"
28        echo "Sum of 5 and 7 is 12"
29        exit 1
30    fi
31    num3=`expr $num1 + $num2`
32}
33
34num1=5
35num2=7
36
37add_two_numbers_ver2 num1 num2
38echo "Value of sum is $num3"
39
40add_two_numbers_ver3() {
41    if [ $# -ne 2 ]
```

Now what is unique about this function 1 it does not return any value what it does is? It just takes parameters input parameters and then prints the result of the input parameters. So in that sense should we really call it as a function so we should call it as a procedure so that is what we have seen as a definition previously.

(Refer Slide Time: 08:58)



```
19
20add_two_numbers_ver1 5 5
21
22add_two_numbers_ver2() {
23    if [ $# -ne 2 ]
24    then
25        echo "Usage - $0 Integer1 Integer2"
26        echo "Example: $0 5 7"
27        echo "will print"
28        echo "Sum of 5 and 7 is 12"
29        exit 1
30    fi
31    num3=`expr $num1 + $num2`
32}
33
34num1=5
35num2=7
36
37add_two_numbers_ver2 num1 num2
38echo "Value of sum is $num3"
39
40add_two_numbers_ver3() {
41    if [ $# -ne 2 ]
```

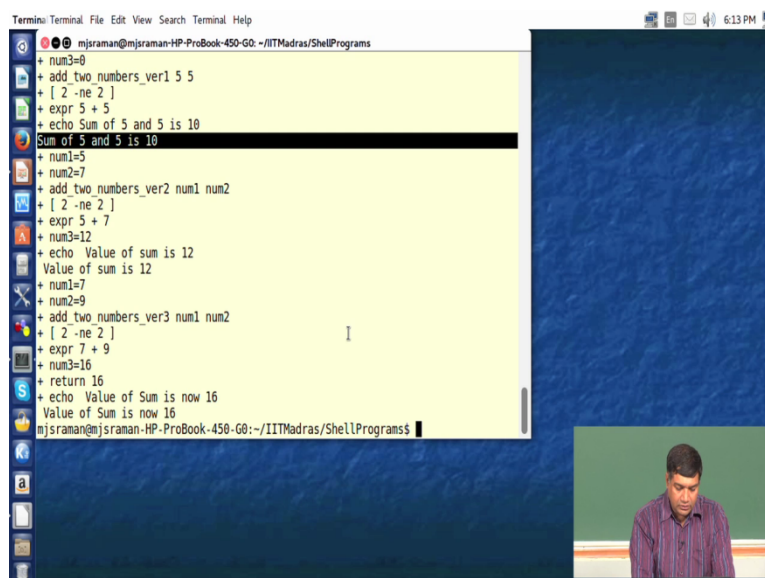
Let us come back and let us look at the second version of the function, ok? So here I am using the same name add two numbers accept that I change it as version 2 and the difference here in this is the way I call this function if you look at this function if you look at line

number 31, what I had done is I had declared variables in line number 16,17 and 18 num 1, num 2 and num 3; and what I do is in line number 35 I actually give num 1 is equal to 5 and num 2 is equal to 7 and you see one of the things is before you use the function you should define it in shell, ok? So that is one of the conventions that you should follow otherwise, first thing is it confuses the user if you use something like this without defining it,ok?

So what I do is I actually have this definition of the function which is add underscore two underscore numbers underscore version 2 and in this case if you see we had initialize the variable number 3 to 0 and we are changing the value of number 3 inside the function, this is known as side effects such kind of programming is extremely dangerous but I think we should leave such things the shell script allows you such things but be very careful when you use such kind of side effects.

Now in this case what we do is we just pass parameters num 1 and num 2 instead of the values 5 and 5 so what we do is we assign the value of num 1 to be 5 num 2 to be 7 and then we pass the parameters as num 1 and num 2 and we do the calculation inside the function and but we prove the result outside the function so this is another way of using a function, so in this case let us say the number that should be printed is 12, so what happens is when the shell comes to execution of this it actually calls this function, ok? And once it calls this function all the lines are executed and your value will be printed.

