

**SEER AKADEMI
LINUX PROGRAMMING
AND SCRIPTING
PERL 2**

Hi everyone again welcome to this lecture on the Linux programming and scripting. I hope you are enjoying the class so far last week we started with the Perl programming this is when the course is getting serious so in terms of the programming itself ,so today we will be continuing the next discourse on the next programming focusing on the Perl last week we introduced the basic commands actually we saw how to actually get data from the terminal or some external input and then how we can actually manipulate that input and then produce an output. Which is like world program and it also gives the user name that is what we saw and then we also saw the basic data types this L or data type the arrays and then the associative arrays today we will continue that looking at some more the data types one of the special types of array of string and then we look at that and then we will also like start some of the operations using these data structures , and how you can build programs. So we will build it little by little.
(Refer Slide Time: 01:40)

String Matching

- ◆ To match a string with `$_`, use `/string/`. The result is true if the string appears anywhere in the variable.

```
print "What is your URL? ";
chop ($_=<STDIN>);
if (/seerakademi\.com/) { ## Yes, our
    students
} else { ## No, outsider
}
```

20

So let us look at some of the strings so string is essentially like anything in an array of four characters but together they have a special meaning this is they are treated as a single variable so there is a concept of a matching which is essentially what it means is whether the two strings are exactly the same so like , I mean in equality of number systole we will say like 2 plus 3 equal to 5 and 5 on one hand and then 2 & 3 on the other hand if you do a addition operation they are equal that we in established but in a string.

How can you establish whether string is equal to another particular string so it is mainly it is what we call it the comparison or matching and essentially the matching is achieved through this double slashes as you can see here basically for flash string slash there are a couple of mnemonics here one inform this dollar underscore it actually has a special meaning, this one here which shows here the dollar underscore refers to the current line current line or essentially current value basically as to where it is so here.

You can see with all of underscore is directly refers to what it was standard in essentially like ,I mean so again this particular statement you can see that it is a compound statement it basically gets this value into dollar the score, and then it also does the chop function on it one thing is basically this dollar underscore in the some of these variables you do not really need to specify actually like okay.

I am going to operate on that pull kind of implicitly assumes that if you do not specify anything you are referring to dollar underscore so that is again another reason why like I mean in this statement we do not specify dollar underscore equal to or something again pay attention to the two slashes and the string ,so in order to match this particular any string with the dollar underscore we just simply use the string in the middle of two slashes so here so the thing is basically if the result is true .

I mean then essentially like ,I mean that that path is taken if you are using like an if statement and again like I mean you can use this essentially like I mean if the string appears anywhere in the particular line in anywhere in this line again, so the dollar underscore denotes the line and this particular string if it happens to be they are given within that line then the result is becomes true when you do the just the /string bin slash so in this example actually, so your people with the seerakademi calm in as URL essentially like.

I mean so you can specify like I mean anything basic with your image students or electrical engineering students dot clear academy seekarademi as soon as finds out that seerakademi .com is there in the particular as a substring of this particular string then the result becomes yes so and then if you give any other one of course result it no ends of the LCD, so this is a simple matching function so as we see basically like a student becoming more and more now like okay we have from data structures how we can do programming this test data structure.

This will be the discussions that will go over and then we will see like how we can manipulate the data itself using various commands.

(Refer Slide Time: 06:20)

Regular Expressions

◆ Perl provides every regular expression feature found in standard Unix utility.

```
print "What is your URL? ";
chop ($url=<STDIN>);
if ($url =~ /seerakademi\.com$/) {
    ## Yes, our students
} else {
    ## No, outsider
}
#Matches "seerakademi.com" at the end of the
variable.
```

21

So regular expression is the other thing that we will be talking about actually this inverter introduces the regular expression but we will go into more details later on so in general so we saw that basically like if it is a dollar underscore here then we use just the slash but in general like. I mean the variable may not be just the score you will be actually using other variables to do the extreme laughing so in order to do goes string matching essentially we use this tilde after the equal to and so equal to tilde means like okay.

