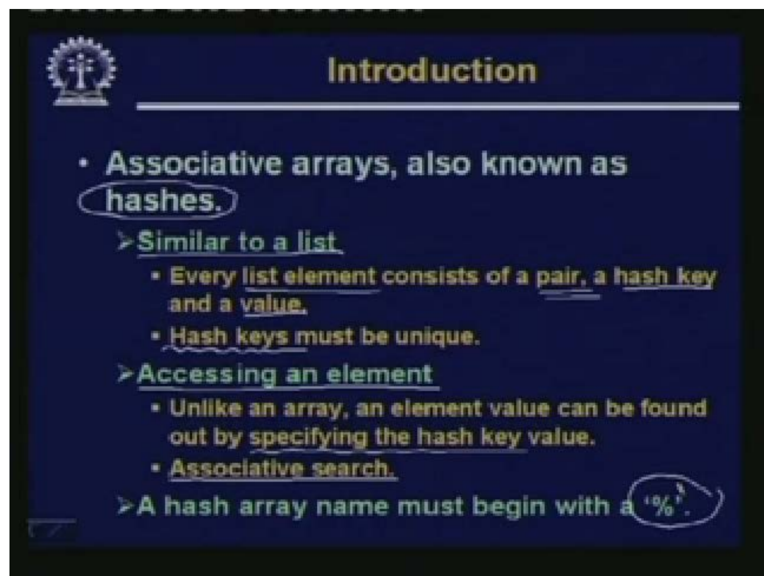
Internet Technology
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No #24
PERL – Part IV

In this lecture we would be first looking at some more additional features of language Perl. Then we shall see how we can use Perl to write or develop CGI scripts with the help of some examples. So first let us see about something which is called Associative arrays in Perl. Now we have already seen or we have already talked about the conventional or normal arrays or lists. An array is just a collection of list items which can be accessed element wise. Now we have seen that an array is basically a collection of items which can be accessed by specifying the index of an item in the list. The index starts with 0, so can say the array numbers array element number 0, element number 1, element number 2 and so on. Now as the name associative implies associative means we are trying to access by content. If you recall what an associative memory is an associative memory is something which you do not access by specifying an address. Rather we specify the contents and try to search whether there is any memory location with that content present. So an Associative array conceptually is very similar. It is an array where the primary mode of accessing is by specifying the value of an element rather than the index if the element in the array. So the associative arrays.

(Refer slide Time: 02:33)

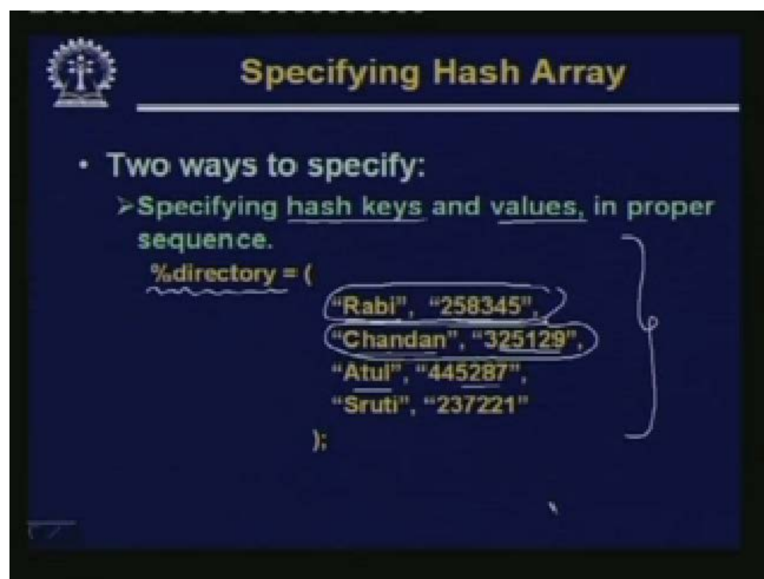


Now in the terminology of Perl which is also known as hash list or simple hash. Now an Associative array to start with looks very similar to the list. Similar to the list in the sense that it consists of a sequence of elements arranged as a list. But the element in the list has a very specific ordering relationship maintain among them. How? Here every list element of a hash or Associative array consists of a pair of sub elements. The pair, the first element of the pair is called hash key and second element of the pair consists of a value. So if you look at the contents of such a list you will see that you have a hash key, a value, a hash key, a value in this way it continues. So hash key value pair appears one after the other. So that is why the pair hash key and value are considered to be one element of an Associative array or

associative list. Now the constraint here is that among all the elements you have in this hash or Associative array the hash key values must be unique. For example you have a list consisting of the roll numbers and names of the students in a class.

So you have roll number, name, roll number, name. In this way the constraint is that the roll number has to be unique. The reason for this constraint is fairly obvious because we are trying to search by specifying the value. There should not be two entries in the list containing the same value of the roll number. That is why roll number has to be unique. So as said the characteristic Associative array is that when you are accessing an element unlike an array the element value can be searched by specifying the value of the hash key. Associative search means searching is done implicitly on all the elements of the array. So you need not have to write a loop kind of structure in a program while you are searching the list. One element at a time you give a single command, you can search the whole array and find out where a match is found. Now just to indicate that it is a hash array the name of a hash array must begin with a percentage sign.

(Refer slide Time: 05:21)

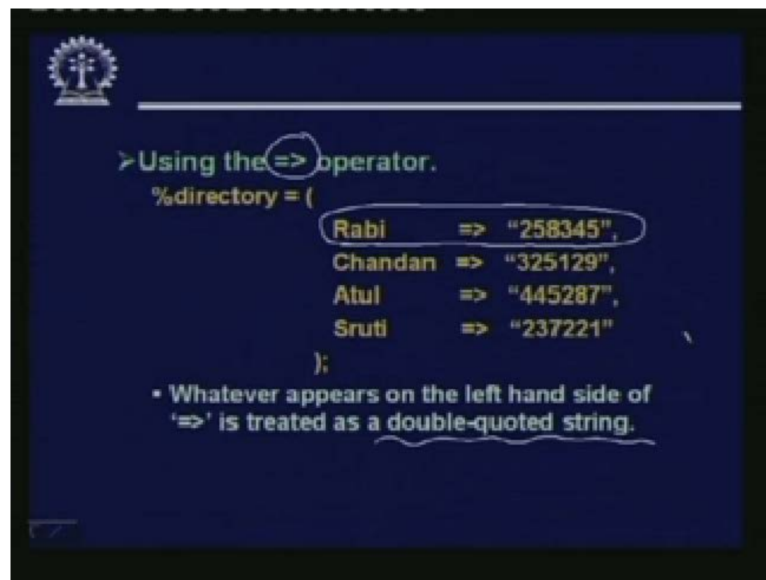


Now let us see how we can specify the values in a hash array. Now had said that the basic element structure of a hash array consist of a hash key and a corresponding value in proper sequence. This example gives you one way of specifying. The initial value of a hash array. This variable name starting with a percentage indicates that it is a hash array. This opening bracket and closing bracket indicates the boundary. Now within this boundary we are specifying a set of values separate by commas. Now in a hash array as had said that the values will be appearing in pairs. So conceptual these will the pairs. So actually the interpretation of this example is that the first element of the pair is a name and the second element may be a telephone number.

So actually we are trying to store the names of certain persons and the telephone number. Since they logically appear in pairs a set of such name and telephone pairs will be stored as an associative array. So this is the first way of specifying it where you simply separate out all the elements separated by commas as if it is a normal list. But the associative array will be taking the numbers as pairs. But the first elements of the pair that means the name will be

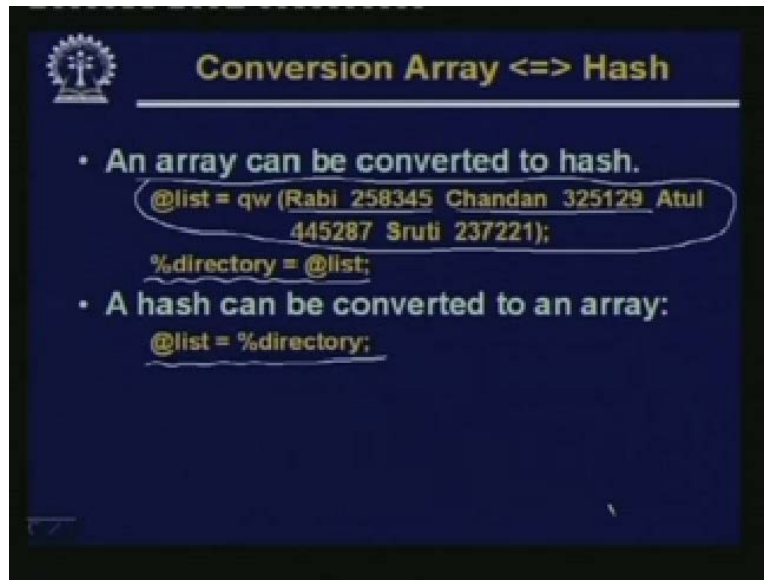
taken as the hash key and the second element. The telephone number will be taken as the value. So name is the hash key value is the telephone number.

(Refer slide Time: 07:20)



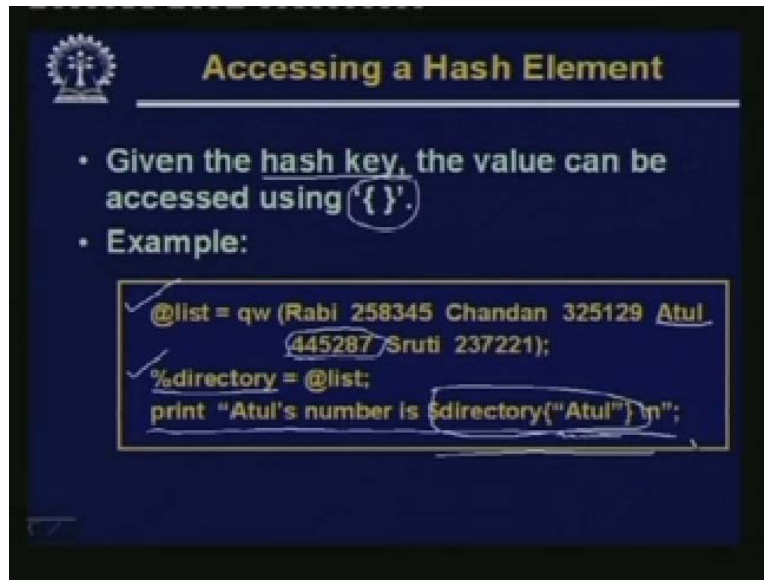
The second way of specifying it is like this using the equal and greater than operator. Here it is quite similar, but you see that the way we specify is little more meaningful or little easier to understand we specify it like this. First we specify the name as usual then the symbol equal to greater than. Then the telephone number then a comma. Now in this notation when you are using the equal to greater than operator, whatever appears on the left hand side of it for example here Rabi. It is implied that whatever is in left hand side is enclosed within a double-quoted string. So the double-quoted string here is optional. So whatever appears to the left of equal to greater than is considered to be a string within double quotes. So this is a more natural way of specifying it is much easier to understand that well this particular name is associated with this particular value. In the previous case everything was separated by commas. So unless you separate out the pairs across lines it will be difficult for you to find out the exact correspondence which is the key and which is the value. So this is the alternate way of specifying the initial value of a hash or an associative array.