(Refer Slide Time: 11:18)



The image shows a terminal window with a yellow background and a blue desktop background. The terminal window title is "Terminal: File Edit View Search Terminal Help". The user is "mjsraman" and the host is "mjsraman-HP-ProBook-450-G0". The terminal shows the execution of a shell script with the following commands and output:

```
+ num3=0
+ add_two_numbers_ver1 5 5
+ [ 2 -ne 2 ]
+ expr 5 + 5
+ echo Sum of 5 and 5 is 10
Sum of 5 and 5 is 10
+ num1=5
+ num2=7
+ add_two_numbers_ver2 num1 num2
+ [ 2 -ne 2 ]
+ expr 5 + 7
+ num3=12
+ echo Value of sum is 12
Value of sum is 12
+ num1=7
+ num2=9
+ add_two_numbers_ver3 num1 num2
+ [ 2 -ne 2 ]
+ expr 7 + 9
+ num3=16
+ return 16
+ echo Value of Sum is now 16
Value of Sum is now 16
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
```

In the bottom right corner, there is a small video inset showing a man with dark hair, wearing a red and blue striped shirt, looking down.

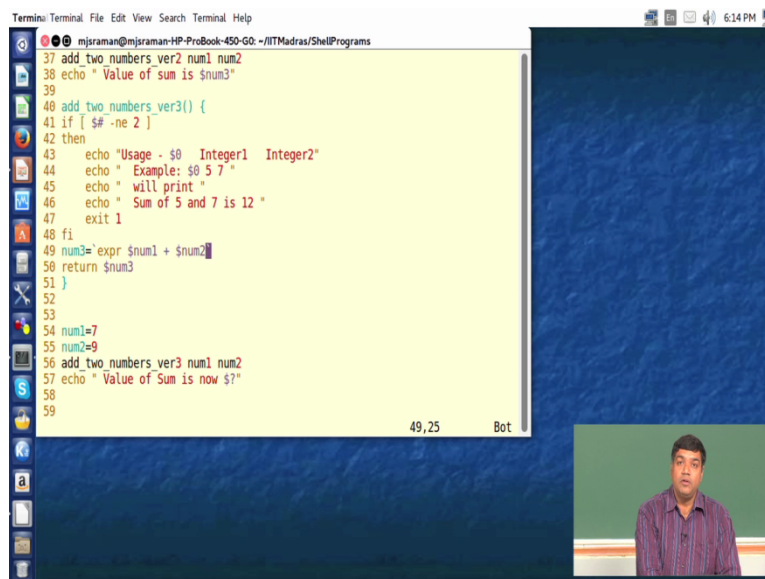
So let us try to see whether this happens, so the value that we have given as 7 and 5 so it should print a value of 12, so it says value of sum is 12 if you look at the second line that is printed, so what we can do is we can also see whether by using the debugging whether such a thing happens so if you look at this if you look at the place where we assign the values we assign the value of num 1 to be 5 and num 2 to be 7 then it calls the function call ok add two numbers version 2 and we are passing num 1 and num 2 then it compares whether the arguments number of arguments are 2

So then what it does is if it takes this num 1 takes the value of 5 here and num 2 takes the value of 7 so when I calculate the expression of 5 plus 7 it gives 12 which is assigned to num 3 and what we do is at this point the function ends execution and what we are happening is that the echo statement comes after this function but this occurs in the main shell script that we are using. So what happens in a shell script is whenever you do a function called and this is similar in the first case too.

See in the very first case where we had discussed giving add two numbers version 1 with 5 and 5 ok? what had happened was? if you look at this let us look at this so if you look at this we see that it tried to check whether there are two parameters and after this the function actually the shell script actually call the function call and inside the function call we were doing the addition of 5 plus 5 and then we echoed this saying 5 plus 5 is 10 which is from the shell script,ok?

