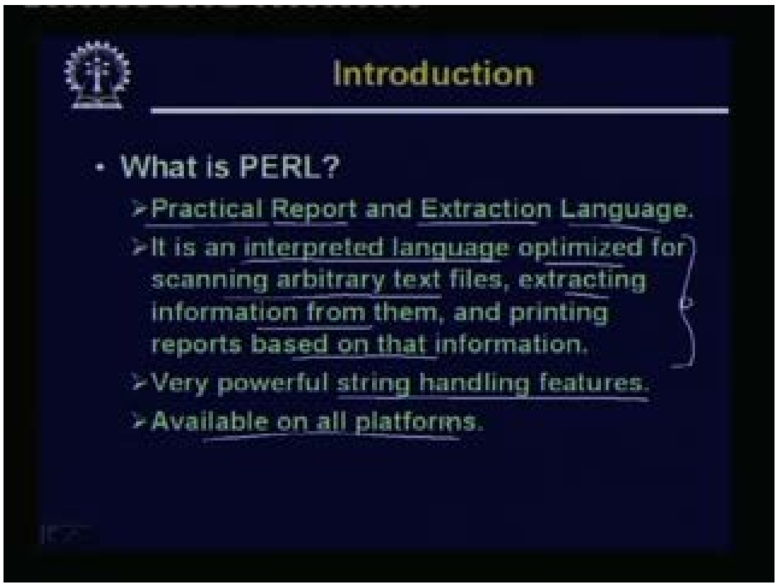
Internet Technology

Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No #21
PERL – Part I

Earlier we had seen or we have told or talked about that the language PERL is very important in the context of developing interactive web pages on the internet. So in the next few lectures would be looking at the various features and syntax and semantics of PERL language with particular emphasis on how you can use this language to develop interactive web pages to develop cgi scripts and the like.

(Refer Slide Time: 01:16)



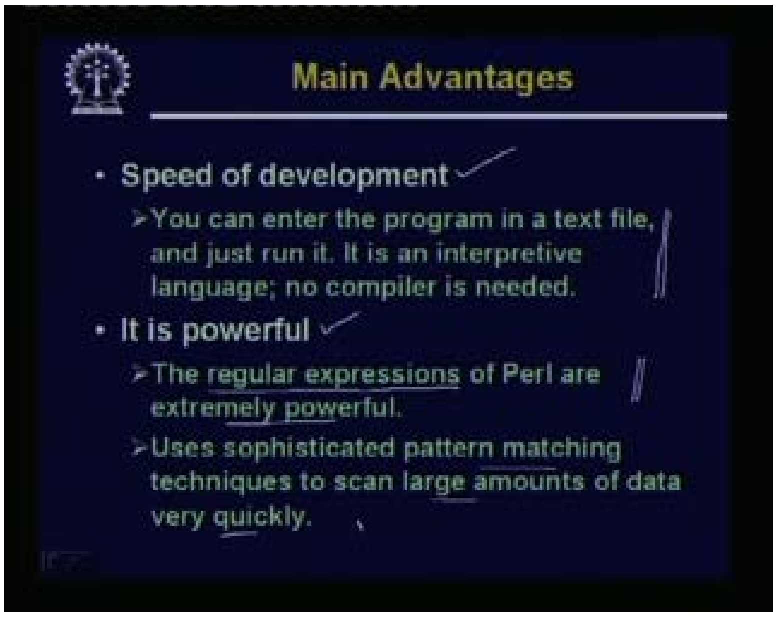
Introduction

- What is PERL?
  - Practical Report and Extraction Language.
  - It is an interpreted language optimized for scanning arbitrary text files, extracting information from them, and printing reports based on that information.
  - Very powerful string handling features.
  - Available on all platforms.

So we start with some basic background about the language PERL. The PERL the full form of this acronym is Practical Report and Extraction Language. Now PERL is an interpreted language. Interpreted language means that you do not have to compile a PERL program in to a machine code and then execute it. There are two ways you can run a program. The conventional way you write a program in C compile it in to an object code or machine code then execute the machine code. The second way is that the way languages like Java or PERL works or operates. You write the program in some language well of course in java we have to compile. In PERL we do not have to compile it at all. The program remains in the PERL's source code form.

There is an interpreter program PERL interpreter which takes the source program directly and executes the PERL code. This is how an interpreter works it takes the source code directly and statement by statement it tries to interpret and execute. And PERL is a language which has been optimised for scanning text files, extracting information from them and printing reports which are the main requirements and needs for web based applications. In particular you will see PERL has very powerful string handling features. String handling means string comparisons string replacements and a number of other related features and this is available on all platforms. You can have PERL installed on a machine without any problem.

(Refer Slide Time: 03:19)

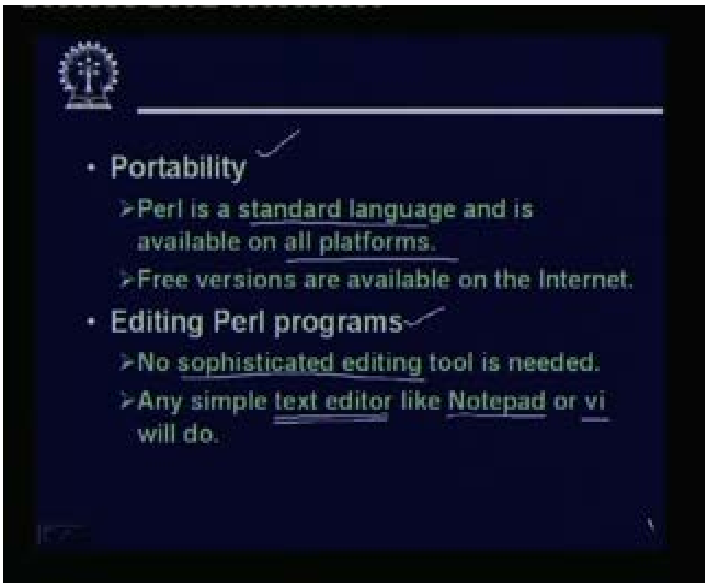


Main Advantages

- Speed of development
  - You can enter the program in a text file, and just run it. It is an interpretive language; no compiler is needed.
- It is powerful
  - The regular expressions of Perl are extremely powerful.
  - Uses sophisticated pattern matching techniques to scan large amounts of data very quickly.

The main advantages of PERL are several. First is of course is speed of development. Since PERL is interpretive you need not have to compile a program every time you make some modifications. You can simply type the means type and enter the program and simply run it through the interpreter. There is no additional overhead required for compilation. As I said it is very powerful for some application domains. There is something called regular expressions in PERL which will shall see later. It is an extremely powerful feature and this lends the main advantage of PERL which regards to the string handling capability. It uses sophisticated pattern matching techniques to scan large amount of data very quickly.

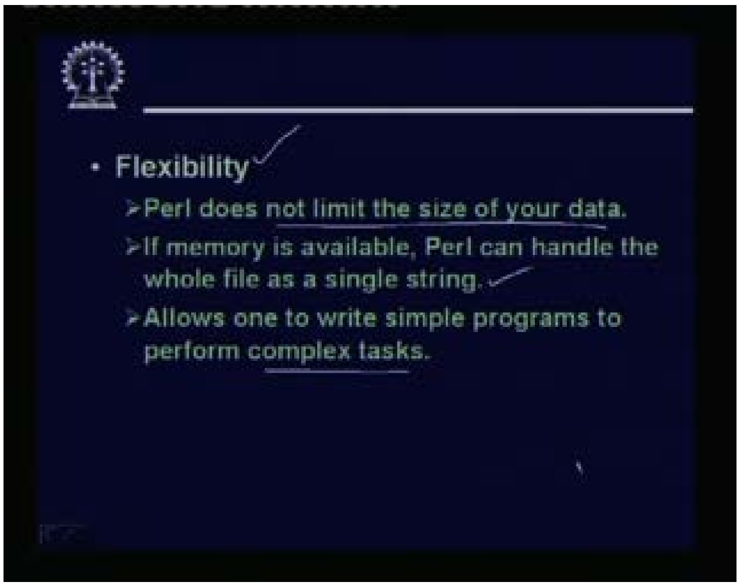
(Refer Slide Time: 04:13)



- Portability
  - Perl is a standard language and is available on all platforms.
  - Free versions are available on the Internet.
- Editing Perl programs
  - No sophisticated editing tool is needed.
  - Any simple text editor like Notepad or vi will do.

PERL is portable, it is a standard language. All implementations are based on this standard. It is available on all platforms; Windows, UNIX, Linux, Macintosh, everywhere and the second most important thing is that this is the language you need not have to purchase. It is available for free on the internet. You can download the PERL interpreter on a machine and you can start using or writing a PERL programs. For creating PERL programs you do not need any specialized editor. You do not need any sophisticated editor. You can use any simple text editor like Notepad in Windows or vi under Unix or Linux to create PERL programs.

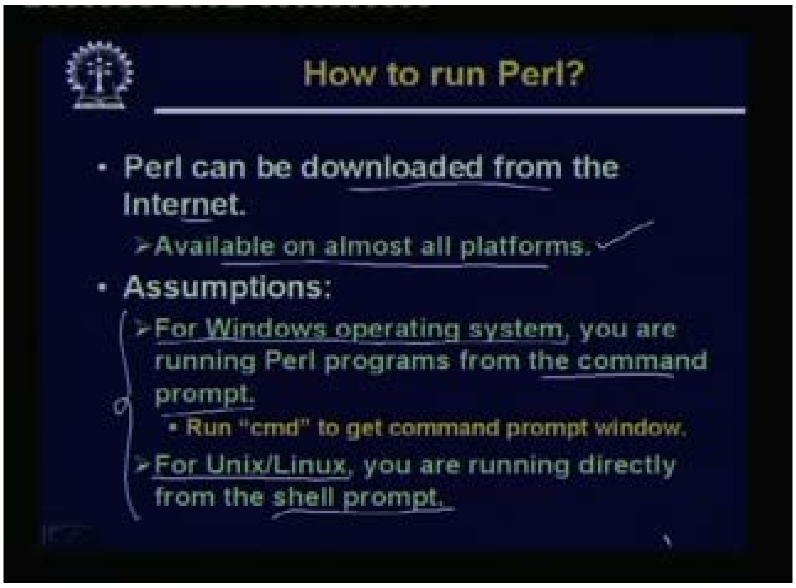
(Refer Slide Time: 05:05)



- Flexibility
  - Perl does not limit the size of your data.
  - If memory is available, Perl can handle the whole file as a single string.
  - Allows one to write simple programs to perform complex tasks.

Flexibility is another thing. The way PERL is defined, PERL does not limit the size of the data which means I am giving an example in PERL, and you can define a string. Now a string can be one word, it can be one line. The entire content of a file can also be regarded as string and it can be stored in a string variable. PERL does not impose any limit to the max size. You may be having 50 Megabyte long file that entire file content you can assign it to a single string variable. As long as you have sufficient memory available, virtual memory is sufficient PERL will not create any problem for you. This is what I have said and this because of this flexibility you can write some program to perform very complex task without worrying about memory availability and other problems.

(Refer Slide Time: 06:07)



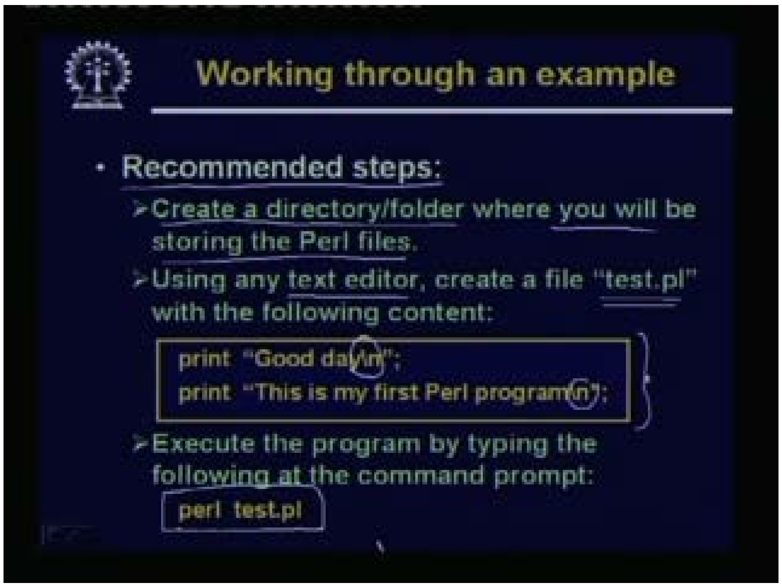
How to run Perl?

- Perl can be downloaded from the Internet.
  - Available on almost all platforms.
- Assumptions:
  - For Windows operating system, you are running Perl programs from the command prompt.
    - Run "cmd" to get command prompt window.
  - For Unix/Linux, you are running directly from the shell prompt.

Now the next question is how to run PERL? As I said PERL can be downloaded from the internet, there are number of websites available. And versions are available to run on almost all platforms. Now one of the examples that we have shown here that is will show the [06:32 word not clear] assumptions are. Well if you are trying to run PERL under the Windows operating system, then I am assuming that you are running them from the command prompt. Command prompt comes whenever you go to the start button on the left bottom you select run and then you enter cmd or command depending on your Windows version and a command prompt window comes you type the commands directly on that window.

So here all the examples that I have shown we are assumed that you are typing the commands directly on the command window. And if you are a doing it on a UNIX or Linux system, then the commands you are typing directly from the shell prompt. So we have made this assumption because of both the cases the appearance will be similar. We have windows where you directly typing the commands and getting back your results. This is unlike the most other windows application where you have very fancy menus and buttons. You click on the menus using mouse select something. No here we are not assuming Windows here we will be creating the files in text editors. We will be giving command on the so called command prompt line and we will be getting the result from there directly.

