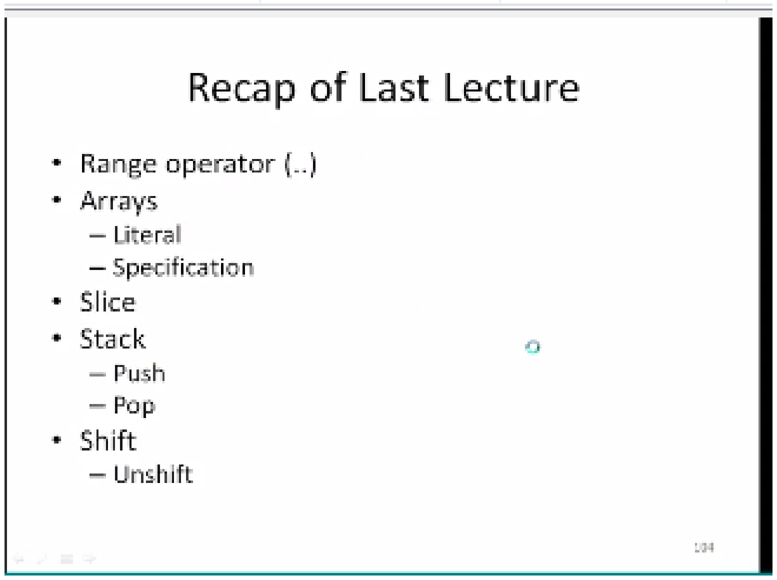
SEER AKADEMI
LINUX PROGRAMMIMG
AND SCRIPTING – PERL 5

Hi everyone again welcome to this lecture this is again the I want to remind you about the class that we are talking about are Linux programming and scripting. Today we are going to continue our lecture on goal we covered a lot of things about Perl. So far we started with just doing some really simple programming Perl.

Then we went into more kind of like we started learning about the data structure he lent a couple of them one is the main data structure that that is pretty much the workhorse of Perl is the scalars we learned about scalars and then we went into the arrays actually in between we also learnt about strings we touched upon it I think like. I mean there is more two strings that we will be seeing in the future classes in couple of future classes.

But, now we will be we are talking about arrays and I want to continue our discussion on more a and today I will be finishing up with arrays. So that you have a good understanding of how to represent our ratings how to use arrays in your programs and how to manipulate various data using arrays. So, all these things are the key things be learning around objectives from this session so before I start with the today's lecture I want to give you a brief recap of the what we learned in the last lecture so let us begin with that discussion you.

(Refer Slide Time: 02:50)



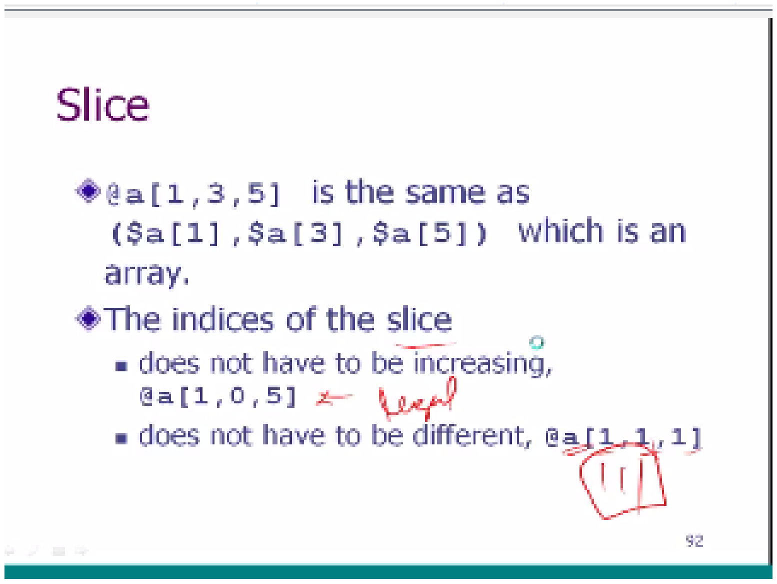
Recap of Last Lecture

- Range operator (..)
- Arrays
  - Literal
  - Specification
- Slice
- Stack
  - Push
  - Pop
- Shift
  - Unshift