Now match and then we then build a string and then and match it me here you can see that actually I am also like putting a dollar at the very end this dollar indicates that that is the end of the line so or end of the line or end of the variable is where the tolerance and we look at the previous one actually here , we do a chop on as well basically that job actually chops anything white characters after the white characters between the non-white space and the dollar video you can think of it that way.

So in this case like when we are doing the chop again here on the dollar URL so first we are attaining the standard in to the dollar URL that is the big label and then we perform a chop so it removes all the whitespace characters so all you are left with is a non whitespace character followed by a dollar, so that is why I like now we will be do a matching essentially which is now right here with this tilde it is the one that is used for matching.

Now we can match it and then basically with this dollar essentially anchors that particular string at the end of available so it looks for this particular thing at the end of a table so can anyone tell me like what is or the beginning of the variable so to go to the beginning or basically like match from the beginning of the variable we use current this current is essentially in your keyboard you

can look at as the shift 6 essentially that is the current, so we use that for anchoring the beginning of the variable and the dollar is used to enter the end of variable. So essentially the regular expression itself we have not really introduced in this section but now we will talk about the regular expression because the regular expression is one of the key things in Perl which enables it to actually be used by many script writers. (Refer Slide Time: 09:59)

Examples

```
◆ $name =~ /^John/  
◆ $name =~ /^John\b/  
◆ $name =~ /^John/i  
◆ $name =~ s/^http/ftp/i  
◆ $name =~ tr/A-Z/a-z/
```

So here the dollar name is matched with John and a can anyone tell me like the where this John usually comes so as you can see basically imperative use actually that the variable is anchored at the beginning so this John has to be the first or character of the string, in order to match now we follow it basically with a slash B or backslash B which is essentially like denotes a blank character or a white space.

And in this one actually it is kind of different basically, we now say that it is John and then /I the eye actually gives us essentially like a mean wall that is we do not have to match the exact case so upper case download is done both can become true in this case , so that that is another unique thing about you now here there is even longer ink basically so here if you notice actually there are you there are two slashes or actually three slashes one on here and then there is also this s so this is basically like.

I mean there we now start substituting the string with another string in this example the string HTTP is substituted with the other string the FTP so essentially like I mean you can think of this as like I mean from beginning basically like a mini division and HTTP then we replace that with the variable the string FTE and again this is you this particular command is in fact used very widely like.

I mean pull that kind of string matching and then basically taking form of the strengths you so this is another example.
 (Refer Slide Time: 14:35)

Examples

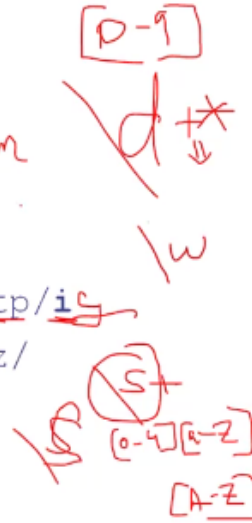
◆ \$name =~ /^John/g\$m

◆ \$name =~ /^John\b/

◆ \$name =~ /^John/i

◆ \$name =~ s/^http/ftp/is

◆ \$name =~ tr/A-Z/a-z/



Okay, so as I was explaining so we can use like the various matching, so a simple matching useful form here where the string is matched against or the dollar name is matched against the spring down one thing that, I want to highlight is here basically you can see that actually matched with the beginning at the beginning of the string to the column name variable matched from the beginning so you can think of this as like it is anchoring point ,so it anchors either in the beginning or at the end the beginning enter is this caret and as we find the previous one. The ending anchor or the towards the end how we do it is with the dollar the dollar signs in so this is the simple matching basically where it just these better this is the same amount so you can use this matching inside conditional statements to see whether they are getting the right value amount now the next one is with the vacuums be again this is bachelor these character that is used to match blank character ,so the John followed by blank is what match down. The next one is some /I this is essentially like a minute invention like makes it like case-insensitive in this case actually the even the day we will the case of the day will also be more so this will match either like lower case John it will also match day and then uppercase o H n whichever one if the cases is does not matter it just matches, so there is this set of characters does not have to be negative uppercase or no you then the next one is essentially like any before we copy this is where we are replacing the character. It should be with FTP and whether it is uppercase HTTP Lord it is a significant matter because of this I the default the end basically then mom we do that directly and more let us look at this

one this is also like a replacement and anyone tell me like what is being replaced, so if you look at the difference between this and then we call it like TR and this side it is essentially it is S the S is essentially for a string replacement.