(Refer Slide Time: 08:00)



Working through an example

- Recommended steps:
  - Create a directory/folder where you will be storing the Perl files.
  - Using any text editor, create a file "test.pl" with the following content:

```
print "Good day\n";
print "This is my first Perl program\n";
```

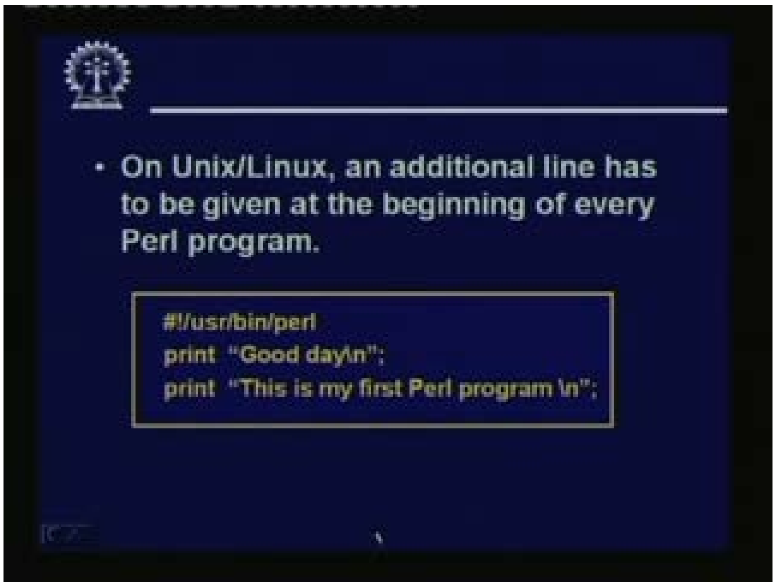
  - Execute the program by typing the following at the command prompt:

```
perl test.pl
```

Let us take a small example and see how we can go through this example and run, run it execute it. These are the recommended steps because some of the steps are optional. But it is always a good practice to do this. For example the first step it is always good to create a separate directory or folder where you will be storing the PERL files. Otherwise your PERL files will get mixed up with a rather different kind of files. So it is always good to have a separate folder where in which all the PERL files will be stored. Now using any text editor you create a file. For example give the name as test.pl which will contain these two lines. Print within double-quotes, something print within double-quotes something.

Now if you look at these two lines I think this is. This is absolutely self-explanatory. First one prints the character string and the reverse slash n like in C. This means a new line character. So good day will be printed, and then on the next line “This is my first PERL Program” will be printed then on the next line will get printed. Now this program can be typed. It can be saved in a file text.pl and you can execute the program directly by typing this particular command on the command prompt PERL test.pl. PERL will be PERL is the name of the PERL interpreter when you give the PERL command, PERL interpreter will start running test.pl is the input program to the PERL interpreter. PERL interpreter will be reading the commands. From there it will be executing it directly. So executing PERL program is as simple as that.

(Refer Slide Time: 09:56)

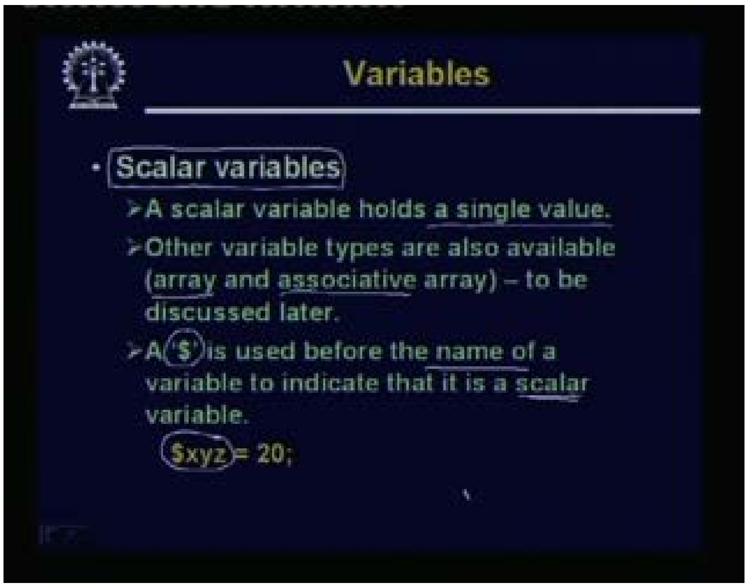


- On Unix/Linux, an additional line has to be given at the beginning of every Perl program.

```
#!/usr/bin/perl
print "Good day\n";
print "This is my first Perl program \n";
```

Well if you are working on UNIX or Linux, you need one additional line at the beginning. This tells you in which directory the PERL interpreter is stored. So as you can see this syntax, this is hash the exclamation sign, then the path name. Say for example I have given usr bin PERL this is the complete path name for the PERL interpreter. So in UNIX or Linux you normally require this so that when you execute, this kind of program you need not invoke the PERL interpreter directory. You directly run this program and the first line of the program will tell you which program to call it, will be called automatically and that program will be taking the other commands and executing them this is exactly the same. As executing shell script from a UNIX command prompt because in a shell script you typically give something like usr bin sh where sh is the name of the shell interpreter. Here instead of sh you are giving Perl. So using this you can execute the programs.

(Refer Slide Time: 11:27)



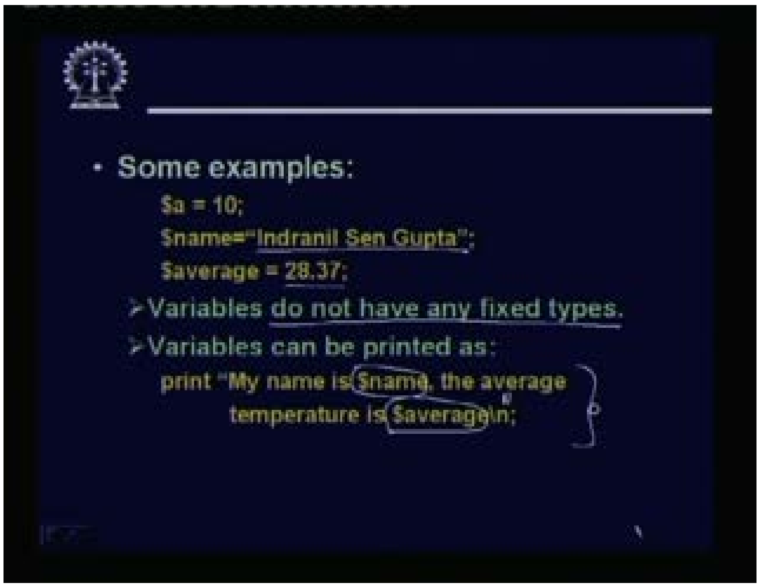
Variables

- Scalar variables
  - A scalar variable holds a single value.
  - Other variable types are also available (array and associative array) – to be discussed later.
  - A $ is used before the name of a variable to indicate that it is a scalar variable.

    $xyz = 20;

Now let us come to the details of this language. Some of the constructs we start with the concept of variables in Perl. Now in Perl, broadly there are three kinds of variables Scalar, array and hash. The second two we would be looking at later for the time be look at something called Scalar variables. Scalar variables as the name implies it can hold a single value. So as I said and hash are associative these will be considered later. The way of defining a variable is that a dollar sign before the name indicates that, this is a variable for example dollar xyz is a variable. It is a Scalar variable dollar indicates it is a Scalar. If you give dollar xyz is equal to 20, it means you are assigning a value 20 to this Scalar variable dollar xyz.

(Refer Slide Time: 12:41)



- Some examples:

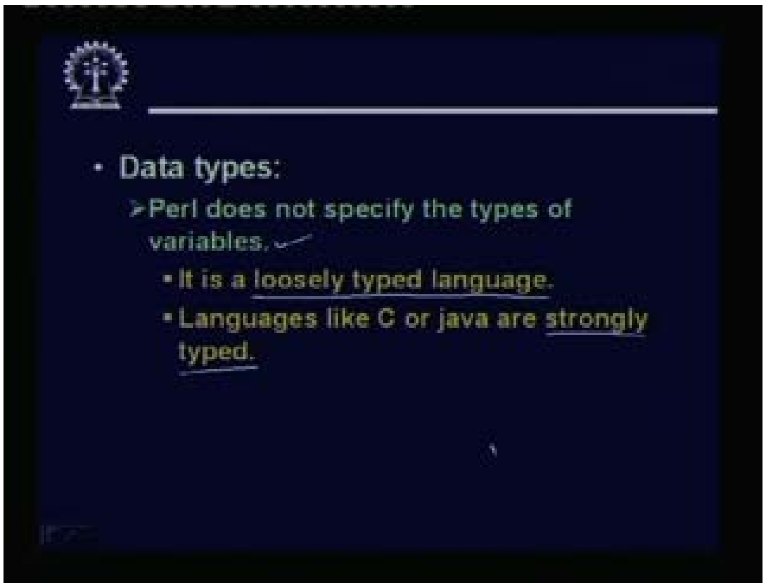
```
$a = 10;
$name="Indranil Sen Gupta";
$average = 28.37;
```

- Variables do not have any fixed types.
- Variables can be printed as:

```
print "My name is $name, the average
        temperature is $average\n";
```

Some more examples dollar a equal to10. Dollar name equal to within double-quotes, this is string Indranil Sen Gupta, dollar average equal to “28.37”. Now you can see from these examples so may you must wonder and is in fact true that in PERL variables do not have any fixed types. There is nothing like this variable is integer variable. This is character, this is floating point. Well when you store something in a variable name everything is stored as string. When you process them as you the need requirements it can be treated as a number. If we are using a variable in an arithmetic expression, then the content of the variable is treated as a number or otherwise it will be treated as string. So the context in which you are using a variable will determine its type how it is interpreted explicitly. The user is not specifying which variable is holding which type of data. Variables can be printed like this, will talk about this later. Print within double-quotes “My name is $name the average temperature is $average”. There will be a closing quote here. So just like php I took the same example. If this dollar name appears within double-quote it will replace by its value. Similarly here, so exactly what we printed is “My name is Indranil Sen Gupta the average temperature is “28.37”. This is what we printed. This is what we have mentioned.

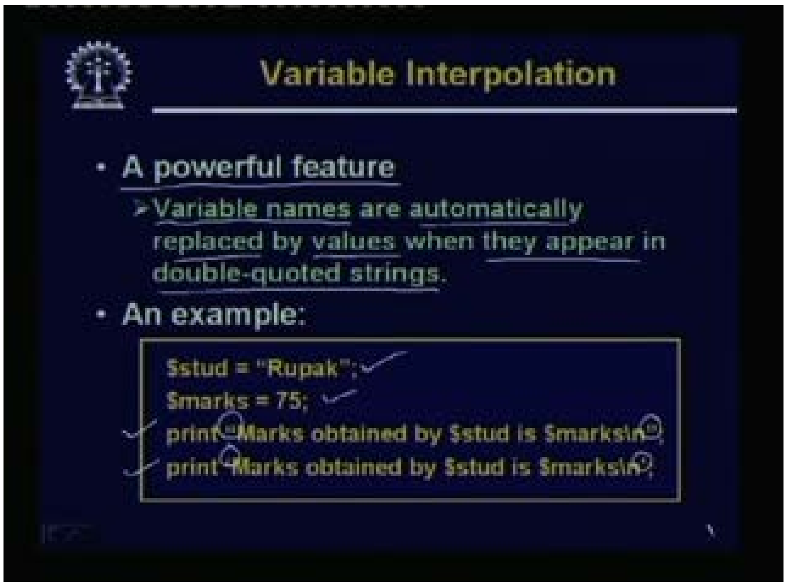
(Refer Slide Time: 14:37)



- Data types:
  - Perl does not specify the types of variables.
    - It is a loosely typed language.
    - Languages like C or java are strongly typed.

Perl does not specify the type of variables. That is why it is sometimes called loosely typed language to distinguish it from languages which are so called strongly typed. Strongly typed means they have very strong type declaration syntax or rules in Java or C each and every variable must be assigned or defined with a type and it must abide by that type at all. Time that particular type at all points of time you cannot change the type of a variable in between. But in PERL you can in general.

(Refer Slide Time: 15:20)



Variable Interpolation

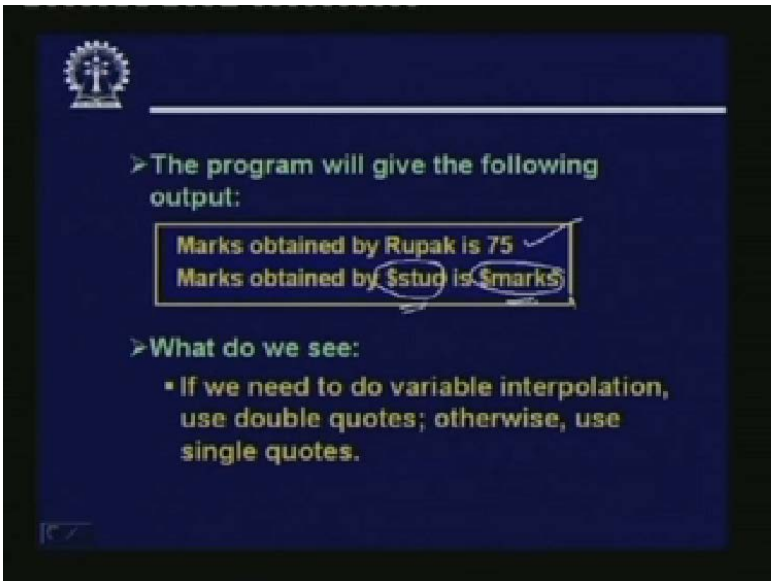
- A powerful feature
  - Variable names are automatically replaced by values when they appear in double-quoted strings.
- An example:

```
$stud = "Rupak";
$marks = 75;
print "Marks obtained by $stud is $marks\n";
print 'Marks obtained by $stud is $marks\n';
```

Variable interpolation I had just shown, it as an example. Earlier a couple of slides back. But let us look at it. More formally, this is very powerful feature in Perl. This says that variable names where ever you use them they will be automatically be replaced by their corresponding values whenever they appear in double-quoted strings. This means if you have a string enclosed in double-quotes and within this string a variable appears. Then that variable will be replaced by the contents of that variable whenever. That string is processed, you may be printing that string you may be doing something else with the string also. [16:15 word not clear] Similarly by counting the number of characters in the string, for example the value will be replaced and then the processing of the string will start.