So there are two aspects here one the shell script that calls the function and the other aspect the execution inside the function.

(Refer Slide Time: 13:22)

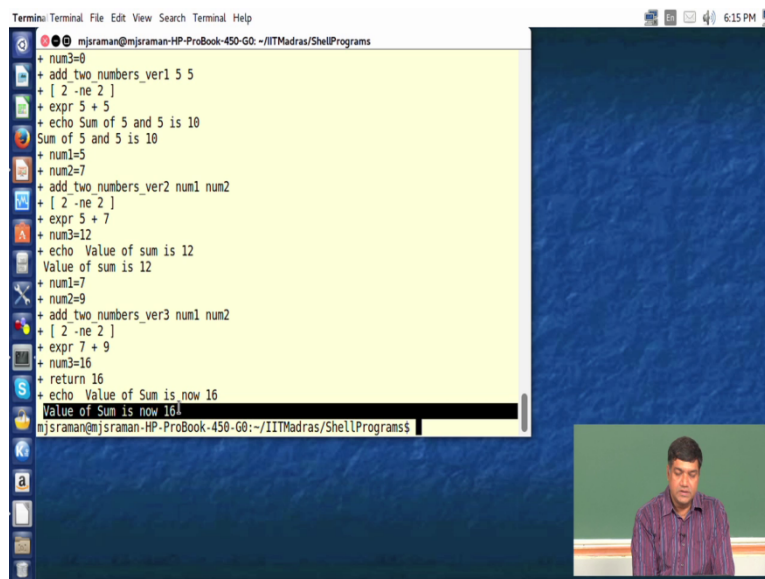


```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/ITMadras/ShellPrograms
37 add_two_numbers_ver2 num1 num2
38 echo " Value of sum is $num3"
39
40 add_two_numbers_ver3() {
41 if [ $# -ne 2 ]
42 then
43     echo "Usage - $0 Integer1 Integer2"
44     echo " Example: $0 5 7 "
45     echo " will print "
46     echo " Sum of 5 and 7 is 12 "
47     exit 1
48 fi
49 num3=`expr $num1 + $num2`
50 return $num3
51 }
52
53
54 num1=7
55 num2=9
56 add_two_numbers_ver3 num1 num2
57 echo " Value of Sum is now $"
58
59
49,25 Bot
```

Now let us take look at the next example so if you look at this third example what we are doing here is we are working on something which is slightly different if you look at the difference between the previous version of the function and this version so I told you you should not allow side effects side effects are extremely dangerous in programming so what I do is I actually if you look at this line number 50 I told you the definition of a function a function should return a value.

So in this case what we will do is we will actually declare the variable num 3 and return its value explicitly now this is a better method of coding rather than using side effects and once we returned this value what happens is that now if you remember in our previous session we had told that the return value of any function can be caught by using this variable dollar question mark, now that is exactly what we are doing right here since it returns a value of number 3 we are actually printing the value of number 3 using dollar question mark, so this is another way of returning a value from a function.

(Refer Slide Time: 14:49)




```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
+ num3=0
+ add_two_numbers_ver1 5 5
+ [ 2 -ne 2 ]
+ expr 5 + 5
+ echo Sum of 5 and 5 is 10
Sum of 5 and 5 is 10
+ num1=5
+ num2=7
+ add_two_numbers_ver2 num1 num2
+ [ 2 -ne 2 ]
+ expr 5 + 7
+ num3=12
+ echo Value of sum is 12
Value of sum is 12
+ num1=7
+ num2=9
+ add_two_numbers_ver3 num1 num2
+ [ 2 -ne 2 ]
+ expr 7 + 9
+ num3=16
+ return 16
+ echo Value of Sum is now 16
Value of Sum is now 16
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms
```

So if you look at this ok? And let us try to execute it before we move on to the other parts. So let us try to execute this function if you look at this we are passing the values as 7 and 9,ok? and then we are using add two numbers version 3 with the inputs of number 1 and number 2 which are nothing but 7 and 9, then it test for whether the operands for the number of parameters are ok then what it does is it adds 9 plus 7 which is 16 and it assigns to number 3 and then what we do is we return the 16 that is dollar number 3 is what we return and once we echo this the echo statement prints 16 here.