(Refer slide Time: 08:47)



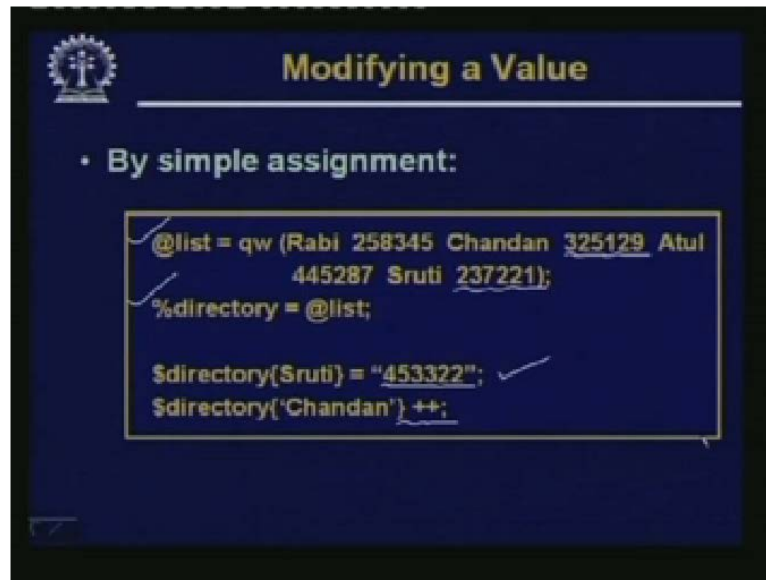
Now here we see how a normal array containing some values can be stored in a hash array or can be converted to hash? So let us take an example. First here we see a case where you have normal array called list which consists of values Rabi. Some number telephone number Chandan and some number so on. So actually names and telephone numbers appear in sequence plus but it is a normal array. However if you make an assignment like this that this array list is to be assigned to a hash array directory. Then the pairs or automatically extracted out. These pairs are automatically extracted out and the first values of the pairs become the hash key and the second value of the pairs will become the hash key value. Similarly if you use the reverse kind of an assignment then whatever is present in a hash array will be assigned to a normal array where the elements will be appearing in order. So a normal array can be assigned to a hash array. A hash array can be assigned to a normal array. But essentially the values are the same in the two arrays. So what is the main difference is the way in which you can access the elements? As you know in a normal array we access the elements by specifying the index. But in an associative array as we will see we can access the element by specifying the value of the hash key.

(Refer slide Time: 10:33)



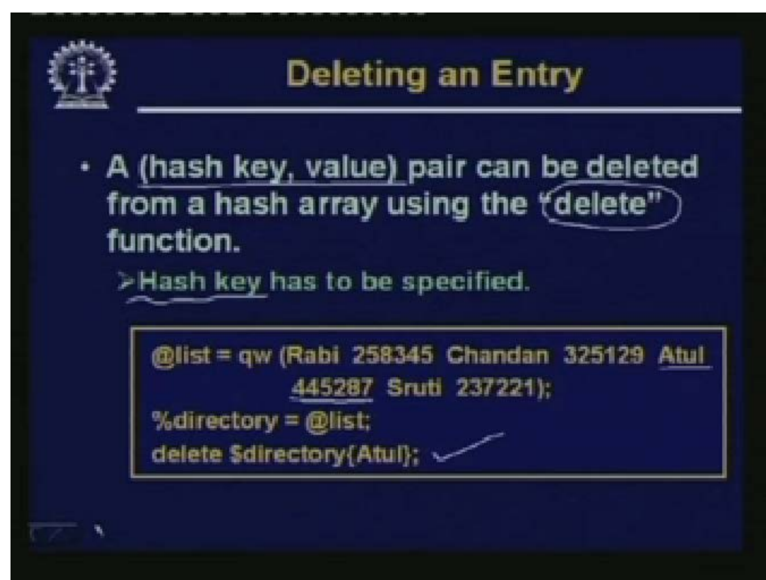
So for accessing a hash element in an array you will have to specify the hash key and you use this symbol curly bracket to indicate that well here we have the given hash key. Well an example is given here. The first line shows you the definition of a list, but the names and telephone numbers are appearing as pairs. Then you are assigning that array to a hash array directory. Then there is print statement it says Atul's number is well you look at this part. Dollar directory. Directory is the name of the hash array. Whenever we specify dollar directory it means we are trying to access a particular element of that hash array and after dollar directory within curly bracket. As you can see within curly bracket we are specifying a value of a hash key. So in this case Atul is the hash key value and the corresponding value which is this. This will be returned by this directory. Atul access to the hash array so here what am saying is that you are accessing as if an element of an array. But instead of specifying the index you are specifying the name of the hash key. Atul in return what it gives you is the name of the value which in this case is the telephone number. So what will be printed is Atul's number is 445287, this is what will get printed.

(Refer slide Time: 12:36)



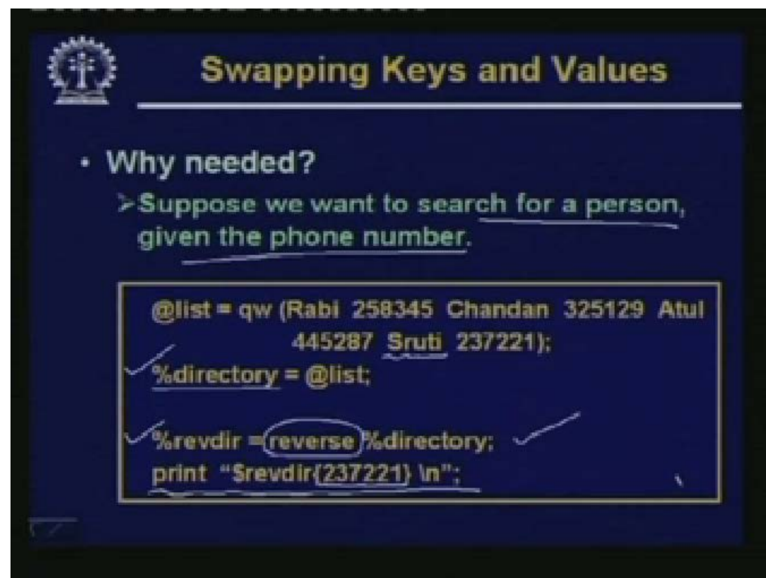
You can modify the value of a hash array by using simple assignments like this. Or similarly you can say the same example we are assigning. These values to a hash array directory. The first example shows you that how you can assign a particular value to a value part of a directory. That means we are trying to change telephone number of Sruti. So initially the telephone number was 237221. So after this assignment the telephone number will get changed to this particular value, right the second example shows that we can use the plus plus operator. For example or even you can use it in a normal arithmetic expression to update the values. For dollar directory Chandan will mean whatever was the telephone number of Chandan. This will get incremented by one, so the number will become 325130. So this how you can modify the value part of the hash key and value pair in an associative array. But in all these cases you are accessing the elements by specifying the value of the hash key and in return you are being able to access the corresponding value of the data.

(Refer slide Time: 14:00)



Well you can also delete an entire entry from the hash table. Entire entry means the hash key and value pair. This pair can be deleted by using a function called delete. Now here again whenever you are deleting you have to specify the value of the hash key. So the same example if you give a command like this, delete four directory Atul which means this Atul as well as the corresponding telephone number both will get deleted from this list. So using the delete command you can remove a pair of key values means the hash key and value from the associative array totally.

(Refer slide Time: 15:00)

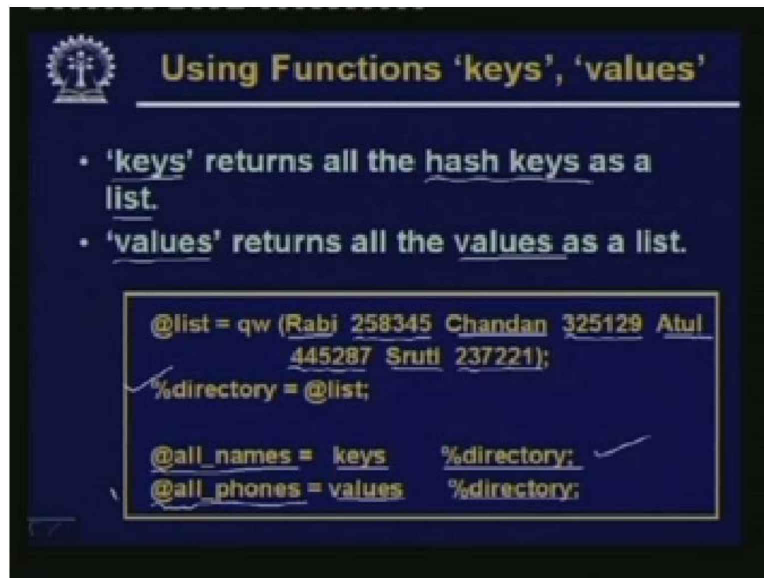


Sometimes we need to swap the keys and values of an array. Well why it is required? If we come to the same example, say in the example we have given we are storing the names and the telephone numbers, like they appear in the telephone directory normally people search the telephone directory by name. That was our assumption the name was chosen as the hash key. So if we specify the name we will get back the telephone number. But suppose want to search in the reverse direction know a telephone number have found out a telephone number. I want to know who is the owner of that telephone, want to find out the name. But in an associative array we are not allowed to do this kind of search given a value part cannot find out the hash key part. So in a normal hash array you are not allowed to do this kind of reverse searching. But some application may require you to do this kind of reverse searching. So in that case what do you do? What you do is very simple. You reverse the roll of the hash key and the values. You tell somehow that whatever was the value earlier.