Now another small example, the first line here is assigns a string Rupak to a variable dollar stud. Second line assigns a number 75 to a variable dollar marks. There are two print statements which are very similarly it say "Marks obtained by dollar stud is dollar marks" with a new line reverse slash n. The only difference is the first print statement uses double-quotes, while the second print statement uses single-quotes. This is the only difference. But in terms of semantics of the meaning this single quote and double quote really makes some difference here. Here we are talking about variable interpolation where a variable is replaced by a value this interpolation occurs only when you are using double-quotes.

(Refer Slide Time: 17:35)



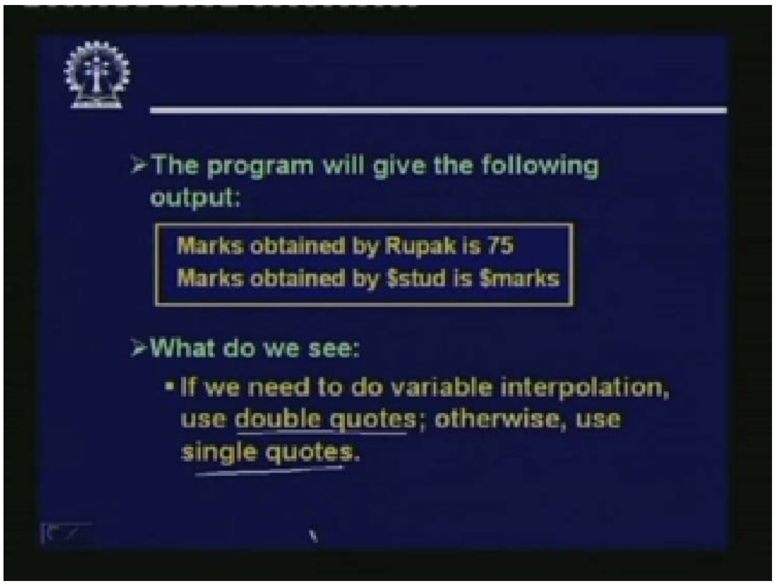
- The program will give the following output:

```
Marks obtained by Rupak is 75
Marks obtained by $stud is $marks
```

- What do we see:
  - If we need to do variable interpolation, use double quotes; otherwise, use single quotes.

Like for the previous example, the first print statement will give an output. Marks obtained by Rupak is 75. The second one will give an output Marks obtained by dollar stud is dollar marks which means the replacement of variables by their values have not taken place. They are taken the 4 names of the variables are regarded as strings. Let us just go back to the program once more. To see Marks obtained by dollar stud is dollar marks in the first one dollar stud is replaced by Rupak. This is replaced by 75. But in second one it does not. So in a string if we need to do variable interpolation, we use double quotes otherwise use single quotes.

(Refer Slide Time: 18:31)



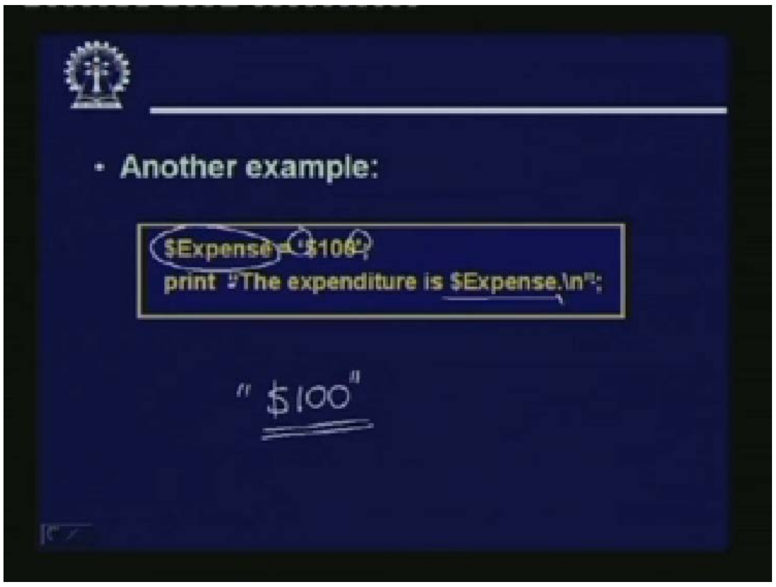
- The program will give the following output:

```
Marks obtained by Rupak is 75
Marks obtained by $stud is $marks
```

- What do we see:
  - If we need to do variable interpolation, use double quotes; otherwise, use single quotes.

But if it is simple sting, without any embedded variable, then single or double quote does not make any difference. You can use either of them.

(Refer Slide Time: 18:42)

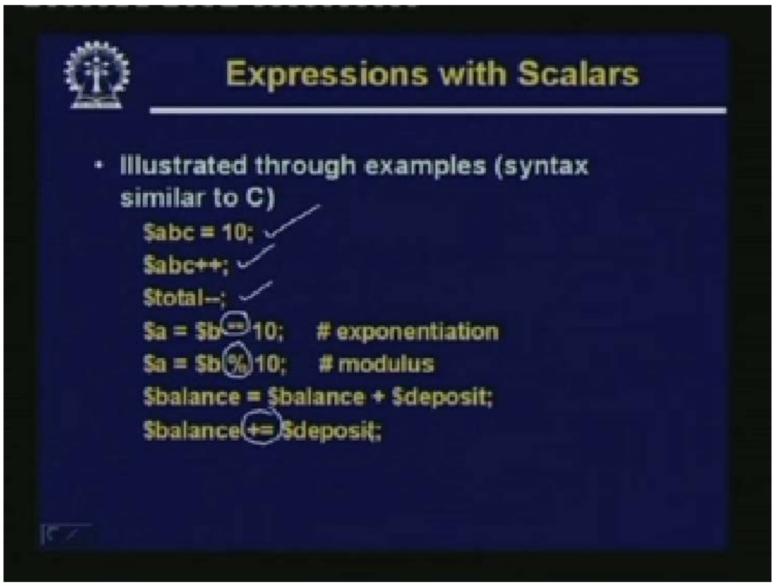


- Another example:

```
$Expense = '$100';
print "The expenditure is $Expense.\n";
```

This is a particular example where string contains the currency symbol dollar 100. Now here we have to use single quote. Because if we use double quote $100 like this, then the PERL interpreter will be misled. PERL interpreter will regard $100 as a name of a variable and try to replace the value of the variable which is not true. Since this is not a variable at all, it is always safe to enclose it within single quote and not confuse the PERL interpreter. And when you are printing the content “The expenditure is $expense” within double quotes, so actually what will be printed is “The expenditure is $100” because dollar 100 is the contents of this variable. So this tells you when you want to use or means when you need to use double quote and when you need to use single quote.

(Refer Slide Time: 19:55)



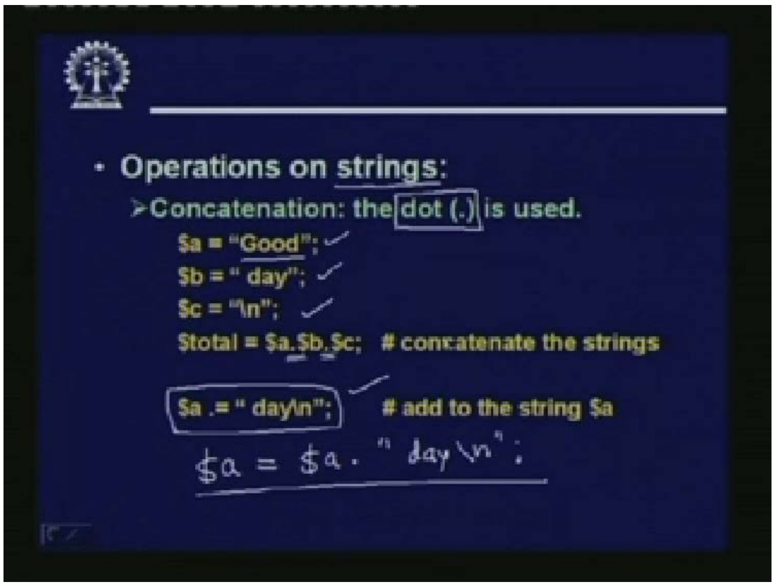
Expressions with Scalars

- Illustrated through examples (syntax similar to C)

```
$abc = 10;
$abc++;
$total--;
$a = $b ** 10;     # exponentiation
$a = $b % 10;      # modulus
$balance = $balance + $deposit;
$balance += $deposit;
```

So let us now look at how expressions can be written down in Scalar variables while syntax is very similar to C. So I am not spending much time on this. I am just giving a few examples to illustrate. What all things you can do? Dollar abc is a variable, $abc is equal to 10. You have auto increment and auto decrement dollar abc plus plus dollar total minus minus. Here exponentiation is indicated double star dollar b star star 10 means this variable raised to the power of 10. This gets assigned to dollar. A percentage means modulus dollar b percentage 10 means you divide b by 10 and the remainder whatever is coming that is assigned to that is the modulus. Balance is equal to balance plus deposit or you can also write in the convention C short form. Balance plus equal to deposit. So almost all the things which you can do in C you can do in PERL as well the syntax is all most the same.

(Refer Slide Time: 21:15)

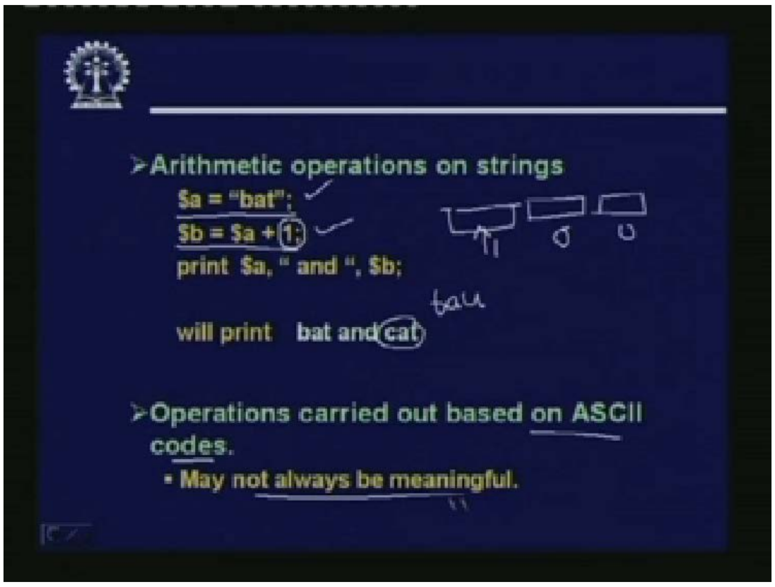


- Operations on strings:
  - Concatenation: the dot (.) is used.

```
$a = "Good";
$b = " day";
$c = "\n";
$total = $a.$b.$c;    # concatenate the strings

$a .= " day\n";       # add to the string $a
$a = $a . " day\n";
```

Talking about strings, there are few additional operations you can do on strings which are convenient. Like let us take an example. See there are three strings we have defined good, blank day and new line reverse slash n. These are the three strings we can carry out string concatenation using the dot operator. You can write dollar total, dollar a dot, dollar b dot, dollar c. So the three strings get concatenated and will get a single string good day with a new line which is assigned to total. Well if you look at this statement, this says dollar a dot equal to this dot equal to means dollar. This is the short form dollar a equal to dollar a dot day slash n. This also creates the same string dollar a contain good. Good is concatenated with day new line. So good day new line is assigned to the same variable dollar a, this is doing this.

(Refer Slide Time: 22:41)



➢Arithmetic operations on strings

```
$a = "bat";
$b = $a + 1;
print $a, " and ", $b;
```

will print bat and cat

bau

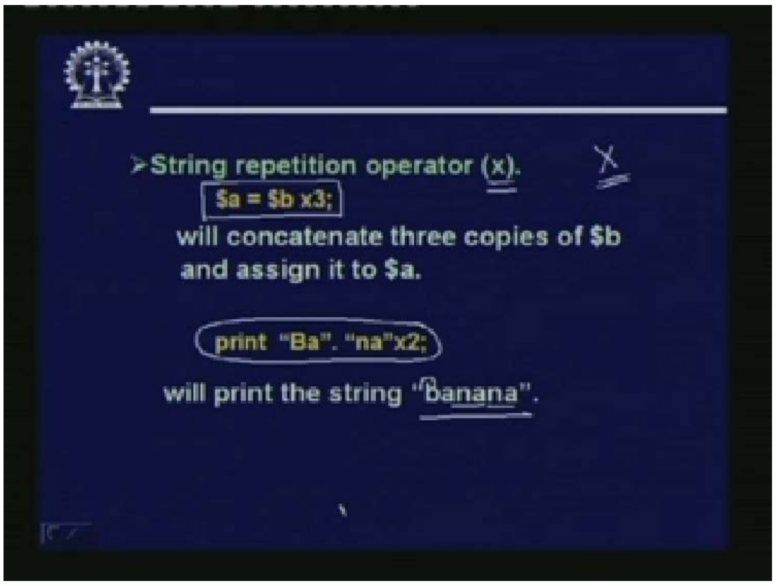
➢Operations carried out based on ASCII codes.

- May not always be meaningful.

So we sometimes do arithmetic operations in strings but of course very rarely but PERL supports it. Let us see what it means in terms of semantics. Say there is a string dollar a, which we have assigned a value bat b a t bat to it. Then you give a statement dollar, dollar b is equal to dollar a plus. Dollar a plus one is an arithmetic addition you want to do. That means you asking to add one to a string now. Now means actually what will happen here is that this fellow one will be added to the ANCII covalent of the number number represented by a string. So actually if you add this one, this well b a t will become not really cat. Cat will be done. If you add to the first one it will actually become b a u.