So the string replacement essentially means that like we just replaced exactly the same string with on based in this case the TR stands for the transliteration operator and TR essentially changes the case in the in this example actually all the uppercase will be converted into a lowercase, so in that in so it takes the dollar name so it here actually like a minute it is basically it takes the dollar name and there are HTTP is found with TP.

In this case all the characters are used this translation and then in this way that translation works is it takes the uppercase once and then convert any to go so if there is already some lowercase it leads Isis but if there is any effort is that it is become alone, so these are some of the examples of how we do string matching and kind of we are beginning to now talk about regular expressions.

Which is essentially like a mean way to represent strings which then becomes easier or get being operated on so that is the way like and you can looked at it and the end character is also like not just to be introduced inside there are other characters, that we can use at the end for example we can use G so if here instead of I can use D, D is essentially matching a global so this probably matches only the first occurrence.

But if you put a D then that is all the occurrence of John in this particular variable and then we can also do string matching or matching against multiple line this is by using M and then there are a few more characters which we will study in the future lectures but they are like little bit more complicated and then a couple of things that we have not really understood yet is now we know exactly how to Mack the crack swing and walk we play strings and also let some good characters.

There are some shorthand notation for various form these functions our these patterns for example the P represents or backslash B this will represent a digit basically expect anything within one or actually in 0 and 9 so a shorthand notation or this is this / D there is one character and then if you add a plus this becomes like one or more careful if you put a star instead of a plus then that becomes zero or more of that characters.

So these are some of the nifty things that we can use for evaluating and then there is also elected on w which is essentially a backslash w is a letter essentially beginning which is from a movie, so and then we also have all there are two of them two different characters, one is slash backslash F the lowercase s and then the other one with backslash upper cases the lowercase s signifies space character it is similar to like this backslash T.

Which is essentially like I have been used for denoting white space between two words and then you can have multiple voices by just adding plus sign and then the upper case s is a non whitespace character which could be same thing basically nine or a and the uppercase a 2 also possible and any other special characters we all included in the uppercase S so the B kind of uses acting like more and we will the recommendation match exactly like some specific things inside this entire spring or better line.

And then we can actually use that information in processing that people but I think like I mean yeah this is all like probably it is a little bit more than what we wanted to look at the beginning of the lectures but I think like, I mean it will be to become more and more clear as we go along perform these are the things that we use to match and more fine insufficient.
(Refer Slide Time: 24:46)

Functions

◆ Function Definition

```
sub print_header {  
    print "Content-type:  
    text/html\n\n";  
}
```

