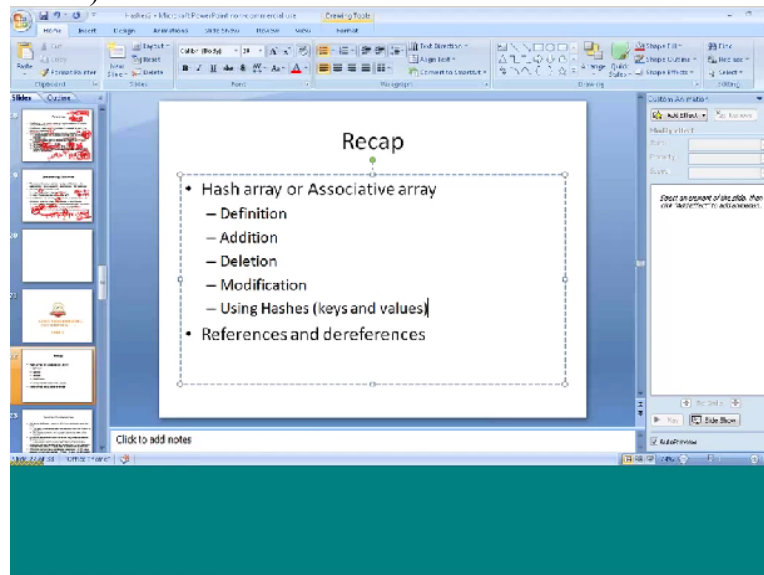
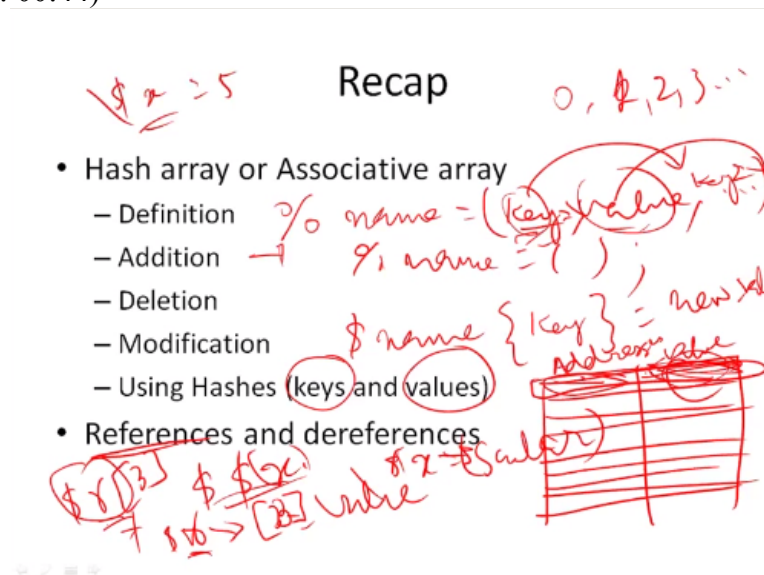


(Refer Slide Time: 00:01)



LINUX PROGRAMMING AND SCRIPTING- PERL 8

Hi everyone again welcome to the LINUX programming and scripting course today we will be continuing the lecture on Perl, programming language today lecture of eight Perl programming language so last lecture we covered several topics actually two main topics, then we went into the detail of those two I will recap that then topic of today, so let us look at last time.
(Refer Slide Time: 00:44)



What we covered the main the two topics that we covered last time the hash array or associative array and the references, referring of variables the Hash or the associative array essentially this is

defined we saw the definition it is basically define as % and then the main then we also saw how to add the objects into the array, how to modified the objects.

Typically I like mean the modification is \$ name and then we use the {} essentially denote that which object that we want to remove one key thing that we noticed what actually the when we actually define it has key value there and usually we see the association by using => greater than symbol to form the association typically all these key value etc that is how we can define the \$ name array, each of so this keys and value access the modification simply by calling the key. And then changing that into new value so corresponding changed and then we also saw that to initialize hash array we will just simply do the % name => just open those parenthesis that will initialize the array itself and then we use the hash arrays we use it with either the keys or the values when we can actually like just get all the keys or all the values one key thing key difference between hash array regular charm array is that hash array as only association between key and the value it does not have any association with the key one and key two for example. So there is no relation or value one and value two in a regular array.

We know that actually the value implemented by 0 1 2 3 etc, so each object will have it is own index, essentially like I mean special key and the values so and the gain we can access the keys basically and then we can sort on the values and then we can get various resorted keys what is the value foe resorted value what are the keys those kind of things we can easily get from this array.

This is every useful as two dimensional array has I mentioned we can denote like points of in a plain things like that essentially we also saw one example which is kind of dictionary creation either a dictionary creation or even a phone book creation were we can have key indexes as the name of the people and the values are the corresponding phone numbers so and then couple of other thing that we saw actually like the addition is not actually, does not follow any rule.

It is just get added and element, so we cannot say this is okay course in the end of the Q or end of the array or beginning of the array we cannot say anything like that it can go anywhere in the middle because it is almost random access so essentially like give the keys, that particular element we can access directly we can have serially go through 1 2 3 4 kind of keys before accessing the element so lot of good features.