The last one to make it b a t you have to compute the corresponding number that you will have to make an addition here because ultimately b a t will get stored as three ASCII characters. So to make that in to cat you have to add one to the first [24:01 word not clear] and 00. Here it will be 1 and 00, so accordingly you have to find out the number to be added so that the first byte gets incremented. So these operations are based on ASCII but usually we do not use this because this kind of arithmetic operations on strings are not very meaningful and it also sometimes misleads the designer or reader of the program. So it is not a good practice to use this.

(Refer Slide Time: 24: 37)



➢String repetition operator (x).

```
$a = $b x3;
```

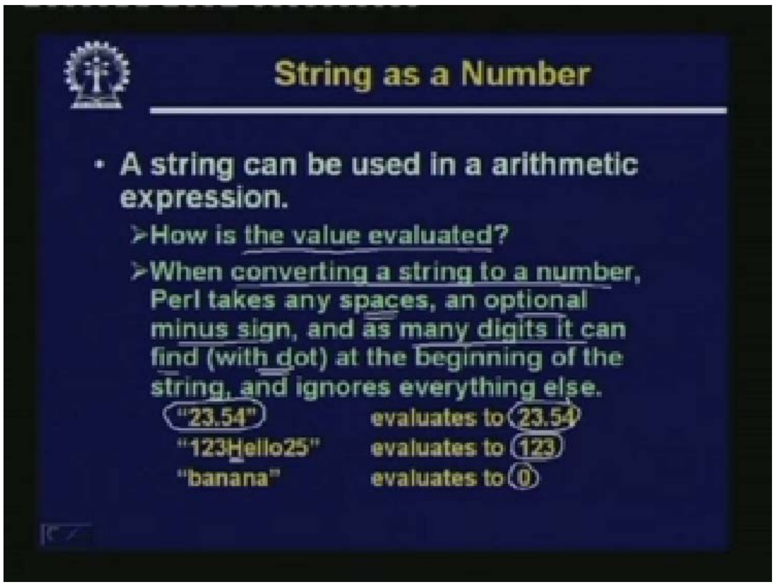
will concatenate three copies of $b and assign it to $a.

```
print "Ba". "na"x2;
```

will print the string "Banana".

There is another interesting string operator this is called the string repetition operator x. Well you can repeat a string by a number of times following x. Like you take this example, dollar a equal to dollar b x3. X3 means dollar b, whatever is the content of this will be repeated, three times. This will concatenate three copies of b and assign it to a take this specific example print within double quotes Ba, then concatenation then n a repeated twice. So actually it will be printing the string bana of course this b would be capital it will be printing this string banana. This n a will be appearing twice. So many we may need to do this kind of string repetition. We can use this x operator for the purpose.

(Refer Slide Time: 25:40)



String as a Number

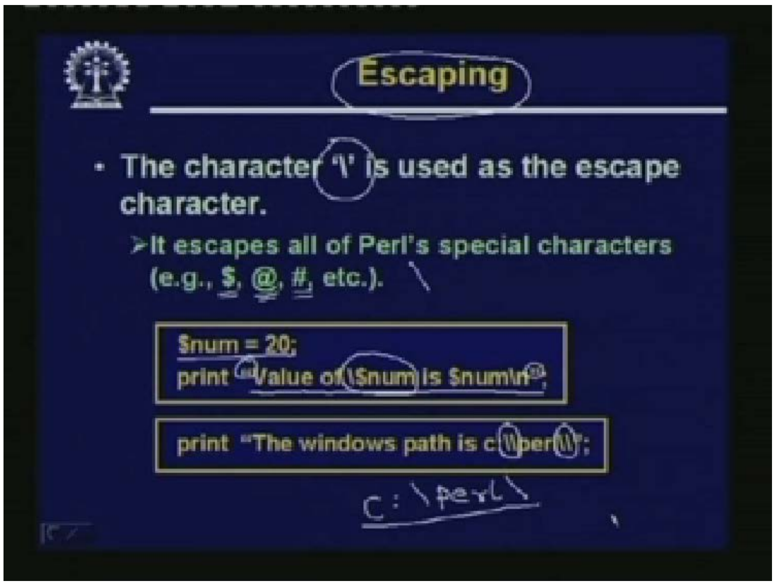
- A string can be used in a arithmetic expression.
  - ➢How is the value evaluated?
  - ➢When converting a string to a number, Perl takes any spaces, an optional minus sign, and as many digits it can find (with dot) at the beginning of the string, and ignores everything else.

"23.54"	evaluates to 23.54
"123Hello25"	evaluates to 123
"banana"	evaluates to 0

Now string sometimes can be represented or evaluated as a number also in terms of an arithmetic expression. Now if it is a full fledged arithmetic expression other than simple increment of decrement, the question arises if it is string. How do we evaluate the value?

There are some very different rules before using it in a full-fledged arithmetic expression. PERL converts the string to a number. When converting it to a number PERL will take in the beginning if there are any spaces it will consider that. If there is optional minus sign at the beginning it will also consider that and as many digits it can find with a possible decimal point at the beginning. But after that it will ignore everything else. For example if it is “23.54” it will take the whole thing “23.54” as a number. If it is 123 Hello 25, then it will scan from the left. It will get 123 and after that there is some non-numeric character. So it will stop here. It will take the value as 123. If it is banana there is no numeric character. So the value will be taken as 0. So these are the results or these are the rules using which a string if used in an arithmetic expression can be converted in to a number.

(Refer Slide Time: 27:18)



Escaping

- The character '\' is used as the escape character.
  - It escapes all of Perl's special characters (e.g., $, @, #, etc.).

```
$num = 20;
print "Value of \$num is $num\n";
```

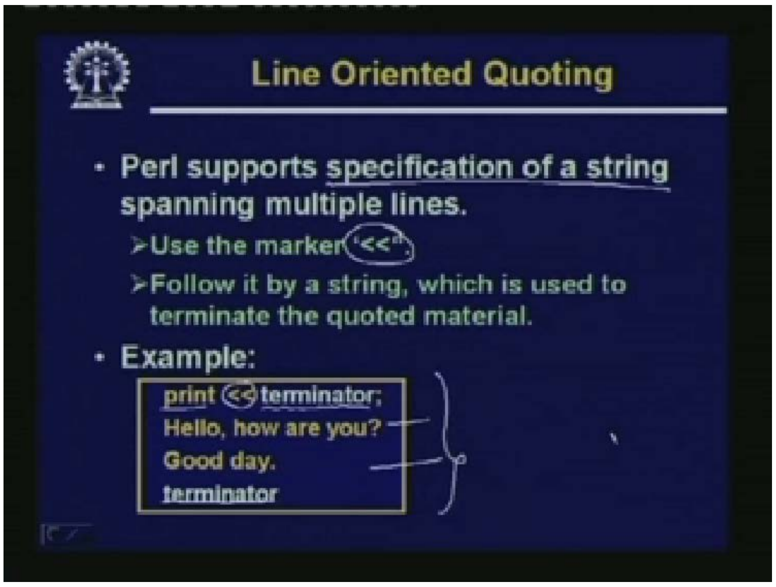
```
print "The windows path is c:\\perl\\";
```

c:\perl\

Now sometimes when you are defining strings we need to have escape sequences because sometimes special characters in PERL there are number of escape characters. There are number of special characters. So dollar ampersand hash reverse slash is also there. There are many such special characters which are there in Perl. So if you want such special characters to come as a part of a string, then we need to use an escape sequence. Now in PERL the reverse slash character is used as the escape sequence. Like you take this example, first one dollar num equal to 20. Print Value of reverse slash dollar num is dollar num. You see you have double quotes. So you need variable interpolation, but what you require in the output is the output you want that is should come as value of dollar num is 20.

So in one case you are replacing it by its value in the other case you are not. So when you do not want to replace it or do not want the dollar to be treated as special character precede it by a slash. So this is treated as a normal dollar as part of the string and will be printed as Value of dollar num is num. And suppose you want to print a path the path name will be like this c colon slash PERL slash. Slash is a special character, again so in a string if you want to write this string, then this reverse slash has to be escaped by proceeding with the escape character reverse slash. So there will be double reverse slash at the end again double reverse slash. So in a string whenever you use some special character which you want to be treated as an ordinary character. It is better to use or you must use an extra reverse slash at the beginning.

(Refer Slide Time: 29:48)



Line Oriented Quoting

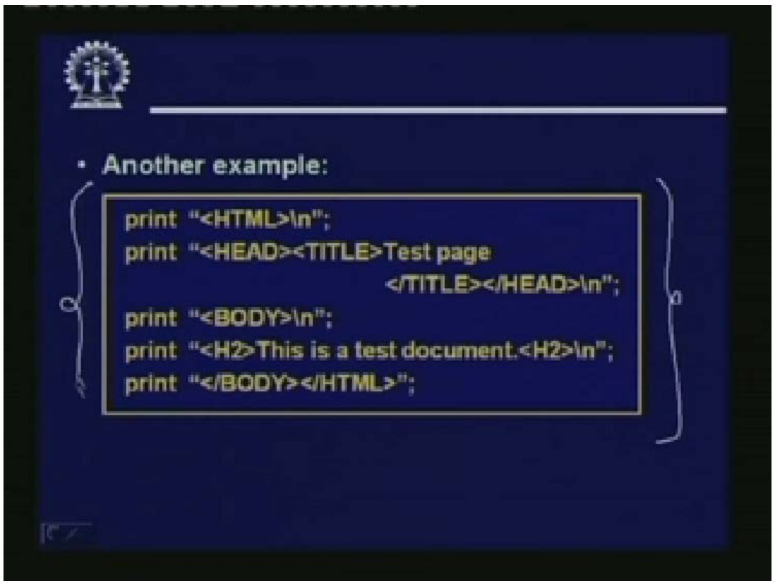
- Perl supports specification of a string spanning multiple lines.
  - Use the marker "<<".
  - Follow it by a string, which is used to terminate the quoted material.
- Example:

```
print <<terminator;
Hello, how are you?
Good day.
terminator
```

Now there is something interesting. Here there is something called Line Oriented Quoting which simplifies writing something in PERL sometimes will see an example here. Here actually what we are talking about is we are talking about specification of string. Normally in PERL when you specify a string we specify by enclosing it with either double quotes or single quotes. Now if it is very long string, then enclosing it in quotes like this becomes inconvenient. Long string means a string with spanning ten hundred thousand lines very big string. So there is a mechanism which is called Line Oriented Quoting. This uses the marker double less than. Double less than the way its use is like this. You give a print comm sup. Suppose I want to print a very long string, you give a print command give, this double slash follow it up with a string.

Say here I have chosen a string called terminator some word. After this I can write down the lines composed comprising my string one by one. Finally whenever this particular string terminator appears again the string is treated as being finished. So the same thing I could have written as print within quote "Hello, how are you" within quote good day with a reverse slash at the end. So if there are 100 lines, you would have written the word print 100 times. But using this double less than mechanism, using this kind of Line Oriented Quoting we can give the print only once followed by the string itself without any quotes or anything. With a delimiting marker at the end, this is a very convenient way of specifying a string.

(Refer Slide Time: 32:15)

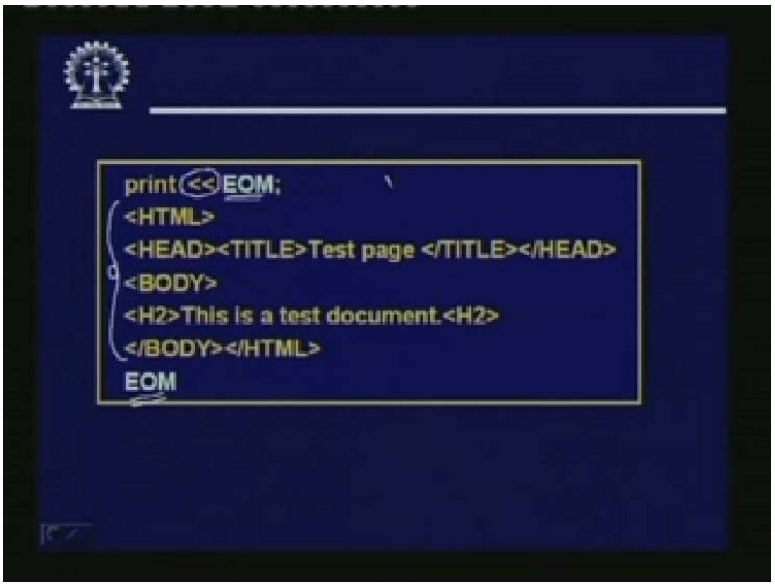


- Another example:

```
print "<HTML>\n";
print "<HEAD><TITLE>Test page
                    </TITLE></HEAD>\n";
print "<BODY>\n";
print "<H2>This is a test document.<H2>\n";
print "</BODY></HTML>";
```

I am giving an example in the particular context of designing cgi scripts. Look at this small PERL program snippet. PERL is trying to output some very small HTML document using print with double quotes HTML. New line print HEAD TITLE Test Page end TITLE end HEAD, new line print body, new line print H2. This is a test document H2 new line print BODY HTML new line. This you can understand what it is doing. PERL is generating an output of very small HTML document. Now here we have to write so many statements. We have to write some quotes. At the end of each line you have to write a new line slash n semi colon so on.

