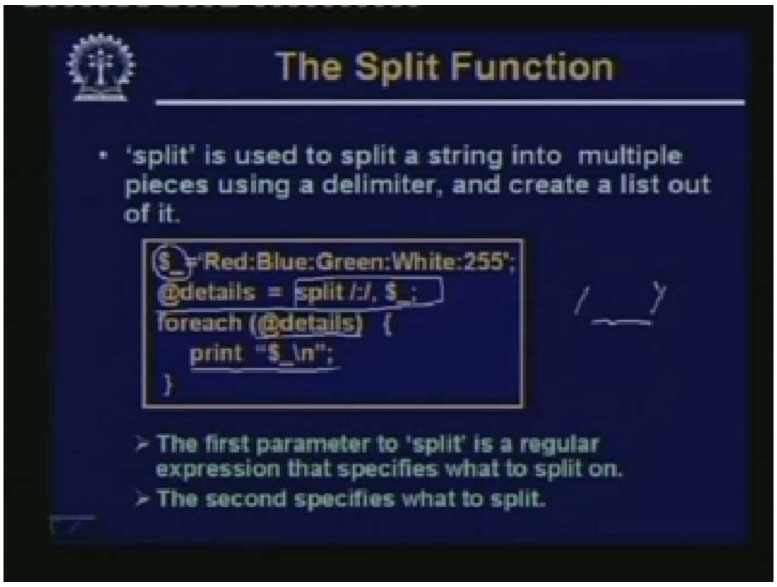
Internet Technology
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No #23
PERL – Part III

In this lecture we continue with the discussion on the Perl language. Now if you recall, we had already discussed the concept of variables expressions, how the expressions are evaluated in Perl. We talked about arrays and we talked about a few other string manipulation features that are available in Perl. Now in the present lecture we would first start looking at a couple of very interesting and important string functions followed by something which is considered to be one of the most powerful features of Perl namely, regular expressions. So we start by looking at some of very interesting string functions.

(Refer Slide Time: 01:35)



The Split Function

- 'split' is used to split a string into multiple pieces using a delimiter, and create a list out of it.

```
$_ ='Red:Blue:Green:White:255';
@details = split /:/, $_;
foreach (@details) {
    print "$_\n";
}
```

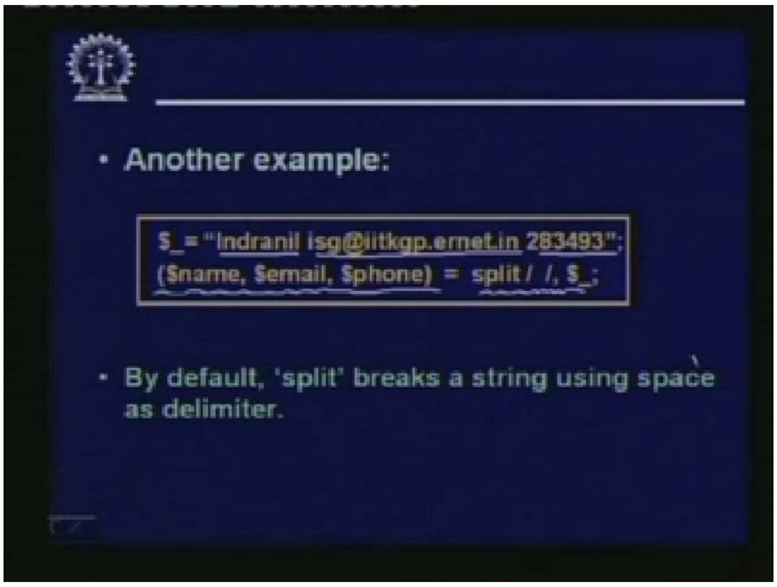
> The first parameter to 'split' is a regular expression that specifies what to split on.
> The second specifies what to split.

Well the first function is called Split. Split as the name implies is used to split a given string in to multiple pieces using a specified delimiter. What this means is that giving a string there are certain delimiters. Some defined delimiters; it can a colon it can be a space. They this particular delimiter is existing in several different points in the string and the whole string will be cut at the locations of the delimiters and the pieces that you get after the cutting, it is formed as a list. So the result of a split is a list of elements. So split breaks up a string in to multiple pieces and creates a list. Let us take a small illustrative example. Suppose I have a list like this. Red:Blue:Green:White:255 separated by some character colon, the whole thing is a list. Now you look at this assignment we have assigned this string to a variable dollar underscore. Well we could have used any conventional variable names. But we have chosen to use the dollar underscore character.

Now dollar underscore has a particular significance dollar underscore is some kind of default variable in Perl. See suppose you assign it to a variable say dollar abc, then whenever you want to refer to the string you should mention the name of the string dollar abc. But this dollar underscores which is considered to be default. If you use it in a string in many operations you will see that it is not required to specify the name of the string. By default it will considered or assumed that the string is stored in the special variable dollar underscore. So in this example we have considered that dollar underscore where we are storing the string. Let us look at the next statement. The right hand side uses split. Split is the function followed by a slash after slash. You have the delimiter in this case it is a colon and again a slash. So between the two slashes you have the delimiter. This is the basic syntax. Now here you give a comma, you give dollar underscore means from which string you are trying to split. So if it is dollar underscore, the second parameter is optional.

But here we have explicitly mentioned split slash the delimiter again a slash comma name of the string. So after this split is executed, the splitted value will get stored in the array called details. So the value will be Red: Blue: Green: White: 255. Now in the next statement for each details, now if you look at this syntax of for each as we have seen earlier we had used it as follows. For each name of the scalar variable then within parentheses name of the array but here we have dropped the name of the scalar variable by default scalar variable will be taken as dollar underscore. So you see this, for each at the rate details, so the elements will be accessed from the array one at a time. They will be stored in the variable dollar underscore and in the body of the loop you are printing the value of dollar underscore with a newline. So basically this program will print the values of the different components of the string which are separated by colon.

(Refer Slide Time: 05:54)



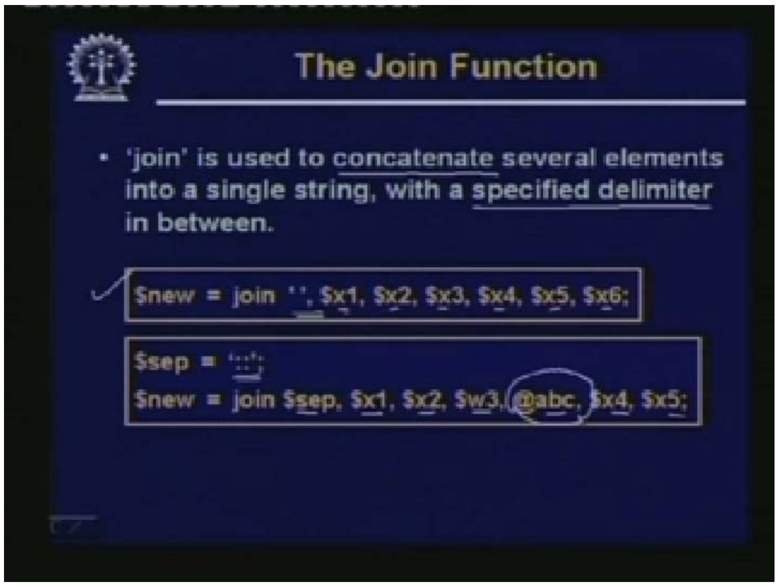
- Another example:

```
$_ = "Indranil isg@iitkgp.ernet.in 283493";
($name, $email, $phone) = split / /, $_;
```

- By default, 'split' breaks a string using space as delimiter.

Let us take another example. Here the string contains a name, an email address and a telephone number. They are separated blanks. Now instead of a array variable name on the left I can also specify a list of scalar variable names like dollar name, dollar email, dollar phone. So when I execute a split on a blank from dollar underscore then the three values will be extracted and will be stored in name email and phone. Now we will see some examples, some practical examples of this split function. When we use Perl for writing cgi scripts, this we shall be looking at our next lecture. Now you understand when a form is submitted assuming it is a POST method of GET method also. The same thing will be there, the data which is coming to us is coming as a name value pairs name equal to value ampersand name equal to value and so on. So what logically we need to do at the first, we need to split the whole string with respect to ampersand. So that the individual name value pairs get extracted. Now with respect to each of the individual name value pairs we can further split with respect to equal. So the name and the value will get separated out. So this is one practical example of split which we shall see later.

(Refer Slide Time: 07:36)



The Join Function

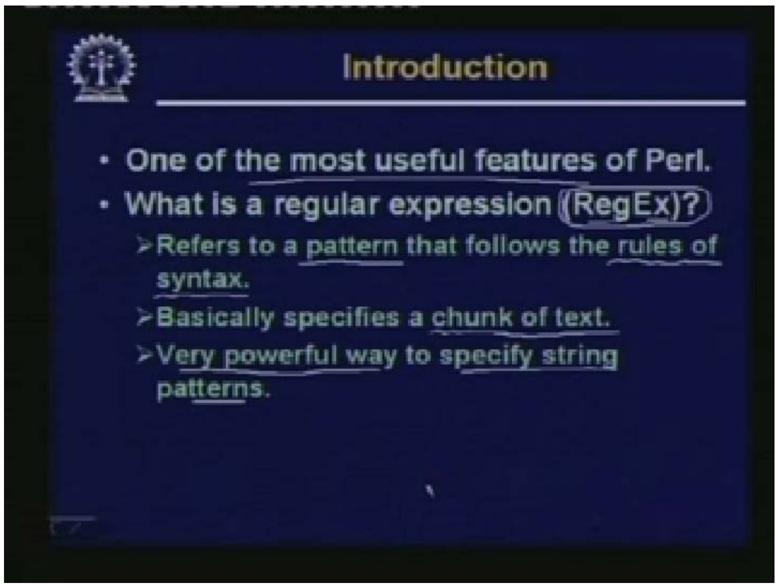
- 'join' is used to concatenate several elements into a single string, with a specified delimiter in between.

```
$new = join ' ', $x1, $x2, $x3, $x4, $x5, $x6;
```

```
$sep = '::';
$new = join $sep, $x1, $x2, $w3, @abc, $x4, $x5;
```

Join is like the reverse of split. Join is used to concatenate are connect several elements together in to a single string. Well here also you can specify a delimiter. So when the strings are concatenated the delimiter is automatically put in between. Normal string concatenation is a blind concatenation. Whatever is the content of string they are put side by side. But here you are putting a special marker in between the strings. When you are putting them concatenating them together like an example here the first example. Join; the first parameter of a join is this specified delimiter within quote I give a space. Then the names of the variables, the six variables will get concatenated. Whatever they contain with a space in between, the second example shows two things that instead of a single variable the delimiter can be any character string containing multiple characters. For example here double colon this is my delimiter. So here I have asked to join several different items and the point to note is that one of the items is an array. So when the new string is constructed, the elements of the array abc will be expanded and the whole thing taken together including the components of the array. They will be concatenated and the double colon delimiter will be put between each pair of them consecutive pairs.

(Refer Slide Time: 09:33)

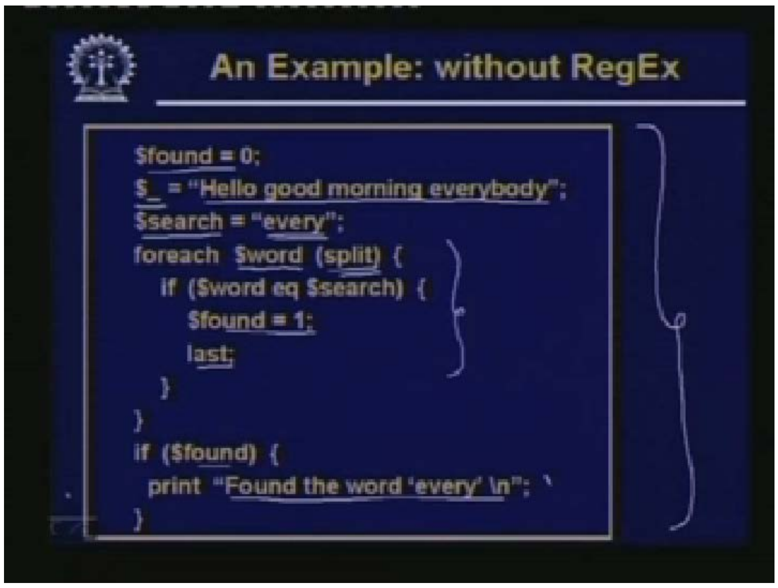


Introduction

- One of the most useful features of Perl.
- What is a regular expression (RegEx)?
  - Refers to a pattern that follows the rules of syntax.
  - Basically specifies a chunk of text.
  - Very powerful way to specify string patterns.