That one thing that I mentioned last time also that lot of programming actually revolt around nationalize in fact he will see any kind of practical applications this is a whole bunch of Hash array that we get use in order to actually do this meaning to programming on main program you

suppose and then two popular features Perl that gets use more often one of them is a Hash array the another one talk about yesterday.

Once you match the these two elements you can pretty much write any program inside Perl of course there are some data structure that we covered in the next lectures, couple of lectures also the other thing that we noticed or we learnt in the lat lecture was referencing and dereferences the variables so what is reference of a variable is thing but the address where it is stored so you know that actually the memory structure.

Is usually there is all these it almost like a table you can think of this and then one side is addresses and then this is a map so a memory with lie a 32 gate will have like 32 kids of addressing space each one store a byte or bit 32 based on you can have that many storage capability so bit system may be like a addressing meaningful here is byte 8 ,8 bit can be address a on e short so the 8 bits will have unique address correspond to it .

So variable is nothing but one of this chunk of data base it an address and corresponding value and typically we only we say like \$ X=5 only have the \$ as the value now what is the address where this 5 is stored in order to get that is what we mean by referencing, the referencing we can achieve it by an character like a \ \$ X this gives scalar value and this scalar represent the address of that particular variable.

So like Arrays things like that so we can easily get these references we also saw like a module show you couple of things so here essentially like I mean list can be created by actually this [] this reference to Hash as if created by curly braces essentially so these are the some of the things that we already saw and then all of them are the noticed the \$ because it is a kind of a variable which is scalar quantity, so which is in denoted by just by \$.

So once we have this how do we reference a reference already given the reference in order to do that we use double \$ with X and then that gives as basically back provide the access the scalar quantity which is a reference itself and we can get back the actual value so then we can use this a \$ \$ X to modify the value the reference structure and that we also we saw here actually this is array actually.

I like modify one of the variables we use the – arrow to – greater than actually modified the element the particular element and one thing that we noticed was this concept of \$ R 3 is not = to \$ R arrow I think I mean we have to noticed basically for a given R the R is not changing basically the same here the R is use as an array variable here R used as a reference that the big difference why they are not equal, so wanted to understand the this one.

So I think like I mean this is we can introduce today's topics, today we will be talking about functions which is the natural profession for the topic, and then we will also talk about some of the, the other biggest topic of the mostly most widely used item in Perl which is the regular expressions.

(Refer Slide Time: 11:50)

Function Fundamentals

- A function definition consists of a function header and the body
 - The body is a block of code that executes when the function is called
 - The header contains the keyword `sub` and the name of the function
- A function declaration consists of the keyword `sub` and the function name
 - A declaration promises a full definition somewhere else
- A function call can be part of an expression. In this case the function must `return` a value that is used in the expression
- A function call can be a standalone statement. In this case a return value is not required. If there is one, it is discarded

So let us look at functions essentially function is the short form notation for repeatedly used code, so essentially you can so if you here using something basically like that you are over and over and in order to compute certain things, you can put it under function and then call this function whenever you need it, so this reduces the is of programming, in Perl has you know like there is no concept of compilation.

So in a regular programming language like C or C++ when you write function the function are actually compiled during the compiled time and then they are they come in to the actual programs, and they can get the various parameters, similar to like subroutine and the functions here in Perl since it is not a compile language only have functions there are no subroutine as such, so the function consist of a function header and the body.

So the body is a block of the code that executes when the function is called, the identification of the header, is through this keyword `sub` and then the name of function, so when we call like `sub` and then followed by the name that means that it is the it is function, and then so, so the again the same thing basically like mean so the function declaration consists of the keyword `sub` and then the function name the declaration promises the full definition somewhere else.

So you can just define it and then basically you can actually write the function somewhere else, and the function call itself definition actually use the function in a particular code, this can be

part of an expression as well, so in this case a function must return a value and that is that we get used in the expression, the function call can also be a standalone statement, in this case return value is not required, if there is one, that is pretty much discarded. So sometime why there is a return value which is just used for testing purpose like I mean whether the function was executed correctly or not.
(Refer Slide Time: 14:35)

Function Return

- When a function is called, the body begins executing at the first statement
- A return statement in a function body causes the function body to immediately cease executing
 - If the return statement also has an expression, the value is returned as the value of the function
 - Otherwise, the function returns no value
- If execution of a function reaches the end of the body without encountering a return statement, the return value is the value of the last expression evaluated in the function

So whenever a function is called the body begins executing at this at the first same of this function, the return statement in a function body causes the function body to immediately stop executing that particular function, and then return the control back to the original program, and then the return statement is also has an expression if the return statement also has an expression values return as a value of function itself.

So return statement is another key thing basically it determinate the function by using the return statement and then if the return statement just has the plainly just it when we have return statement without any expression that is the function that returns value, and if the execution of the function reaches the end of the body without encountering the return statement, then the return value is the value of the last expression value in the function.

So Perl assumes the implicit return essentially like so you got to be careful so if you do not specify anything basically like you have a junk of code, the last expression evaluated will become the return value for the particular function, so when you are using it you go to be careful.
(Refer Slide Time: 16:14)

Local Variables

- Variables that are not declared explicitly but simply assigned to have global scope
- The `my` declaration is used to declare a variable in a function body to be local to the function
- If a local variable has the same name as a global variable, the global variable is not visible within the function body
- Perl also supports a form of dynamic scoping using the local declaration
 - A `my` declaration has *lexical* scope which works like scope rules in C, C++ and Java

So the variable, so here there is a concept of a local variable, one thing is essentially a in a function the variable hat are not declare expressively, but simply as sign to have a global scope, so you do not have to hang actually like a define declared explicitly, and they all have the global scope which means that actually it stands across the whole program, in order to use a local variable this is something that we saw in the previous example also.