This will now become the hash key. Whatever was the hash key earlier this will become the value now. So in this example we want to search for a person given the phone number. The way you do is like this. Up to this it is exactly as in the previous example. The hash array directory contains the name telephone number pairs of the lists. There is a function called reverse in this line. We are applying this reverse on this hash array directory. What this reverse will do? It will reverse the rolls of the pair. That means it will reverse each pair the second element will become the first element. The first element will become, the second element of the pair. So now the telephone number will become the hash key and name will become the value. So after this if you assign it to another hash array, let us call it revdir. Then we can use this revdir to access by specifying the telephone number like if you give revdir

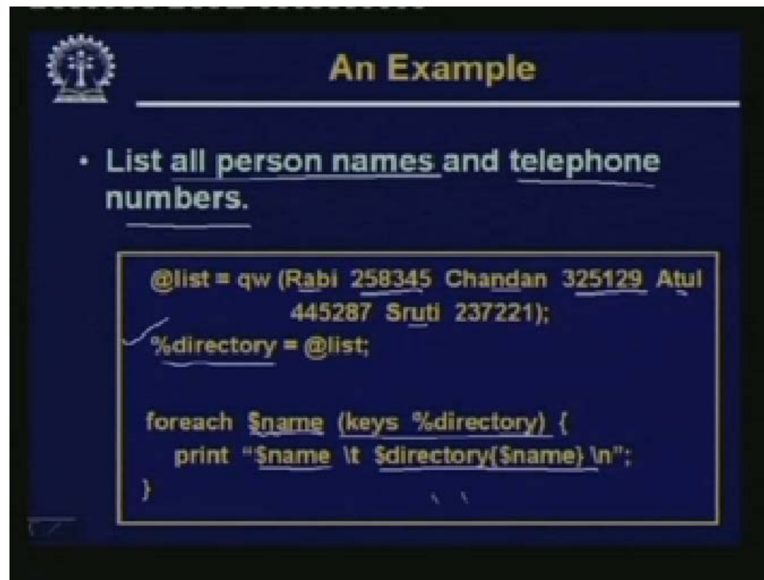
237221, then we will get back Sruti as the name. So here what will be printed will be Sruti. So this is how we can reverse the rolls of hash key and values.

(Refer slide Time: 17:47)



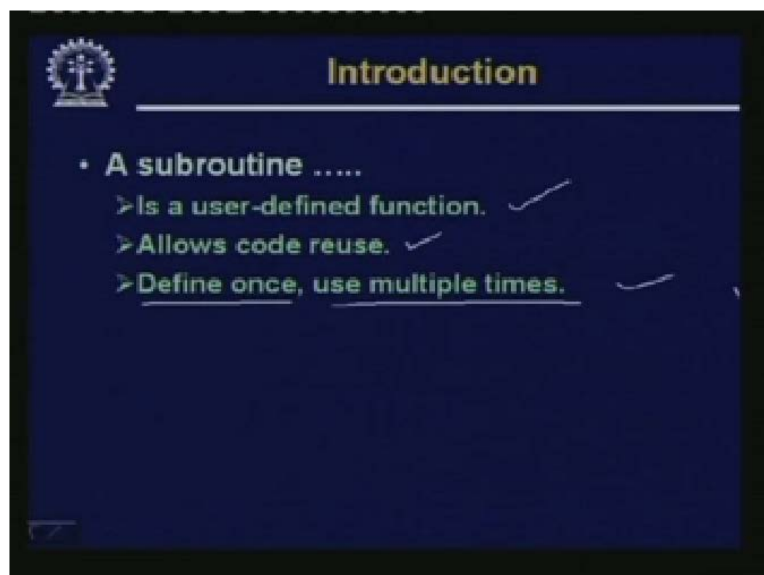
There are some functions called keys and values which are also used in a hash array quite frequently. The keys functions return all the hash key values as a list. Similarly the values will return all the value parts of the pair like in the previous example. Maybe I have an application where want to list all the names of the persons or want to list all telephone numbers that are stored. So here we need to use the function keys or values. So an example illustrates the usage. The same arrays have taken as an example. In this line we have used the function keys. Keys are applied to the hash array directory. Keys directory actually will give you a list which will consist of Rabi Chandan Atul and Sruti. So this four element list will be assigned to an array all names and if we use the function values in the same directory you will get back the values or the telephone numbers. So now these four telephone numbers will form a list and will be assigned to another array called all phones. So you can see that how by using keys and values functions. We can extract the first or the key part whereas second of the value part of the key value pair in the hash array.

(Refer slide Time: 19:45)



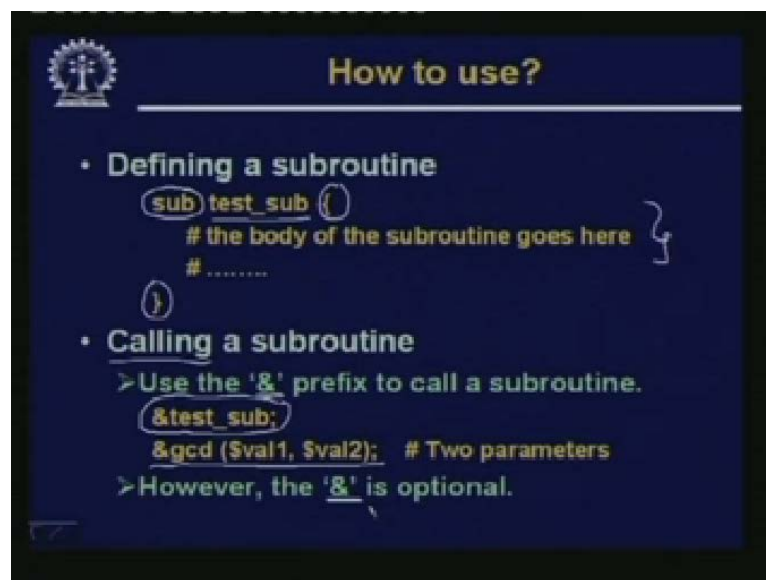
Now a very simple example to illustrate its usage. Like want to list all the names of the persons and their corresponding telephone numbers. So first have this list assign it to a hash array directory. Here we use a foreach loop. Using foreach name keys directory. So keys directory will be returning a list containing all the names Rabi, Chandan, Atul, Sruti. So this will be a for loop and in this for loop the variable name will be taking one of these names at a time and it will be looping 4 times and in every loop what it will print? It will print the value of the present name. So in the first line it will print Rabi. Reverse slash t means tab it will give some gap and directory within bracket name. It is a normal hash array access it will return the value of telephone number. In the second line the name will be Chandan. So Chandan and the corresponding telephone number and so on. So this way can print the name and telephone number pairs of all the persons whose names are there in the directory. Now let us look at another very important feature of the Perl language namely subroutines. Now you all know why we use subroutines.

(Refer slide Time: 21:31)



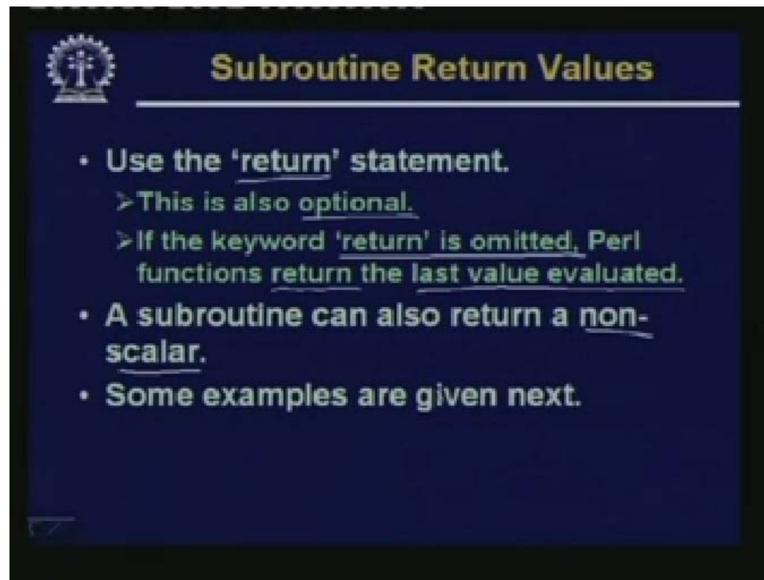
Subroutines are used as a user-defined function motivation of usage is to allow code reuse. Code reuse means write the subroutines once but can call it or use it multiple number of times. So it makes the effort on the purpose of the code designer easier it also makes the program code shorter. Instead of writing the same function several times write it only once and call it multiple number of times. Let us see in Perl how we can define subroutines.

(Refer slide Time: 21:54)



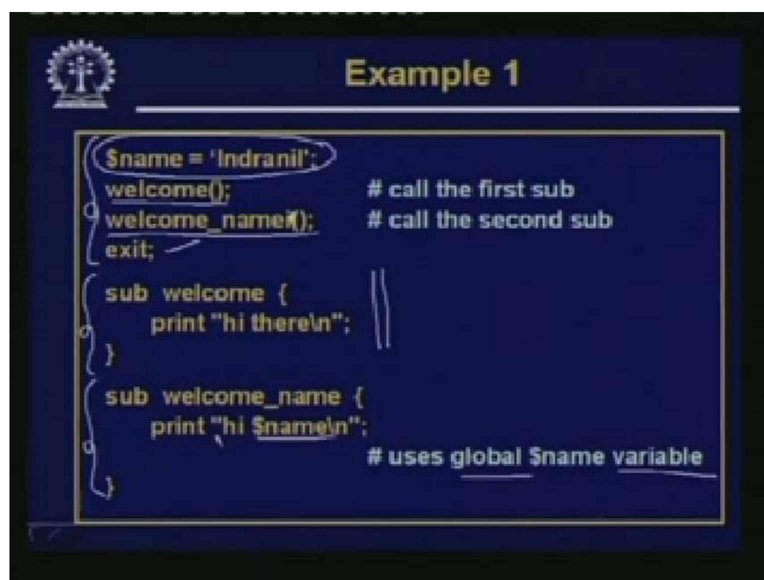
Defining a subroutines is easy. It starts with a keyword sub. So any subroutine mu must start with this keyword followed by the name of this subroutines. Then an opening curly braces ending curly brace. So whatever comes in between will be the body of the subroutine. This is how we define a subroutine. Now the subroutine can be defined in any place in a given program but we you are trying to call it. Normally we use this ampersand prefix to call a subroutine. For example in this example in this case test underscore sub was the name of subroutine. When you call it we give an ampersand before it. Ampersand test underscore sub a subroutine can also have parameters. We will see how parameters can be specified for example the second case. This is maybe a subroutine which computes the gcd of two numbers val1 and val2 and this ampersand gcd means that this is a function. But Perl says that this ampersand optional if you do not test sub are gcd will be by default taken to be the name of a subroutine. So this ampersand is not mandatory thing to use. You may use; you may not use it.