Now let us have a look at Perl regular expressions. As I had mentioned at the very beginning, regular expression is one of the most useful features in Perl. Regular expression sometimes in short we call RegEx regular expression and the first question is; what is a regular expression? The regular expression is basically you can say it is a way; it is a method to specify a character string. It is a very powerful method to specify a string depending on our need. The string may be arbitrary complex as we shall see. Basically it refers to a pattern a string pattern that of course will follow some rules the pattern has to be specified following some rules which are guided by the syntax of the language. Regular expression is nothing but a character string as I have said which is actually a chunk of text; it is an extremely powerful play as you shall see to specify string patterns because sometimes we need to search for something in a string. Well simple string searching we have seen using the eq, less than equal, greater, these kinds of operations, you can compare two strings. But sometimes we need to search whether a particular element belongs to this string or not and this condition may become arbitrary complex as we shall see.

(Refer Slide Time: 11:07)



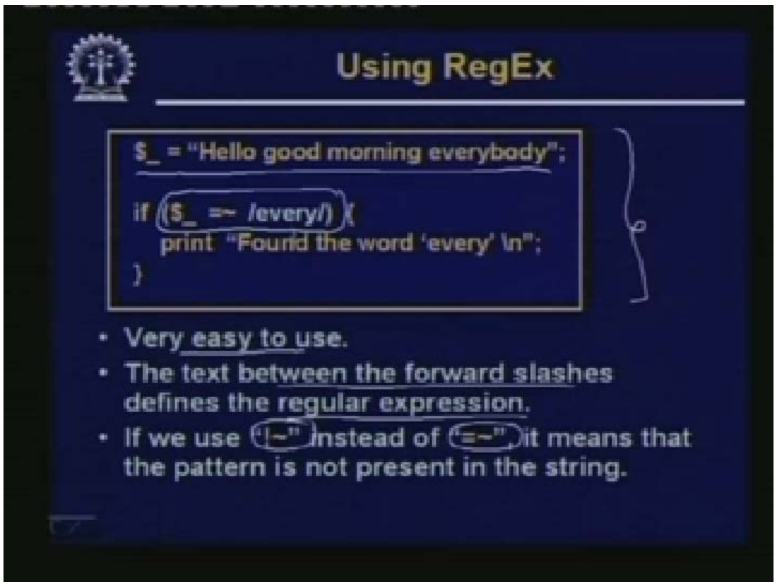
An Example: without RegEx

```
$found = 0;
$_ = "Hello good morning everybody";
$search = "every";
foreach $word (split) {
   if ($word eq $search) {
      $found = 1;
      last;
   }
}
if ($found) {
  print "Found the word 'every' \n";
}
```

Let us first start with a simple example that how regular expressions help us in making a program simpler. This is a program which has been written without using regular expression. See this starts with a variable found initialized t zero there is a character string “Hello good morning everybody” that is stored in variable dollar underscore and what we are trying to do in this program is we want to search for a word every which we are storing in another variable search. Now the program is as follows. For each word see we have given written only split here. See split only, writing split means number one see means if you look at the usage of split we had shown earlier. There were two things we are specifying after split within forward slashes, we are specifying the delimiter with respect to which we have split. Secondly after a comma we are specifying from which string we need to split out. Now here string is the default string dollar underscore. So we did not specify it. It is optional and secondly there is also another optional thing in split which we did not mention earlier that if the delimiter is space that is the default case.

We need not specify that also. You simply mention split that will take care of everything. It means that you are splitting the string dollar underscore with respect to space. So within bracket the condition is only split s, you are splitting this string it become “Hello good morning everyday”. There are four elements and the four elements will get splitted in a list and the list, the elements of the list will be assigned to variable word in every iteration. Now within the loop you are checking if word is equal to search then you are setting the flag found to one and you are immediately exiting the loop by last. Because you have already found it otherwise you go on iterating till you find or do not find. So after you come out of foreach loop you check the flag 0 or 1 if it is true then you print a message that you have found out the word. Now the same program this program has been written without using regular expression the same program. If you use regular expression it becomes so small.

(Refer Slide Time: 13:54)



Using RegEx

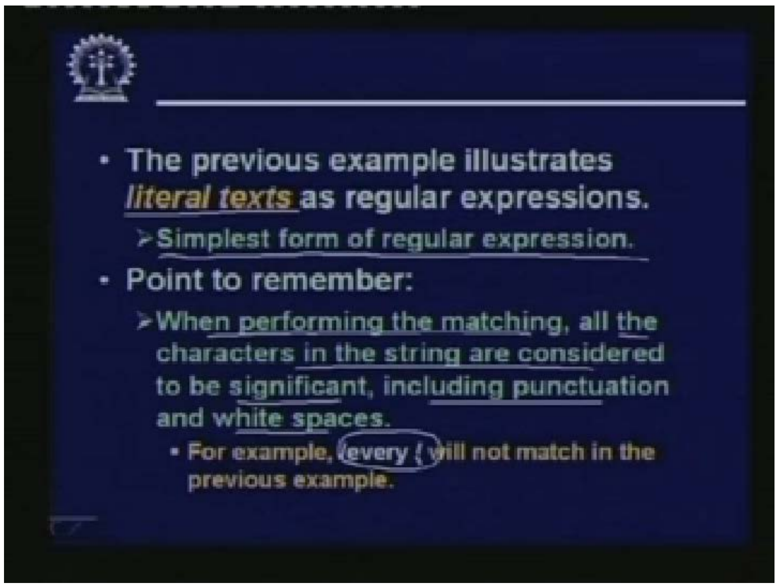
```
$_ = "Hello good morning everybody";

if ($_ =~ /every/) {
   print "Found the word 'every' \n";
}
```

- Very easy to use.
- The text between the forward slashes defines the regular expression.
- If we use "!~" instead of "=~" it means that the pattern is not present in the string.

Just try to understand what this means. First line as usually this is the string you are defining in the if-statement. The condition says well this we shall explain in detail this actually means if this means in English if the string every appears in dollar underscore then print this. We can write in a single if statement in a single line you can understand this kind of facility gives Perl its power this is an example of regular expression see within this forward slashes I am writing every, this is a a simple example of a regular expression and this equal to tilde. This is an operator which means you search. You are searching if the string specified in right hand side is present in the string on the left hand side. You can see this is extremely easy to use.

As I had mentioned the text between the forward slashes this defines the regular expression and here we are using equal to tilde to check if it is present. This one now as an alternative we can use exclamation sign tilde which means the negation of the condition. This means that the pattern is not present in the string. So in place of equal to tilde if I write this exclamation tilde, this would mean that if this string every is not present in dollar underscore then print something. So whether you want the truth value of the condition checked in the if statement to be whether the string is present or not present, you can use either equal to tilde or exclamation tilde.

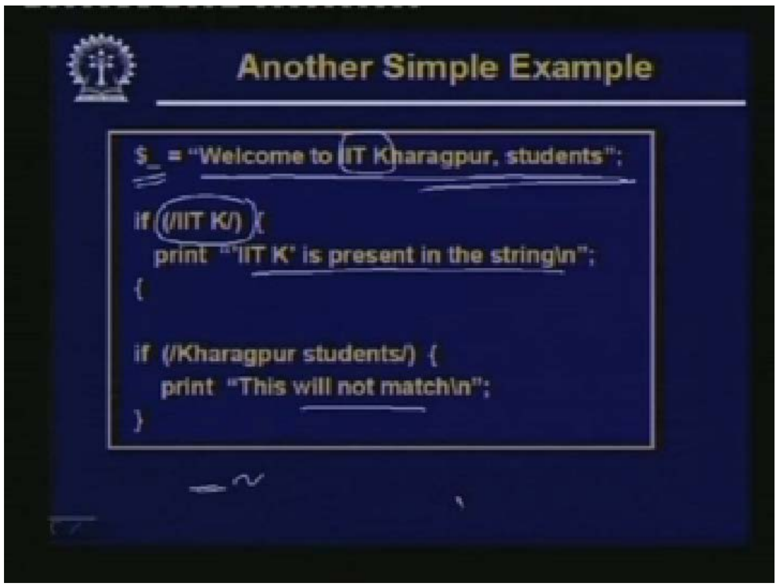
(Refer Slide Time: 15:54)



- The previous example illustrates *literal texts* as regular expressions.
  - Simplest form of regular expression.
- Point to remember:
  - When performing the matching, all the characters in the string are considered to be significant, including punctuation and white spaces.
    - For example, /every / will not match in the previous example.

So the example that we have seen, that is the example of literal text. Literal text means within forward slashes we are specifying simple text to be searched for. This is the simplest possible form of regular expression. Some text enclosed in forward slashes you give and that you can search whether it is present or not. There is something to remember here that when you are performing the match all the characters in the string are considered to be significant. This you must remember, all the character means this includes punctuations symbols like comma, semi colon, dots and also spaces. For example in the previous example if you give within forward slash every followed by a blank then the match condition will not be true. Because if you look at the previous example this string contained everybody. So there was no space after every b was immediately after it.

(Refer Slide Time: 17:04)

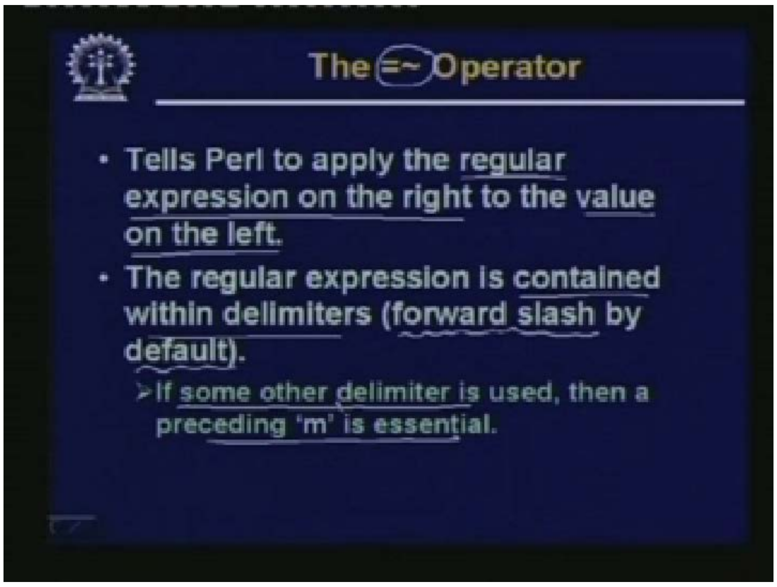


Another Simple Example

```
$_ = "Welcome to IIT Kharagpur, students";

if (/IIT K/) {
  print "'IIT K' is present in the string\n";
{

if (/Kharagpur students/) {
   print "This will not match\n";
}
```

So if you write like this than there will be no match. Let us look at another example. A string dollar underscore contains welcome to IIT Kharagpur students. Here in the first case we have checking using within slash. We are checking IITK blank K. See this is a default check simi, again if I want to search in dollar underscore we need not have to specify anything. Just in the if condition if I specify the regular expression is slash Perl will interpret it that well. This is the regular expression and we need to search if this is present in the default string dollar underscore. So everything is specified in this simple thing within brackets. So if you write like this we need not have to explicitly specify that equal to tilde or not equal to tilde just within slashes you write IIT black K. If it is true then you can print IIT K is present in the string. Now here actually IIT K is present IIT K was there. Now the second if statement says if Kharagpur blank students, this is the whole regular expression then print something. But here you see Kharagpur blank students will not match because what is present in the string is Kharagpur comma blank students and that said comma is significant. If you do not compare with the comma the match will not return true.

