

SEER AKADEMI
LINUX PROGRAMMING
AND SCRIPTING-

PERL 7

Hi everyone once again welcome to this lecture we are actually the course is Linux programming and scripting we have been talking about the Perl, Perl programming language in today is Perl lecture 7 we have made a lot of progress respectable and I hope you are now much more familiar with the inner workings of Perl and are confident to do programming in Perl but mainly like I mean we looked at several data structures.

So far and we will continue our discussion still with the data structures in Perl then we will take it take upon the controls while we are doing this itself you are learning somewhat some of the controls so with it really good actually so let us recap as to what we saw in the last class.
(Refer Slide Time: 01:23)

Recap of Lecture 6

- Operations on Array
 - Reverse
 - Sort
 - Chop
- Array and Scalar context
- Avoiding array references for scalars
 - Three methods
 - `$ { }`, Concatenation, `\[`

So in the last class we continue the discussions on array we looked at some operations on array for example the reverse how to reverse the array essentially like the order of the array using the reverse function you also understood about the sort function sort an array and then we also did some chopping use the Chop function to chop the last character of every element in array then we also looked at the , the data from array context as well as the scalar context. And we noted the differences between how they are perceived oh they will be perceived if you are in the array context and scalar context we also studied how to avoid array reference for

scalars and we studied principally like the three methods as to how to define a scalar when we use like the array notation extension, so a \$ variable followed by a [] is always an array that we know about before but then how to distinguish between a \$ and basically a scalar variable With just a [] one method was we can define the variable mean under the [] for the [[]] essentially which is actually shown here.
(Refer Slide Time: 03:02)

Associative Array (Hash Array)

- An *associative array* uses general data, often strings, as indexes
 - The index is referred to as a *key*, the corresponding element as a *value*
- Since a hash table is often used to implement an associative array, these structures are known as *hashes* in Perl
- Elements in a Perl hash do not have a natural ordering
 - When a list of keys is retrieved from a hash there is no definite relationship between the order of the keys and either the values of the keys or the order in which they were entered into the hash

So they saw this notation the other one is the concatenation notation there we can actually use a. to actually concatenate variables essentially so that distinguishes completely like I am in between a scalar variable and [] or the third option is this one there we can escape the [] using a \ I thing like I mean this is pretty much like I mean what we talked about in the last lecture with this I think we are ready to move on what I want to really touch upon today is the associative array or the hash array or the hash array.

Is also known as hash array is and we will learn about that today then we will also see like I mean some of the common challenges the two main things in the Perl language if you do not know anything that you should really know about is one of them is the hash array how to you hash array and then the second one is the regular expression so we will see the regular expression in more detail in the coming classes.
I always promise you that so we are not ,we are not it compete with the Perl programming language so we will be looking at that but today we will talk about the associative array.

And the various things associated with that the functions and also think how to use them and we will also look at some simple application essentially does in how we can do it and do it and I will also give you some problems to work on to understand the concepts of the hash array so without delaying further let us look at what it is essentially so hash array a couple of things one is it does not have a numerical index it uses.

The general data often strings as indexes are indexes the index itself is referred to as key in a hash array and then the corresponding element is its value so these two terms are very important because we will be looking at this from these two terms so I want you to remember this a hash array or an associative array is always referred to the indexes are referred to as key and then the corresponding element in the array is referred to as value.

So the hash table basically like I mean so we use the hash table to implement the associative array so hence these structures are known as for hashes in Perl hashes is not probably like new to you lay essentially hashes something which is unique in the sense that it has a very random access capabilities built in and essentially recommend that is what we are striving to achieve then use the then using most of the way.

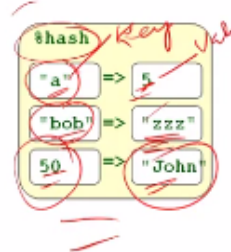
So you do not have to do a linear search to get to a particular element instead you can directly go into that particular image using the hash function and so the other just resulting from that is that the elements in the Perl hash do not have any natural ordering so there is no relationship between each of the keys or each of the indexes in the array in the associative array where as in a regular array as you know that the numerical values are incremented one at a time.