The keyword `my` declaration is used to declare the variable in a function body and that is local function, if the local variable as a same name as a global variable, the global variable is still not visible with the function body, so only used that the local variable is what is more get used, it also supports the form of dynamic scoping using a local declaration, so and then the `my` declaration is what is the eating and that has a electrical scope it works similar to the rules of the C, C++ and java.

We were also see some of the other types of actually declaring variables in the other topics in the future, some of the other programming concept s basically how we can pass variables that means the subroutines are the functions parts.

(Refer Slide Time: 18:00)

Parameters

- Parameters used in a function call are called actual parameters
- Formal parameters are the names used in the function body to refer to the actual parameters
- In Perl, formal parameters are not named in the function header
- Perl supports both pass-by-value and pass-by-reference
- The array @_ is initialized in a function body to the list of actual parameters
 - An element of this array is a reference to the corresponding parameter: changing an element of the array changes the corresponding actual parameter
- Often, values of @_ are assigned to local variables which corresponds to pass-by-value

Sub xyz (\$A, \$B, \$C)
 formal
 Actual
 Sub xyz (\$P, \$Q, \$R);

Now let us talk about another key thing that is the parameter, so these all like I mean actually this one, we know that block of the code that is basically like what do we get, the header actually contains the sub and then the name of the function and we can also declare parameters at the point, the parameter that are used in the function call are called the actual parameters, the formal parameters are used in the function body.

And refers to the actual parameters, so actual parameters is linked to the formal parameters, so you can say is declare the function sub XYZ \$A, \$B, \$C, and then when you are using XYZ, \$P, \$Q, \$R and like system execute within the code, so if you look at this one basically like that, so in this subroutine essentially like an as a, these are the actual parameters and this are the form, so in the formal parameters are not named in the function header.

So we do not have to actually specify this, without this they will get passed, and the Perl actually supports both pass by value and pass by reference as means of passing the parameters in the function, so the special array which is like that array underscore is initialized in a function body to the list, to the list of actual parameters, and the element of this array is a reference to the correspondent parameter.

Changing the element of the array, changing the correspondent parameter, so again we already talked about the references and the dereferences so think about this one basically in this array we only store the references so as we change the elements that corresponding those parameters are actually the correspondent actual parameters changed as well, so the values of this particular array are assigned to the local variable which correspondent to the pass by value okay.

So these things will be like clear when we go through an example, we have time to do an example today if not we will be doing an example in the next lecture.

(Refer Slide Time: 21:30)

Parameter Usage Examples

- This code causes the variable \$a to change

```
sub plus10 {  
  $_[0] += 10;  
}  
plus10($a);
```

Handwritten notes: $a = 0$ Actual Parameter, $a = 10$
- The first line of this function copies actual parameters to local variables

```
Sub f {  
  my($x, $y) = @_;  
}
```

Handwritten note: $@_$

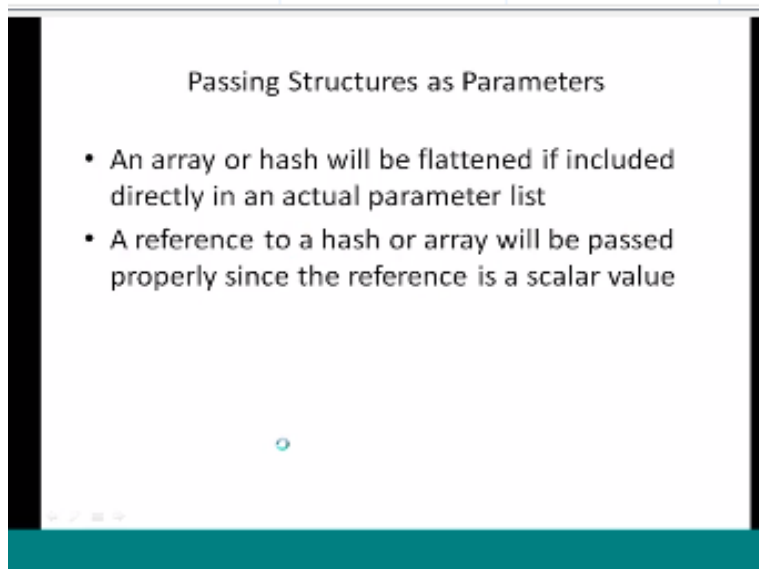
Okay Here is some, some other examples here, so here we declare sub plus 10 as the function and basically like here we do not have to specify the formal parameters, so here we just specify \$ - 0 + = 10 this mean that expanded has return a value which is whatever the parameter. That you get + 10 so that is name of the function. And then the usages considered here basically so this become the actual parameter. So we do the + 10 of \$ A, the \$ A becomes whatever the P dash value + 10. So previously like \$ A you are assign to 0 then after + 10 \$ A, \$ A will power value of 10. So the first line of this function for the action parameter to the local variables. So there is this function that we can think of which takes basically like in the values essentially. And it is basically assigns the local variables from these from the array on the scope.