(Refer Slide Time: 18:54)

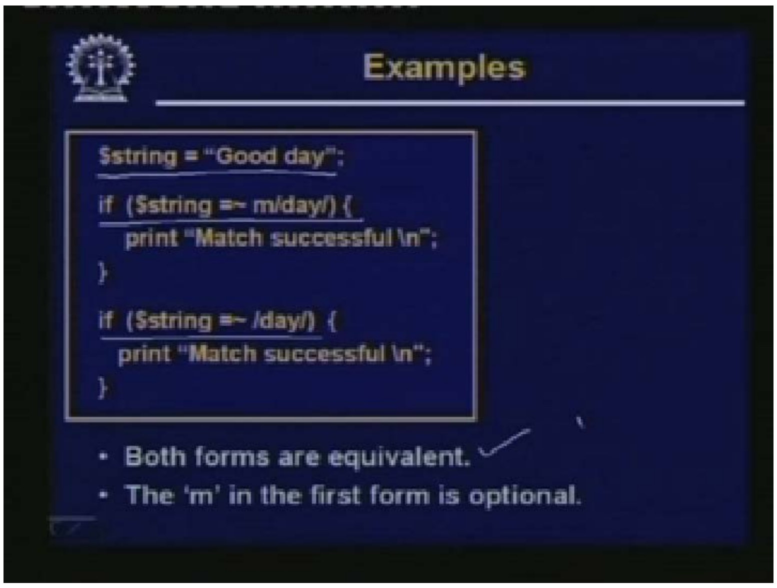


The =~ Operator

- Tells Perl to apply the regular expression on the right to the value on the left.
- The regular expression is contained within delimiters (forward slash by default).
  - If some other delimiter is used, then a preceding 'm' is essential.

Now broadly the types of regular expression can be categorized as Matching and Substitution. In the Matching case [19:08 word not clear] means here basically you are Matching a string against a substitute. We are checking if a given string contains a substring. The example that we had shown earlier was an example of Matching. Therefore Matching we use the symbol m as we shall see some examples. But this m is optional if forward slash is used as delimiter. In the previous examples we were using forward slash that is why we did not use this m. We will see some examples when this m is necessary. In the second type Substitution we can replace a substring by another substring in a given string for Substitution, we use the symbol so we two examples. First let us look at matching. Matching as you have seen can be specified by the equal to tilde Operator. This operator tells Perl to apply the regular expression on the right to this string value specified on the left. The regular expression that was specified should be contained within the delimiters as I mentioned, again I am mentioning forward slash is the default delimiter. If the default is there you need to specify anything else but instead of slash if you are trying to use some other delimiter then a proceeding m is essential.

(Refer Slide Time: 20:57)



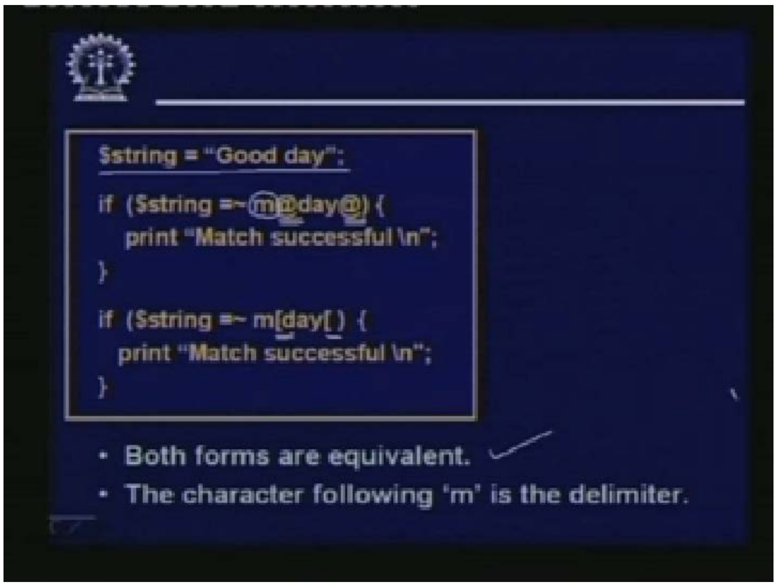
Examples

```
$string = "Good day";
if ($string =~ m/day/) {
   print "Match successful \n";
}
if ($string =~ /day/) {
  print "Match successful \n";
}
```

- Both forms are equivalent.
- The 'm' in the first form is optional.

All right let us see how, let us take this small example. Let us define a string Good day and store in a string dollar string. In this if statement we are checking if string matching m slash day slash m is the symbol for matching, I have mentioned print Match successfully. As an alternate form I can omit this m also. Since we are using forward slash as the delimiter whether or not we use m it does not make any difference. Both forms are same or equivalent in this case. But if you are using some other delimiter not slash.

(Refer Slide Time: 21:45)

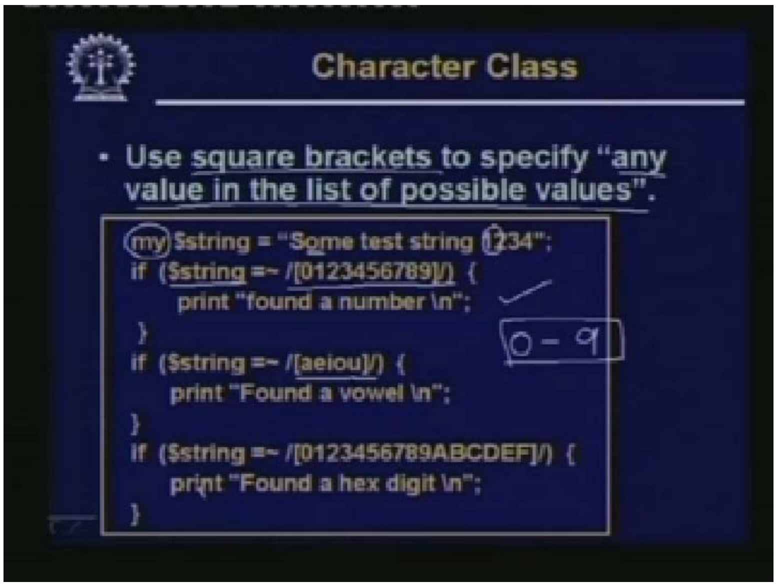


```
$string = "Good day";
if ($string =~ m@day@) {
   print "Match successful \n";
}
if ($string =~ m[day[ ) {
  print "Match successful \n";
}
```

- Both forms are equivalent.
- The character following 'm' is the delimiter.

Like this example shows here the string is same. But instead of slash we are using a delimiter at the rate. Now this m is mandatory. This symbol that immediately follows m that is identified as the delimiter. And similarly the same symbol will appear at the end to mark the end of the string, end of the matching string. Now this is typically used whenever the symbol forward slash also appears as a part of the matching string. Otherwise normally we do not want to change the default delimiter. Here is another example where the opening square bracket is used as the delimiter. So this we are checking for day [22:35 word not clear]. Both these forms are equivalent again in the first case we have used at the rate; in the second case we have used the open parentheses as the delimiter.

(Refer Slide Time: 22:45)



Character Class

- Use square brackets to specify "any value in the list of possible values".

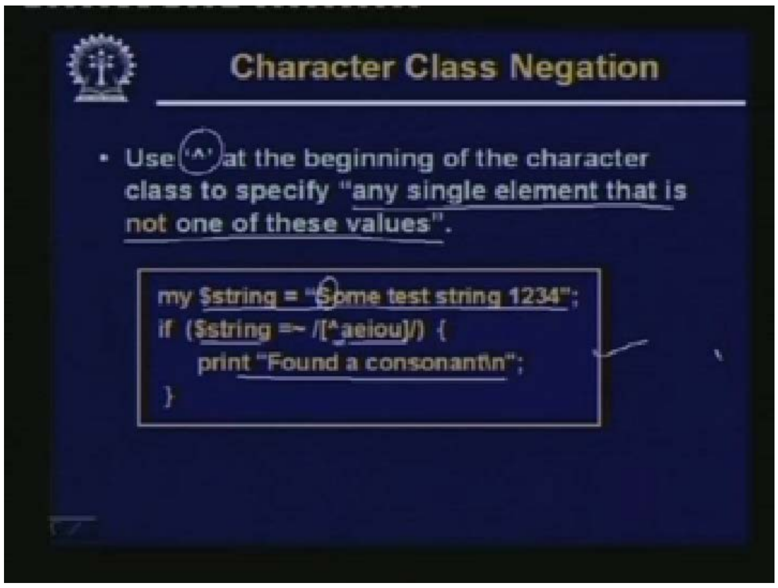
```
my $string = "Some test string 1234";
if ($string =~ /[0123456789]/) {
    print "found a number \n";
}
if ($string =~ /[aeiou]/) {
    print "Found a vowel \n";
}
if ($string =~ /[0123456789ABCDEF]/) {
    print "Found a hex digit \n";
}
```

0 - 9

There is something called Character Class in Character Class. We use square brackets to specify any value in the list of possible values. Whenever something is specified within square brackets it means any one of these values. Let us take an example. First line says string equal to "Some test string 1234". This is a string, well you look at this key word my at the beginning this is something we are using new here. We did not use this. This my keyword specifies that this is something like a local variable you know the concept of local and global variables in other programming languages like C or Java local means with only within the body of this particular procedural the value will be accessible if you are having a separate procedural or a subroutine you cannot access it from there. Well look at this if statement if dollar string matches, look at this within slash we give a character class within square brackets you mention 0123456789.

This means if the given string contains any of these digits then the if condition will be true. So it will search the string from left see at the first one it will find a match. So it will print "found a number". So this if condition will return true. The second if condition it checks whether any one of aeiou is present. Then it presents "found a vowel". Here the match will occur here, Some this o in Some. The third example shows the case string equal to again 0 to 9 ABCDEF. Well if you find any one of these strings you can say Found a hexadecimal digit hex digit. Here are some examples, so within square bracket you can specify certain list of characters. So if any one of them matches you say that the match returns true. Now just one point to note this 0123456789, these are consecutive values. There is a short form also you can write 0-9. This also you can write within square bracket instead of writing so many symbols. Negation of Character Class is also possible.

(Refer Slide Time: 25:33)



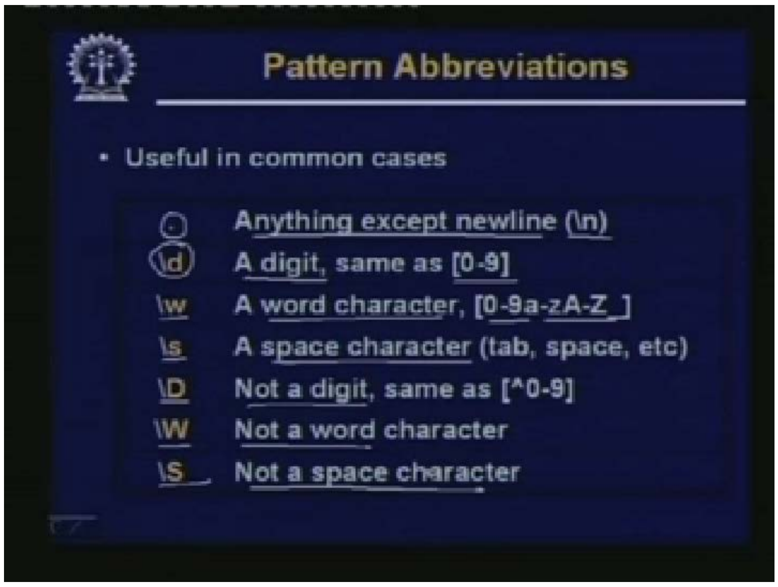
Character Class Negation

- Use '^' at the beginning of the character class to specify "any single element that is not one of these values".

```
my $string = "Some test string 1234";
if ($string =~ /[^aeiou]/) {
    print "Found a consonant\n";
}
```

If we use the carrot symbol at the beginning of the Character Class, this will mean any single element that is not one of these values. Let us again take an example. So again this string is given, we are checking matches aeiou vowel with carrot at the beginning. This means that the match will be returning true if none of these symbols appear in the string or the first symbol you get is none of these. Because it always tries to find out the first match. Now if it is a Negation it will get a match in the first symbol itself S. Because S is not one of these right, so it will return a match in the first symbol position itself and it will print the message. So in this examples you just note one thing that it is trying to find out the first match it can get in the string all right not trying to determine all the matches in the string.