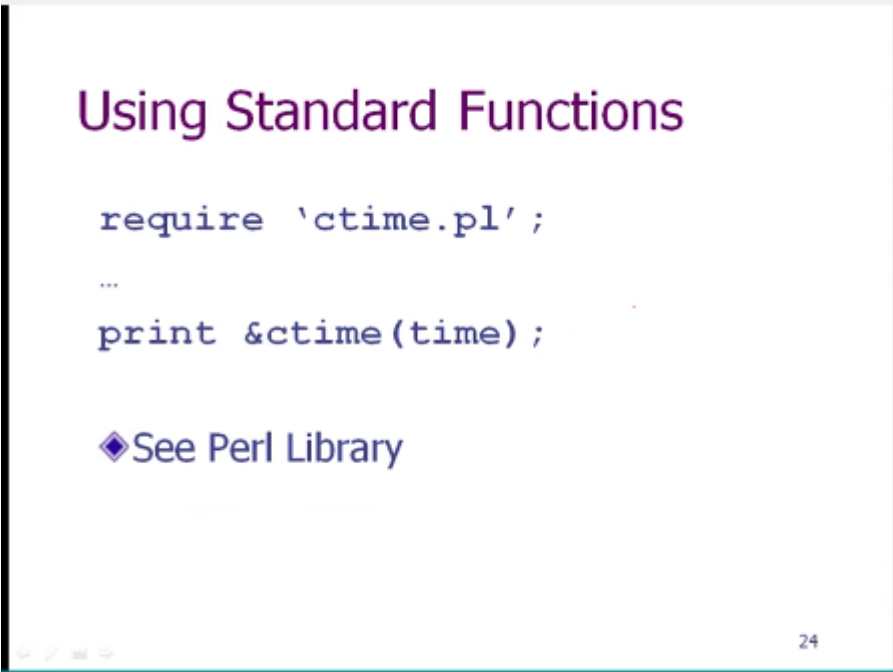
◆ Function Call

```
&print_header(); or  
&print_header;
```

So now let us look at the function definition. We actually use the function in the previous examples of the job function of the team we even though the definition of that is included as part of the first kind of neighbors but the way I define is using this SUV okay, now let us look at the functions essentially.

So we actually saw a function previously which is the chop, itself actually that I more like a library based function ,so it is like a standard function ,provided by the perl itself in general the function definition starts with the keyword sub ,so here you can see the sub then the function name itself which is here print header, we can have like some argument here, there are no arguments, if there are arguments ,we will give that within the parentheses and then, we start this bracket and then .

We just specify what exactly is the content of this particular function ,so here the content is actually like this print and then, we just say what is the content type ,whether it is a text or HTML and then followed by I thing like this one ,you are familiar with now, this is the newline character ,so it prints a couple of new lines ,so essentially like I mean this may be a print header ,header information on an HTML page, if you want to print something out . And if this particular subroutine, actually prints nothing and then prepares it for the subsequent content to print it out ,and then the call is essentially a print header with either this parenthesis, are simply ,so we will be dealing with some of these functions and function definitions and function calls in the later section, because the program is itself, is not fully done ,if you do not have these things by the way . We already talked about it in the first class that in Pearl ,there are no like, there is no main routine, which actually be executed first here, basically it just works from top ,down ,so the function calls could be anywhere . (Refer Slide Time: 27:58)



Using Standard Functions

```
require 'ctime.pl';  
...  
print &ctime(time);
```

◆ See Perl Library

24

And you can call that any time basically, so the other way to specify a function ,is to specify this particular command called require, this is essentially like I mean that these are functions ,that are already predefined and kept in the library, so when we specify this require as a beginning, at the beginning of the whole script, then this particular function will be incorporated within your program or it will be called so that whenever you use, this the function call. (Refer Slide Time: 28:40)

Functions

◆ Function Definition

```
sub print_header(){  
    print "Content-type:  
    text/html\n\n";  
}
```

◆ Function Call

```
&print_header(); or  
&print_header;
```

23

The one thing I forgot to mention in the previous one is when, we call the whatever the function calls, we also use this & at the beginning, & in general you can think of it as make an address, so the & print header is essentially making the thing, how to call a particular function, so here again for the C time, we call with this & C time, and then here the argument is time and then it prints out, while the output we should say, so this one, you can actually like both the Perl library and actually book for c time and see what it does.

(Refer Slide Time: 29:32)

Reading from a File

```
    ↑ FH      Filename  
open (LOG, 'access.log');  
while (<LOG>){  
    chop;  
    print "Accessed host: $_.\n";  
}
```

25

So now let us come to how do we open a file, the opening of a file is achieved through this command open and then there are two arguments to that, here you can see that, so this is straightforward which is the file name and what is this, this is all file handle, you can think of

this as the address of that particular file ,so file name the name itself does not mean anything but once we start .