(Refer Slide Time: 23:20)

Passing Structures as Parameters

- An array or hash will be flattened if included directly in an actual parameter list
- A reference to a hash or array will be passed properly since the reference is a scalar value

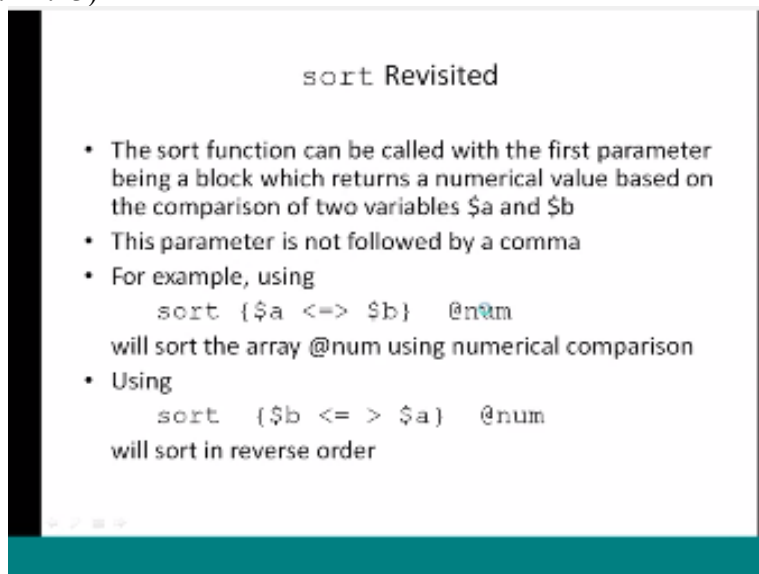
Okay so let us look how you do we pass the structures as parameters.
(Refer Slide Time: 23:48)



Passing Structures as Parameters

- An array or hash will be flattened if included directly in an actual parameter list
- A reference to a hash or array will be passed properly since the reference is a scalar value

Trouble of thinks that note here is one is array or hash will be flattened it included directly in an actual parameter list. So it own keep in has is basically they have flattened the reference to the hash or array will be pass properly since the reference the reference is a scalar value. So if you passing is by reference those references will be passed correctly.
(Refer Slide Time: 24:13)



sort Revisited

- The sort function can be called with the first parameter being a block which returns a numerical value based on the comparison of two variables \$a and \$b
- This parameter is not followed by a comma
- For example, using

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

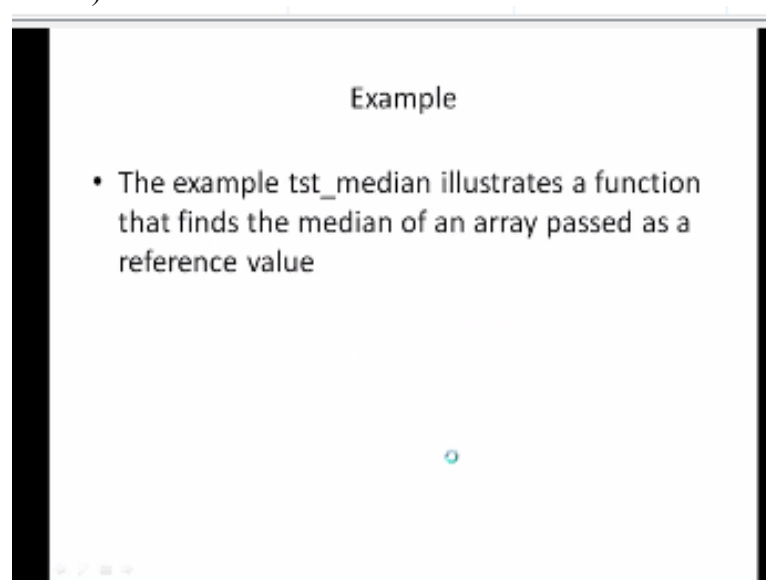
will sort the array @num using numerical comparison
- Using

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

will sort in reverse order

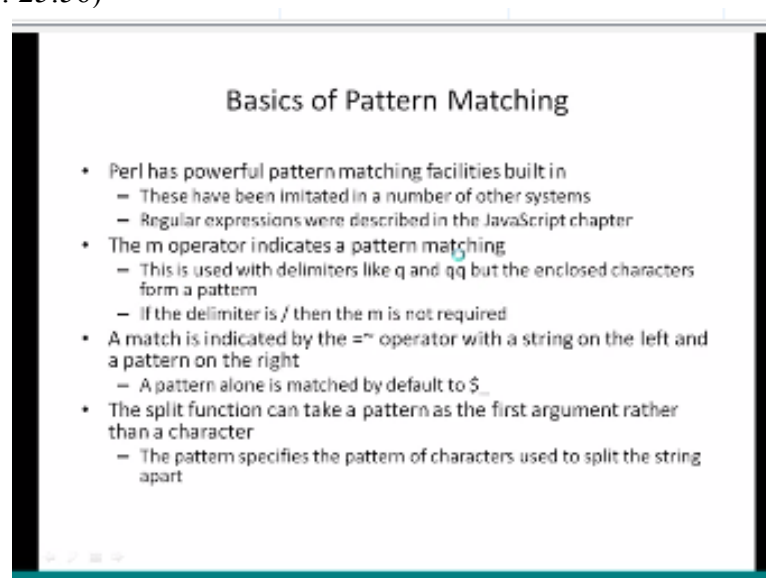
So let us look at sort function, the sort function can be called with the first parameter being a block which contain the numerical value based on the comparison of two variable \$ A and \$ B. And this parameter is not followed by comma. For example like using the sort there is a <=> b the sort will numerical array the Num using the numerical comparison. Where has when we do this A and B in the other way than little sort in reverse order? So this way basics okay.