Starting from 0 things like that basically like there we do not implicitly with specify so the index itself so here there is no definite relationship between the order of the keys and either the values of the keys or the order in which they are entered into the hash so we just enter values and they can be attached anywhere within the array I will show you some examples essentially as how we had how we delete objects from the hash array.

(Refer Slide Time: 08:01)

Hash Variables

- Hash variables are named beginning with the character %
- If an array is assigned to a hash, the even index elements become keys and the odd index elements are the corresponding values
 - Assigning an odd length array to a hash causes an error
- Curly braces are used to 'subscript' a hash
 - If %h is a hash, then the element corresponding to 'four' is referenced as \$h{'four'}



Let us look at more things essentially a couple of more syntax for issues one is the , the hash variables of hash array the names begin with this character % if an array is assigned to hash the even index elements becomes the keys and then the order in index elements are the corresponding values so assigning an odd link array to hash will cause an error so these are some of the basic things that you may want to look at meaning like I mean this is like I mean by mistake like I mean you are instead of & you.

Do not use that and insert you % and this is what a kind and then the {} are used to subscript the hash so if %h is an hash then the element correspond to four you can do it as \$ h and then the four essentially is insane so let us just an example here so here this % hash essentially all three elements so this is what is this is key and this is the value , so again another key is bob and that value Z and then the % in the hash as a t of 50 and the value is jump.

So look at that I right I mean so he can be just a character string or a numerical value same thing with the values essentially they can be just a number or string or any other data structures so ,so like I mean so this is this is the key thing basically we will see like I mean how we use this and how we insert it to this one.

(Refer Slide Time: 10:17)

Hash – an associative array

An *associative array* (or simply – a *hash*) is an unordered set of pairs of keys and values. Each key is associated with a value.

A hash variable name always start with a %.

my %hash;

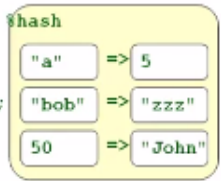
Initialization:

```
%hash = ("a"=>5, "bob"=>"zzz", 50=>"John");
```

Accessing:
you can access a value by its key:

```
print $hash{50};
```

Tip you can reset the hash (to an empty one) by `%hash = {};`



So let us look at so in a nutshell essentially again it is an unordered list unordered set of as of keys and values and each key needs to be associated with a value otherwise it is an arrow so we mentioned that basically it always starts with the % so here I am declaring my %hash this is again a declaration essentially like I mean this my term in fact Perl does not impose any kind of fiction you can use variables as you please you do not have to declare.

But it is a it is a good idea and a good practice and a good style will actually declare your variables up front using the my notation again unlike other programming languages you do not have to specify whether it is a strain whether it is a character those kind of variables you can just simply say like my then particular variable name here the initialization essentially like one way to initialize so basically assign the key a-5 the value 5 and then t-bob to value ZZZ and then 50 to value john.

So, so far it is fairly easy declaring the variable and then this assignment now how do we access it so we can do like a dimension like in the {} here so within the {} we specify the T so in order to access from that particular value so it is a print \$ hash 50 that gives you jump , and if you want to reset this hash table essentially like I mean you just remove all these things for all though key value pairs and simply do this and then that will reset So now let us look at how do we modify what happens when we saying hash bob is AAA so now what happens to this one the value simply changed so Z will been changes to a.