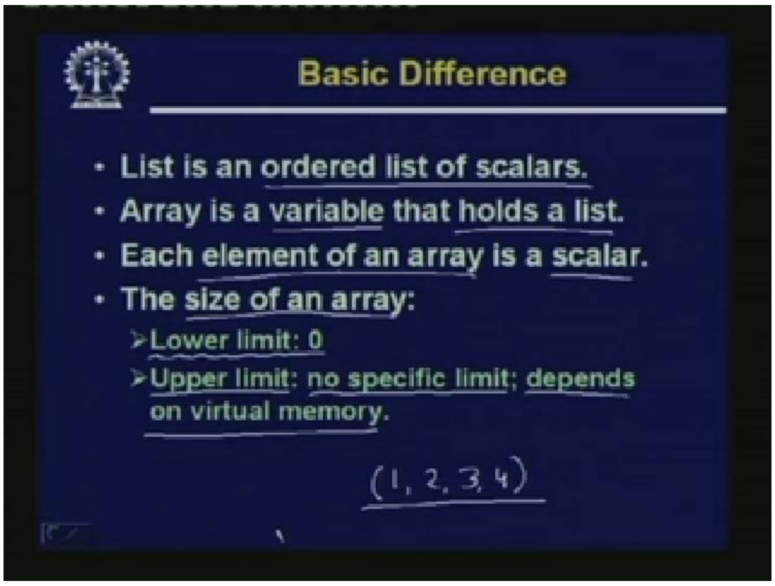
(Refer Slide Time: 33:22)



```
print<<EOM;
<HTML>
<HEAD><TITLE>Test page </TITLE></HEAD>
<BODY>
<H2>This is a test document.<H2>
</BODY></HTML>
EOM
```

So if you look at the alternate version of this same specification which uses the double less than construct. Print double less than some marker EOM. We choose here EOM is somewhere here between that I have straight away written the HTML page as if it is a normal text without any quotes or anything. This entire thing will be treated as a string and it will print. As you can see in the context of writing cgi scripts this kind of a thing will occur frequently will be having lot of lines to get printed. Verbatim and if we have a mechanism like this, your program looks much more neater and if you also need to type lot less and the chances of errors are also less quotes, double quotes semi colon you may be missing something in between. Now let us talk about something called Lists and Arrays in Perl. Now we have so far talked about Scalar variables where a variable can hold a single value. Now that single value can be a number the single value can be a string can be anything. But lists and arrays we shall see it is some sort of collection of scalars, a number of such Scalar quantities if you combine together. You can call it a List or Array.

(Refer Slide Time: 35:03)



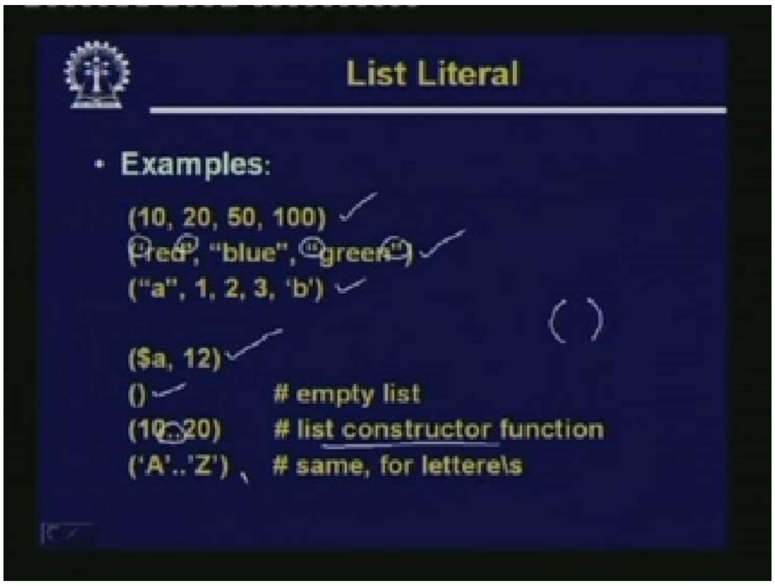
Basic Difference

- List is an ordered list of scalars.
- Array is a variable that holds a list.
- Each element of an array is a scalar.
- The size of an array:
  - Lower limit: 0
  - Upper limit: no specific limit; depends on virtual memory.

(1, 2, 3, 4)

Let us try to understand the basic difference between a List and an Array. List is an ordered list of scalars. Suppose if you have a List of scalar variables say 1, 2, 3, 4, this is a list. Array is a variable that holds the list. Just try to understand the difference list is the contents. Certain numbers arranged in an ordered fashion ordered fashion means I can say this is the first element. This is the second element, this is the third element, certain items arranged in an ordered fashion. That is a List and a List can be stored as a variable as a special variable called an Array variable. So array represents a variable which can store or hold a List and a list is an ordered collection of Scalars. So as I said each element of an array is a scalar. Now like C the index of an array starts with 0 and an array in the, you can say in the lowest case can have zero elements. This is the smallest array or an empty array in the maximal case. There is no specific limit that PERL imposes. You can have an array with one million entries, you can have an array with one billion entries, and you can go on doing this as long as the virtual memory of the system supports it. PERL does not impose any restriction on this. So the size of the array lower limit is zero, upper limit there is nothing that PERL imposes.

(Refer Slide Time: 37:01)



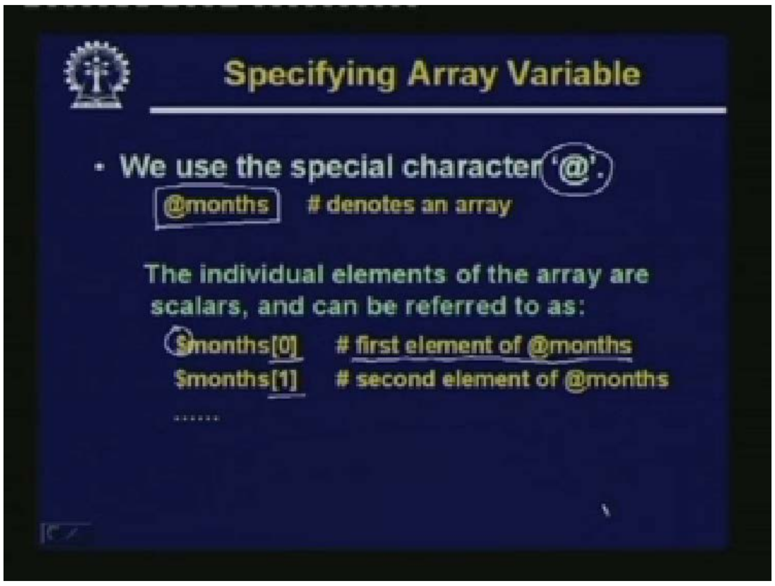
List Literal

- Examples:

```
(10, 20, 50, 100)
("red", "blue", "green")
("a", 1, 2, 3, 'b')

($a, 12)
()              # empty list
(10..20)        # list constructor function
('A'..'Z')      # same, for letters
```

Here are some examples. Some examples of list literals or constants. First one defines a list with four elements 1, 20, 50, 100. Second one red, blue, green. So as you can see these are simple strings does not matter whether I include them within single quote or within double quote it does not matter. Now a list can contain any mixer of strings and numbers. Here is a list with five elements. There are some other notations. This one, this is also a list where the first element of the list is the content of the scalar variable dollar a. The second element of this is 12. Here this is an empty list, if we give the two brackets without anything between, just like this represents an empty list. But we can use the dot dot operator as a list constructor, 10.. 20 actually means 10, 11, 12, 13, 14, 15, 16 up to 20. So instead of writing all these numbers separated by comma, I can straight away write 10.. 20, this will mean that there 11 elements in the array in the list. Similarly I can do the same for the characters, A.. Z means all the characters from A to Z. All 26 letters, so if you have this kind of regular values that you want to assign to the elements of a list. You can use this kind of constructed operators using dot dot operation.

(Refer Slide Time: 39:05)



Specifying Array Variable

- We use the special character '@'.

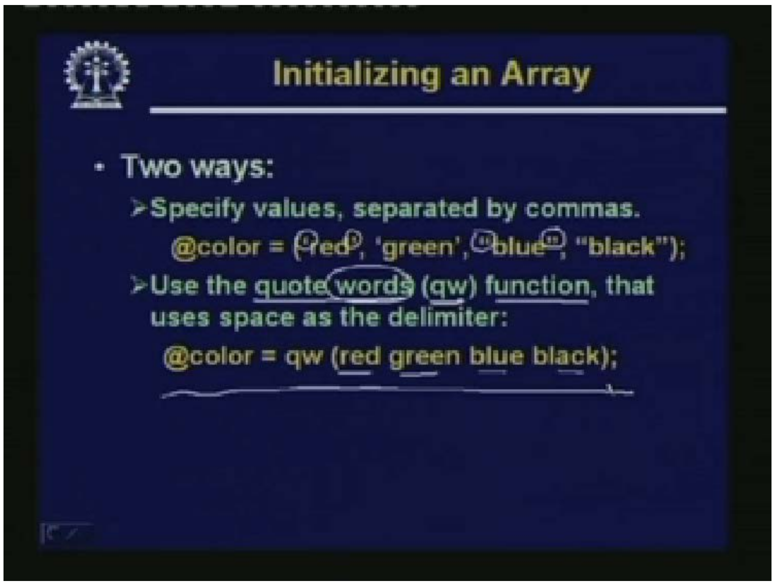
```
@months     # denotes an array
```

The individual elements of the array are scalars, and can be referred to as:

```
$months[0]    # first element of @months
$months[1]    # second element of @months
......
```

Now this is how you can define a list. Now let us come to array variables that said an array variable can hold a list. Now PERL uses a special character to uniquely identify an array. So any variable which starts with at the rate symbol is regarded as an array variable. This at the rate months this denotes an array. This array can have many elements. Now the individual elements of the array, they will obviously be scalars because array is a list. When I talk about the first element, second element, third element of the array, they are individually scalars, they are not arrays. So when we talk about the individual elements of the arrays the way we refer the indices is very similar to C within square bracket. The index number and index number start with zero just like C. But the point to note is that when I want to refer to first element of the array, I start it with a dollar because although ampere, this means at the rate months is an array dollar months zero is a scalar. Because when you are talking about the first element it is a scalar and in PERL any scalar value or any reference to scalar quantity must precede with a dollar. So if it denotes one element of an array it has to precede with a dollar to indicate that this is a scalar quantity.

(Refer Slide Time: 40:57)

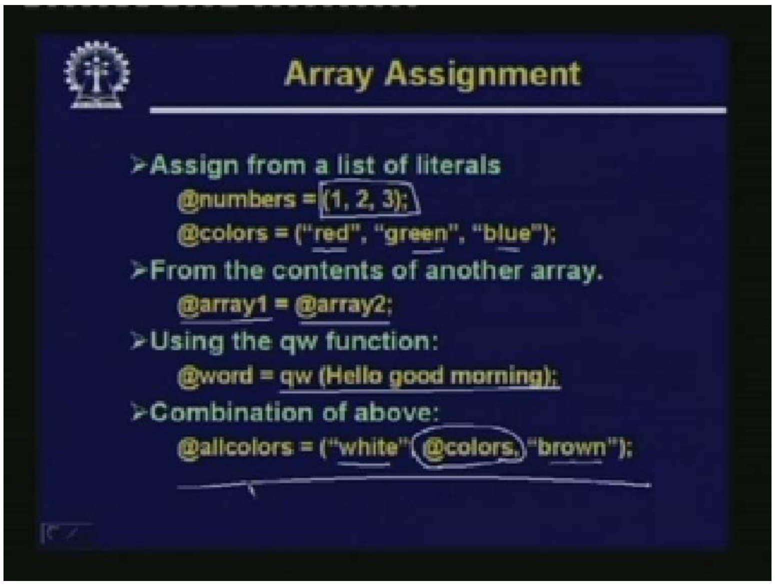


Initializing an Array

- Two ways:
  - Specify values, separated by commas.
    @color = ('red', 'green', "blue", "black");
  - Use the quote words (qw) function, that uses space as the delimiter:
    @color = qw (red green blue black);

So now let us see or let us look at some of the simple operations on an array that how you can initialize an array with some known values. Broadly there are two ways. First way is to specify values separated by commas. For example at the rate colour equal to red, green, blue, black, again for some cases you can use single quote. For some cases we can use double quote, no problem. This can also contain numbers 1, 2, 3, 4, 5, whatever you want this is the first way of representing or specifying the value. But one problem is that you have to type a lot. So many quotes, so many commas, there is a more convenient way using the quote words of qw function. This quote words or qw is used more conveniently when you are specify an array. But elements are of string type single word string. That is why it is called words. Multi word strings are not allowed here. Well you can use it like this. Color this array, color equal to qw within bracket you simply write down. The values of the array elements separated by spaces red, green, blue, black. So the element number zero will get red, element number one will get green, element number two will get blue and the last one will get black. So these are the two ways in which you can initialize the value of an array.

(Refer Slide Time: 42:36)

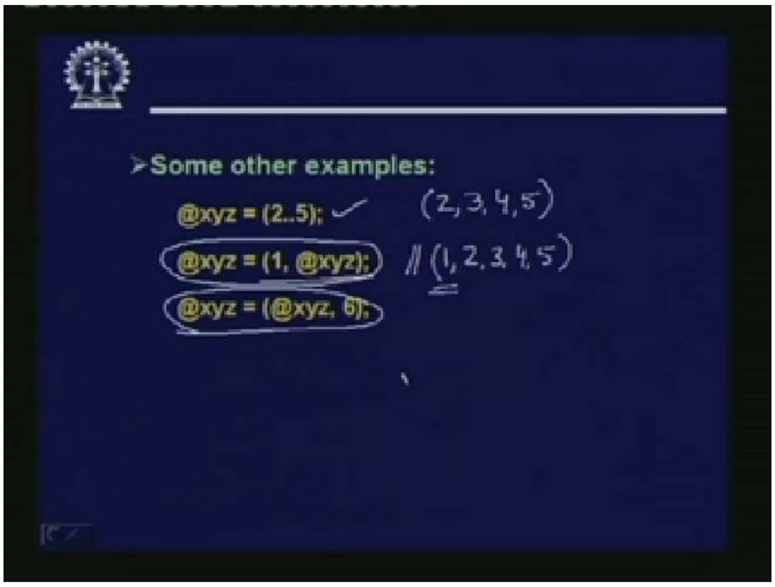


Array Assignment

- Assign from a list of literals
  - @numbers = (1, 2, 3);
  - @colors = ("red", "green", "blue");
- From the contents of another array.
  - @array1 = @array2;
- Using the qw function:
  - @word = qw (Hello good morning);
- Combination of above:
  - @allcolors = ("white", @colors, "brown");

Now some array assignment just like you can assign something to a scalar. You can also assign something to an array like at numbers equal to in bracket 1, 2, 3. This is a list, so this list will be assigned to this array. So the new value of numbers will be 1, 2, 3. Similarly colours equal to red, green, blue. You can assign the contents of one array to another at array. One equal to at array two, you can also assign something using the qw function, qw something equal to or you can use any combination of the above. Like as this example suggests you see at all colors equal within bracket, you are specifying the list first and the third elements are scalar while the second element is an array. So before being assigned this array will be expanded. This at colours means, for example red, green and blue. So actually what will be assigned? It will a five element list white red, green, blue, brown. So when you are assigning, you can specify an existing array so that all the values of that array will also go in to the new array along with the new values as this example showed.