So the name is associated with just this file handle and then after opening the file we always refer to it ,with its file handle , so here essentially the file handle is using this log essentially and then basically like `>` and `<` , so now can anyone tell me , how do we read input from the terminal essentially, now as you must have seen and basically as we learnt in the very very beginning of the this course ,all the utilities ,all the resources in the unit system or Linux is all represented all files .

So even the terminal is also a file and that file name is this `stdin` ,so that is why there are standardized ,define the predefined file handles ,the `stdin` and `stdout` is error and then there is also like some `argv` , once essentially `argv` and `argc` ,we will learn about that but think of it ,this way right the `stdin` and `stdout` there all like files ,that gets written out or read in and then we use the same five handles to read various variables or radiant of these variables at issue right up.

So it is very similar to the same log and in this case actually, we are opening a custom file with custom file handle , where as `stdin` ,`stdout` or they are always redefined and they are given to you already , so a typical construct is this construct which is like `while` and then this end, so in this Perl program essentially like I want to find every line, so when we specify that `while` and then this `>`, `<` ,log this gets line by line ,so the first line will be there and then we can process the first line and then.

Since this is a `while` loop , it goes and processes the next thing , reads one line at a time from this particular file, so here the key thing is again, it just gives the access log and then because I it prints out access log and `$_` here the dollar underscore directly is equal to this log , so it is typically you can think of this assignment is like this `=` ,this is the same , so `while` then basically like this and then do this same thing.

So the `$_` is a special character as a invention earlier, this is to denote the current line of current variable, whatever is there in the system, there are similar dollar variables they are called dollar variables we will be actually learning some of them, naturally because they are also like very crucial and in fact, you can manipulate things with these kind of using the dollar variable.

For example how do we read a particular file, so that so here like I mean one thing you notice is actually , when we do this `while` loop, always it gets one line at a time , how do we change or can we do something to change this to read multiple lines and use something else as a delimiter, so delimiter is essentially ligament that is how it reads essentially it knows that the end of line is basically the dollar are essentially the end of line character.

So it reads the end of line, so now say like I mean we want to read like any like multiple lines but on a semicolon, the semicolon can come maybe or even on a period basically, we want to get one sentence at a time from a text file ,how do you do that and then these the sentences can be like spanning multiple lines, the only delimiter on the sentences with the period, so think about that in fact , that may be one of the exercises .
That we will ask you to do at some point there are ways to do it actually like to change the delimiter from the dollar to something else and how do we do it .
(Refer Slide Time: 35:42)



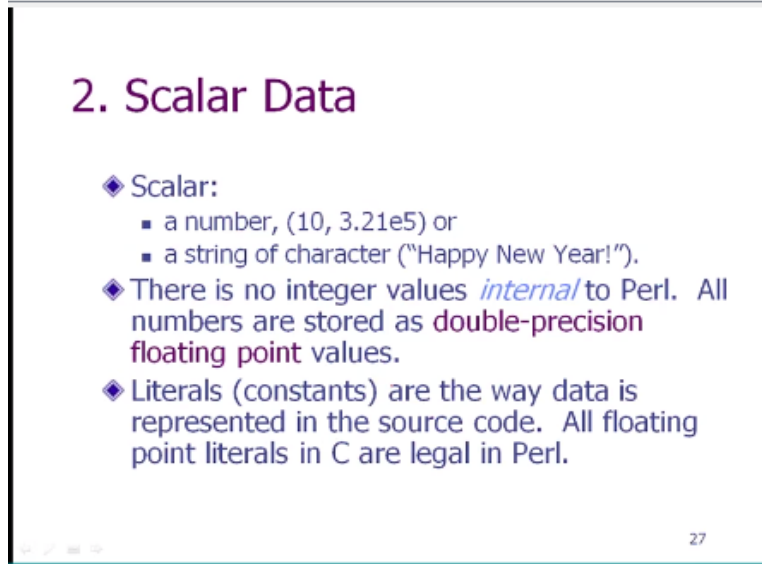
```
Writing to a File

open (MYFILE, '> MyFile.txt');
print MYFILE "Welcome.\n";
...
close (MYFILE);
```