(Refer Slide Time: 13:01)

Hash – an associative array

modifying:
`$hash{bob} = "aaa";` (modifying an existing value)

adding:
`$hash{555} = "z";` (adding a new key-value pair)

You can ask whether a certain key exists in a hash:
`if (exists $hash{50}) ...`

You can delete a certain key-value pair in a hash:
`delete($hash{50});`

\$hash

"a"	=>	5
"bob"	=>	"aaa"
50	=>	"John"
555	=>	"z"

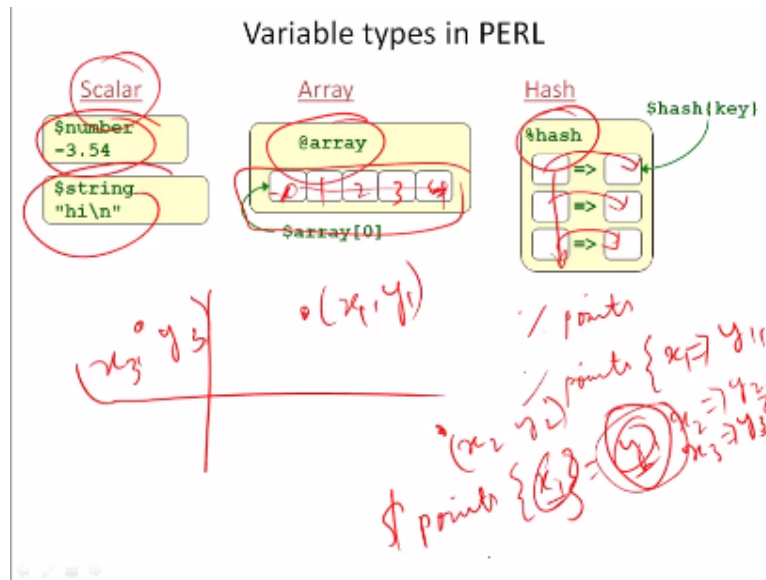
Now if you want to add a new key value pair you just simply specify hash and then that particular key with what the value is and then this gets added so basically now the hash difference the number of elements become 4.

And then the, the law the value that you mentioned is getting inserted please note that actually there is no order here I am inserting at the end it can be anywhere in the middle or wherever it is basically just gets added to this a hash array that is how we should look array because there is no order that is associated with it so you cannot say like hash end and then you get this one essentially like I mean you need to access it using the key and then you get the value.

So again there is no order in this segment, the way that certain key whether it exists use this function Perl exists so we can say like I mean this actually like a is actually doing two things one is exists this essentially like I mean it is actually you know finding whether there is a value that exists corresponding to this key essentially which is kind of testing whether there is a key that is existing in that.

Now we can also do a delete essentially like I mean that is simply because you really function and then we specify the particular key value pair the key value pair is denoted as just the array followed by the, the keys essentially let go the key and the value both get deleted in this case so for example here we pay the 50 which is associated with free John so when say we do this both of them get deleted So it is not just it is so moving the 50 yards removing this with john if so moving both the key and the value from that array.

(Refer Slide Time: 15:25)



So now so far we have learned the 3 different data types essentially the variable types is essentially the one is the scalar, scalar could be a number or a string the array data type which is essentially like a string of objects essentially like so the array has an value of 0 which is the first one and then the last one is what over the entering there as a for a hash essentially hash has just key value pair so notice that actually like a here.

Element for all of time together because this starts from 0 1 2 3 & 4 etc whereas here there is no or any order here essentially only connection is this element is connected do this and that is all so you can think of hash array as a 2-d two dimensional arrays which is how we can make use of it in real world applications and then also like I mean it is you can also call it as a sparse array so this is again a useful say for example like I mean you have a coordinate system.

Where point is denoted by X Y and then you have given like them inside with three points to identify these points you can actually build a hash array where you can say like I mean % points and then you can say like %points and then the key can be this X and that corresponds to I wanted to say that this is x 1 y 1 this is X 2 y 2 and this is X 3 y3 I can just simply note that 1, 2 y1 x2 y2 x3 y3.