(Refer Slide Time: 25:13)



So we can actually now write a function essentially missing whatever we have learn this could be a good day to actually find out how to write the function we will discuss this example in the next lecture. But we know try actually write this function test median especially like to find median of an array. Passed as a reference value, okay so this could be an example that we will discuss in the next lecture.

(Refer Slide Time: 25:56)



So I think like mean so this is mainly the functions how much we define the function hash, now let us go to pattern matching this is another second most important topic in Perl. Once image the pattern matching and the hash array is pretty much you got a 80% of Perl covered already. So let us look at the pattern matching some of these things that we talked about these in the earlier

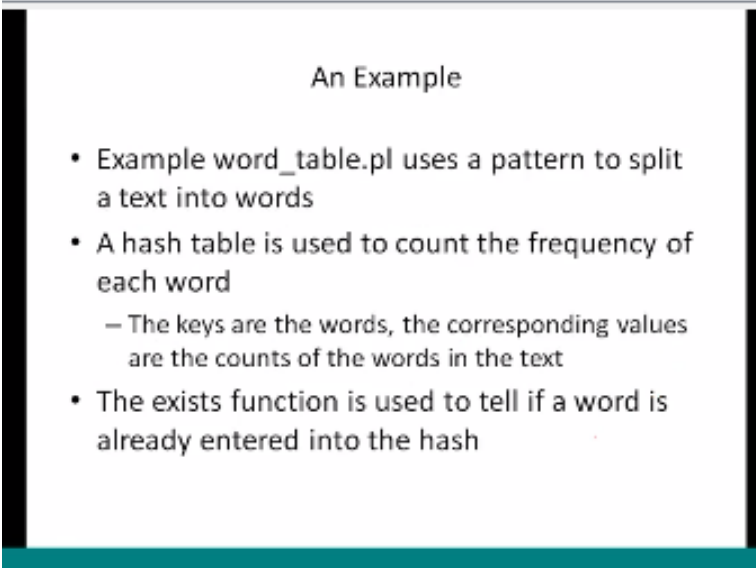
lectures. So I would like to remain some of them and essentially Perl has a very powerful pattern matching facility built in.

Is having imitated in a numbers others with the regular expression that is scribed in Java script chapter. And this is also like pumple that Java scripts forward these regular expressions, so the pattern matching itself is a indicated by this M operator. M we can think of M has a match operator, and this is used with D limited like a Q and Q, Q but a enclosed characters form a pattern essentially.

So the d limited is just the / then the M itself is not required the match itself is indicated by the = till the operator. With the string of the left and then the pattern the sting variable should be from variable on the left, and then the pattern on the right. Actually like in the variable containing the string or even dist a nation string is obtain in the scale actually. Okay, so if you just give a pattern the default string that is assumed \$.

And the scope which is barren line the think of and then the split function can take the pattern has the first argument rather than a character. So even for a split function we can declare the pattern. Now the pattern itself, the pattern specifies the pattern of characters used to split the string into multiple cases.

(Refer Slide Time: 29:04)

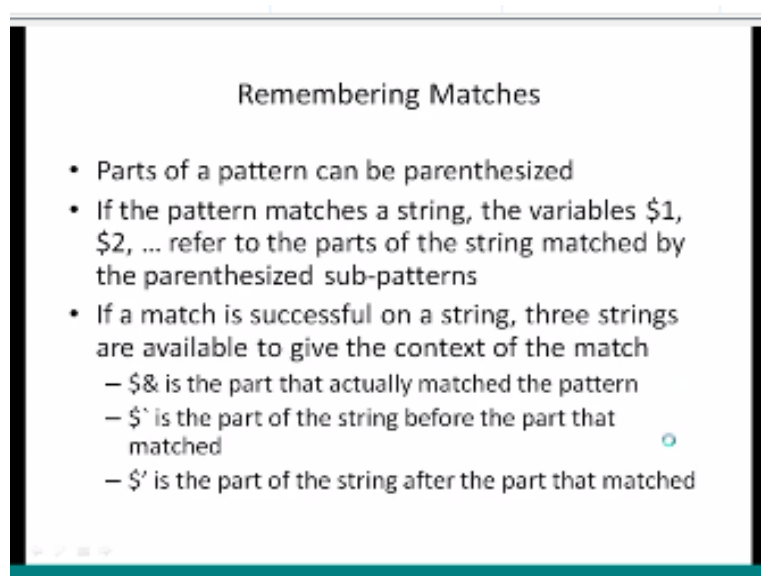


An Example

- Example word_table.pl uses a pattern to split a text into words
- A hash table is used to count the frequency of each word
 - The keys are the words, the corresponding values are the counts of the words in the text
- The exists function is used to tell if a word is already entered into the hash

So let us look at the some of the examples, so actually like a example itself here it is a properly not there we will come back to this so simple match essentially like a defend think of is valuation \$ on the scope = you can specify M operator then / and then the particular pattern followed by the /. So this is one matching structure essentially like a imagine pattern and whenever this pattern is matched it can generate like and it we can do some act which matching essentially.