(Refer Slide Time: 15:30)

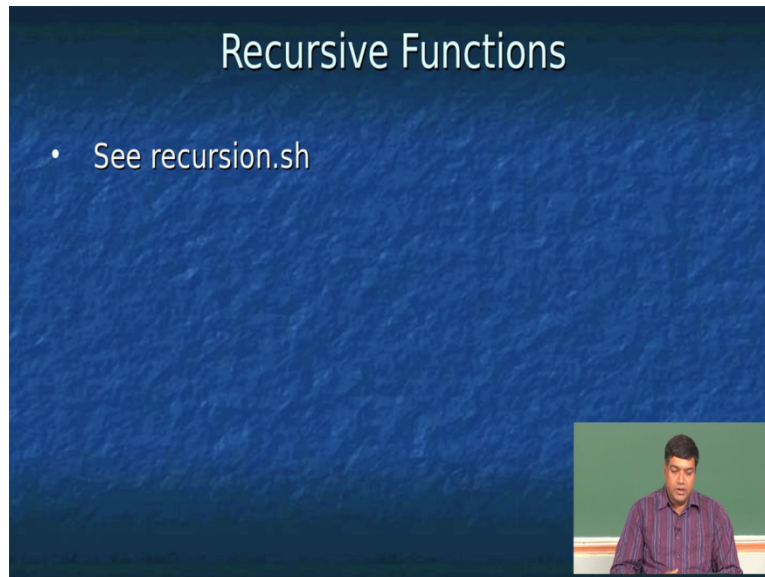
Functions

- Can change the state of a variable or variables
- Can use the exit command to end the shell script
- Can use the return command to end the function
 - return the supplied value to the calling section of the shell script



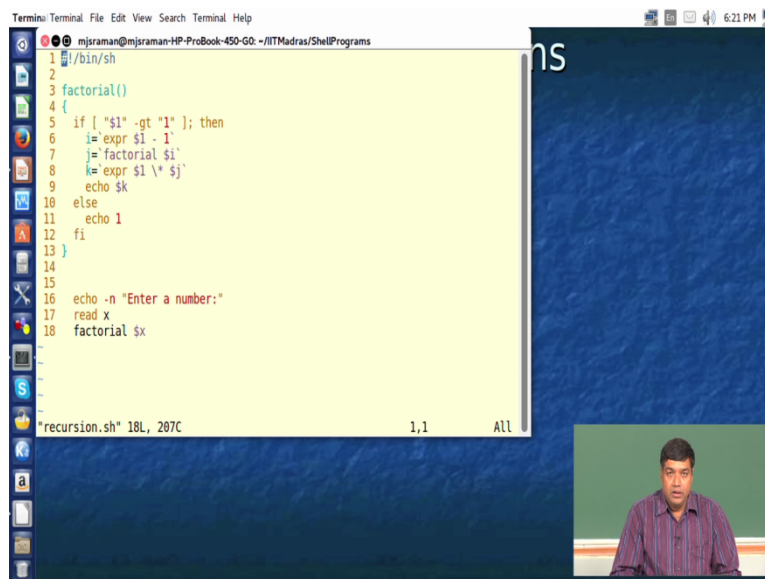
So in this way if you look at our presentation we have said that we can change the state of the variable we can use the exit command to end the shell script and we can use the return command to end the function so the return value is supplied to the calling section of the shell script is what we are printing.