(Refer Slide Time: 26:56)



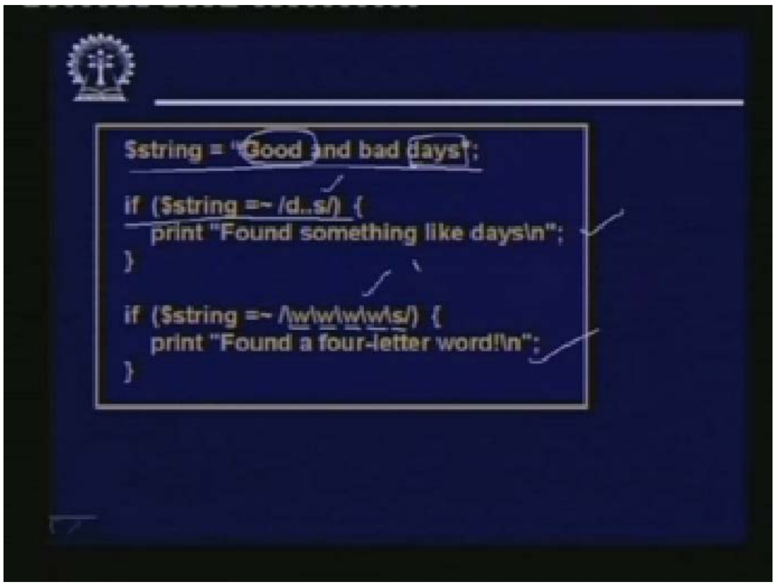
Pattern Abbreviations

- Useful in common cases

.	Anything except newline (\n)
\d	A digit, same as [0-9]
\w	A word character, [0-9a-zA-Z_]
\s	A space character (tab, space, etc)
\D	Not a digit, same as [^0-9]
\W	Not a word character
\S	Not a space character

There are some Abbreviations Pattern Abbreviations you can use in specifying more complex regular expression. A simple dot, dot means any single character except newline. So if there is a dot in a regular expression, it means it can represent any character. So if you write within slash c a dot then it can be c a t cat, c a d cad anything dot can represent anything reverse slash d means A digit which is same as square bracket 0 to 9. Small w means A word that means either digit or an alphabet. These are called word characters it is an alpha numeric character. Reverse slash s is A space character space character means tab space enter carriage return [27:53 word not clear] these kind of characters. But if you use this d w and s in upper case it represents the negative condition. Capital D means Not a digit, capital W means Not a word, capital S means Not a space character. So you remember this.

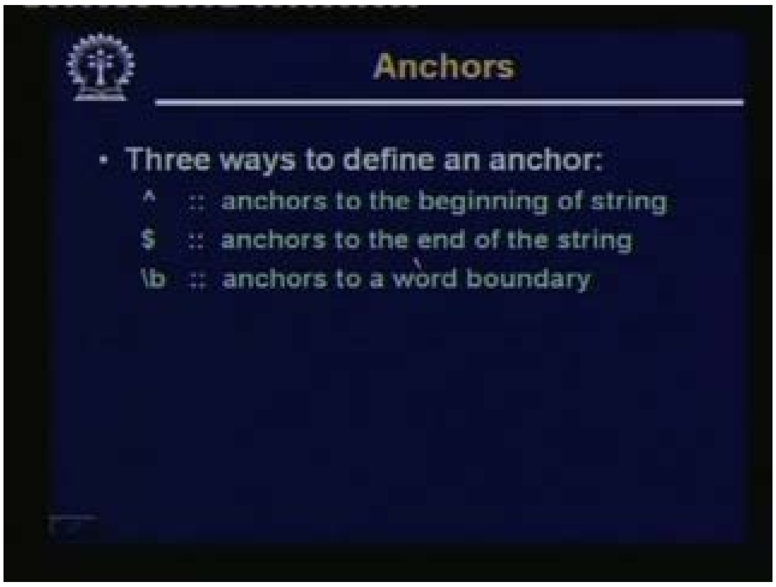
(Refer Slide Time: 28:15)



```
$string = "Good and bad days";

if ($string =~ /d..s/) {
    print "Found something like days\n";
}

if ($string =~ /\w\w\w\w\s/) {
    print "Found a four-letter word!\n";
}
```

Let us look at this example. You have a string here "Good and bad days". Here we are trying to compare something. Slash d dot dot s, d is a character s is a character two dots in between dots can match any single character. That is what I have said. So this will match d a y s, days d dot dot s. So it will match output. Let us look the second case. Within the forward slashes we have given w w w w, then a space character. So there are four alpha numeric character followed by a space character. The first match that will occur here is here G o o d, G o o d followed by a space. So it will print "Found a four-letter word". These are some slightly more complex way of specifying the regular expressions.

(Refer Slide Time: 29:27)

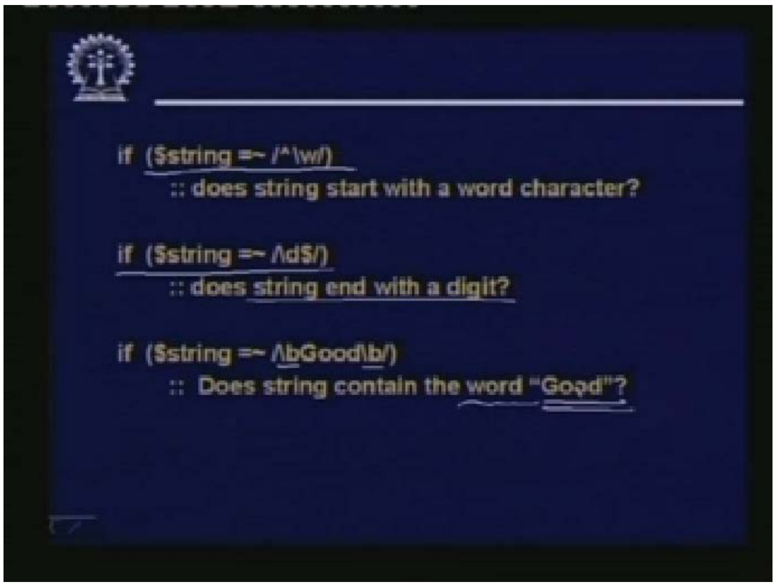


Anchors

- Three ways to define an anchor:
  - ^ :: anchors to the beginning of string
  - $ :: anchors to the end of the string
  - \b :: anchors to a word boundary

There is something called Anchors which sometimes we need to specify when talking about a regular expression. Sometimes when you specify some search condition that means we want to search for a word. We may need to specify that well search for something which is happening at beginning of the string search for something. At the end of the string search for something which is located at some word boundary something like this. So these Anchors are used specifically for this purpose. So there are three ways to define an anchor we can use the carrot symbol. Carrot means beginning of the string anchors to the beginning of the string. Dollar means end of the string and reverse slash b means to a word boundary. Word boundary means if you have a string like this good day then this space is considered to be the word boundary. So the match will take place starting from a boundary, such a boundary.

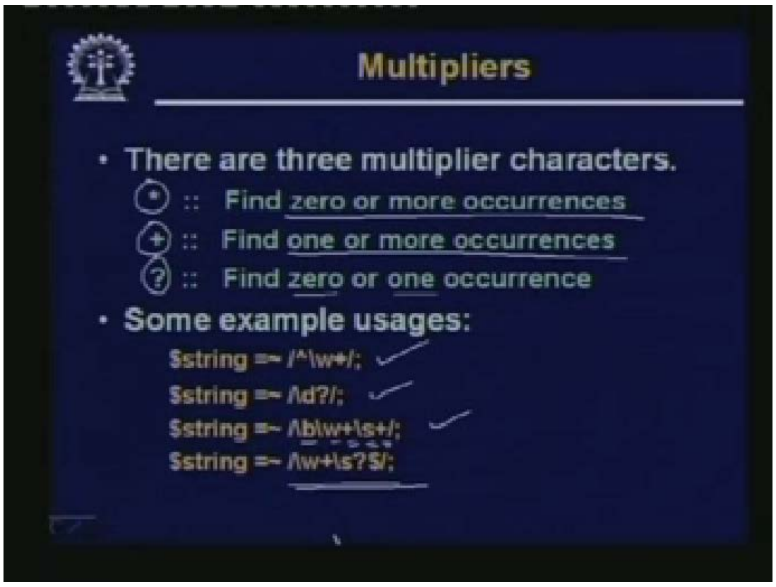
(Refer Slide Time: 30:29)



```
if ($string =~ /^\w/)
        :: does string start with a word character?

if ($string =~ /\d$/)
        :: does string end with a digit?

if ($string =~ /\bGood\b/)
        :: Does string contain the word "Good"?
```

Some examples are here, first example if dollar string matches within slash carrot slash w, slash w means a word character an alpha numeric carrot means beginning. So what we are saying that the string must begin with a word character. Carrot means we are anchoring at the beginning then slash w means a word character. This means we are checking if the string is beginning with an alpha numeric character. The next one if string match slash d, dollar d, digit dollar means anchor at the end. So we are checking if the string is ending with a digit the last character of the string is digit or not. The third example which is a slightly complex condition there is a slash b here. There is a slash b with Good in between and this entire thing is enclosed within forward slash. This actually represents that there must be a word boundary followed by Good there must be another word boundary which means you try to find out whether Good appears as word in this string. Word not a substring because Good can follow something together. But there has to be a space or delimiter after that it is appearing as a separate word. So this if condition checks whether Good appears as a word in the string or not.

(Refer Slide Time: 32:25)



Multipliers

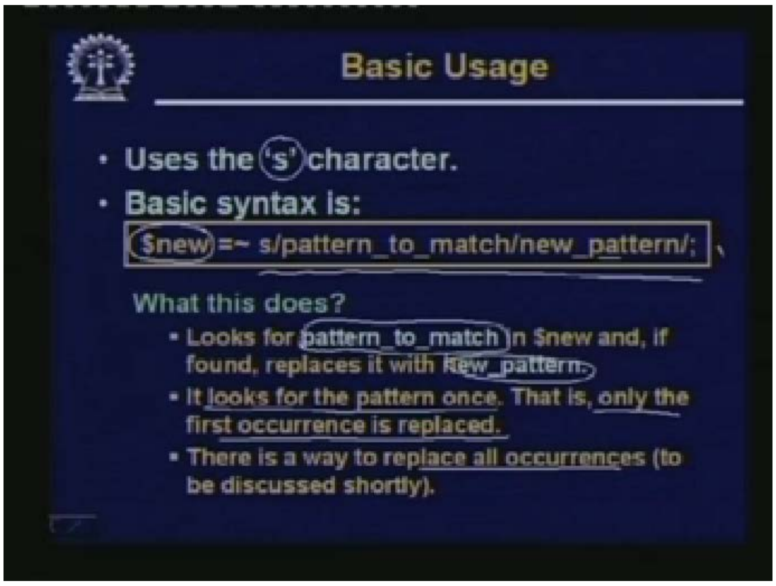
- There are three multiplier characters.
  - * :: Find zero or more occurrences
  - + :: Find one or more occurrences
  - ? :: Find zero or one occurrence
- Some example usages:

```
$string =~ /^\w+/;
$string =~ /\d?/;
$string =~ /\b\w+\s+/;
$string =~ /\w+\s?$/;
```

Multipliers are another important thing where we sometimes specify repeated value. Well we may want to check something like this, that a string which begins with a character. After that it can contain any number of alpha numeric digits but should end with a digit this. How do we specify? There are some Multiplier characters we can use for that. There are three such characters we can use. Star. Star means zero are more occurrences, plus means at least once one or more occurrences, question mark means zero or exactly one occurrence. So in a string we can use this symbol to make the regular expression more powerful. Let us look at some examples. First one this says within slash, carrot means beginning of the string, beginning a word plus which means the string must begin with a word and there must be at least one character after that because equal means one or more characters. So it must begin with a character followed by at least one more character.