(Refer Slide Time: 30:24)



So how do we remember the matches essentially like coming to we can use parenthesized and then we can also denote variable string variables to explicitly contain matches. What this mean this is essentially like $A / \$ 1 =$ and $\$ X =$ some match $X, Y, G A, B ,C$ so the parenthesized elements will be remember into this particular once. And then in fact if you tone events specify this just $X, Y G A, B, C$ or $G F$ so this particular matching pattern is always set $\$ 1$ and this is the $\$ 2$.

So the pattern matches is string the variable $\$ 1, \$ 2$ refers to the part of the string matched by parenthesized sub patterns. So for this sub pattern 1 it is remember has $\$ 1$, for the sub pattern 2 that is remember $\$ 2$. So this is basically like the way actually if do it and then essentially like you can write a match $\$$ on the scope = then you can say like $\$ 1$, if $\$ 1$ is whenever and then you can exsiccated part.

So it can actually tap into the matched component and then two processing based on the matched component. So if the matches successful on a string there are three string that are available to give the contacts of the match. So the first one is the $\$$ ampersand that give the part that actually match the pattern. And then the $\$$ this back take is the part of the string before the part that Matched and then $\$$.

The normal take which is the part of the string the after the part so in this example like even if you do not specifies like $\$ 1$ actually still available to you basically $abc PQR$ and then this the match that you wanted essentially like, or this is the string and then basically see, then the this will contain abc this is contain xyz , so these are all the like some of the shortcuts few of the matches.

(Refer Slide Time: 34:02)

Substitutions

- The `s` operator specifies a substitution
 - `s/pattern/new-string/`
- The new-string will replace the part of a string matched by the pattern
- The `=~` operator is used to apply the substitution to a string
 - If the operator is not used, `$_` is operated on by default
- A `g` modifier on the substitution causes all substrings matching the pattern to be replaced, otherwise only the first match is changed
- The `i` modifier cause the pattern match to be case insensitive

So, matching itself a to identify a particular line then will be the process line and we actually go into those matches and then identify how to actually do this matches and then how to work, make use of matches so if you are searching matches essentially like I mean you can think of it has way to search to a document for a given matches, now once we search and find out something this is like a I mean a word document we are doing in file and various occurrence of a particular string.

But the next logical step for in your those kind of matching which is find something we need to actually replace with this in new system so invert of word you can think of this as point and replace commands but how do we do it perl that is what is known as use so we know that these we saw actually `m` for matching now we introduce an new operate which is a `s` operator which is used for substitution.

Again in substitution actually find the pattern and then replace it with a new string, and then the new string replace part of the string matched by the particular pattern so we say basically like `abc` replace it where `123` then initial string is actually `s is $, xyz abc pqr` when we do this basically `$s =replace s/abc 123` the `abc` will be replaced by this one pattern the new string is actually `$s xyz $123` and then that is what a this stands for again the `=~` operator used.

Applied the substitution if the operator is not used then `$_` is operator font by the font so do not specify anything like I maen want to say that `s abc 123` then the string `$_` which is actually assign to `$_`, and then now there are things that we can attached the end I think like I mean this is something that we saw like wekly we talked about this one of the earlier lecture this perl, and there are several modifiers essentially like I mean we can add to the end actually.

One such modifier is the g modifier the g modifier on a specific causes all the sub strings matching pattern will be replace so g stance for global, and then this also another modifier I, I modifier makes is in sensitive so we sue I think of in censitive more essentially is a case sensitive so and upper case abc same as lower case, so we can put it as I g it is also like I mean with so these are all like different ways to substitute.

(Refer Slide Time:38:33)

The Transliterate Operator

- This is written `tr/char-list1/char-list2/`
- When applied to a string it causes each character of the string that appears in the first list to be replaced by the corresponding character in the second list
- If the second list is empty, the characters from the first list are deleted from the string
- The `=~` operator is used to apply the transliteration
 - If the operator is not used, `$_` is operated on by default

Now we also talk about this operator in the previous lecture in the earliest lecture this is the transliterate operator also known as tr essentially in the transliterate operator essentially we can replace a character liss and another character list so it is not only actually so think about here we this is the string first one is a pattern and the second one is has to be string we are actually taking a pattern and then replace use the string.

Here actually it can be a just a character list and the char list and this can be just pattern essentially so, when we apply this translator operator is string it causes each character of this string it appears in the 1st list will be replace it corresponding character in the 2nd list with the 2nd list is empty then the characters in the first list are just deleted from the string so use this operator with caution.

So again here also like I mean be use `=~` are the prefix operator at apply the transliterated and if you do not specify any operator and , it is basically `$_` this operator on by default.

(Refer Slide Time:40:05)

File Input and Output

open
close

- To carry out file input and output, a filehandle must be created for each file
- The open function is used to create a file handle
 - The first parameter to open is the name of a file handle
 - By convention the name is all capital letters
 - The second parameter to open is a string value naming the file and, optionally, including a character to indicate the mode of opening the file
 - < indicates open for input (default)
 - > indicates open for output, deleting the content of an existing file
 - >> indicates open for output, appending to a file that already exists

open(FH, < myFile);
open(FH, > myFile);
open(FH, >> myFile);

So let us look some other input file and out put ,so we have this concept we introduce the concept of a file handle in the previous lectures when we even we talked about perl with talk about where the talk about file handles the context of unix,in are carry out of file input and output file handle must be created form each of the file the open function is used to create a file handle so there are 2 functions in perl.