Now if I access like \$points , x1 this will get me the y1 so now you can see that actually like I can preserve both x1 and y1 and the points that are relevant for me I can store it within them so this is one way to actually store like a two dimensional array keys with the relevance and then now I can do any kind of operations between them with x1 I know y1 we can get 2x1 from y1 there is an interesting problem that I will post later on.

There I want you to get x1 given what is the y1 so all that I will give you like more functions essentially so let us look at some more functions to work with hash arrays.
(Refer Slide Time: 18:57)

Hash – an associative array

An *associative array* of the phone book suggested in the first slide
(we will see a more elaborated version later on):

Declare

```
my %phoneBook;
```

Updating

```
$phoneBook{"Dudi"} = 9245;
$phoneBook{"Dudu"} = 7693;
```

Fetching

```
print $phoneBook{"Dudi"};
```

\$hash

"Dudi"	=>	9245
"Dudu"	=>	7693

So here another practical example is so essentially a phone book we can associate various users with their phone numbers and put them in a hash array and then when we say like phone book for a given user it prints their numbers.
(Refer Slide Time: 19:16)

Changing a Hash

- Values can be assigned to a hash reference to insert a new key/value relation or to change the value related to a key
- A key/value relation can be removed from a hash with the delete operator
- The undef operator will delete all the contents of a hash
- The exists operator checks if a key is related to any value in a hash
 - Just check `$h{'something'}` doesn't work since the related value may be the empty string or 0, both of which count as boolean false
- A hash variable embedded in a string is not interpolated
 - However, a reference to a hash element is interpolated

%hash = ();
 undef %hash;
 if (exists \$hash{something})
 true

This is pretty much what we saw basically like how we change the hash array the values can be assigned to hash reference to insert a new key value relation or change the value related to the

key these two we saw already for an examples a key value relation can be removed did the delete operator this is something that we saw already and then this is another way to delete all the contents of hash array.

We know that one ,one other method is that % hash equal to this will get rid of everything the other operator is basically like under % hash and this also will just remove all the contents from the hash and then the exists operator this is something that we saw basically like a move if this any key is related to any value in the hash and then if you just do this basically like a I mean a stick the \$a something.

It does not work since the related value can be empty string or 0 both of which is containable for Boolean false box so we do not want to just feel like, like if this kind of a thing \$ edged, if you specified this kind of a check essentially if something is actually has a 0 it will still termed as a Boolean false back here such as it should be Boolean false.

So this is what this means this essentially so do not do this instead use the exists function to check whether it really exists or not exist on the exists function if it is a 0 it still returns it true value and only like it is a false only such object does not exist in the hash array and then a hash variable embedded in a string it is not interpolated this is another key concept however that a reference to the hash element is related.

So if you are embedding some variable inside a string you cannot interpolate our hash variable that is embedded in string cannot is interpolated.
(Refer Slide Time: 22:10)

Iterating Through a Hash

- The keys operator returns a list of the keys in a hash
- The sort operator can also be applied to iterate through the keys in order

So just remember that mean you are missing so now we want to iterate through the hash actually there are two function again in order to do this the first one is the keys operator essentially so the keys operator will iterate through all the keys the other one we will learn a little bit later then also the sort operator is a other one which can also be applied to iterate through the keys in order so let us see like how we can use it
(Refer Slide Time:22:47)

Iterating over hash elements

It is possible to get a list of all the *keys* in %hash

```
my @hashKeys = keys(%hash);
```

Similarly you can get an array of the *values* in %hash

```
my @hashVals = values(%hash);
```

The diagram shows a hash `%hash` with three entries: `"a" => 5`, `"bob" => "222"`, and `"Joan" => "Joan"`. Below it, an array `@hashKeys` is shown with the first two elements: `"a"` and `"bob"`. A large green box obscures the rest of the diagram. Red handwritten notes and arrows are present: a red arrow points from the `keys(%hash)` code to the `@hashKeys` array, and another red arrow points from the `values(%hash)` code to the `@hashVals` array. There are also red handwritten notes like `hashKeys` and `hashVals` near the arrays.