So it should not be a single character string. The second example it starts with a digit followed by a question mark. Question mark can be zero or more. So it is either a single digit or a digit followed by some other character. Third one a word boundary followed by a character then as many number of characters after that followed by a space. Again as many number of characters after that, so in this way you can specify any complex kind of condition here is some other complex condition mentioned here. So the whatever you want to specify using a combination of these symbols we have mentioned we can specify that in the form of regular expression it can be arbitrary complex. The next important operation that Perl supports you can do in Perl are something called Substitution. Because we want to match characters, but sometimes we also want to modify something in a string. That is substituting, a part of a string by something else. Let us see how we can do this.

(Refer Slide Time: 35:09)



Basic Usage

- Uses the 's' character.
- Basic syntax is:

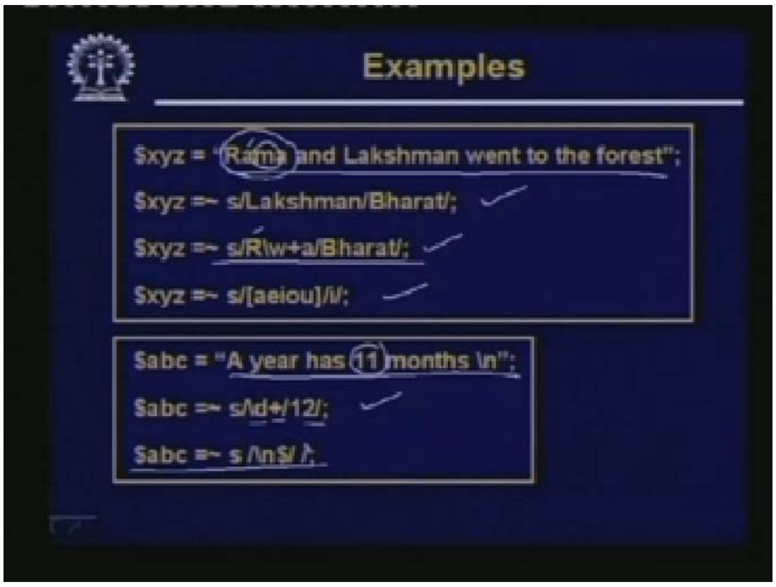
```
$new =~ s/pattern_to_match/new_pattern/;
```

What this does?

- Looks for pattern_to_match in $new and, if found, replaces it with new_pattern.
- It looks for the pattern once. That is, only the first occurrence is replaced.
- There is a way to replace all occurrences (to be discussed shortly).

Basic Usage of the substitution mechanism we have mentioned is the using the s character and the general syntax are like this. On the left hand side we will be having some variable while this will refer to some string, dollar new is string. This is an operator equal to tilde, but if it is a s after, that this means a Substitution s slash, some pattern to match I have just written like this. Here actually pattern will be appearing in the form of a regular expression again. A slash new pattern well here again you can specify this other regular expression slash close. So this one what this will do is that it will be looking for this specified pattern to match in the given string specified in the left hand side. And if it is found it will replace the string with the new pattern whatever is mentioned here. Now the point to note is that if you give the Substitution like this then only the first occurrence of the string pattern to match will be substituted. Only the first occurrence will be replaced, however will see that there is a method to replace all the occurrences if you put a g at the end g means global then you can replace all the occurrences of string by the other string. This we will see with examples.

(Refer Slide Time: 36:48)



Examples

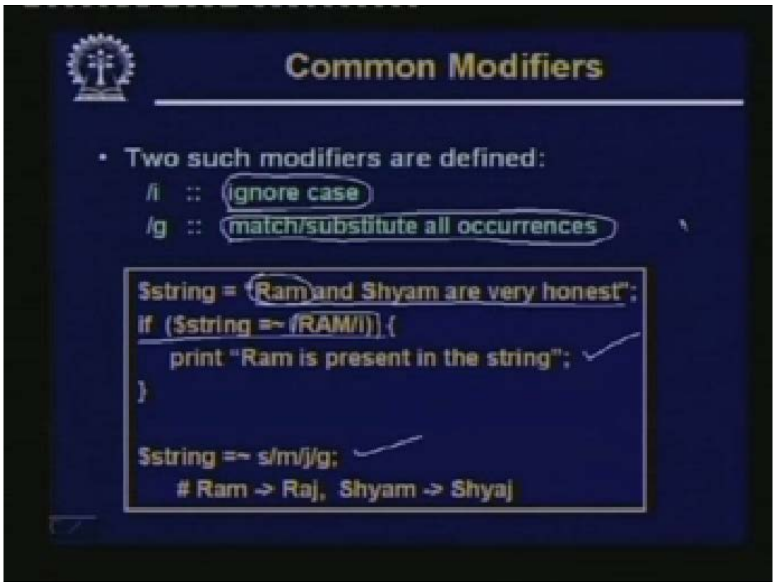
```
$xyz = "Rama and Lakshman went to the forest";
$xyz =~ s/Lakshman/Bharat/;
$xyz =~ s/R\w+a/Bharat/;
$xyz =~ s/[aeiou]/i/;
```

```
$abc = "A year has 11 months \n";
$abc =~ s/\d+/12/;
$abc =~ s /\n$/ /;
```

Let us see the first example assigns a string like this to a variable xyz. The first one says substitute Lakshman by Bharat. So here the new string will become Ram and Bharat went to the forest. But if you give something like this, what does the right hand side specify? Right hand side specify the first argument is a regular expression. It says the first character is Rthen a word. Plus means one or more occurrences followed by a. You see that the first such match will take place here because we have a R ends with a. We have this word character, this a is that word and plus means one or more. This m is her one or more occurrence. So Rama R a m a matches this regular expression. So this Rama will be replaced by Bharat in this example. Now the string will become Bharat and Lakshman went to the forest. If you look at the last one, here substitute the first one says aeiou within square bracket that means any one of these slash I. So you search the string for the occurrence of a vowel.

The first occurrence R a m a, this a is the first occurrence replace it by I. So Rama will become Rima R I m a. So this will be the effect of the third this Substitution. Now here we have another example. Here the string is A year has 11 months. In the first Substitution example we have substituted for a digit followed by one or more characters by 12. Now in this case the first match here will be this digit followed by one or more characters. Similarly last one, see this string contains a newline. The last one shows an example where the new line is removed from the string. This specifies newline dollar. This dollar anchor means form the end of the string, if there is a newline you replace it with by nothing. There are two slashes one after the other. So replace the newline by nothing means you actually delete the newline. So this is one way to remove the new from the end of a string. So just to recall you can do the same thing using the chomp function, but this is another way of doing that.

(Refer Slide Time: 39:40)



Common Modifiers

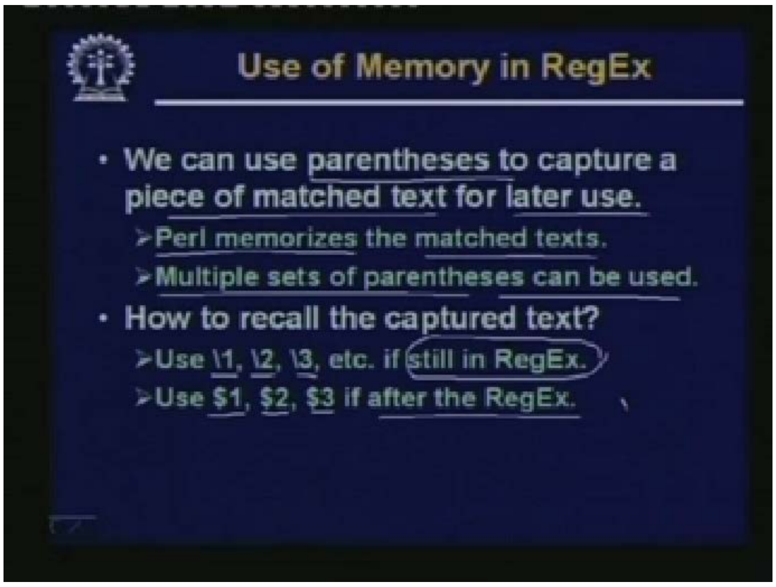
- Two such modifiers are defined:
  - /i :: ignore case
  - /g :: match/substitute all occurrences

```
$string = "Ram and Shyam are very honest";
if ($string =~ /RAM/i) {
    print "Ram is present in the string";
}

$string =~ s/m/j/g;
    # Ram -> Raj, Shyam -> Shyaj
```

There are couple of command frequency. Modifiers which can modify the meaning of a match or a Substitution. One is slash I which is used to indicate that we are doing case sensitive matching. Case can be ignored, that means we do not make any distinction between lower case and upper case. By default Perl is case sensitive, it considers small a and capital a to be different but we can make it ignore case by giving this slash I and slash g, as I mentioned before, if you give this then it will match or substitute all the occurrences of the substring. So some small examples here the first one says Ram and Shyam are very honest in this if statement we are making a match.

This RAM is all capital but we have followed it with an I RAM slash I. This means you are mentioning that you make this match ignore the keys. So this RAM and this Ram match will be successful although some letters are lower case some letters are upper case here. So this match will be successful and this string will be printed. In the second example you see here we have given substitute. The letter m by the letter j followed by slash g; g means global all occurrences. This means you substitute all occurrences of m by j. So in the string in Ram there is one in Shyam there is one. So after this execution Ram will become Raj Shyam will become Shyaj Ram and Shyaj are very honest, this will be resulting string.

(Refer Slide Time: 41:42)

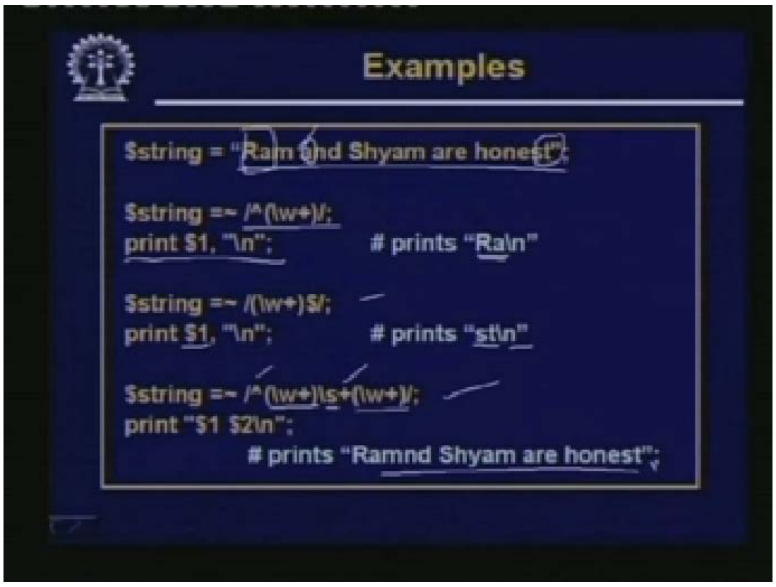


Use of Memory in RegEx

- We can use parentheses to capture a piece of matched text for later use.
  - Perl memorizes the matched texts.
  - Multiple sets of parentheses can be used.
- How to recall the captured text?
  - Use \1, \2, \3, etc. if still in RegEx.
  - Use $1, $2, $3 if after the RegEx.