One is perl open the other one is close, so the other function is used to create the file handle, the file handle is similar to the file handle and unic essentially is a address of file but here we Have this address generate by the perl slide of a program that now allows that perticular program access the file so it is not the same as what is specified in unix file system but it is very similar to that context that concept so, the 1st paramater to open function is a name of the file handle the convention basically like keep the name as all caps which is just so that distinguish between normal variable, and then the 2nd paramater to open the string value nameing the file and optionally including the characters which is the mode of opening the file.

So typically this will be like open, sh, so this perticular declaration now opens my file we and the file handle of sh, so read means is basically I mean a it is only for input, and one thing that notice basically I do not but any characters here that because the input is the default so do not have to specify this or it is specific by spcify open, sh ,so this perticular declaration will be match specifically file has only a read file.

If you want write to the my file can we open the my file has Fh->file this one will enable the my file it become an output or essentially we can write into the my file and here the my file will be completely like over writtenby this statement of any previous content will get deleted the third options is esentaily we can do open FH double arrow and then followed by my file thi sis

essentially like mean I mean which indicate the open for output appending to the file which is already existed ,so then we do this &&, I mean double arrow or >> this actually causes the file to be appended at the end by any new cahnges so the file would be open but previous content will be perserved and then any new content.

(Refer Slide Time: 44:54)

The slide is titled "Input and Output Operations". It contains a list of bullet points with handwritten annotations in red ink:

- The print function is used to send output to a filehandle.
 - print OUTHANDLE "data" "more data";
 - Note that there is not comma after the OUTHANDLE
 - This is important, otherwise the value of the handle will be displayed on the output console
- The input operator <> can be used on an input file handle
- The read function reads a number of characters into a given array.
 - The function returns actual number of characters read
 - The function parameters can indicate that characters are to be stored in the array somewhere other than at the beginning
- The seek function can be used to position the filehandle cursor at a different position in the file

Handwritten notes include: "open(my \$fh, >> 'file.txt')" at the top right, "while (\$?) {" and "}" on the right side, "array read" next to the read function, and "next;" at the bottom right.

So now how do we actually write it in to the file essentially for that we need to use the file handle in the statement front so by default actually like I mean she is going into the standard out that when we, we can also specify even file handle and then to write particular paramour, variables into that one section so in this example basically like out handle is something that is already defined in which we are actually adding the data and more data.

So couple of thinks notice there is no comma after the out handle and or there is no special parenthesis or anything all the print function has these parameters essentially like ,I mean file handle followed by the string and essentially like I mean we omit this file handle because standard out is the default for front, now there is also another operator called the input operator which can be used on the input file handle the read function reads the number of characters includes the one arrow essentially the, the function returns actual number of characters.

The function parameters can indicates that characters are to be stored in an array somewhere than, other than at the beginning. So these are all the two difference other functions so the input operator itself it used to read from this file, so you can think of it as basically like and so typical here programs are open and file handle with my file and then we will save while we can save for example this implicitly calls reading this file and then we will reading it line by line and this delimiter is essentially like established stanza already mention it in is \$ / so if you modify then

you can change the default is such as \ so this \$/ is actually \ n and it I read up to that value and then so that is how we write programs essentially. And for specific read essentially like we can actually use read function itself this reads number of characters into an given array the function returns actual number of characters read, function parameters can indicates that characters are to be stored in the array or somewhere other than at the beginning, so you can actually do a man read to get more information. And then the seek function it this is can be used position the file handle cursor at different position in the file. We share all some other advance functions you can typically read this line by line and then if you are not if you do not want process any line you can have to go with next as a simple way to access move passed or move the file handle passed the current position location instead of using C functions, C function is much more upcoming and actually gives it to really fine tune where you want to put cursor essentially. So to just doing a recap essentially today we saw some key topics essentially like I mean number one that we saw was the function calls essentially. (Refer Slide Time: 49:58)

Function Fundamentals

- A function definition consists of a function header and the body
 - The body is a block of code that executes when the function is called
 - The header contains the keyword sub and the name of the function
- A function declaration consists of the keyword sub and the function name
 - A declaration promises a full definition somewhere else
- A function call can be part of an expression. In this case the function must return a value that is used in the expression
- A function call can be a standalone statement. In this case a return value is not required. If there is one, it is discarded

The function essentially like this consists of the function header and the body and then the it is basically like the keyword the stop identify the function the declaration essentially like I mean, here so the , function itself essentially like I mean the can be return anywhere we can declare the function have the active function essentially some errors, the function can is essentially like mean it can be a part of the expression if it is the part of the expression return value that we used inside the expressions. If the function calls makes stand alone then ther is the no return value for even the function returns of the space discarded, and then the function is getting call the body build in the first step

statement the return statement is the an encounter the return statement are the function stop execute one keep thing to notice is basically we see we do not have any explicit return statements the last expressions are evaluate the return as the return value of function.