(Refer Slide Time: 15:47)



Until now we have used a shell script which is called a function now is it possible that the function can call itself, yes such functions are called recursive functions. Now how are we going to use recursive functions well I mean we can only show some examples and how we apply it to your UNIX operating system depends on the task at hand. So what we are demonstrating here is the kind of facilities that are provided by the shell scripts in order to achieve a task these are all set of tools. How you use your tools is left to your intelligence, so what is recursion? A recursion informally can be called as a function calling itself. What we will do is? We will see how recursion operates and how we can write recursive programs using shell.

(Refer Slide Time: 16:37)



```
Terminal: Terminal File Edit View Search Terminal Help
1 #!/bin/sh
2
3 factorial()
4 {
5     if [ "$1" -gt "1" ]; then
6         i=expr $1 - 1
7         j=factorial $i
8         k=expr $1 \\* $j
9         echo $k
10    else
11        echo 1
12    fi
13 }
14
15
16 echo -n "Enter a number:"
17 read x
18 factorial $x

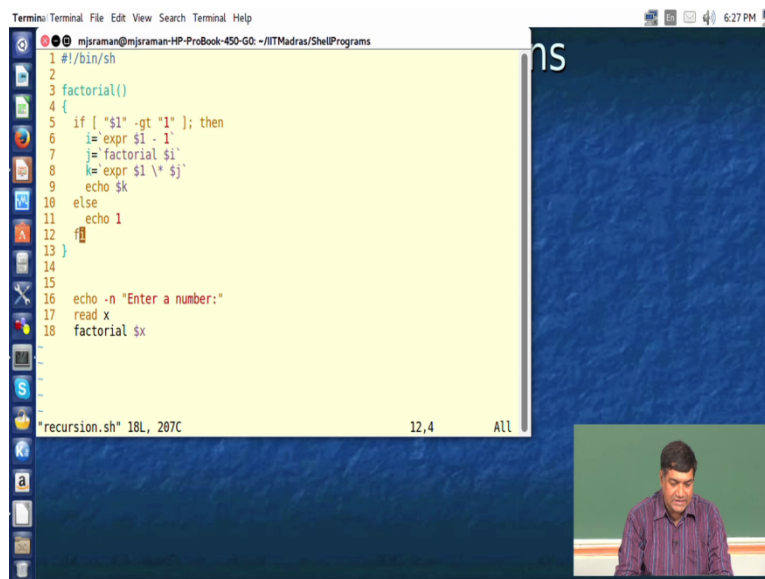
"recursion.sh" 18L, 207C 1,1 All
```

Let us look at this function called recursion dot sh. Now what we are doing is we are writing a very small shell program to get the factorial of a number. Of course there are very many efficient algorithms to get a factorial of a number but what we are trying to do is we are trying to just demonstrate the calculation of recursion so in this case what we do is we first define the function called factorial. I hope you are very familiar with the way we call functions, ok?

So what I do is when this factorial function actually expects a number to be put so that is what it says and then you can call this function with a given number. Now what happens once it comes to this line it actually calls the factorial function, ok? And then if you dollar x becomes dollar 1 ok? And the given number is greater than 1 because if the given number is 1 ok? Then I say the factorial is 1 and come out of this program. So once this else part is executed then finally there is nothing therefore this program comes out.

So where is this recursion used? If you remember we have a command in LINUX which is LS space minus l capital R this capital R actually tells you that you have to go a directory within directory sometimes find out list out all the files so this is an example of recursion where we have a system calls itself but only thing that is different is the parameter that you pass.

(Refer Slide Time: 18:17)



```
Terminal: Terminal File Edit View Search Terminal Help
mjrsaman@mjsraman-HP-ProBook-450-G0: ~/ITMadras/ShellPrograms
1 #!/bin/sh
2
3 factorial()
4 {
5     if [ "$1" -gt "1" ]; then
6         i=expr $1 - 1
7         j=factorial $i
8         k=expr $1 \* $j
9         echo $k
10    else
11        echo 1
12    fi
13 }
14
15 echo -n "Enter a number:"
16 read x
17 factorial $x
```

recursion.sh 18L, 207C 12,4 All

ns

6:27 PM

Video inset: A man in a striped shirt speaking.