Now in regular expressions we can memorize a few things. We can use parentheses to capture a piece of matched text which you can use in later comparisons like we are making some kind of matching. Now we want to use the result of that matching in a next statement or somewhere little later. So using this concept of memory we can use this kind of a feature. So the basic idea is that Perl memorizes the matched texts and you can have in general multiple sets of parentheses. To memorize multiple things well will some examples to illustrate how this work. The first thing is that how to recall the captured text because whatever you specify in parentheses will be remembered. That is what we have said there are two ways. You use reverse slash 1, 2 or 3. If you want to use match text while you are inside the regular expression itself. The same regular expression then you use slash 1, slash 2, and slash 3. But if you have finished the regular expression and you are outs ide it and still you want to use the last value, matches then you use dollar 1 dollar 2 dollar 3 if it is after the regular expression. Let us take some example.

(Refer Slide Time: 43:15)



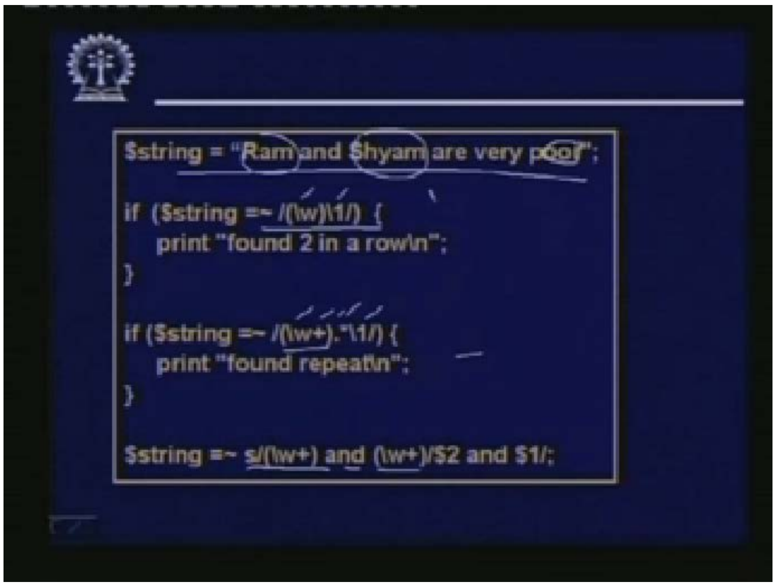
Examples

```
$string = "Ram and Shyam are honest";

$string =~ /^(\w+)/;
print $1, "\n";            # prints "Ra\n"

$string =~ /(\w+)$/;
print $1, "\n";            # prints "st\n"

$string =~ /^(\w+)\s+(\w+)/;
print "$1 $2\n";
            # prints "Ramnd Shyam are honest";
```

Here is a string “Ram and Shyam are honest”. The first example you see there is a match. Here we are matching carrot, means from the beginning of the string within bracket word followed by one or more characters. Now here the first or the smallest substring for this match will occur is R and a here. And since it is in bracket this R and a will be remembered and if later we print this is outside the regular expression, if I give dollar one then that matched value will be printed. Let us take the second example. Here again within bracket we give word plus dollar. So from the end there must be word character at least one or more character. So the smallest string is s and t.

So here again we give dollar one so that statement will print here and this newline. Now the third example is a case where we are using multiple parentheses you see form the beginning of the string. This carrot a word followed by one or more character followed by a space, followed by one or more character plus again word followed by one or more character. Now if you print this the first one will print Ram. The second one will print this. There is a space this space will come out this plus minimum will be this a and word followed by this plus typically will take up to the end of this string. So “Ramnd Shayam are honest” this is what will get printed.

(Refer Slide Time: 45:25)

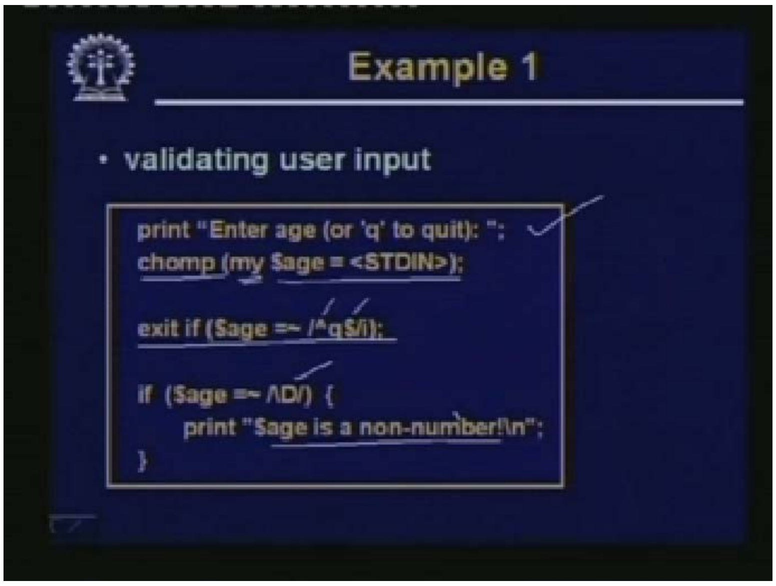


```
$string = "Ram and Shyam are very poor";

if ($string =~ /(\w)\1/) {
    print "found 2 in a row\n";
}

if ($string =~ /(\w+).*\1/) {
    print "found repeat\n";
}

$string =~ s/(\w+) and (\w+)/$2 and $1/;
```

This is another example which shows you the use of the values within the regular expression itself. So again the same string look at this comparison with bracket a word followed by slash one slash one means the last match this means that whatever was matched the same thing appears immediately after that. So, on this string if you make this kind of comparison this will becomes true out here. There are two o's appearing one after the other. Right there is a word character followed by the same thing this means [46:06 word not clear]. Next example tells you that there is a word character followed by one or more followed by a dot.

At least one more character star zero or more followed by this match. This means this checks whether the same thing repeats more than one time. This is a way to check whether this particular string is repeating. The last one is a method to swap two things. Say here you are substituting word one or more and word one or more substitute it with dollar two and dollar one. This you are outside the regular expression this slash has end. That is why using dollar for the first match should be Ram next match should be Shyam should be replaced by Shyam and Ram. So the new string will be Shyam and Ram are very poor.

(Refer Slide Time: 47:13)



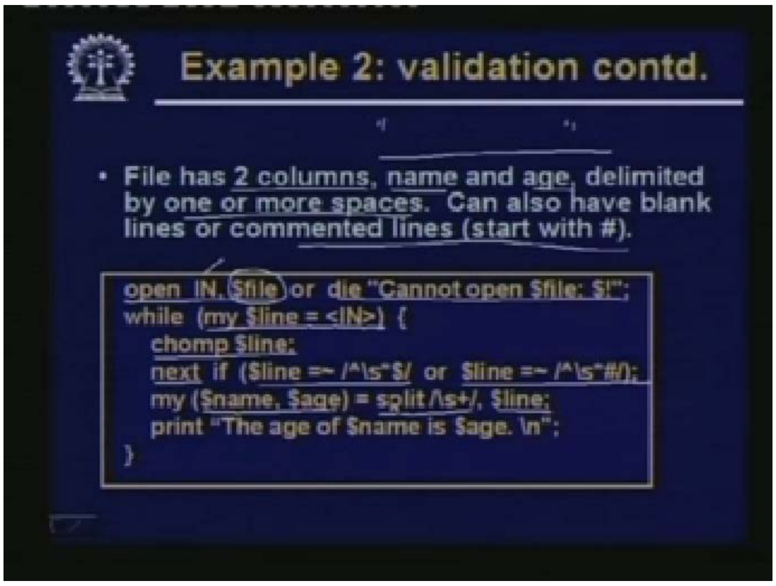
Example 1

- validating user input

```
print "Enter age (or 'q' to quit): ";
chomp (my $age = <STDIN>);

exit if ($age =~ /^q$/i);

if ($age =~ /\D/) {
      print "$age is a non-number!\n";
}
```

Now some examples where we are trying to validate some user inputs. We will see that how this can be used. Well here basically what we are trying to do is. [47:17 word not clear] A prompt is displayed where the user is asked to Enter the age and if the user wants to exit user, will type q. So the user enters the age from the STDIN, these are local variables. That is why I used my removing the newline character by chomp. We are exiting if age equal to q from the beginning q, that means q is the only character in the line. It has an anchor at the beginning also at the also has an anchor at the end and I means you can type in small q or capital q ignore case right after that you check if age. If you have a capital D means not a digit, you have read something if there is any non-digit character. There then you print that it is a non number y have entered something wrong. Here is one simple example. Let us take another example.

(Refer Slide Time: 48:20)



Example 2: validation contd.

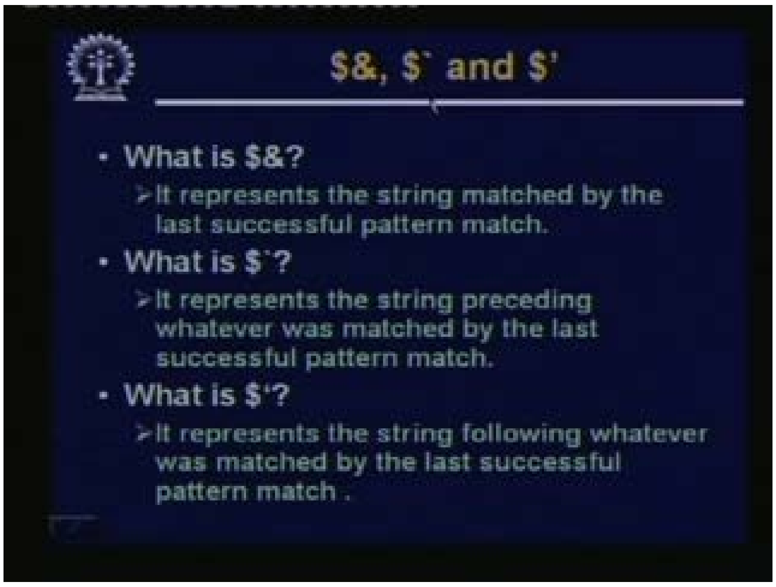
- File has 2 columns, name and age, delimited by one or more spaces. Can also have blank lines or commented lines (start with #).

```
open IN, $file or die "Cannot open $file: $!";
while (my $line = <IN>) {
   chomp $line;
   next if ($line =~ /^\s*$/ or $line =~ /^\s*#/);
   my ($name, $age) = split /\s+/, $line;
   print "The age of $name is $age. \n";
}
```

Suppose you have a File of text, there are two columns. First columns contains name, second columns contain age and between name and age there will some spaces. Just assume for the time being. That this name [48:37 word not clear] within double quotes fine there can be blank lines commented lines starting with hash also. So we want to validate this kind of File whether the data is more or less correct or not. Firstly we are opening the file, suppose we assume that the file name was there in dollar file you are [49:06 word not clear] the file handle is IN. Die if it Cannot open we need a print error message. In a while loop while we are going through the file contents line by line, the first line we are reading dollar line equal to this file handling IN. We are removing the newline character from the end of the line. We are skipping this particular loop.

Next means skipping the loop and go to the next iteration. If S line slash star dollar means it is a blank line the whole line is a blank or slash star hash means it is starting with a hash. So if either of this is true means that means this is either blank line or comment line you can skip it otherwise. You take this name and age and split. Split with means the name and age with respect to space. It can one or more space that is why we are giving plus from line then you can print a report "The age of name is age". So that by looking at the printed report you can check whether you have entered the data correctly or not.