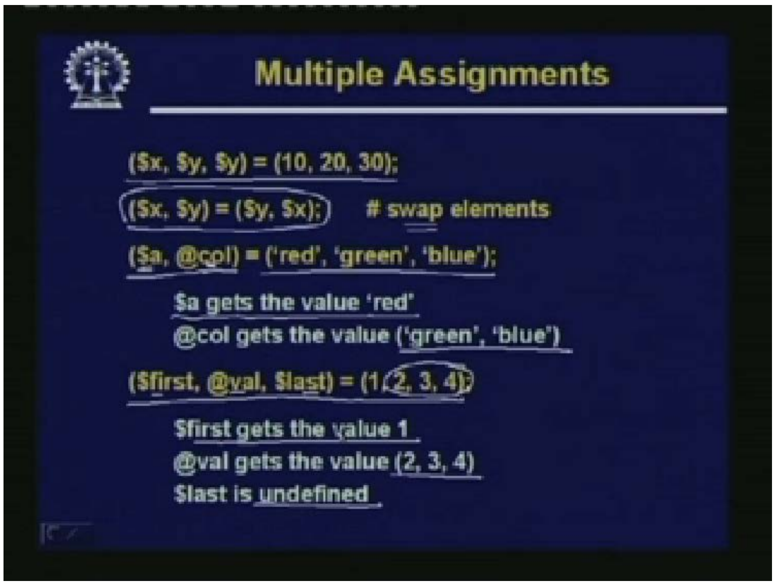
(Refer Slide Time: 44:03)



- Some other examples:
  - @xyz = (2..5); ✓ (2, 3, 4, 5)
  - @xyz = (1, @xyz); // (1, 2, 3, 4, 5)
  - @xyz = (@xyz, 6);

Some other examples, first one element 2, 3, 4 and 5 will go. Second one see this is interesting. In the first case the array value will be 2, 3, 4 and 5. Second one, we are going to assign a list to the [44:27 audio not clear] list xyz. The list starts with a number or scalar one followed by the array xyz. So the first element will be one followed by xyz. Whatever it had xyz was 2, 3, 4 and 5. So this will be the new value of the array. You can say this is a way to insert a new element at the beginning of the array. This one has been inserted at the beginning. Similarly the last example inserts a 6 at the end of the array. So six will get inserted after xyz right. So this way you can simply do some insertion at the beginning or end of the array.

(Refer Slide Time: 45:06)

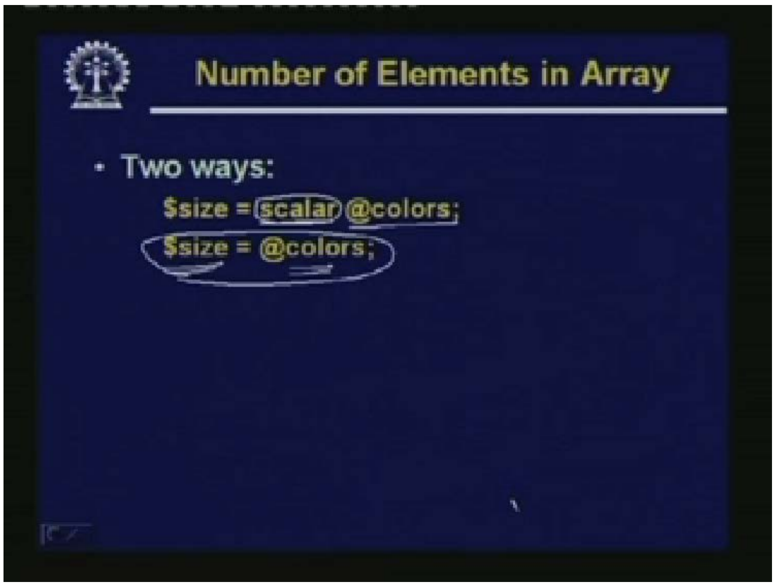


Multiple Assignments

```
($x, $y, $y) = (10, 20, 30);
($x, $y) = ($y, $x);     # swap elements
($a, @col) = ('red', 'green', 'blue');
    $a gets the value 'red'
    @col gets the value ('green', 'blue')
($first, @val, $last) = (1, 2, 3, 4);
    $first gets the value 1
    @val gets the value (2, 3, 4)
    $last is undefined
```

Multiple Assignments. Using a single statement you can have multiple assignments 10 to x, 20 to y, 30 to y, again so the final value of y will be 30. You can swap two elements using a single line y will go to x, x will go to y and this will be simultaneous. That is why swapping will actually takes place if you write an assignment like this, dollar a comma an array. So the first one is the scalar, second one is an array and there is a list on the right hand side. Since the first one is scalar a single value can go in to it. That is why dollar a will get the value of red and whatever remains will go in to this array col. So col will get the remaining which is green, blue. This is how multiple assignments will be taken. This is an interesting example take this one. First one is a scalar, second one is an array, third one is a scalar. You are assigning 1, 2, 3, 4, since the first one is a scalar, the first element will be assigned. So first will get the value one. Second one is an array. An array can take as much as you can give. So it will gobble up the remaining 2, 3, 4, whole of it. Last there is nothing to assign to, so last will remain undefined right. This is what will have here.

(Refer Slide Time: 46:59)



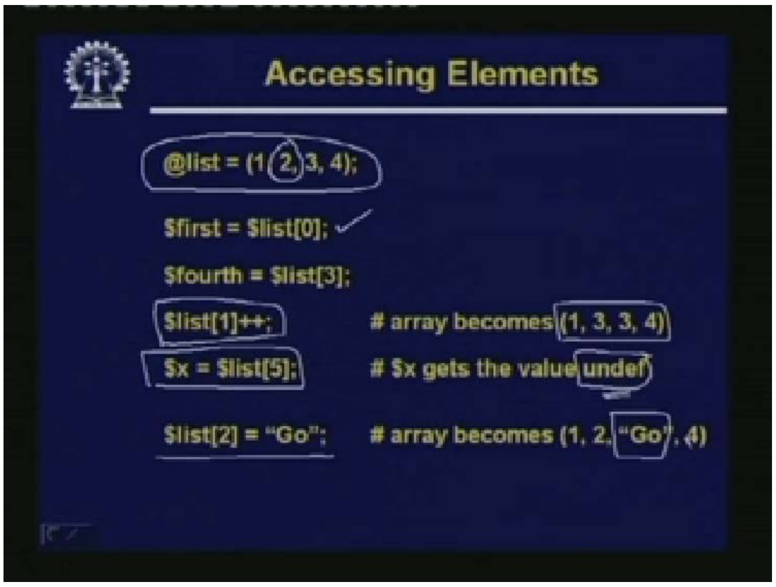
Number of Elements in Array

- Two ways:

```
$size = scalar @colors;
$size = @colors;
```

Number of Elements in Array. This if you want to in a particular program, there are two ways. One is a function called scalar or an operator called scalar. If you precede the array name by scalar it will return the number of elements or you can simply drop this keyword scalar and write a scalar equal to an array. An array cannot be assigned to a scalar, but if you write a statement like this, what will happen is that the number of elements in the array will be assigned to this scalar which is the size of the array. So there are number of ways in which two certain things in PERL as we can see, but you need to under understand the basic mechanism. To know that why these various alternatives work or they can be used, this alternative construct we shall see later because there are certain things you need to understand in order to appreciate its working. How it works? Why it works? These kinds of questions.

(Refer Slide Time: 48:10)

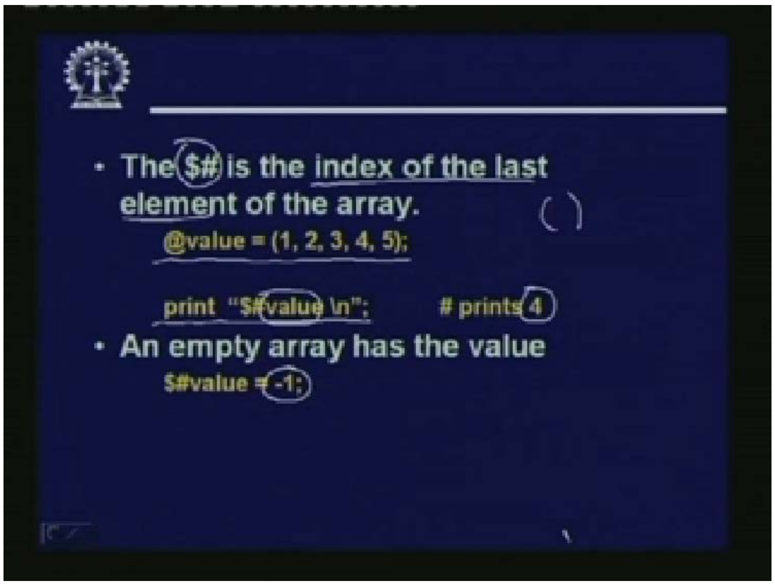


Accessing Elements

```
@list = (1, 2, 3, 4);
$first = $list[0];
$fourth = $list[3];
$list[1]++;          # array becomes (1, 3, 3, 4)
$x = $list[5];       # $x gets the value undef
$list[2] = "Go";     # array becomes (1, 2, "Go", 4)
```

Now talking about accessing elements in the array, suppose I have an array called list containing elements 1, 2, 3, 4, I can simply access elements like this. List zero means the first element which is one third means the fourth element which is 4. List three. List one plus plus means the one first element of the list, first means one that means the second one which is two. This will het incremented by one. So now the new list will be 1334. If you have a statement like this, where you are accessing an element which is out of bound, so it cannot return a value. So x will get the value undefined. Now this undef u n d e f is a special keyword which is used to indicate undefined values. So undef is it is like a constant or it is like a literal which is used to indicate undefined. If we give list two equal to a string Go, then list 2 is third element. So this string will replace the third element it will be 1, 2, Go and 4.

(Refer Slide Time: 49:31)



- The $# is the index of the last element of the array.

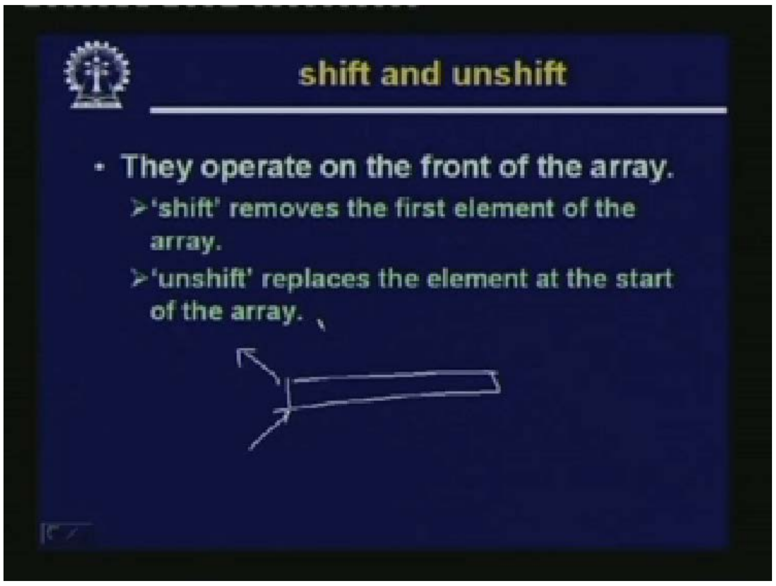
```
@value = (1, 2, 3, 4, 5);

print "$#value \n";        # prints 4
```

- An empty array has the value

```
$#value = -1;
```

There is a special marker indicated by dollar hash. Dollar hash represents the index of the last element of the array. Suppose you have an array value which contains 1, 2, 3, 4, 5. You can use dollar hash like this. Dollar hash followed by the name of the array without this at the rate this will return the value 4 which is the index of the last element of the array because index will be 0, 1, 2, 3 and 4. 4 is the last index, so 4 will be written. If the array is empty there will be no elements. If the array is just an empty array and if you give this dollar hash, name of the array, name of the array then it will return minus 1. Minus 1 indicates that the array is empty, zero means it has only one element because zero is the index of the last element.