(Refer slide Time: 23:34)



Now subroutine like in any other language compute something and it can return some value. Again in Perl the return statement is optional. If return is present then the return statement will tell you that, what are the values that have been returned just like the return statement in the language C. However if the return is omitted. What Perl assumes is that the last value that was evaluated that by default will be returned. So the subroutine will be returning the last value that was evaluated. Now unlike other languages a subroutine can return non scalar. Also like it can return in general an array it can return a list which normally other languages do not support in C. You can return a pointer but you cannot return a whole list. Some examples we shall see.

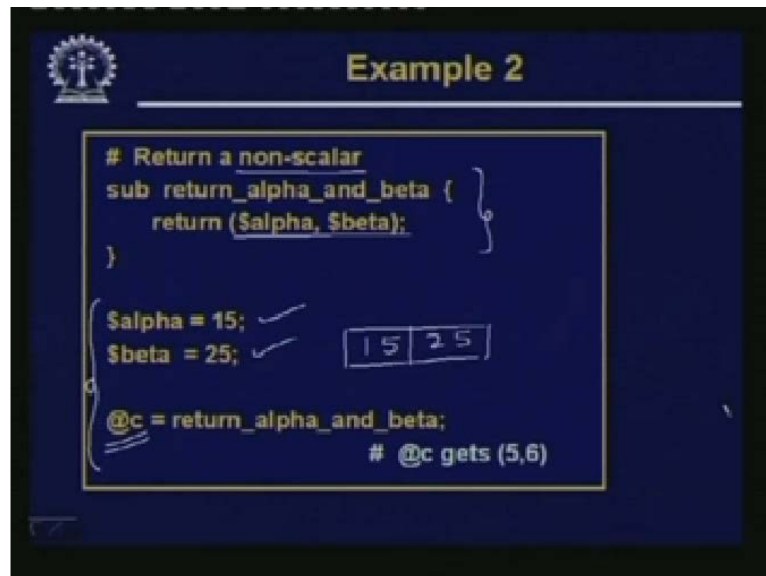
(Refer slide Time: 24:45)



This is a very simple example. This is the main part. This is one subroutine. This is another subroutine. In the main part we have defined a variable called name we have called a subroutine welcome. We just see that we have not given the dollar. Then you have called

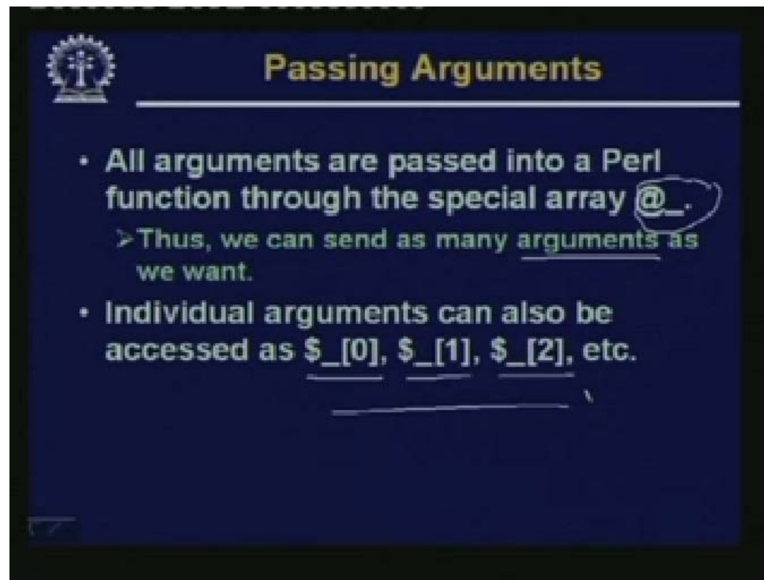
welcome name exit means end. The program ends here. When welcome is called if you look at this particular routine, it simply prints a line hi there that is all. But the second function welcome name namei. It should be name no is there. So here it is printing hi. The value of this dollar name. Now you recall this dollar name is a variable which was defined here. So this is termed as the global variable. A variable which is defined outside the definition of a subroutine and just like any other language like C. If a variable is defined outside a function it is accessible inside the function. Unless of course you redefine another with the same name variable inside it. So in this way you can access global variables.

(Refer slide Time: 26:15)



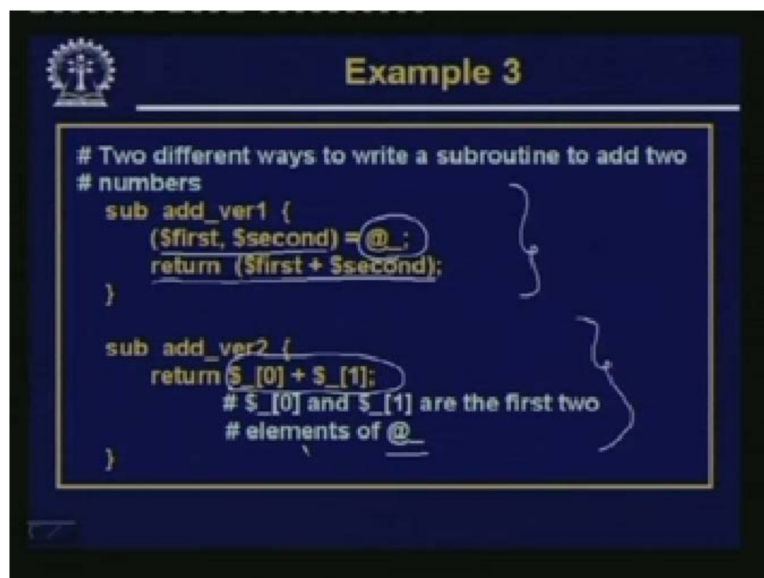
The second example here, there is a subroutine. This returns a non-scalar. Well this does not carry out any computation. Just for the sake of illustration it returns a pair of values. A pair of values within brackets constitutes a list. Dollar alpha dollar beta and this is the main function. So this example tells you one thing that a subroutine can also be defined before the main function. There is no particular ordering and here you are assigning 15 to alpha 20 to beta. Then we are calling this function. Here alpha and beta will be treated of as global variable again because these are defined outside the definition of the subroutine before this function is called. So it will be returning alpha and beta which means 15 and 25. So this list will be assigned to an array c. So c will finally be a list with two elements; the first element will be 15 the second element will be 25.

(Refer slide Time: 27:34)



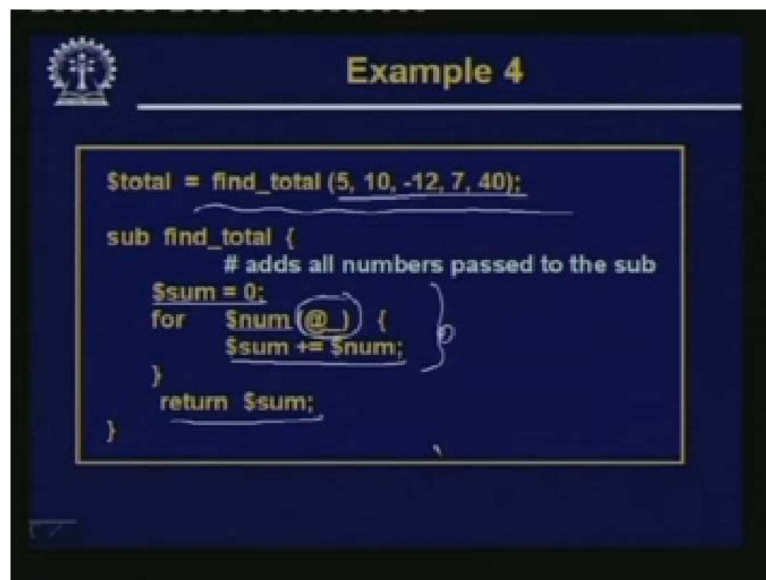
So far we have taken example where there were no arguments to be passed. Now let us see that how we can pass arguments to a subroutine in Perl. Now Perl makes it very flexible. Perl assumes that there is a special array called at the rate underscore. This is a special array which contains a number of elements and these elements are considered to be the arguments. So since this ampersand underscore does not have any name ampersand underscore, the individual arguments can be accessed as dollar underscore, zero dollar underscore, one dollar underscore, two and so on. Or you can copy the contents of an array to some other list and you can access the elements of that other list also. It really does not matter, you can use it anywhere in any way you can. The only thing you have to remember is that arguments are passed through the array at the rate underscore. You can take the argument list as the entire array. Or if you want to access the individual arguments you can also access it as dollar underscore, zero dollar underscore, one, two and so on like this.

(Refer slide Time: 29:06)



Let Us take an example. Suppose we are trying to add two numbers. Here we are showing it in two different ways. The first is like this. There is a subroutine which is actually adding two numbers as said. So here when this subroutine is called the parameter which was there in at the rate underscore. This array is assigned to a list. So the first element will go to dollar first, the second element will go to dollar second. The subroutine then returns dollar first plus dollar second; their sum. This is second version. Second version does not access the elements from array at the rate underscore. But it directly access it as dollar underscore zero and underscore one and adds them up. So basically dollar underscore zero and one are the first two elements of at the rate underscore. So in these ways you can access the individual elements.

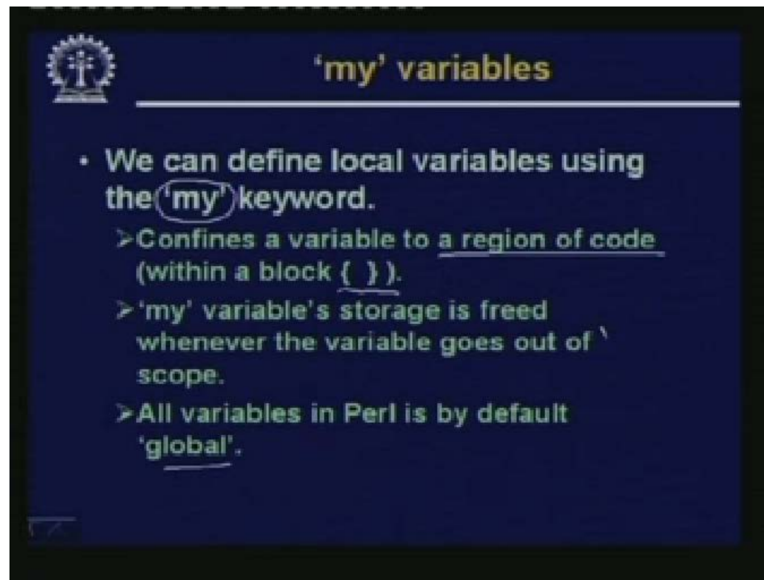
(Refer slide Time: 30:20)



Let us take another example where we want to write a generic function called find total. That will compute the total of a list of elements within parameter. You can specify a list of elements. Now one thing you understand that in a language like C whenever you pass a parameter you cannot give unconstrained number of parameters when you are calling it. When you are defining the function you specify that there will be two parameters 4 or 5 whatever. When you call it you have to specify the same number of parameter. But in Perl you can make it flexible it is not fixed by definition. Like you see when am calling the function and calling it in this way. Here within in bracket have specified 5 different values, 5, 10, minus 12, 7, 40.