26

We talked about it ,so now for writing to a file essentially ,we use the redirect hello essentially which is the < , so we open the file, but if you use this one in front of the filename then it knows that okay this file handle, like exactly it is going to be a right file essentially ,so in the previous one this was like a read-only file ,we are reading it from the access dot log but now the right file is essentially from my file.txt and then essentially whenever we use .
Whenever we want to write something to the file, we also add the file handle to the command, so if you do just print and then welcome, so that does not print into the file ,but if you print with this file handle ,my file and then welcome ,now this welcome will go into this , so one question I have is, if you do just print and welcome ,what does it do, does it write to someone or how does Perl react .
So the short answer is essentially as you know ,everything is a file , so when you say like print basically without any arguments, it assumes that is what you are asking it to do, is to write it into the standard out ,so it uses the standard out as the file handle and then this write out, as the output and then finally ,we have this particular statement at the close and then the close

statement essentially like closes the file ,usually like when the Perl program finishes running the operating system goes and actually close all the files and open during the operation. But it is a good idea, that we close the file yourself, because sometimes you may be using like multiple files ,multiple times you may have to read and write files, so you will miss out as to whether you really close or not and for closing any file ,we have to give it the correct file handle ,here the my file is the file handle ,which is going to specify in order to close the file . (Refer Slide Time: 38:27)



2. Scalar Data

- ◆ Scalar:
 - a number, (10, 3.21e5) or
 - a string of character ("Happy New Year!").
- ◆ There is no integer values *internal* to Perl. All numbers are stored as double-precision floating point values.
- ◆ Literals (constants) are the way data is represented in the source code. All floating point literals in C are legal in Perl.

27

So we will come to some of the other interesting ways to manipulate data , later on ,so now let us look at the data types and basically like or how do we operate on some of this data, so number one is this scalar data ,we already saw the scalar variable, but now the data is essentially like, either it can be numbers or string of characters, so any particular string is also a scalar string and one thing to note is essentially like as I mentioned earlier Perl is not a strongly typed language . And there are no integer values ,all the numbers are stored as double position floating point numbers ,so again this is one key difference, so that you can operate integer with floating point number and the Perl will accept it and then produce it produce a result ,there are some rules that you need to follow which I will explain as we go along, and then the other one is to basically be the literals or the constants are way the data is represented in the source code so all the floating point of little you have legal in pole, so these are the things that you want to keep in mind. (Refer Slide Time: 40:17)

Literal Examples

- ◆ Floating-point literals
 - 1.234
 - 1.23e45
 - -6.54e3
 - -12e34
- ◆ Integer literals
 - 12
 - 987
 - -1997
 - 0177 Octal 177 or 127
 - 0xff Hexadecimal FF, or 255

28

So what are floating point with little come up with examples are one point two three four on point two three, e to the power 45 or minus six point five four minus six point five for e to the power 3 minus 12 e to the power 30 for these are all floating particles, the integers are essentially like what we have shown here, FF could be an integer, octal one, seven, seven is actually in the value of 127 hexadecimal SS is 255.
(Refer Slide Time: 50:58)