(Refer Slide Time: 50:37)

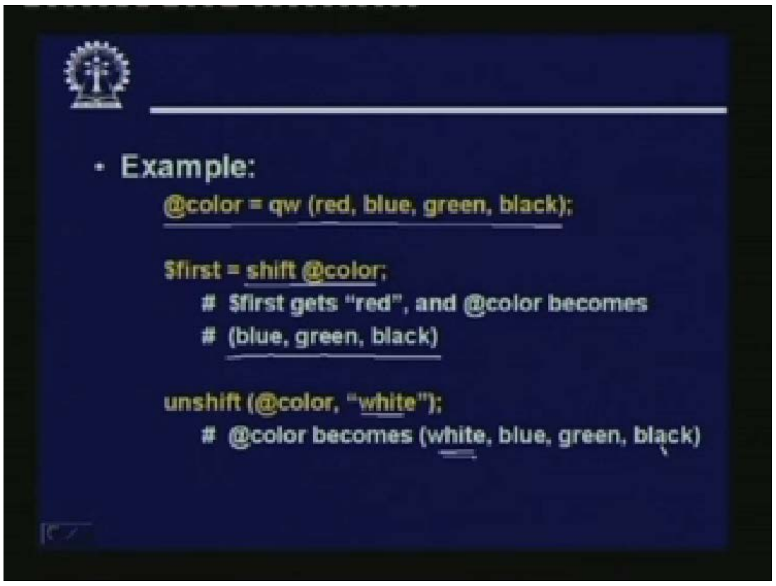


shift and unshift

- They operate on the front of the array.
  - 'shift' removes the first element of the array.
  - 'unshift' replaces the element at the start of the array.

There are some additional operations on arrays. Shift and unshift; given an array if you regard an array like this shift and unshift work. At the beginning of the array shift removes the first element of the array and unshift replaces the element at the start of array. Let us see how it will work.

(Refer Slide Time: 51:08)

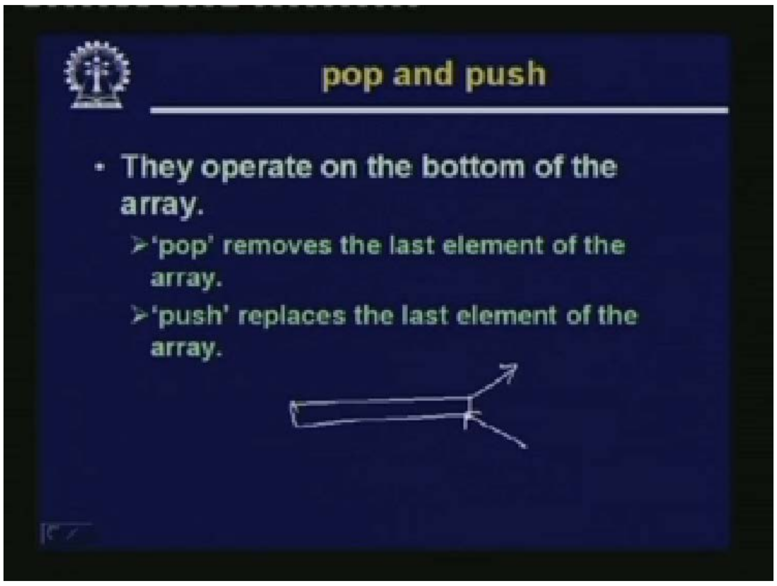


- Example:

```
@color = qw (red, blue, green, black);

$first = shift @color;
    # $first gets "red", and @color becomes
    # (blue, green, black)

unshift (@color, "white");
    # @color becomes (white, blue, green, black)
```

Take an example like this. If you give a shift like this, shift at color, so the first value red will get replaced and gets assigned to first and color will be now only blue, green, and black. Unshift is a function, if you give an array name with a value then white will replace the first element red, that will be white, blue, green, black.

(Refer Slide Time: 51:35)

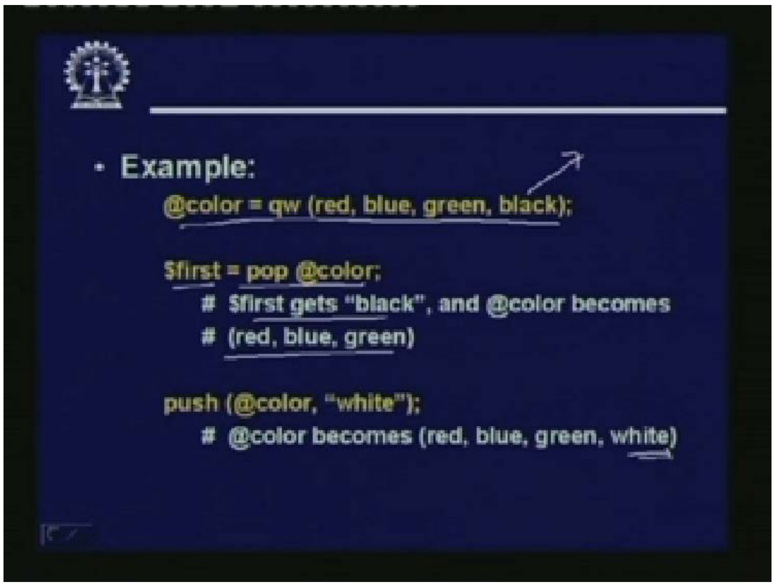


pop and push

- They operate on the bottom of the array.
  - 'pop' removes the last element of the array.
  - 'push' replaces the last element of the array.

Similarly there are a pair of functions pop and push which works at the tail of the array. At the end of the array pop removes the last element of the array while push replaces the last element of the array.

(Refer Slide Time: 51:53)

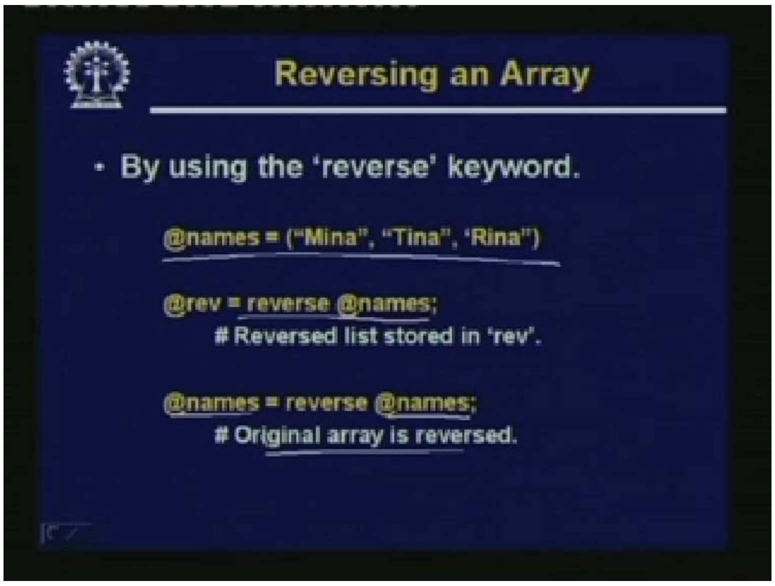


- Example:

```
@color = qw (red, blue, green, black);

$first = pop @color;
    # $first gets "black", and @color becomes
    # (red, blue, green)

push (@color, "white");
    # @color becomes (red, blue, green, white)
```

Some example again if color is red, blue, green, black, then if you give pop color, the last element black will get removed. There it will get assigned to color the first this is variable and the array will become only red, blue, green. But if you push white then this black will become white last one.

(Refer Slide Time: 52:18)



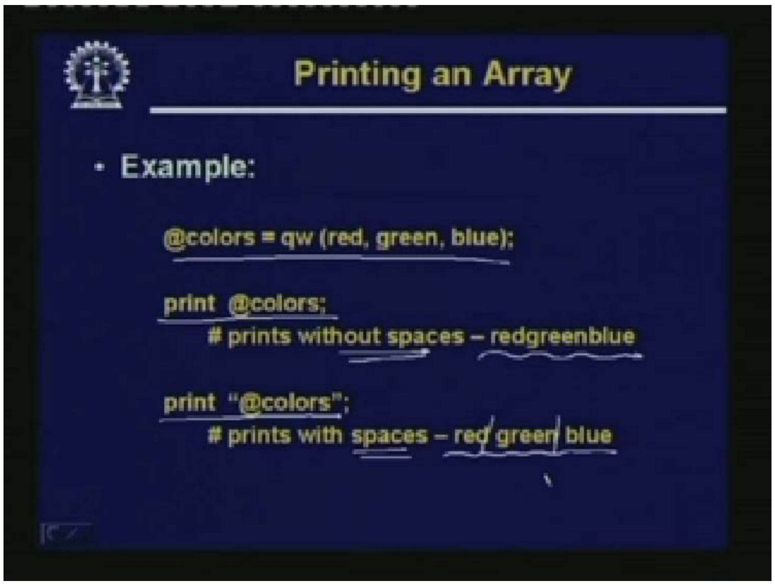
Reversing an Array

- By using the 'reverse' keyword.

```
@names = ("Mina", "Tina", 'Rina")

@rev = reverse @names;
    # Reversed list stored in 'rev'.

@names = reverse @names;
    # Original array is reversed.
```

So an array can be reversed by using the reverse keywords. Suppose you have an array like this, you can simply use reverse this array name. The entire array will be reversed, the elements will get reversed if I use this same name here and here then the original array will get reversed.

(Refer Slide Time: 52:58)



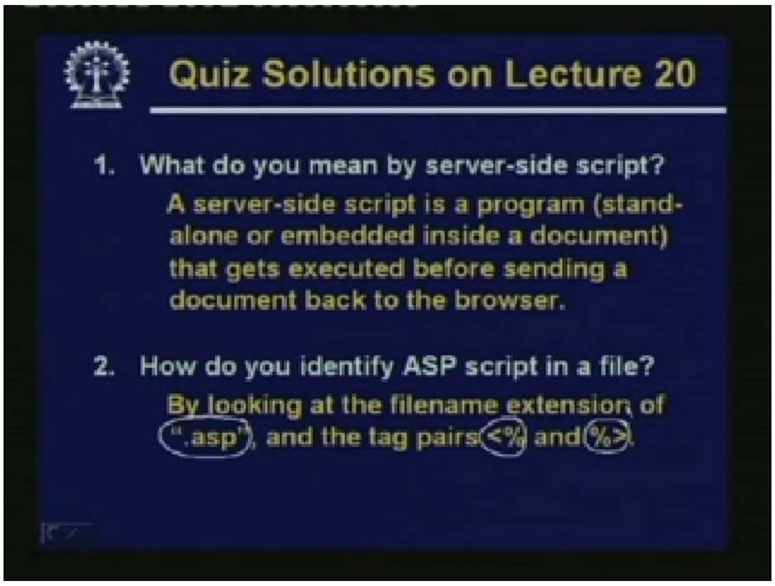
Printing an Array

- Example:

```
@colors = qw (red, green, blue);

print @colors;
    # prints without spaces – redgreenblue

print "@colors";
    # prints with spaces – red green blue
```

So printing an array is equally simple. If you have an array like this, you can simply write print at colors or within double quotes. If you do not, then printing will take without spaces. That means red, blue, green will come together if you give in double quotes. Then there will be a space between consecutive elements. So it depends on what do you want to do. Now with this we come to the end of today's lecture. Let us have a look at the solutions to last days lecture.

(Refer Slide Time: 53:16)



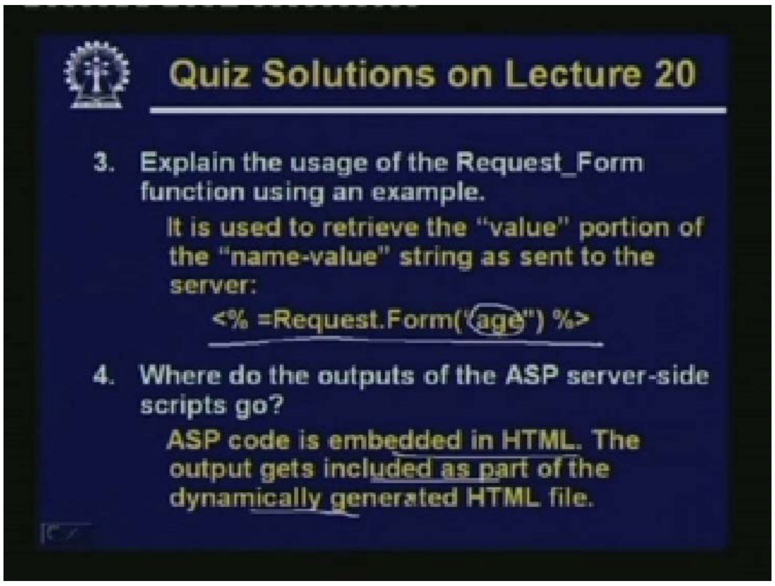
Quiz Solutions on Lecture 20

1. What do you mean by server-side script?
   A server-side script is a program (stand-alone or embedded inside a document) that gets executed before sending a document back to the browser.

2. How do you identify ASP script in a file?
   By looking at the filename extension of ".asp", and the tag pairs <% and %>.

What do you mean by server-side script?
A server is server-side script is a program. This can be standalone or it can be embedded inside a HTML document which gets executed before sending the document back to the browser it is executed at the end of the server.
How do I identify ASP script in a file?
First is that the file extension should be dot asp and these tag pairs should be present.

(Refer Slide Time: 53:44)



Quiz Solutions on Lecture 20

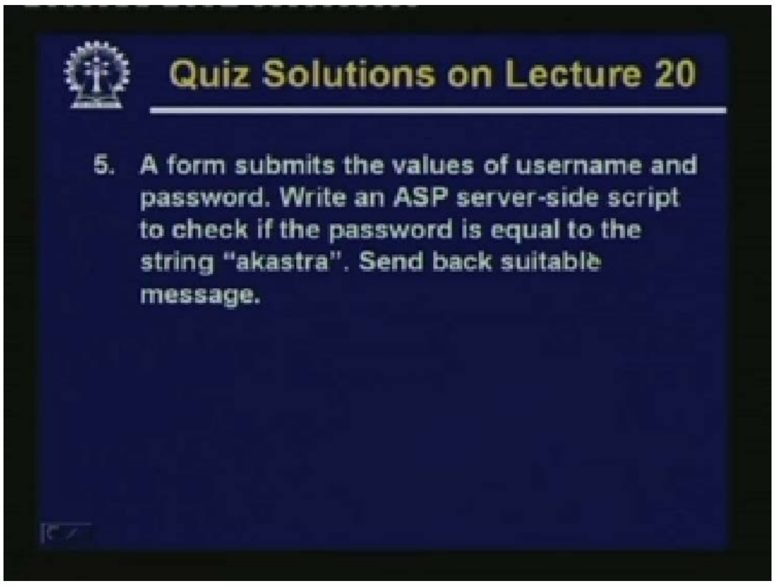
3. Explain the usage of the Request_Form function using an example.
   It is used to retrieve the "value" portion of the "name-value" string as sent to the server:
   <% =Request.Form("age") %>

4. Where do the outputs of the ASP server-side scripts go?
   ASP code is embedded in HTML. The output gets included as part of the dynamically generated HTML file.