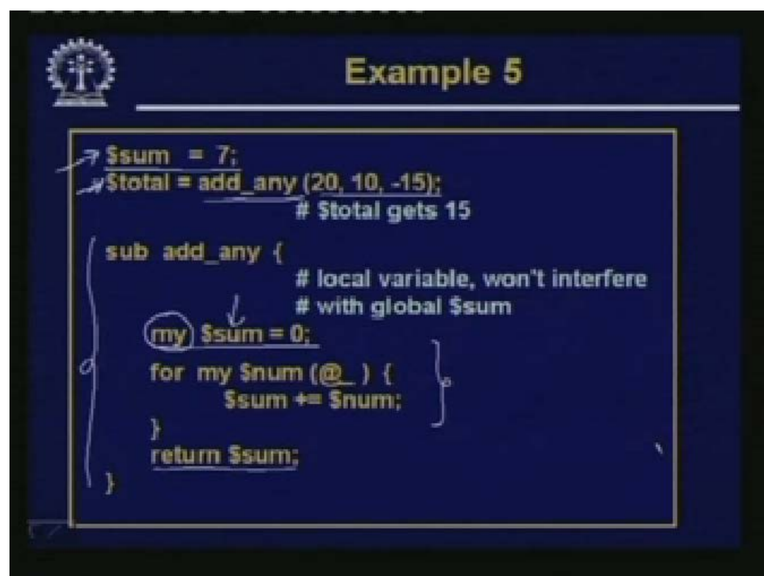
So when the function gets called, these five different values gets stored automatically in an array whose name will be at the rate underscore. This means that we are not explicitly assigning the values to the array at the rate underscore. It gets assigned automatically with the arrays call. We call the array in the conventionnal form within brackets we specify the arguments. So within the subroutine we define a variable called sum initialize to zero for there is a for loop within brackets. We are taking one element of this list at a time. We are storing it in num and within the loop we are adding this num to sum. So finally all the elements will get added and outside the loop we return the value with sum. This is a simple example.

(Refer slide Time: 32:15)



Now in Perl by default all the variables that we define are global variables. If it is defined outside the subroutine it can be accessed inside the subroutine also. But sometimes you may define the variables as local which means that it is only defined or accessible within the scope of that particular subroutine or function. Outside that it does not have any meaning or value. This can be specified using the my keyword and wherever you define a local variable using my the definition is confined to a particular region of code which is typically a block. So as said all variables in Perl, by in Perl by default global and whenever you go out of the scope the variable is available no more.

(Refer slide Time: 33:13)

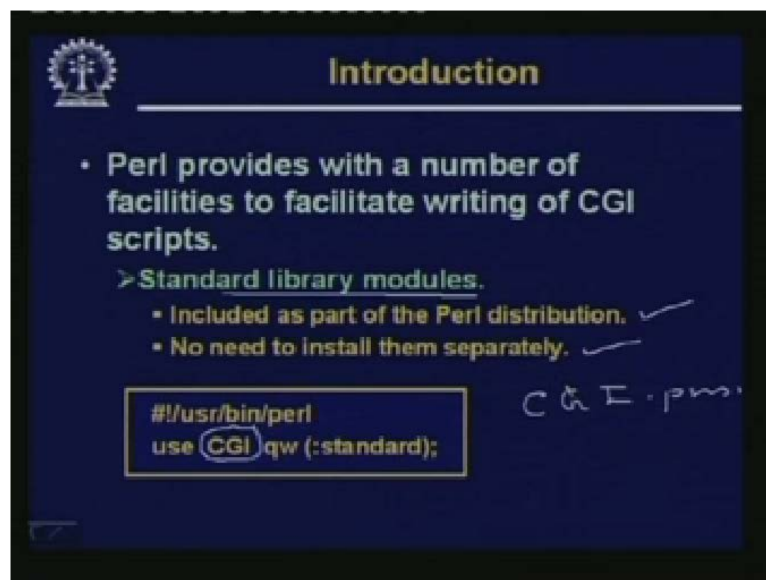


So a small example. The same example as have given earlier similar to that say in the main function. I have assigned a number 7 to a variable sum. Then am calling a function add any with three parameters 20, 10 and minus 15. So in the body of the subroutine am defining another variable the same name dollar sum. But I am specifying the keyword my which

means that this particular dollar sum is local to this subroutine. This is not accessible or available outside. So inside when compute the sum of the elements. In this loop it is this local dollar sum which gets affected and is returned. So after return the value of this total. This will be 15, the sum of these three numbers. But the original sum will remain at 7. This will not change. Now let us see how we can write some CGI script programs in Perl. Now we have said before that Perl is a language which has very strong string handling capabilities.

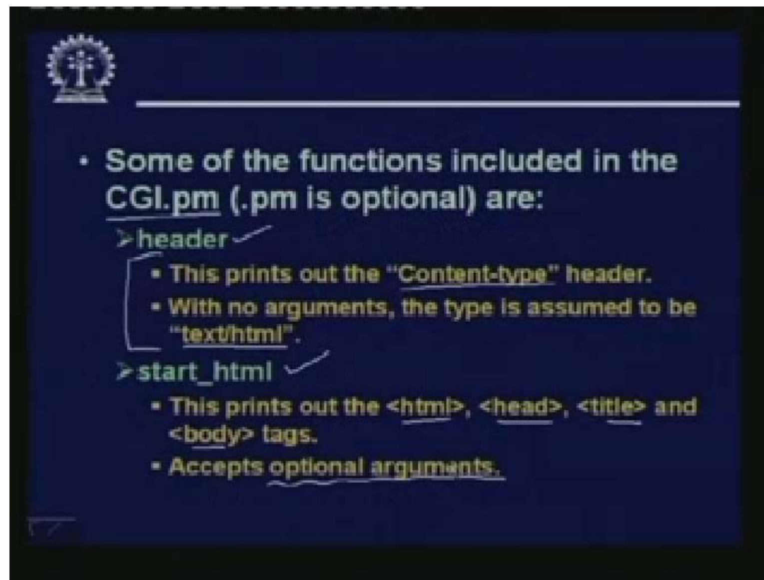
This we have already seen there are some very unique and very strong features of with which you can do some very interesting thing. On strings using very few number of statements. These features of Perl gives us a big advantage when you are trying to develop some applications like CGI script which is essentially a string handling application. Now we shall be looking at two things. First we shall see how a typical CGI script can be written. Second we shall see that Perl comes by default. Whenever you are downloading Perl from some site, from the internet, it will come not only along with the basic features. But also along with some standard libraries. There are some standard library functions which makes it very easy for the programmer to develop applications. So we will some such application and usage of such standard library.

(Refer slide Time: 35:40)



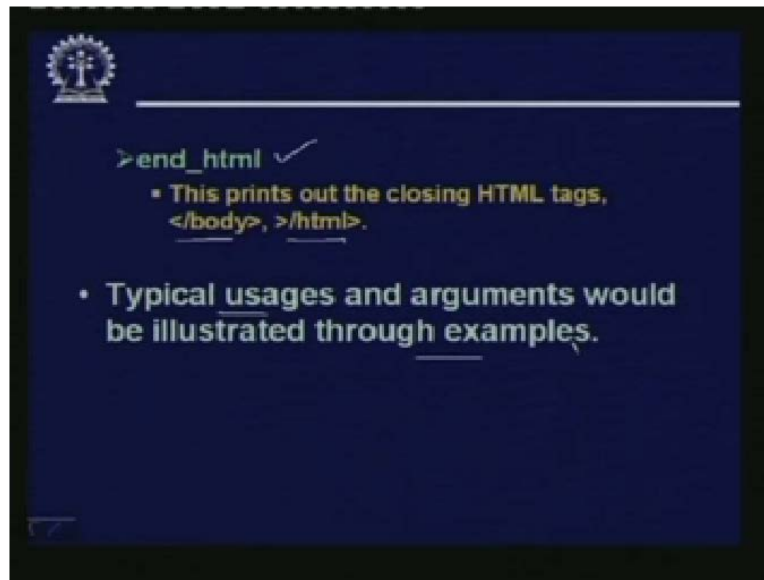
So as we said Perl provides with a number of facilities for writing CGI scripts and there are number of standard library modules which are included as part of the Perl distribution. You get it automatically when you download and install Perl on your machine. There is no need to install them separately. Just an example there is a very popular standard library called CGI dot pm. The libraries are marked pm. So when you give CGI dot pm, now we you call it you just type CGI dot pm you do not specify qw colon standard means you are trying to take it from the standard input. This array this will taken as the input to the CGI program and it will be executed. So will is actually how these are accessed later.

(Refer slide Time: 36:48)



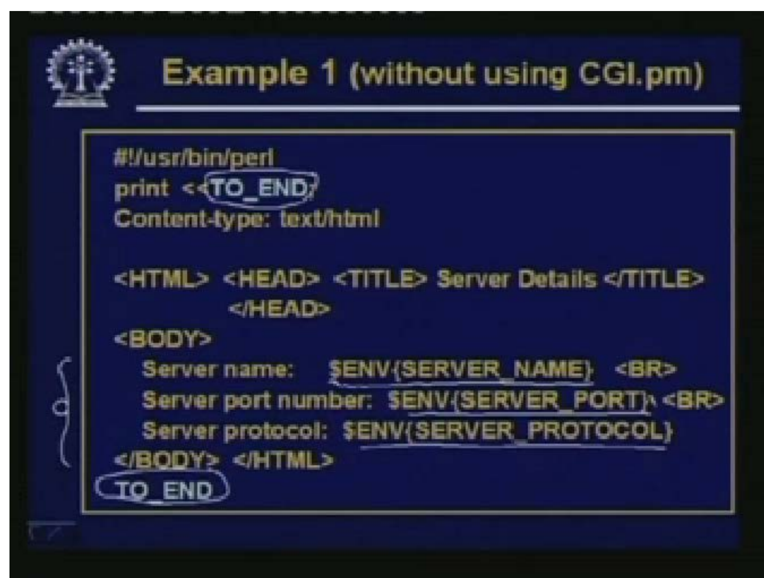
So as said CGI pm is the standard CGI function which is used specifically for writing CGI scripts. In exactly the similar way there are other Perl libraries which are meant for some other applications. Like there is a set of libraries available in Perl for database interfaces and applications similarly for socket programming and so on. Well some of the important functions that are available are there is a function called header. If you simply call header this will automatically generate a standard my encapsulated header as output. Typically this will print out the contents type header and if you do not specify anything by default Content-type will be text html. So it simply prints out Content-type colon html. So that whatever is now going to generate it will be an html file that initial thing has been set up. Similarly there is a function called start html. Start html prints out the initial tags like html, head, title, body, the beginning tags. It can accept some optional arguments for example it can take an argument specifying that what should be the colour of my background, what should be the title of my page because all these tags are generated by this. Now these will not be empty tags. In general you can also specify value of some parameters of these tags. This is start html.