So similarly in this case in this example if you see what we do is we queue we have to call factorial which is a function that we have already defined and we understand that how to define a function in the previous few minutes we have understand that how to define a function. What we do here is we pass parameters to the function which here dollar x is the value that you read from the user here.

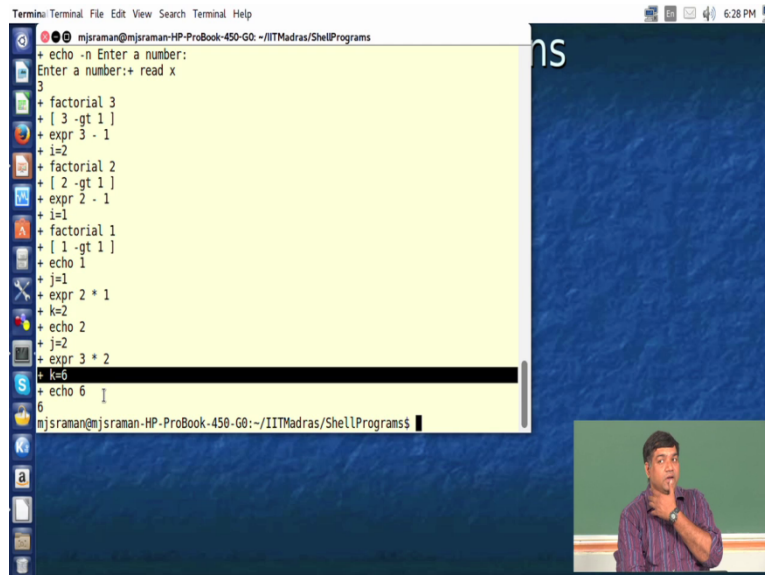
So if this value is 1 the logic says that if this value is 1 whatever is 1 then you have to return the value to the user if the value is greater than 1 then first thing I do is I just decrease the value by 1. So let us assume that I give a value of 2 then I give I make to be 2 minus 1 that is 1 and then inside I catch the factorial of 1 which means this function will again be called , ok? Once this function gets called then factorial of 1 is 1 ok and this value ok so J will become 1 so at that point in time so what we do is the next line gets executed , ok? And here what happens is this 1 gets multiplied by whatever number you have given which is 2 and then finally the result is presented as 2 so let us see how recursion works in this case.

What we are going to do is? First let us apply the value let try to give values of 1, 2, and 3 and then see how recursion works. So suppose I give factorial of 1. So what happens is that the dollar 1 here becomes 1 so 1 is greater than 1 is false therefore I execute echo is equal to 1 So what you see on the screen is if you print a value of 1.

Let us give a value of 2 so in this case what happens is the first line inside the if statement will reduce the value of 2 to 1 because either I will become 1 and then it will call factorial of

1 so once it call we know that the factorial of 1 will return a value of 1 which is j and so k will calculate the value of dollar 1 which is nothing but what is the value of dollar 1? is 1 into j ok j is nothing but the value that you have calculated here,ok? And what happens is that here it will calculate 2 into 1 which gives the value of 2 and then gets it.