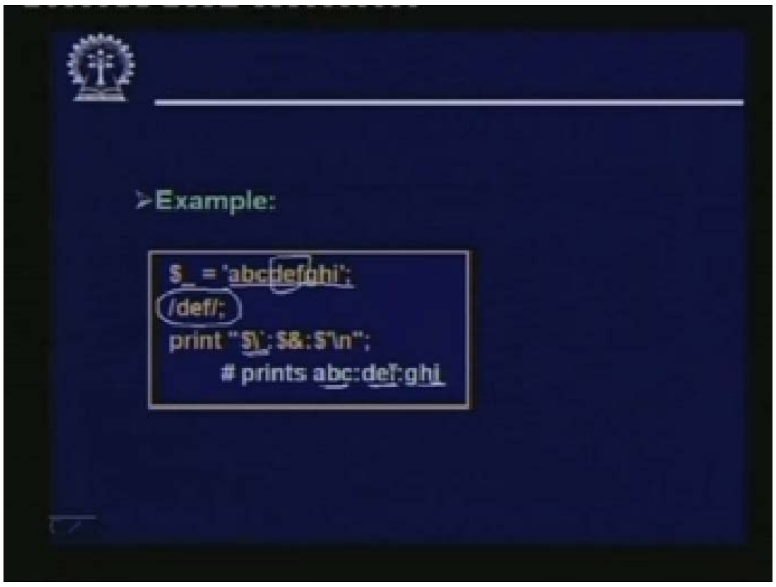
(Refer Slide Time: 50:34)


$&, $` and $'

- What is $&?
  - It represents the string matched by the last successful pattern match.
- What is $`?
  - It represents the string preceding whatever was matched by the last successful pattern match.
- What is $'?
  - It represents the string following whatever was matched by the last successful pattern match .

Now Some Special Variable sin Perl what mentioning, there are three such special variables. Dollar ampersand means, the string matched by the last successful pattern match. Dollar reverse quote means, the string preceding whatever was matched by the last successful pattern match. Dollar forward quote means, the string following whatever was matched by the last successful pattern match. That means in a string if you get a match here we are talking about the preceding string and the following string there are three things. Let us take an example to illustrate.

(Refer Slide Time: 51:13)

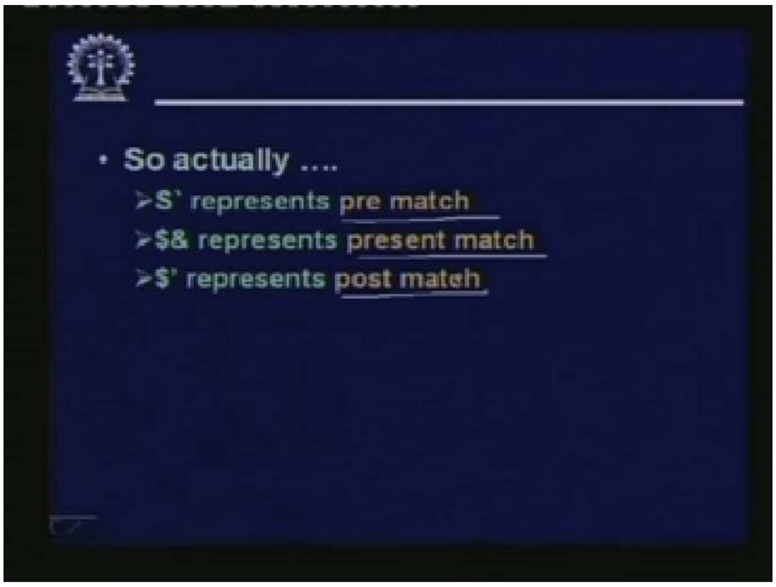


➢Example:

```
$_ = 'abcdefghi';
/def/;
print "$\`:$&:$'\n";
        # prints abc:def:ghi
```

Suppose I have a string abcdefghi, we are making a match. Well since the default string we simply write def we are checking for def. So match has occurred out here in def, we are printing dollar slash see. Since this quote is a special character. That is why we are escaping it with the reverse slash character, [51:40 word not clear] dollar slash, dollar ampersand, dollar forward slash. So in the first case it refers to the preceding string. So whatever there was before the match string abc that will printed followed by a colon dollar ampersand. Means the present match dollar forward quote means whatever is there after the match ghi. So this means these.

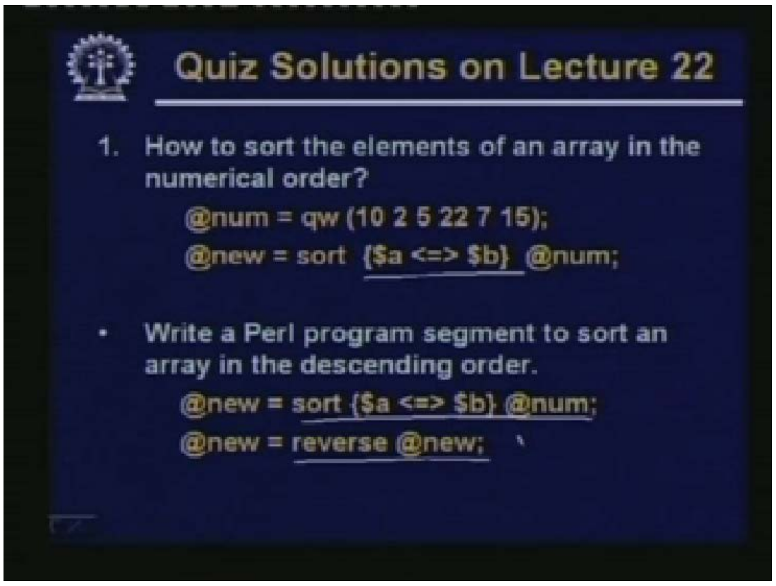
(Refer Slide Time: 52:07)



- So actually ….
  - ➢$` represents pre match
  - ➢$& represents present match
  - ➢$' represents post match

So actually the first one represents something called pre match. Whatever is before there represent the present match the exact substring match occurred and this represents whatever is after that. So with this we come to the end of today's lecture on a discussion on Perl. We shall be continuing our discussion on Perl next week. There are several other things to discuss but before that let us go through the questions from last week quizzes.

(Refer Slide Time: 52:37)



Quiz Solutions on Lecture 22

1. How to sort the elements of an array in the numerical order?

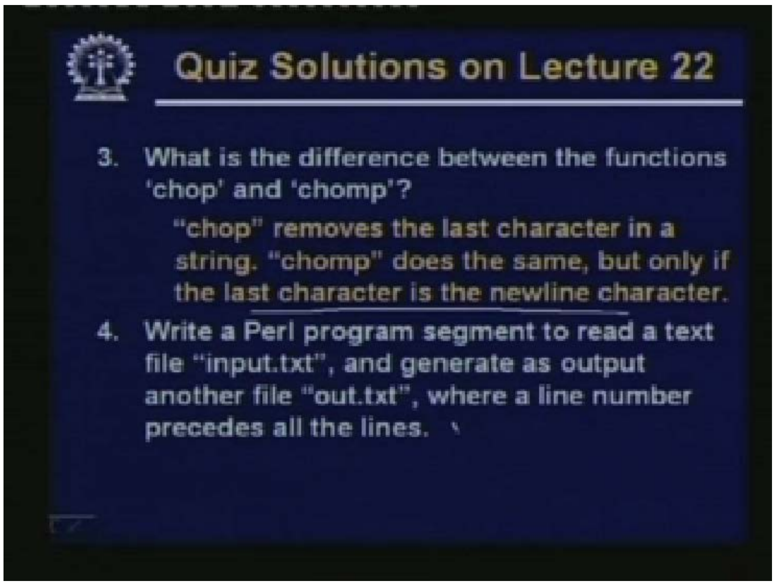
```
@num = qw (10 2 5 22 7 15);
@new = sort {$a <=> $b} @num;
```

- Write a Perl program segment to sort an array in the descending order.

```
@new = sort {$a <=> $b} @num;
@new = reverse @new;
```

How to sort the elements of an array in the numerical order?
Well this we have mentioned by using the special construct in the sort we can do that.
Write a Perl program segment to sort an array in the descending order.
Well directly you cannot sort in descending order. You first sort in ascending order then you reverse the entire array in this two steps you can do.

(Refer Slide Time: 53:03)



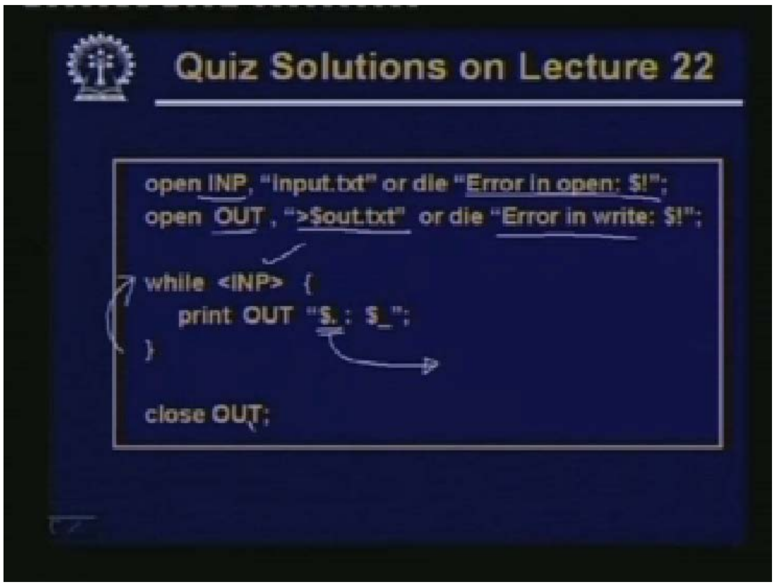
Quiz Solutions on Lecture 22

3. What is the difference between the functions 'chop' and 'chomp'?
   "chop" removes the last character in a string. "chomp" does the same, but only if the last character is the newline character.
4. Write a Perl program segment to read a text file "input.txt", and generate as output another file "out.txt", where a line number precedes all the lines.

What is the difference between the functions chop and chomp?
Chop removes the last character in a string irrespective of what character it is. Chomp will do the same thing but only if the last character is the newline.

Write a Perl program segment to read a text file input txt and generate as output another file out dot txt where a line number precedes all the lines.
So you are basically reading lines from one file outputting it to another file you are putting a number colon then the line. That is how you are outputting it.

(Refer Slide Time: 53:41)

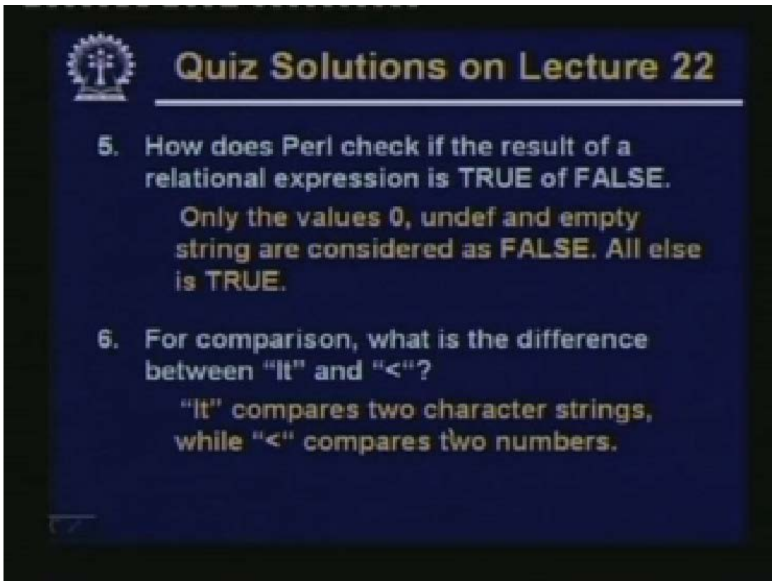


Quiz Solutions on Lecture 22

```
open INP, "input.txt" or die "Error in open: $!";
open OUT, ">$out.txt" or die "Error in write: $!";

while <INP> {
   print OUT "$. : $_";
}

close OUT;
```

So the program will look like this first you open the input file with Error messages open the output file. Greater than means output again Error message while INP means you are reading the lines one by one from the INP by default it goes to dollar underscore. Then you are printing OUT. Dollar dot is a special variable. Again this represents the line number. This represents the line number line number gets printed colon. Then the contents of the line this will repeat throughout the while loop. Finally you close the files.