Explain the usage of the Request_Form function using an example.
Request.Form function as said can be used like this. Just you give the name of form element. Here the value will be returned.
Where do the outputs of the ASP server-side scripts go?

They go now where; they get embedded in the HTML document in which ASP tags are embedded. The output get included as part of the HTML file.

(Refer Slide Time: 54:15)

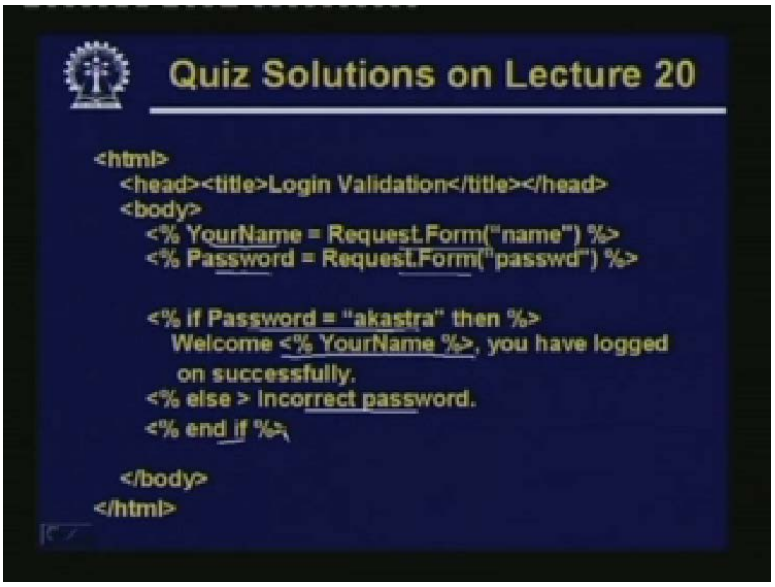


Quiz Solutions on Lecture 20

5. A form submits the values of username and password. Write an ASP server-side script to check if the password is equal to the string "akastra". Send back suitable message.

A form submits the values of username and password. Write as ASP server-side script to check if the password is equal to the string akastra.

(Refer Slide Time: 54:25)

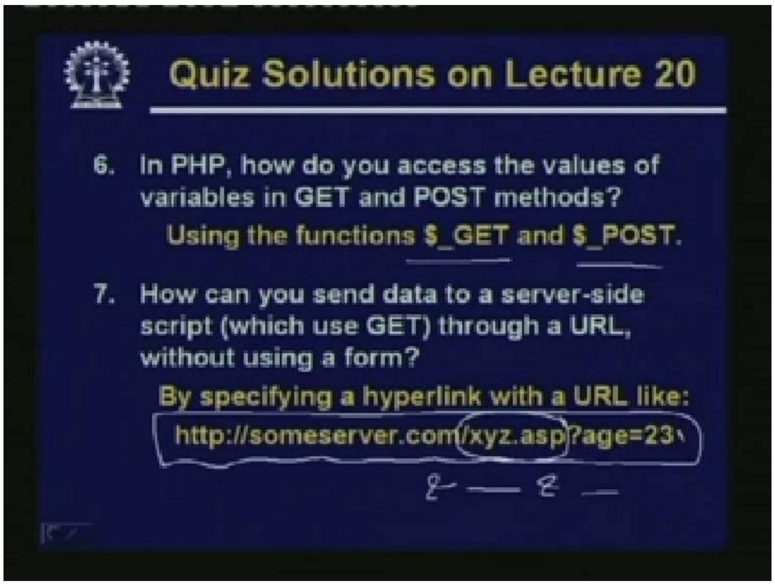


Quiz Solutions on Lecture 20

```
<html>
  <head><title>Login Validation</title></head>
  <body>
    <% YourName = Request.Form("name") %>
    <% Password = Request.Form("passwd") %>

    <% if Password = "akastra" then %>
      Welcome <% YourName %>, you have logged
       on successfully.
    <% else > Incorrect password.
    <% end if %>

  </body>
</html>
```

The code will be like this. So first you read the values of the name, password, through Request form. There is an if-statement, if password is equal to akastra, then "Welcome YourName" gets printed here within ASP tags "you have logged successfully" else prints a message error password or "Incorrect password" end if.

(Refer Slide Time: 54:50)

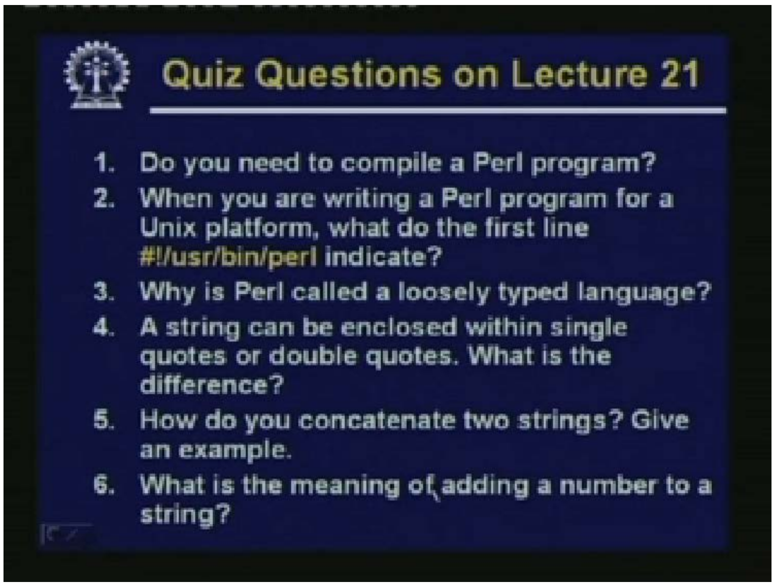


Quiz Solutions on Lecture 20

6. In PHP, how do you access the values of variables in GET and POST methods?
   Using the functions $_GET and $_POST.
7. How can you send data to a server-side script (which use GET) through a URL, without using a form?
   By specifying a hyperlink with a URL like:
   http://someserver.com/xyz.asp?age=23

In PHP how do you access the values of variables in GET and POST?
By using the functions dollar GET and dollar POST?
How can you send data to a server-side script through a URL without using a form?
Well you can use it by straight away coding the hyperlink like this. This is the URL you will be giving, http followed by a question mark, then age equal to 23, if there are more then you can give ampersand and give ampersand the other values. If you put the entire thing as a string then means it will go to server-side. This xyz asp with these values as a GET string. Now the questions from today's lecture.

(Refer Slide Time: 55:32)



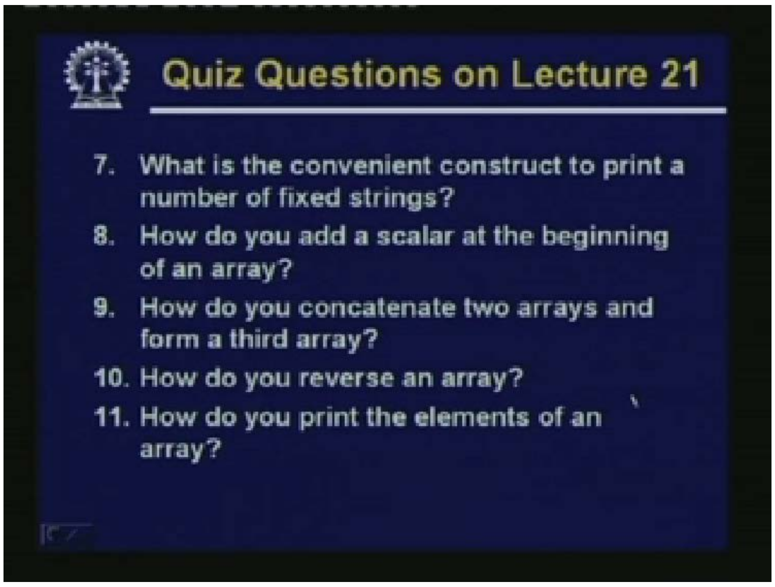
Quiz Questions on Lecture 21

1. Do you need to compile a Perl program?
2. When you are writing a Perl program for a Unix platform, what do the first line #!/usr/bin/perl indicate?
3. Why is Perl called a loosely typed language?
4. A string can be enclosed within single quotes or double quotes. What is the difference?
5. How do you concatenate two strings? Give an example.
6. What is the meaning of adding a number to a string?

Do you need to compile a PERL program?
When you are writing a PERL program for a UNIX platform what do the first line indicate?
Why PERL is called a loosely typed language?

A string can be enclosed within single quotes or double quotes what is the difference?
How do you concatenate two strings give an example?
What is the meaning of adding a number to a string?

(Refer Slide Time: 55:59)

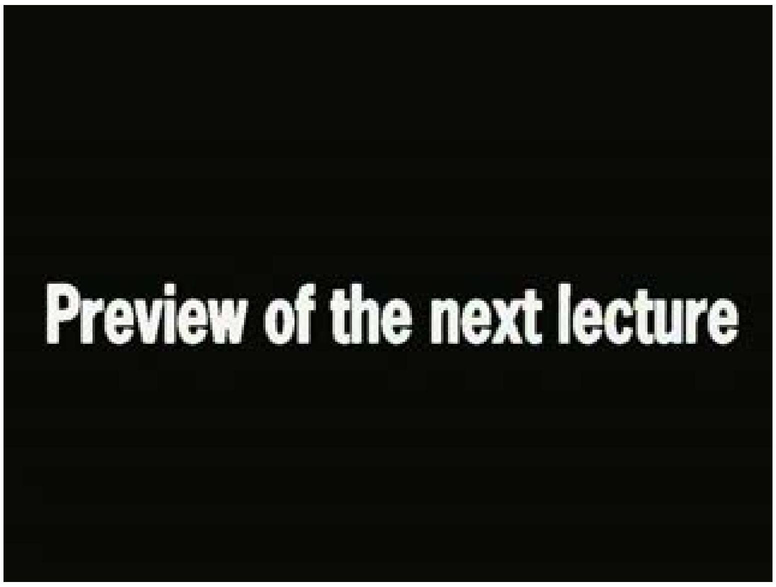


Quiz Questions on Lecture 21

7. What is the convenient construct to print a number of fixed strings?
8. How do you add a scalar at the beginning of an array?
9. How do you concatenate two arrays and form a third array?
10. How do you reverse an array?
11. How do you print the elements of an array?

What is the convenient construct to point a number of fixed strings?
How do you add a scalar at the beginning of an array?
How do you concatenate two arrays and form a third array?
How do you reverse an array?
How do you print the elements of an array?
With this we come to the end of today's lecture. In the next lecture we would be continuing with our discussion on Perl. We will be looking at some more features of the language conditional and other things. Thank you.

(Refer Slide Time: 56:33)



Preview of the next lecture

(Refer Slide Time: 56:35)

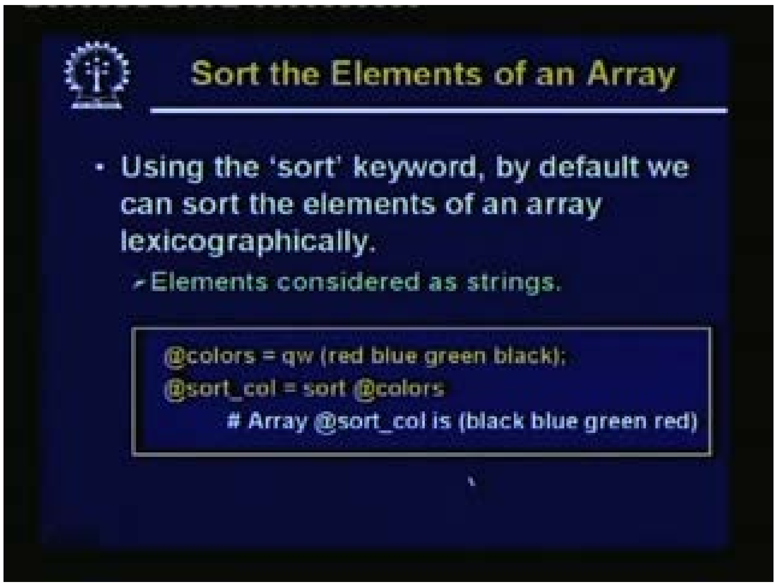
Lecture # 22

PERL - Part II

PERL – Part III

We continue with our discussion on the Perl language and its features. If you recall in our last lecture we were talking about the scalar and the array variables in the Perl. How it can be used? Today we continue with what we are talking about.

(Refer Slide Time: 57:00)



Sort the Elements of an Array

- Using the 'sort' keyword, by default we can sort the elements of an array lexicographically.
  - Elements considered as strings.

```
@colors = qw (red blue green black);
@sort_col = sort @colors
        # Array @sort_col is (black blue green red)
```

We look at some of the other facilities or other you can say features that you can have in association with an array. Let us start with sorting. Perl provides a very simple mechanism to sort the elements of an array using the sort keyword. Default sorting method is we can sort the elements lexicographically. Lexicographically means the elements of the array are treated as character strings and they will be sorted in the order of the dictionary; in the dictionary order. Dictionary sorting means lexicographic ordering. And the sort command by default considers that all elements are a string. For example if you write a number 25 it will considered as a string 2 and 5. So some example colors equal to say using the quote word command you define this array as red, blue, green, black. If you write sort col another array equal to sort colors. So the new array sort col will be this, will be sorted black blue green red right.