(Refer slide Time: 38:43)



Similarly you can have end html towards the end of or closing html tags. This end body and html these are printed. Now some typical usages of these we shall illustrate through some examples.

(Refer slide Time: 39:00)

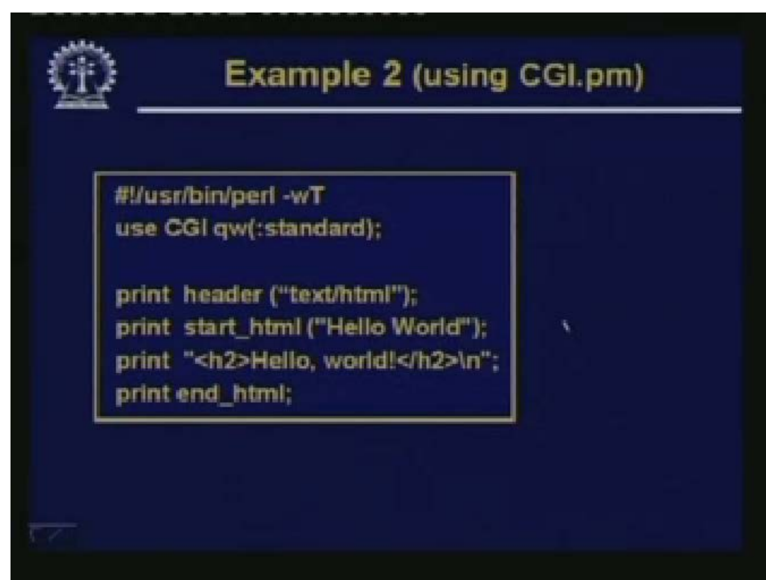


Let us take one very simple example here. Now this is a CGI script. But this CGI script does not accept any data a form. What it does is that whenever someone invokes this CGI script this invocation maybe through the submission of a form. But the CGI script does not read any data which is coming. But rather it simply prints out some information and returns it back to the requesting client. Let us see what it does it starts with this initial header user bin Perl. It basically makes a block print this “TO END” is the special delimiter it is used. So whatever is there between the delimiters is printed as it is Content-type: text/html. Then the headers HTML, HEAD, Server Details, BODY, id server name here we are calling a function dollar

ENV. Dollar ENV will invoke or return the value of the environment variable whose value is there in the parameter SERVER NAME.

So whatever is the value of the SERVER NAME will be printed here. The second one this will print the SERVER PORT and the third one will print the SERVER PROTOCOL. So actually what it will print, then you know that in the client server scenario if you recall, whenever there is a there is a server and someone is sending some request to the server. There will a port number at the connecting point which will be setup which will be different for all incoming connections. Suppose have made a connection and the number that is assigned to me is 2005. So when try to print out this PORT number using dollar ENV SERVER PORT will get this 2005 out here. So that know sitting at my client that have connected to server all right. But which was the port number that was used. So this a very simple CGI script which basically sends out, these information to the requesting client every time. A block is a form is submitted.

(Refer slide Time: 41:51)

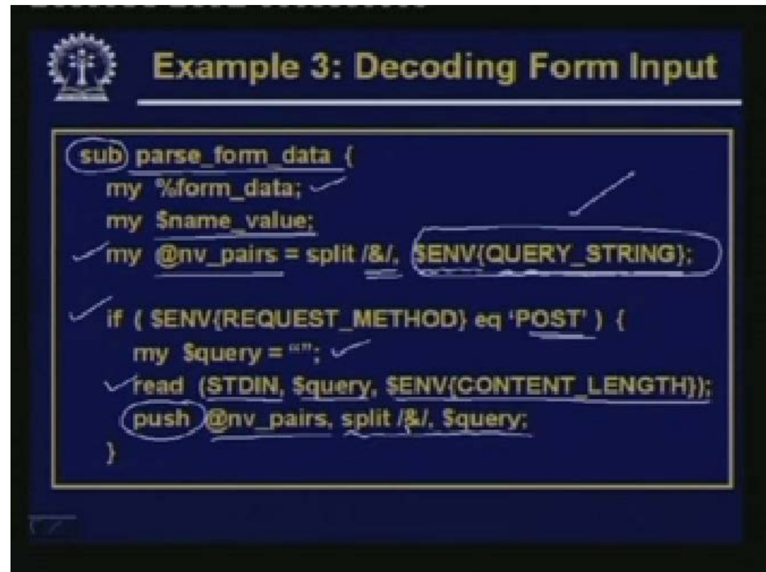


Let us take another example. This is basically the same example you see the previous example. This simply printed some server detail within html tags. Here it is similar in the sense that this also prints a very simple html page. Let us see what it prints. First we are giving user bi Perl. This wT are some optional tags you can use. This second line actually tells you that you are specifying or you are you want to use this CGI dot pm standard library. The first line it uses the function header print header within bracket text dot html. So actually this will generate a my type header Content-type text dot html. If it is something else you are generating instead of text dot html. You can write that if it is image slash jpg you write images like that slash jpg.

Then print start html start html will be printing the starting tags, html, head, title. If you have an optional parameter the first parameter here will indicate the value of the title. So within the title you can specify the value. So this Hello World will appear as the title of the page and whatever comes after that, this is the actual body of the html document. Finally you print the closing lines the end body, end html. These lines so you see that writing a simple CGI script is very simple. If you are using CGI dot pm because you do not have to write lines as in

previous example. HTML, HEAD, TITLE, HEAD, BODY. These are routine things, these routine things are eliminated, here you can use some standard function calls and the standard function calls will automatically generate these routine things for you.

(Refer slide Time: 44:13)



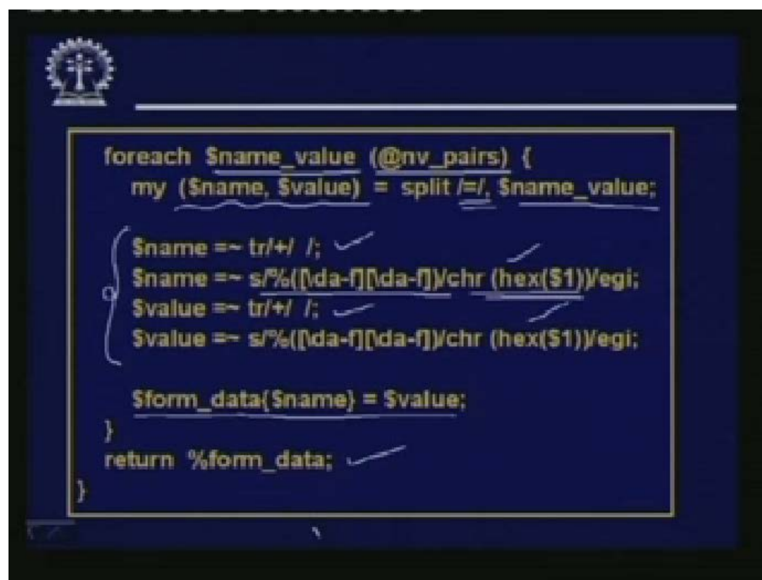
Now let us take an example where a CGI script is used in the proper context. Where we have some form data which is coming to us. Now before trying to understand or explain this example. Let us very quickly recapitulate how form data's, how form data's come to a CGI script? Imagine you are a CGI script and some form data has been submitted to you. First thing you will check whether the submit method or the request method is GET or POST. If it is GET you recall then the data is available in an environment variable called QUERY STRING. If it is POST, then it is available as a continues stream which is coming from the standard input and the content length environment variable will tell you that how many bytes are there total in that string. So this particular CGI program will try to check that and it will act accordingly. Which means irrespective of whether you are submitting the data through GET or POST. This script will work both ways. Next let us try to understand once the data comes suppose you have selected the proper mode GET or POST.

And the data is coming to you, you remember the data coming as name and value pairs name one equal to value, one ampersand name two equal to, value two ampersand name three equal to value three and so on. So you have to do two things first you have to cut the strings at the ampersand points and extract the name value pairs name one equal to value one will be one such name two equal to value two will be another such. Once you have these pairs you cut at the equal. [46:19 audio not clear] So now you have the list name and value. So these are the basic steps you need to follow to actually extract the data that is coming to you individually in terms of elements element by element. Now let us see how this works out here. Here we have written in form of a subroutine. This is a very standard kind of a thing which is required for almost all form data submission. That is why we have written it in the form of a subroutine. The name of the subroutine we have given is parse form data. Here we have defined an associative array called form data.

Now why associative you can understand because data is coming as name and value pairs. So naturally there is a pair concept here, name will be the hash key and value will be the value key and there is a variable called name value. In this statement by default we are assuming that the request method is GET and the data is available in QUERY STRING. If you call the function dollar ENV with parameter as the QUERY STRING, then the whole QUERY STRING will be coming here as a single string. If you call the function split with delimiter ampersand, then the QUERY STRING will be broken up in to a number of smaller strings. Each will be like name one equal to value one [48:09 word not clear] name, two equal to value two will be another string and so on and these individual elements will be stored in an array. We were calling nv pairs, name value pairs. So nv pairs will contain each element will contain a string which is of the form name one equal to value one like this. This is if this is the GET method. Now after that check that if the environment request method if the request method in the environment variable its value is equal to POST or not.

If it is not POST which means it is GET which you have already done. Whatever is to be done we skip this body of the if statement. But if it is POST we do something. There is query variable which we initialize it to null. There is a function in Perl called read which reads from file. As you know it reads from standard input because in POST you have to read it from standard input. After reading you will be storing this string in query and for how many bytes the third parameter tells? For how many bytes CONTENT LENGTH so many bytes? So it continuously reads, so many bytes and the total thing will get stored in to the variable query dollar query. Now the dollar query contains the entire string now. Now just like we had split the QUERY STRING to get the nv pairs. Here we do the same thing. We split the query variable in to nv pairs. But here we use the push function you could have used a simple assignment also. But it does not make any difference. But this is how we do it.