(Refer Slide Time: 54:23)



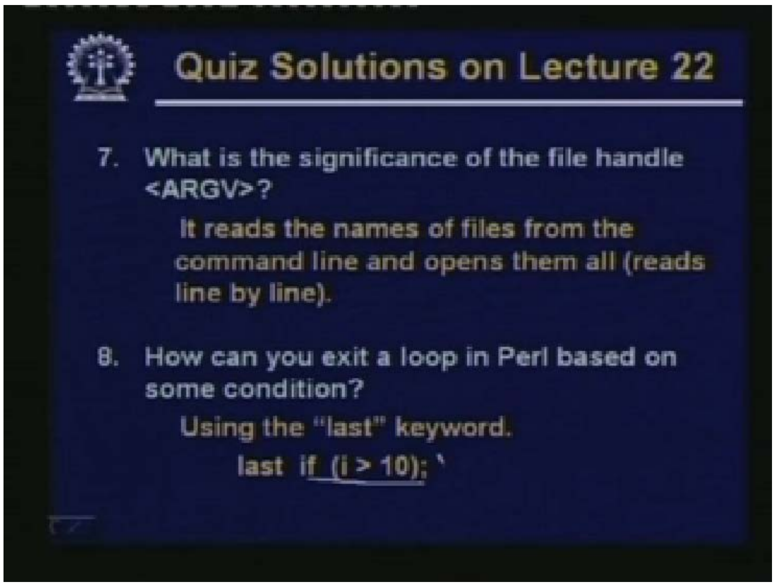
Quiz Solutions on Lecture 22

5. How does Perl check if the result of a relational expression is TRUE of FALSE.
   Only the values 0, undef and empty string are considered as FALSE. All else is TRUE.
6. For comparison, what is the difference between "lt" and "<"?
   "lt" compares two character strings, while "<" compares two numbers.

How does Perl check if the result of a regular expression is TRUE or FALSE?

Well Only the values 0 undef and empty string are considered as FALSE All else everything else is TRUE.
For comparison what is the difference between lt and this less than symbol?
lt we use to compare strings, less than we use to compare numbers.

(Refer Slide Time: 54:48)

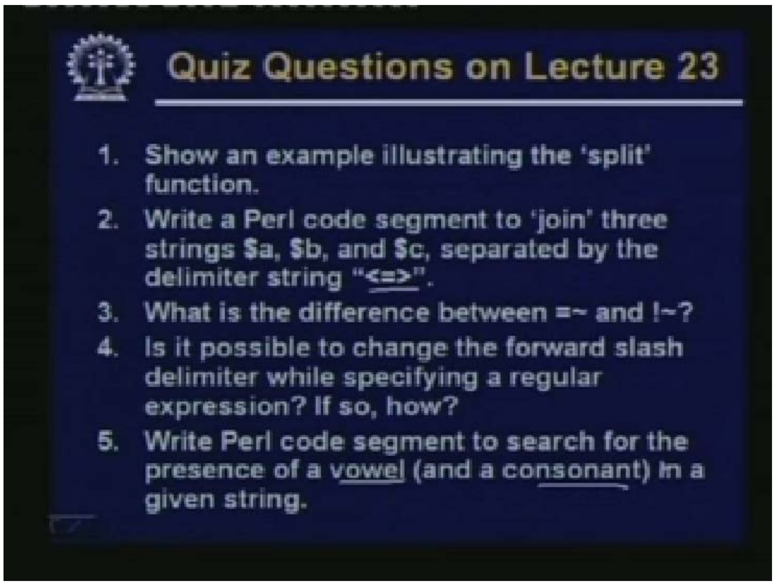


Quiz Solutions on Lecture 22

7. What is the significance of the file handle <ARGV>?
   It reads the names of files from the command line and opens them all (reads line by line).
8. How can you exit a loop in Perl based on some condition?
   Using the "last" keyword.
   last if (i > 10);

What is the significance of the file handle ARGV?
It reads the names of files from the command line and opens them all that means all the files will read line by line and lines will be written in sequence.
How can you exit a loop in Perl based on some condition?
Using the last keyword like last followed by a condition.
Now some questions from today's class.

(Refer Slide Time: 55:12)



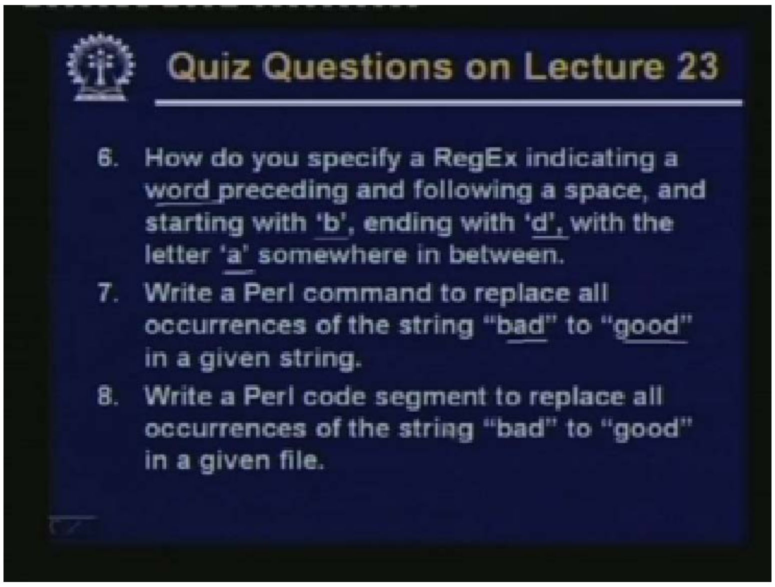
Quiz Questions on Lecture 23

1. Show an example illustrating the 'split' function.
2. Write a Perl code segment to 'join' three strings $a, $b, and $c, separated by the delimiter string "<=>".
3. What is the difference between =~ and !~?
4. Is it possible to change the forward slash delimiter while specifying a regular expression? If so, how?
5. Write Perl code segment to search for the presence of a vowel (and a consonant) in a given string.

Show an example illustrating the split function.

Write a Perl code segment to join three strings a b and c separated by the delimiter string this one.
What is the difference between equal to tilde and exclamation tilde?
Is it possible to change the forward slash delimiter while specifying a regular expression? If so, how?
Write Perl code segment to search for the presence of a vowel and a consonant in a given string? That means these are two questions you search for a vowel you search for a consonant How to do that?

(Refer Slide Time: 55:48)

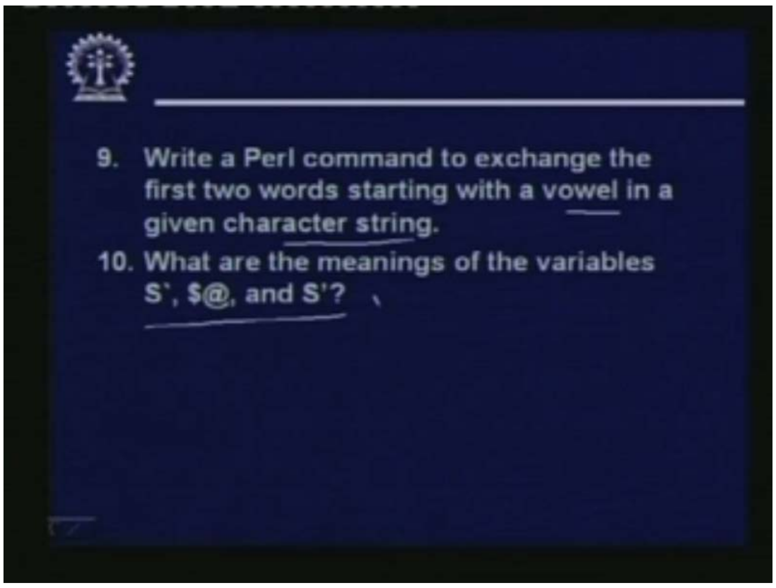


Quiz Questions on Lecture 23

6. How do you specify a RegEx indicating a word preceding and following a space, and starting with 'b', ending with 'd', with the letter 'a' somewhere in between.
7. Write a Perl command to replace all occurrences of the string "bad" to "good" in a given string.
8. Write a Perl code segment to replace all occurrences of the string "bad" to "good" in a given file.

How do you specify a regular expression indicating a word preceding and following a space and starting with b so the word has a preceding space and a following space it starts with bends with d with the letter a somewhere in between?
Write a Perl command to replace all occurrences of the string bad to good in a given string?
The last question is same but not in a string you have to replace it in a given file. So with this there a couple of other questions.

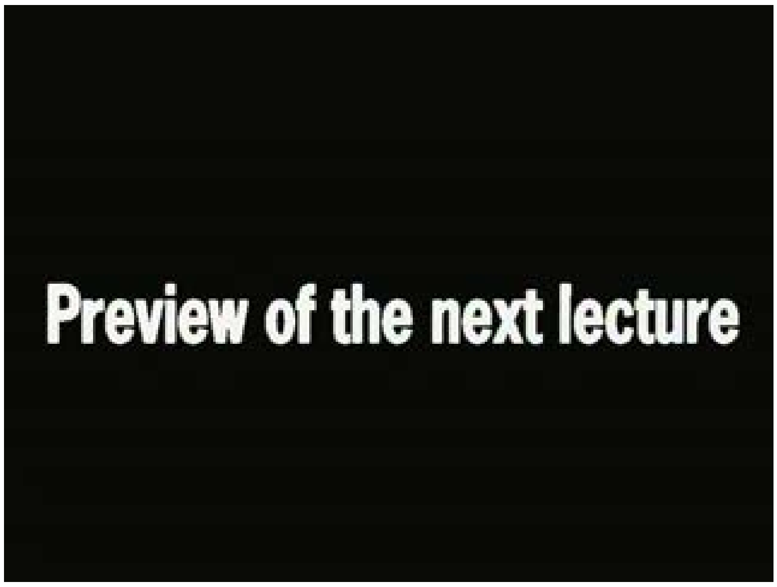
(Refer Slide Time: 56:24)



9. Write a Perl command to exchange the first two words starting with a vowel in a given character string.
10. What are the meanings of the variables S`, $@, and S'?

Write a Perl command to exchange the first two words starting with a vowel in a given character string you find out two such words and then exchange them.
What are the meanings of these three special variables dollar as I have just mentioned?
In our next class will be talking about something called associative arrays in Perl and we shall be actually looking at some real cgi scripts. If you want to write it in Perl, how do we do it? So with this we come to the end of today’s lecture. Thank you.

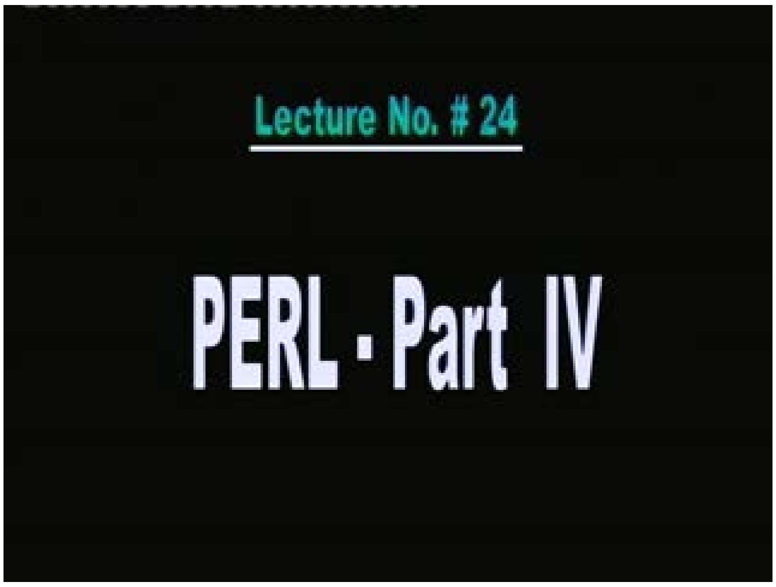
(Refer Slide Time: 57:03)



Preview of the next lecture

Preview of next lecture.

(Refer Slide Time: 57:04)


Lecture No. # 24

PERL - Part IV

PERL – Part IV

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