So in order to get the list of all the keys in the hash we can simply put `keys % hash` and this returns all the keys and we disposed in Boolean value so the array is essentially like hash key this a bob 50 so this is hash key is 0 this is 1 this is 2 so you can actually like in fact print out. These values and then using this so this is the to get that the keys so the other one that I mentioned is using the values or something again the values actually captures all this side of the division so these are all going into here values around going into you so what does values are going to hear what is value look like , so again it is at the same thing so as well as , 0 this 1 and this is 2, so here is a phone book example we started at three so every we basically have we are declaring this hash array or phone numbers initially we are initializing to a null.
(Refer Slide Time: 24:50)

Example – phoneBook.pl #1

```
#####  
# Purpose: Store names and phone numbers in a hash,  
# and allow the user to ask for the number of a certain name.  
# Input: Enter name-number pairs, enter "END" as a name to stop,  
# then enter a name to get his/her number  
#  
use strict;  
  
my $phoneNumbers = ();  
my $number;
```

And then we declare a variable called number.
(Refer Slide Time: 24:53)

Example – phoneBook.pl #2

```
# Ask user for names and numbers and store in a hash  
my $name = "";  
while ($name ne "END") {  
    print "Enter a name that will be added to the phone book:\n";  
    $name = <STDIN>;  
    chomp $name;  
    if ($name eq "END") {  
        last;  
    }  
    print "Enter a phone number: \n";  
    $number = <STDIN>;  
    chomp $number;  
    $phoneNumbers{$name} = $number;  
}  
}
```

Handwritten notes:
- A red circle around the opening brace of the while loop.
- A red circle around the closing brace of the while loop.
- A red circle around the line `$name = <STDIN>;`.
- A red circle around the line `$number = <STDIN>;`.
- A red arrow pointing from the word "value" to the line `$number = <STDIN>;`.
- A red arrow pointing from the word "key" to the line `$phoneNumbers{$name} = $number;`.

And here they are also like I mean have a name essentially and then we basically query the name and the number so we start this while loop here starts here and ends here and then basically if it is not equal to end in this demonic for ending the program or ending this loop so if it is not equal to end then what we say is basically like enter a name that will be added to phonebook we ask this basically and then we capture the standard in whatever be input into the name. We Chomp the last character which is the new line essentially and then we compare the name against end if they are equal then we skip the whole thing and then go beginning of the loop so it

tests this condition and since it is an end it will end the loop itself but if it is not then we will go forward then you ask to the phone number and then capture the number in this number variable which we declared here.

Again we Chomp the number and then we basically just add that into the hash array that they declare the phone numbers with the key as the name and number as value.
(Refer Slide Time: 26:46)

Example – phoneBook.pl #3

```
# Ask for a name and print the corresponding number
$name = ""
while ($name ne "END") {
    print "Enter a name to search for in the phone book:\n";
    $name = <STDIN>;
    chomp $name;
    if (exists($phoneNumbers{$name})) {
        print "The phone number of $name is: $phoneNumbers{$name}\n";
    }
    elsif ($name eq "END") {
        last;
    }
    else {
        print "Name not found in the book\n";
    }
}
```

Handwritten notes:

- key* (pointing to `$name` in the `exists` function)
- value* (pointing to `$phoneNumbers{$name}` in the `exists` function)
- key* (pointing to `$name` in the `print` statement)
- value* (pointing to `$phoneNumbers{$name}` in the `print` statement)
- Do you want to add?* (pointing to the `else` block)
- Yes* (pointing to the `else` block)
- Updating the book* (pointing to the `else` block)
- \$phoneNumbers* (pointing to the `$phoneNumbers` variable)
- \$name* (pointing to the `$name` variable)

So this is the one that we asked for a name and then the print the number corresponding to that name so this, this program is after we built this hash array in this slide so once we build the hash array now what we do so here we again initialize the name again we ask whether the name is end if it is not end then we go into the program and we basically like us to into the name the name we just get it from standard.