(Refer Slide Time: 20:56)



```

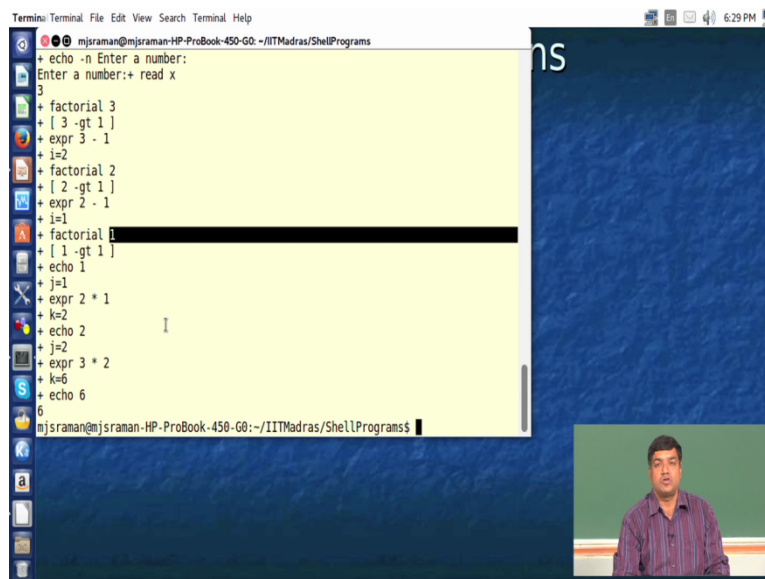
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
+ echo -n Enter a number:
Enter a number:+ read x
3
+ factorial 3
+ [ 3 -gt 1 ]
+ expr 3 - 1
+ i=2
+ factorial 2
+ [ 2 -gt 1 ]
+ expr 2 - 1
+ i=1
+ factorial 1
+ [ 1 -gt 1 ]
+ echo 1
+ j=1
+ expr 2 * 1
+ k=2
+ echo 2
+ j=2
+ expr 3 * 2
+ k=6
+ echo 6
6
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms

```

Let us see the execution here, so I put sh minus x recursion dot sh so I give the value of 2 so if you look at this it first calls factorial of 2 and then I gets a value 1 then it calls factorial of 1 ok? And then what happens is that then I returns a value of this factorial returns a value of 1 that means j returns the value of 1 then I multiply this 2 by 1 the previous value of 1 therefore the value becomes 2.

Let us try to do the same thing with the value of 3. So once I put a value of 3 what happens here is I call factorial of 3 then I becomes value of 2 then it calls factorial of 2 factorial of 2 actually calls factorial of 1 and factorial of 1 returns 1 which is multiplied by 2 ok? And factorial of 2 returns 2 which is again multiplied by 3 and finally you have three into two into 1 the value is 6 and finally it echoes a value of 6. Now you can see that you should spend a lot of time to debug this because recursion is slightly confusing it is easier to code but difficult to debug so recursion in general is a function calling itself on a problem of a smaller size.

(Refer Slide Time: 22:29)



```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
+ echo -n Enter a number:
Enter a number: + read x
3
+ factorial 3
+ [ 3 -gt 1 ]
+ expr 3 - 1
+ i=2
+ factorial 2
+ [ 2 -gt 1 ]
+ expr 2 - 1
+ i=1
+ factorial 1
+ [ 1 -gt 1 ]
+ echo 1
+ j=1
+ expr 2 * 1
+ k=2
+ echo 2
+ j=2
+ expr 3 * 2
+ k=6
+ echo 6
6
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
```

So in this case if you look at this factorial of 3 actually calls factorial of 2 so factorial of 2 is a smaller problem than factorial of 3 similarly factorial of 2 is calling factorial of 1. Now how this whole thing is handled is it makes use of a stack to store the local variables and then it tops out the stack once the recursion ends and all those things. Let us not worry about all those aspects of how a recursion is implemented what we will discuss is how to code programs using recursion this is when you how recursive code is the code will be slightly shorter the another way of this example of factorial is known as tail recursion and there is a proof that any tail recursion can be converted into a for loop or a do while loop and coding using do while loop is much more easier but we just wanted to to show you that a shell scripts allow you to define functions and b You can also call recursive functions.

You should try to practice recursive functions there are lot of examples like tree and something like that but probably you should try to write one or two recursive functions to gain expertise on or to see how the shell scripting works.

Thank You