So use it with caution use always that different values, and then we also talked about the local variable usually the variable declaration will perl actually variables will be carried basically and everything will just have the global scope but my declaration is use d the declare the variable within the function and which are local to the function, and then the my is also like having a so all these variables will have dynamic scope inside the perl. And so it understand th local declaration and then the specify this, and a if the local variable is same name as global name variable global variable itself is not visible in the function one in the local variable in this pattern .
(Refer Slide Time:52:12)

Parameters

- Parameters used in a function call are called actual parameters
- Formal parameters are the names used in the function body to refer to the actual parameters
- In Perl, formal parameters are not named in the function header
- Perl supports both pass-by-value and pass-by-reference
- The array @_ is initialized in a function body to the list of actual parameters
 - An element of this array is a reference to the corresponding parameter: changing an element of the array changes the corresponding actual parameter
- Often, values of @_ are assigned to local variables which corresponds to pass-by-value

Sub args (f1, f2, f3)
Actual

And then the parameter essentially like I mean the parameter for the function, and then , the function call has the parameter that should have the parameter and have those parameters of called actual parameters those are needed are the formal parameters are essentially like used then define the function in later stage and they have a 1 to 1 correspondence with actual parameters.

And in perl the formal parameter are not named in the function header if we do not have the specify them we can directly start use in then itself the functions, and perl suppose both pass by value and the pass by reference in how to initialize the array essentially like I mean so, it is basically like I mean it is a getting assign to various values.
(Refer Slide Time: 53:14)

Parameter Usage Examples

- This code causes the variable \$a to change

```
sub plus10 {  
  $_[0] += 10;  
}  
plus10($a);
```

*\$a = 0 Actual Parameter
\$a = 10*

- The first line of this function copies actual parameters to local variables

```
Sub f {  
  my($x, $y) = @_;  
}
```

And this is something we saw in this example here where can we specify the, the array – is split into the like all this various values using this, other function which is an implicit function. (Refer Slide Time:53:37)

sort Revised

- The sort function can be called with the first parameter being a block which returns a numerical value based on the comparison of two variables \$a and \$b
- This parameter is not followed by a comma
- For example, using

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

will sort the array @num using numerical comparison

- Using

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

will sort in reverse order

Then we talk about the sort a little bit essentially (Refer Slide Time:53:39)

Example

- The example tst_median illustrates a function that finds the median of an array passed as a reference value

We will come back with an example.
(Refer Slide Time:53:45)

Basics of Pattern Matching

- Perl has powerful pattern matching facilities built in
 - These have been imitated in a number of other systems
 - Regular expressions were described in the JavaScript chapter
- The m operator indicates a pattern matching
 - This is used with delimiters like q and qq but the enclosed characters form a pattern
 - If the delimiter is / then the m is not required
- A match is indicated by the =~ operator with a string on the left and a pattern on the right
 - A pattern alone is matched by default to \$_
- The split function can take a pattern as the first argument rather than a character
 - The pattern specifies the pattern of characters used to split the string apart

(\$ =~ m / . pattern /)

Then we talk about the pattern matching we talked about m operator the m operator is the match operator that actually like matches a particular regular expression to the, particular given string then we can actually is if you omit this = ~ operator and implicit with assume that matching on the \$_ the matching gives it is a particular character set is present in the, strings such, one thing we note this actually.

We have an really touch upon how to create the patterns themselves what are the rules of the given pattern which we will talk about in the next lecture but at this time understand that is which came operator.

(Refer Slide Time:55:06)

Remembering Matches

Handwritten notes: \$- = /xyz (abc) pqr (def) /

- Parts of a pattern can be parenthesized
- If the pattern matches a string, the variables \$1, \$2, ... refer to the parts of the string matched by the parenthesized sub-patterns *y(812)*
- If a match is successful on a string, three strings are available to give the context of the match
 - \$& is the part that actually matched the pattern *raw*
 - \$' is the part of the string before the part that matched *abc*
 - \$' is the part of the string after the part that matched *xyz*

Handwritten notes: pqr xyz

and then forward by that actually this is the, & operator but this before we going to the s operator the 2 memorize or to remember the matches essentially known as the matches are successful or not perl provides lot of good syntax.

One is the parts of the pattern can be parenthesis and used this basically, if you have like \$1, \$2 refers to the part of the string that matched, by the parenthesis sub patterns, but how to get to the string itself to the match string with essentially like by using with \$&, \$ back taken or \$over taken so the back take essentially like \$& will have the actual part that matched the pattern so pattern can be specify and basically like this is just string within the string itself the string is split into 3 and then 1 part is basically the matched pattern the back.

Because anything before the most pattern and then the regular normal take is part after the match that is happens.

(Refer Slide Time:56:32)

Substitutions *s/abc/123*

- The *s* operator specifies a substitution
– *s/pattern/new-string/*
- The new-string will replace the part of a string matched by the pattern
- The *=~* operator is used to apply the substitution to a string
– If the operator is not used, *\$_* is operated on by default
- A *g* modifier on the substitution causes all substrings matching the pattern to be replaced, otherwise only the first match is changed
- The *i* modifier cause the pattern match to be case insensitive

Now the *s* operator is used for a substitute essentially like we can substitute one pattern with new string, - this string here, and using the *s* for the substitute pattern here also like I mean the same rules like matching pattern applies with the *=~* for any string and then if we do not specify that *=~* then *\$_* assume the input string and then we have pattern.