And we can do the job operation to remove the last character is into the new line , so no once we have the name then we look for whether that is this in the in our phone numbers hash array if it exists then we just print out this one so look here basically this is the key and here we use the same key to check what the value is so this whole thing this prints out the value , and then again if the name is we typed that end at this point it just goes back to the beginning.

But then if it is not end and if the name is not found with this exists return for false value or a 0 then it goes all the way to this else condition and then we just print out saying that the name is more pump here you can do some more additional tricks and basically me and also say that a it is not found do you want to add , and then if the, if the user says yes then you again go into this updating , the hash array.

And I think like you can easily update hash array by using this the numbers , \$name not a name people to \$ number , okay so if you add this essentially like I mean to make this program we program there if you give a name it looks into its dictionary see whether it is easier that ,that name has a matching number then it put prints of the number if it does not have the match then it prompts you for entering a number so that it can add for that particular person or what the name what the number is corresponding numbers okay.
(Refer Slide Time: 30:13)

A Predefined Hash *set var = value*

- The %ENV variable is defined to be the key/value pairs defined in the environment of the running Perl process
- Many of these are inherited from the run-time environment
- In Microsoft Windows, environment variables can be set through the command-line set command
- In Unix Bourne shell, environment variables may be set by a simple assignment *should primary value*
key

There are some additional concepts in the hash essentially like I mean one of them is a predefined hashes so the %ENV variable is defined to be a key value pair defined in the environment of the running Perl process these \$ this % again ENV variables are inherited from the runtime in oven so again in the windows con context actually like we can actually set up through the command line set command essentially.

Like so when we do like set then we can just say , so we set then variable name equal to value = value in so this is the essentially like I mean so the command line set arrange or even like in ,in the Unix bond shell we can do the simple assignment similar to this we can also do like set A & B and then set and we and then the variable set value ,so the simple assignment Will actually accept these things and then once it is done the %ENV hash these form key value Association already so then it uses that to determine how the operating system to behave so now let me give you like a small quiz essentially.
(Refer Slide Time: 32:29)

Quiz

- Create a hash array with the following data:

AAB	900
Bcd	5000
Vgv	8000
Xyz	50

- Sort the array based on the keys
- Sort the array based on values
- Print both the results

Try to define an ash array with the following data here are actually it is very simple essentially so it is basically like all of them are just strings and this side is all you know it could be like I mean mix and match also and now you sort the array based on the keys try to sort the array based on values and then print both the results and see what ,what we get we will discuss this in the next lecture as to what you should be seen.
(Refer Slide Time: 33:10)

References

- A reference is a scalar value giving the address of another value in memory
- A reference to an existing variable is created by using the backslash operator
- References to literal structures can be created
 - A reference to a list is created by enclosing a list in square brackets, [...]
 - A reference to a hash is created by enclosing a list in curly braces {...}
 - For example `$a = [1, 2, 3, 4]`
 - For example `$h = {'i' => 1, 'v' => 5, 'x' => 10};`
 - Notice the assignment is to a scalar variable since the literal value is a reference

Handwritten notes:
- Above "References": address
- Next to "backslash operator": @
- Next to "scalar variable": \$
- Next to "literal value": literal
- Example: `@a = $h` (where `$h` is a hash reference)

So now we come to the ,the next set of topics essentially one is like I mean how do we reference , the I mean what is the concept of the scalable what is how is it represented in

demanding as you know like I mean every variable is also as a reference to that variable in C language we call it as an address and then essentially like I mean every variable you can think of it as a variable is a ,is a memory location, location in the main memory.

To which a value will be store and then there is an address to it we which will actually is the reference to that variable so if even in Perl we can actually give open this form to the variables essentially, essentially the reference is a scalar value given to be given the giving the address of another value in the memory so whatever the address of this particular location will be the reference and this is a scalar value.