(Refer slide Time: 50:09)

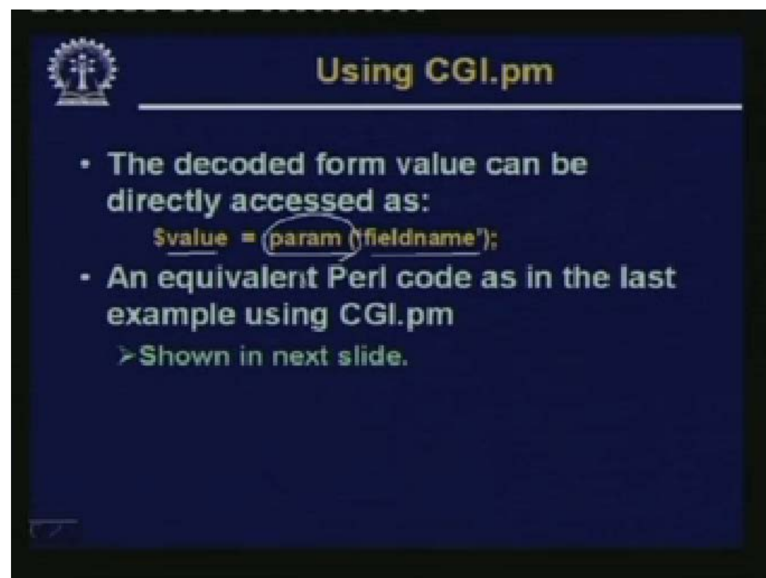


After that see this nv pairs. If you look back this nv pairs will contain some elements name one equal to value name two equal to value two like that. Now for each name value belonging to this array. You now split with respect to equal to this each of the string you get name one equal to value one. On that you split and store it in two variables name and value. Now these four lines are very standard am not trying to go in to detail explaining. It basically the idea is

that you know when you are doing this URL importing. Then sometimes spaces are imported as plus. So where ever there is a plus you translate it in to space. So this line and this line does this and the second and the fourth line actually decodes the encoded values.

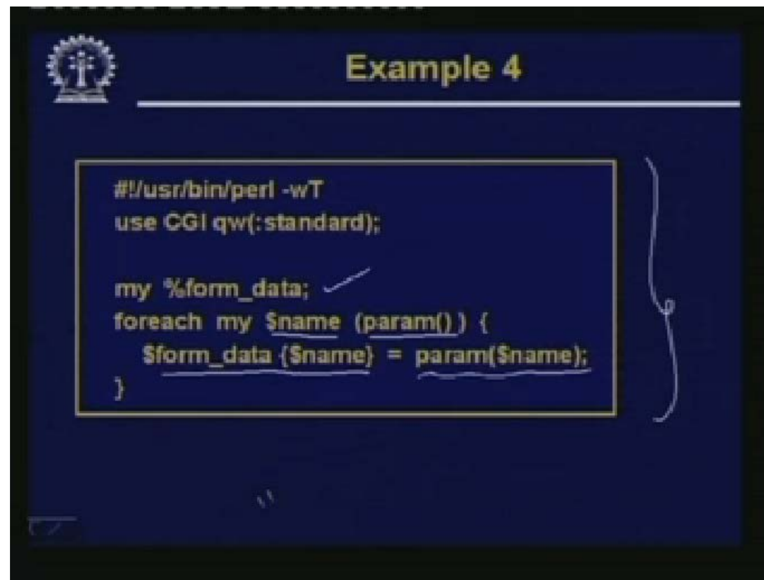
Because you know some special characters are hex encoded. So where ever there are hex encoded characters. You replace the hex encoded characters by their corresponding character value. This chr, hex this is a function call which converts the hexadecimal equivalent of this in to its corresponding ASCII code. So you do this for the name as well as the value part and in the form data name form data is an associative array. So there the name part will store value and you return it finally. So actually what this subroutine is doing is that it is extracting the form data and all the name value pairs taken together. It is storing it in an associative array and the associative array after that you can use in whatever way you want. CGI dot pm as have said that there are some special facilities which are available to you.

(Refer slide Time: 51:59)



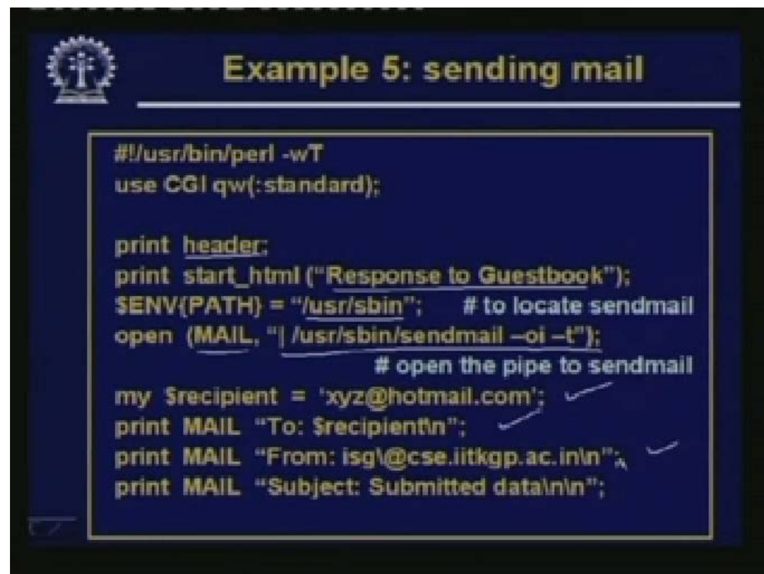
There is a function param which if you call with as an argument, the fieldname you get the value directly. You do not have to do this routine splitting and all these things together. These are already been taken care of in the function param. So an equivalent Perl code of whatever was done here this big function an equivalent Perl code using CGI dot pm is here just this much.

(Refer slide Time: 52:35)



You define an associative array foreach name belonging to param you store form data name the value of param is name. This will do equivalently the same thing which means if you are accessing routinely form data. You need not have to write the program code for that there are some functions param. Here for example which is already available to you, which you can use directly and extract the value corresponding to the name and this splitting and searching all. These will be taken care of automatically within the function this already been provided to you.

(Refer slide Time: 53:18)



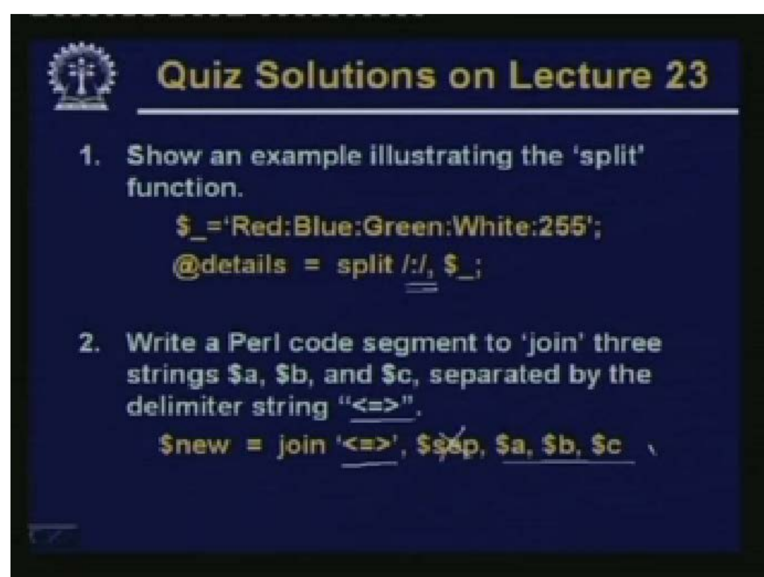
Now another quick example. An example for sending mail. You see here we are printing the header of the html printing start html. This is the title environment PATH this optional you do not require. You open this is a pipe you open MAIL. There is a pipe, this how you use it. So whenever you call send mail whatever is coming to you that will be sending as a pipe as a stream of characters to the sendmail program. These are some flags to the sendmail am not

going in to detail of this. There is a variable recipient. Some standard things you are printing to this to the file handler called MAIL To From Subject.

(Refer slide Time: 54:34)

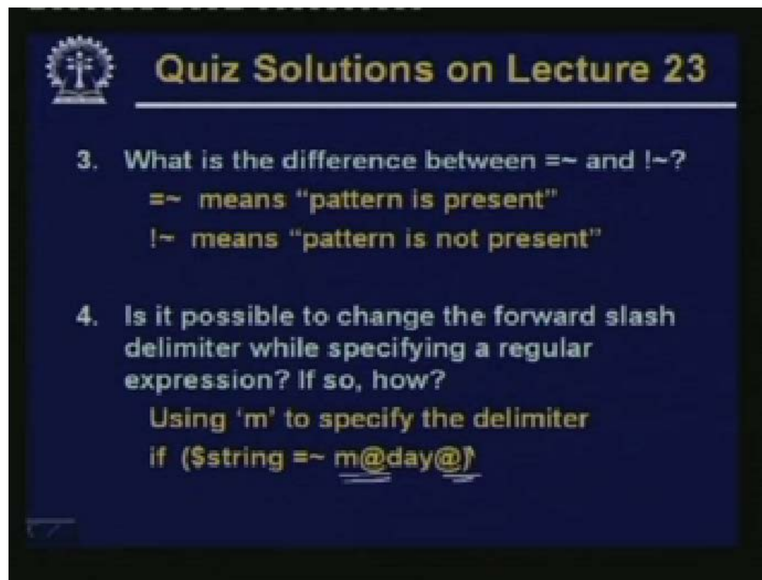
All these things then foreach value in the param. You print xyz is equal to this, the entire thing whatever is coming want to send as a mail the name and value pairs. So, "Thanks for the comments Hope you visit again", this is end. This [54:30 word not clear] EOM, end of file marker we just print it. So actually what you, do we actually this like a Guestbook whenever where ever these is a Guestbook, someone is a specifying that well this in my name, this is my email address and this is my comment. So whenever that Guestbook is submitted what get back as email is that who had sent it, what is his name and what is his comment. This is what will get back as mail. So in CGI as you can see that this is very simple to do. So with this we come to the end of today's lecture. So let us quickly see the solutions to the questions that we post in our last lecture.