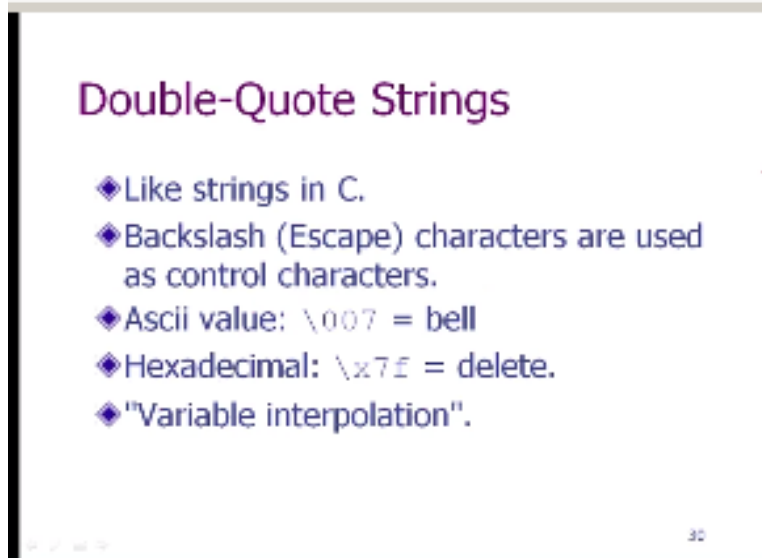
Strings

- ◆ No built-in limit on size (zero to infinity).
- ◆ Single-quote and double-quote strings.

29

So for the string the examples basically there are no size limit can be from anywhere from zero think, there are two different types of strings here, in a Perl one is the, the single coated string and the WPIX string so the single code string that we already saw, in the examples before essentially like I mean that those are we used some of those things with more in those examples the, double quoted strings are more common essentially.

So the, the difference here is like how much and more do the operations on you , you, you in between a double, so the key difference between the, the in the double quote string is that you can actually use variable names within the double quote string and if the Perl can interpret that as available, so that it replaces that, with the actual variable during the execution or when you are running the program but as in a single coated string they are all just taken as a little string so essentially like even if you put a dollar something is the same as what it is.
(Refer Slide Time: 43:16)



So here essentially leggings making the double, double quoted strings essentially there are strings like exactly like in see the you can use also you can use the backslash characters for escaping the control character and also like I mean there they are used as the temple taxes as well so you can spray the valium in backslash-n that is that is one of the things that we saw in earlier this is n plus n represents the newline character. And then number seven in ASCII me represents the bell sound extra decimal X 7, actually like the F denotes that basically and then that is actually necessary X denote the hexadecimal and then the 7s is the character that is for delete, and another example is this variable interpolation in double quotes.
(Refer Slide Time: 44:25)

Some Escape Characters

<code>\n</code> newline	<code>\cX</code> control-X
<code>\r</code> return	<code>\\</code> backslash
<code>\t</code> tab	<code>\"</code> "
<code>\f</code> formfeed	<code>\l</code> lower case next char
<code>\b</code> backspace	<code>\L</code> lower case until \E
<code>\v</code> vertical tab	<code>\u</code> upper case next char
<code>\a</code> bell	<code>\U</code> upper case until \E
<code>\e</code> escape	<code>\E</code> terminate \L or \U

31

So and some of these things that we saw basically our new line character, backslash n the return character or it is the backslash R CX is essentially control act common Emacs for editors, backslash T at T is tab then the spawn speed some of these things are you hardly use them today, but I think again this is a complete list of all the characters that can use essentially you can escape them, I will let you read the some of these things for at your leisure you can think of you can see basically women even like the court themselves can be escaped and so it can print out the real ones.

(Refer Slide Time: 45:31)