The reference to existing variable is created by using the \ operator so that is this one so in the string context you go back \ often to escape the special characters but in variable context then use the \ we actually create a reference to that particular variable or calling its address of the, and then reference to listed structures then can be created so if you want to create reference to a list we can create by enclosing the list in a [].

So this is what we are doing it basically by when we write a array so here at \$ array a dollar a 1 2 3 4 means when we refer then we use the dollar a that refers to this whole which is 1234 and then we use the {} essentially like having to denote the hash array again we saw that this is how we create the hash array essentially what they saying is I can make this is the hash and we create a reference for it by enclosing within the {} again.

Lot of these concepts essentially all revolves around since you recommend there is one way of understanding and but this is the next day to understand the same thing slightly different So the assignment is to a scalar variable since the literal value is reference so here we do not use any of those % H notation or at a these are all now scalar values so are these the same like I mean are at a = \$ A and %h =\$h or anything.

The answer is not because now these are scalar variables so what do they actually contain this is something that you can you can actually like do this experiment do these assignments and print \$a and\$ H and let us talk about that in the next class as to next lecture so what you will see if you do this simple print but it will be interesting to see what we get.

(Refer Slide Time: 37:53)

Dereferencing References

- To access the value pointed to by a reference, the programmer must explicitly *dereference* the reference
- An extra \$ sign can be used
 - If \$a = 5 and \$b = \ \$a then \$\$b is 5
 - \$\$b = 7 changes the value of \$a to 7
- In a reference to an array, -> can be used between the reference and the index to indicate a dereference
 - If \$r = \@list then \$\$r[3] is the element at index 3 of @list
 - \$r->[3] is also the element at index 3 of @list
 - \$r[3] is the element at index 3 of @ completely unrelated

So for a scalar value itself of scalar variable we know that actually like we can refer ,refer to that one by using the \ what I mean is here this is scalar like a \$ x then you can say that basically with \$ ref =\ \$h s now what is \$ ref to \$a and \$h are if we have these as our variables how do we get back to these arrays and hashes and the scalar variables back so this process basically by which we can convert a reference back to the ,the variables.

The original variable that is known as the dereferencing okay so to access the value pointed by a reference the programmer must explicitly be the dereference the reference so if you are given only the reference we can be reference by using 2 \$ signs so a \$\$ D in this case is same as \$8 because \$d is actually \ \$ it makes sense but this is the way to understand that so here, to in order to reference an array the arrow sign can be used.

Between the reference and the index to indicate the dereference so we can one thing that we can do is like I mean when the \$r is at list then \$\$r are you bring back the list and then if you say like number three then that is the index three of the list we can also simply use a \$r And then aero 3 in {} and this is the same this but if you just say like \$R 3 this is not the same any of this, this or even this reason is \$r [3].

Is actually the 3 element or actually in this case the 4 element of another array named at R and that is completely related from this one which is actually be at list so see the difference actually here at our and here it is at list so since these two are different essentially like I mean this is not the same so do not use this for this if you want to use the, the lists 3 element with the reference then use this or this.

And if you want to refer the third element of the 4 element index 3 or 4 then 3 of array R then use this, so I think this is some of the powerful concepts in the Perl language input Perl language. Is not a you want to work more with in order to understand this the concepts are fairly simple but they are very powerful and pretty much 90% of the program's themselves use these concepts extensively and I just wanted to make sure that you can get all these things. Number one thing is we do this for passionately as a new concept of an array the key difference between a hash array and the regular array is so passionate an unordered set of key value pairs so this key value pairs can have any association in some relationship or it can completely be different and Perl treats either case as the same basically like I mean as a hash though there is no association between any of the keys or in any of this values. So the only association that it knows about is in a particular key and a particular value all the others are completely all the other that certain variable and then we denote the hash variable with the % basically in the front you can initialize a hash array using just a simple notation you can also like to make it empty by using UNDEF or just the , the parentheses without any evaluation array.