(Refer slide Time: 55:10)



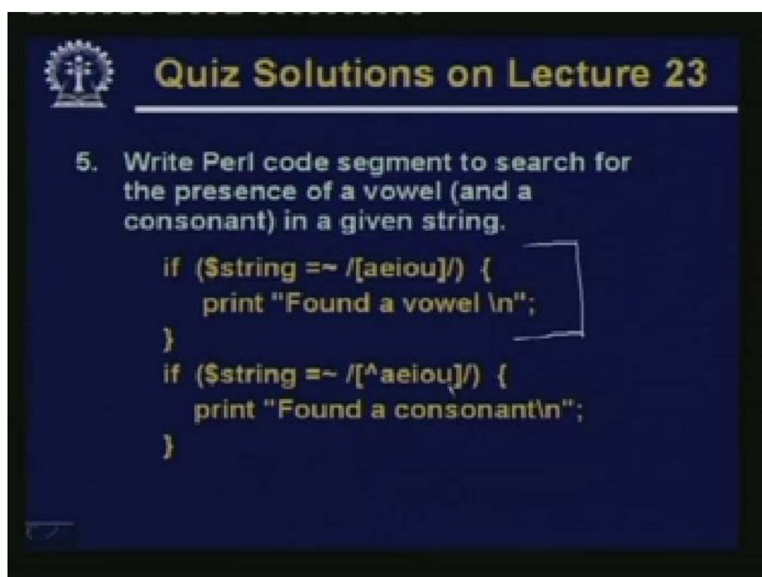
Show an example illustrating the split function.
Well this were also illustrated today. Given an array we can call spit by specifying delimiter.
Write a Perl code segment to join three strings a, b and c separated by the delimiter.
This simply using join you specify the delimiter and then this step is not required. Here in this case you can specify the three strings a, b and c.

(Refer slide Time: 55:42)



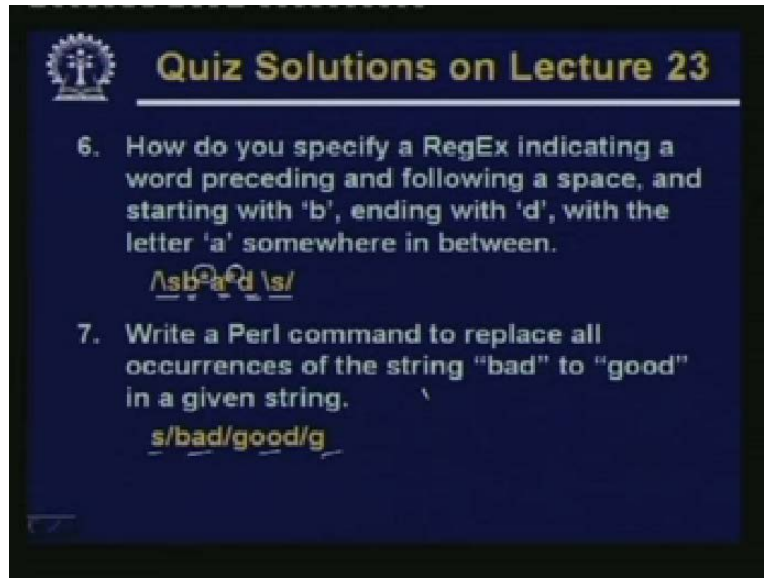
What is the difference between equal to tilde and exclamation tilde?
The first means pattern is present this means pattern is not present. This we use for string match.
Is it possible to change the forward slash delimiter? If so, how?
We use it by specifying m like if you want to change. To this at the rate we use this m m followed by the letter and this will be the last letter also.

(Refer slide Time: 56:14)



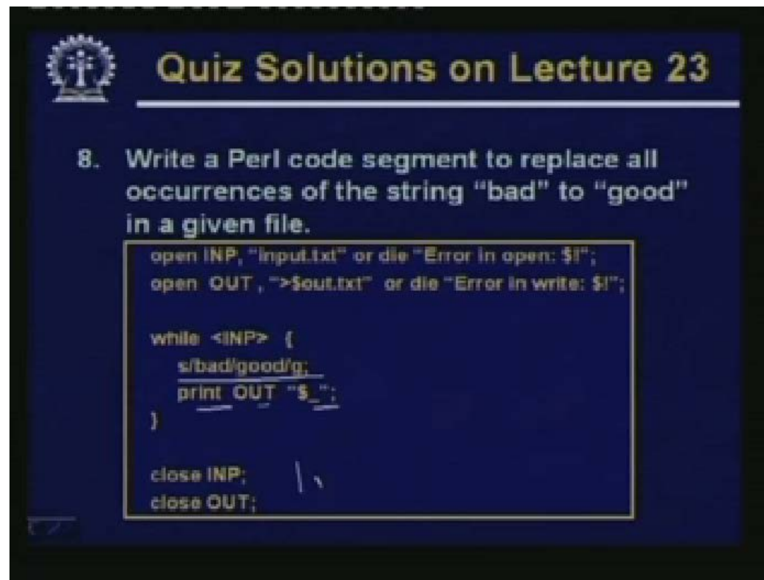
Write a Perl code segment to search for the presence of a vowel and a consonant in a given string.
If it is vowel your code will be like this string matches aeiou. If it is consonant you write like this not aeiou well assuming. That it is only alpha numeric character.

(Refer slide Time: 56:36)



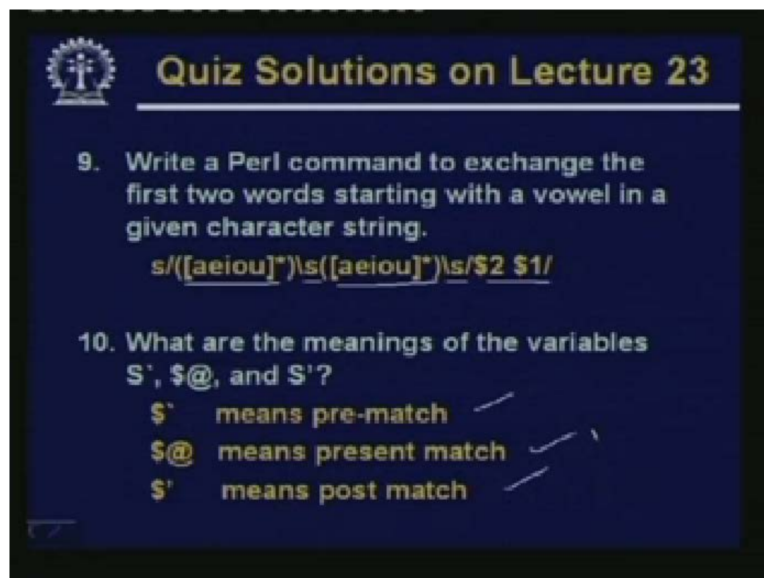
How do you specify a regular expression indicating a word preceding and following a space and starting with b ending with d with the letter a somewhere in between?
You see this regular expression it starts with a space ends with a space after b is the first letter, d is the last letter there is a somewhere in between. This star means any number of characters in between zero or more. So this is what it is.
Write a Perl command to replace all occurrences of the string "bad" to "good" in a given string.
Substitute bad by good global means all if you do not write g only the first occurrence will get replaced.

(Refer slide Time: 57:22)



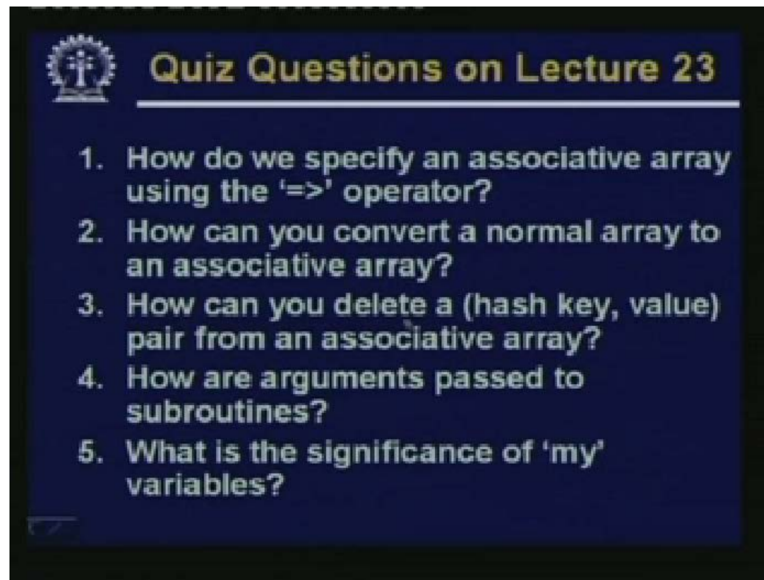
Write a Perl code segment to replace all occurrences of the string bad to good in a given file. It is similar you open two file input and output and while this input is not in case coming you, substitute the lines line by line. Then you print the output file dollar underscore then close the files.

(Refer slide Time: 57:43)



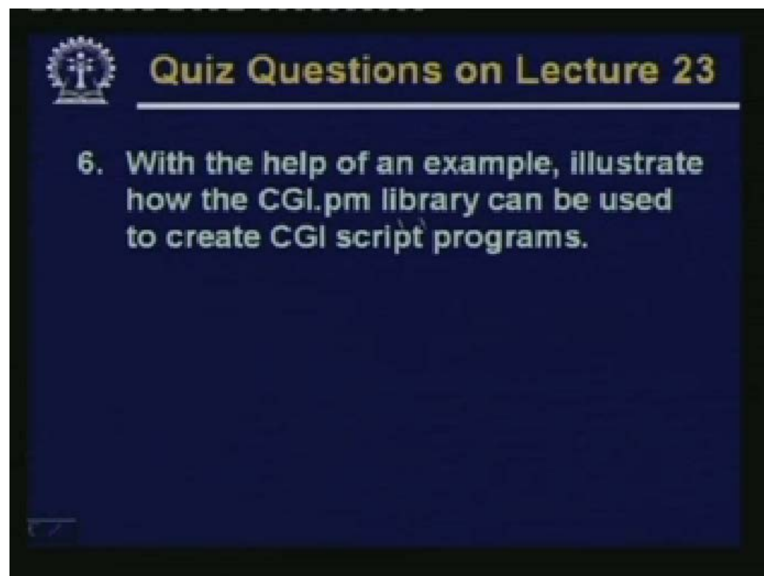
Write a Perl command to exchange the first two words starting with a vowel in a given character string?
This is a vowel corresponding after that anything a space. That means word boundary another vowel a space you interchange them like this.
What are the meanings of the variables these three?
First one means pre-match, present match and post-match.
Then some questions from today's lecture.

(Refer slide Time: 58:10)



How do we specify an associative array using the equal to greater than operator?
How can you convert a normal array to an associative array?
How can you delete a hash key value pair from an associative array?
How are arguments passed to subroutines?
What is the significance of my variables in a Perl subroutine?

(Refer slide Time: 58:35)



With the help of an example illustrate how the CGI dot pm library can be used to create CGI script programs?
So with this we come to the end of today's lecture. In our next lecture we will be starting our discussion on java script. What it is and how it can be used to develop the pages? Thank you.