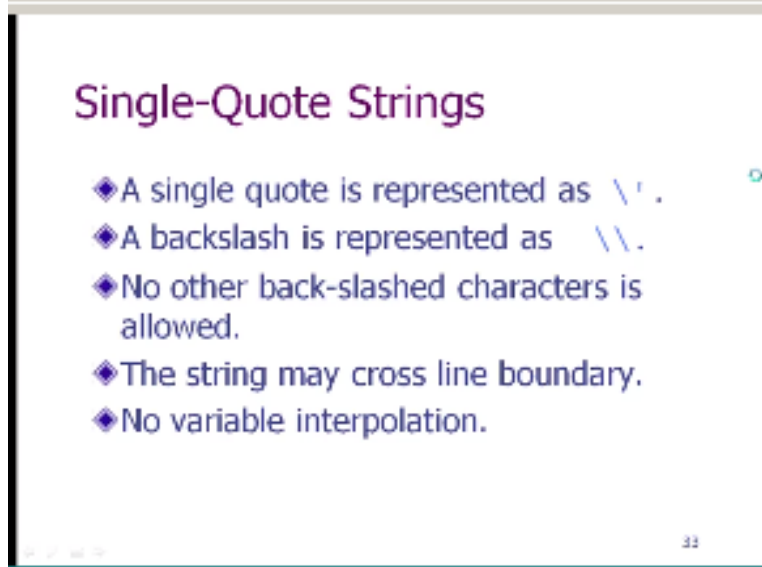
Examples

<code>◆ "hello"</code>	hello
<code>◆ "don' t"</code>	don' t
<code>◆ "hello world\n"</code>	hello world

32

So here are some examples a simple square string with double course this say hello which has prints the hello, now do not it prints just the do not with the single quote within then hello world basically prints hello world with the newline character.

(Refer Slide Time: 45:57)



Single-Quote Strings

- ◆ A single quote is represented as `'`.
- ◆ A backslash is represented as `\`.
- ◆ No other back-slashed characters is allowed.
- ◆ The string may cross line boundary.
- ◆ No variable interpolation.

33

So it goes the next line now a single quote the single quote is also represented as backslash quote the backslash itself is represented as over backwards backslash and then the no other backslash characters are allowed essentially in inside the single code strings, so you can only like escape two things one is the backslash itself or quote itself, and then the string should not cross the line boundary, and then there is no variable interpolation.

So if you are using like `$something` `$sentence` just prints out as `$sentence` it would not hang that to what the equivalent value is so the dollar variables are just representing that edges there is no interpolation feelings, and I think like I mean this is probably the biggest difference between a double quoted string and a psychotic string, in a single code there is no variable expansion whereas in a double quote if you add a variable middle they are interpolated and expanded the variable values are used for Bonitos when you run the program.

(Refer Slide Time: 47:23)

Examples

<code>'hello'</code>	<code>hello</code>
<code>'don\'t'</code>	<code>don't</code>
<code>'hello world'</code>	<code>hello\nworld</code>
<code>'\n'</code>	<code>two-character string</code>

34

So some of the examples like the hello so this is exactly the same as double quoted hello so there is no difference there do not or do not actually we need a stable the coat otherwise the string will end, at tone so if you escape it then it is been for populist do not and then the hello world multiple lines essentially like m4 printer with the new line, that is added here and then new line itself is basically it is a gesture to character Springs prints are exactly the same. (Refer Slide Time: 48:01)

Quiz

◆ What is the meaning of '\\n'?

```

$a = '\\n';
$b = "$a\n";
print $b;

```

35

So one question for you is what does this mean this is to, maybe like more challenging what is this new dollar a equal to put blue put backslash, backslash and the pollen and then dollar B equal to you II Oh \$10 you, so try this program and then see what the cynics we will talk about it time the next class as who exactly that PLC so I think that's the end of today's class, so we will see you in the next one.

So just to recap essentially started talking about some string variables specifically string matching how do we match strings and then we also went through like how to do some of the, the matching itself like I mean how do we operate on the strings we studied about a little bit about the regular expression we have not really introduced big way I will try to do that in the next class and give you like more gory details about the regular expression, and how can we do matching on really esoteric items.

And the and you know recommender there wherever you can find some regularity you can be able to use expression and then use a lot of matching and this, this matching and the associative arrays are probably two main topics in pull, which is useful across where you are and which company work or everywhere you go they will all be using these two principles the subsidiaries and the regular expression matching.

So we saw that and then we also now started talking about the strings as to what kind of string that we can build, what is the difference between a single quote and the double put and the key difference there is the double quote will do the variable enumeration or actually the it will extrapolate all the variables, there as for singing code it would not do those then the table interpolation so I guess that is pretty much, currently like I mean what we have and then we will start talking about and how we go about in the next class okay thank you very much.