We can access by the key essentially so here we use a scalar hash with a particular key value in the {} in order to access that so this is different from other way like the other things that we will learn we learn later on which I will also recap on that front in order to add an element It is very initially like me let us do a modification is basically this modify the you just put a new value for a given key and then that just modifies for value.

So essentially it just replaces the particular value within you and then if you want to actually add it is again very simple basically like add new key and then with the value and then the values that particular p value, and we always use the exists function to find if some particular key value pair exists in the hash array We cannot just do if that particular key value pair or hash of the key to see whether that one particularly.

Because if the elements value is 0 that particular logical test will return a false value so the exists function is the surest way to ensure that the particular key value, shell array then we can delete a particular key value pair by using the delete function and to use the delete function we have to specify hash and the particular key.

And please also note that we need to specify the balances here and I think for exists we do not need to that and then we also saw this difference between the scalar array and the hash essentially like a scalar put the number or string the string is enclosed within the, the "" or a ""

either one is legal syntax and you also know that actually move"" only some \ and only one other specific character that will be getting a manual.

Small values there as known there as in a string that is enclosed within "" even it can interpolate variable speed index string and then you can go from the values so, so that those things we already saw this form the differences and how we can do all these things then the other data structure that we learned was for array the array data structure has starts with the @ sign or aspen sign and then the 1st element is always form or the 1st element is always arrays 0.

So it is the index of 0, 0 and then the index this process will be before but I think hash actually there is nothing just a hash with a key is what is denoted name in that we saw some examples this is the phone book we also saw the example how to denote points on a plane essentially with x1 y1 how do we remove also isolation how to operate on them this is like critical this even you actually do like applications like the basement or routing.

And we want to denote from one point to another point and this using Perl to write certain routines this is very useful then we also see how to access the and the hash array access data elements so if you wanted this access although these we just use the keys function you can also do like a sort on the Perl essentially we are going so sort function all so like useful then you also saw like, the values function essentially like values.

Function gives all the values of the array here essentially living substance then we actually saw a like I mean how the program is being written so the program has like two parts one is where we populate the hash function and then the other one is where we get the name and display what the phone number is one way is essentially we can combine these 2 these 2 sections into one there it asks for name and the name exists it returns the value the name does not exist.

It asked for the phone number and then progressively can go build entire array and that is what is shown here and actually like I mean this could be a good exercise for trying to create a phone book there it not only just gives you the number but if it is not present in it is database it adds that and then, finally gives you all the increase of the phone book and then we also studied about the predefined hashes how to set them then the two main things that.

We talked about was referencing of scalar and also a dereferencing scalar the referencing a scalar value is to reference it is address in the memory so the way to do it is by using \ operator for the lists we simply use for the [] and then for a hash we simply use the {} to create a reference and all these references we wanted were created as scalars and not hash those arrays form or the hashes themselves so these are all like no knows that.

So only like post another \$ is allowed and the other one is now that would not be gotten the references to be this variables are we ready reference them or get back the original array or original lists or original hash functions how do we get back to that and for that we can use the extra \$sign essentially it makes our \$\$ be and then for an array actually.

We use the dash and then the > time which actually is an arrow sign basically this arrow sign is made up of hash > sign and then we saw basically like I mean with we have a list and then if the \$ are is assigned to that list essentially it is the reference to that list \$\$R3 is one way to access the 4th element of the array and the other way is the \$r and then the arrow to the element 3 to access the element but if we do you just R3 that is incorrect.

Because that is not the same as the previously and \$R3 is typically the 4th of the array and , so that pretty much concludes this lecture we will take it up in the next class from this point and meanwhile please try to do to the small exercise that I have sent to you and also think about how do we access based on just the keys and how do we access the access form values and how do we reverse the keys and values , so I guess thank you very much for attending this lecture we will see you in the next one thanks bye.