104

So the last feature we first stopped started talking about the range operator, I hope you remember the range operator which is the two dots which represents essentially a range of numbers and it has two contexts. That we talked about one was irate context and the other one with a la Congo

and what is the difference between those two that also p1 was good basically in the scalar context it also has a Boolean nature.
So that it actually returns a true or false value so that is one thing that. We talked about then we went into a racing more detail form we understood how to specify an array and how the array literal works basically.
What are the rules governing it in terms of putting a $ in front and then it needs to be in a square bracket that you need to specify and then you are a itself with the elements of super fighting the regular brackets basically in the moment. We can specify each and every element inside that array and then work on that and then they are addressed using this form the index indices are index. This is subscribing in square bracket then we also, talked about the slice I just wanted to remind you about slice let us just quickly look into the slice operator.
(Refer Slide Time: 04:19)

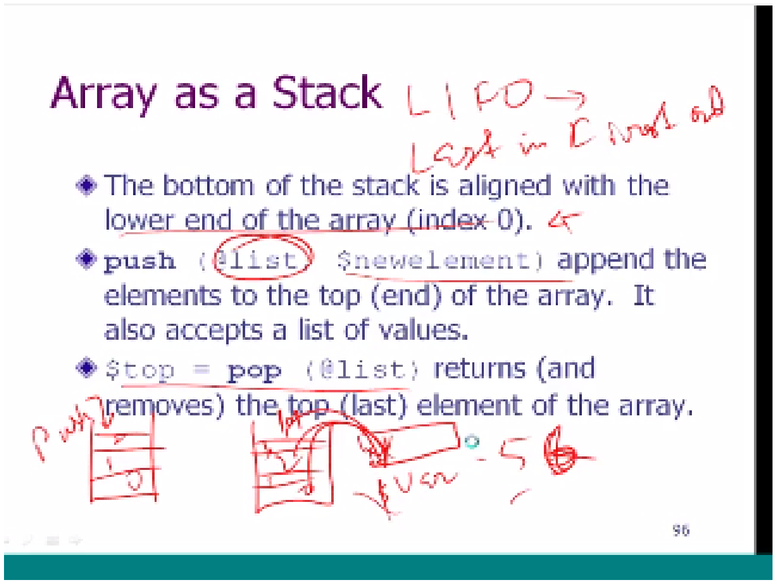


Slice

- `@a[1,3,5]` is the same as `($a[1],$a[3],$a[5])` which is an array.
- The indices of the slice
  - does not have to be increasing, `@a[1,0,5]` ← legal
  - does not have to be different, `@a[1,1,1]`

92

So the slice is essentially we it is a shorthand representation of multiple elements of an array. So here this is something that. We talked about, basically the add sign or the ampersand sign with the name and then. We can specify multiple elements form with comma separated once and it is initially like, I mean that specified with all the ways elements of that array and we also noted some of the rules governing the slice basically, follow the indices may not be increasing and in this is need not be different basically the same indices also constitutes the slice. Then we looked at the stack essentially.
(Refer Slide Time: 05:19)



Array as a Stack

- The bottom of the stack is aligned with the lower end of the array (index 0).
- `push (@list, $newelement)` append the elements to the top (end) of the array. It also accepts a list of values.
- `$top = pop (@list)` returns (and removes) the top (last) element of the array.

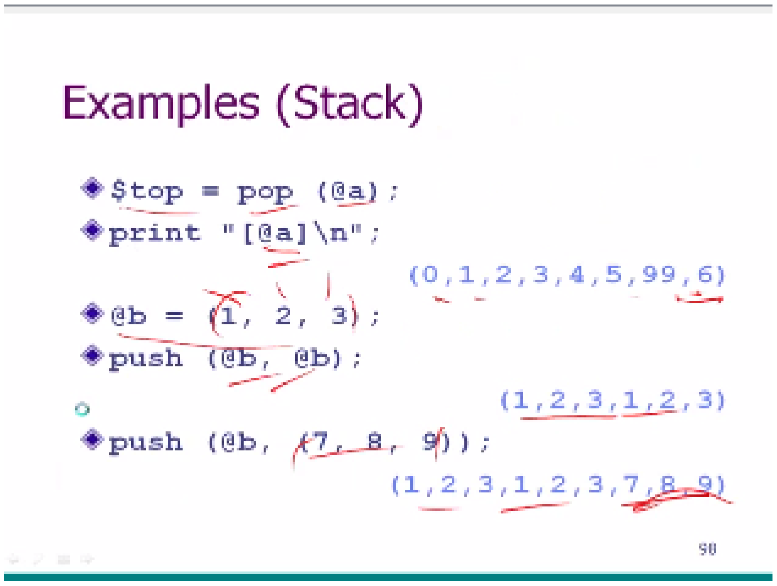
LIFO → Last in First out
push

Stack as I mentioned basically the set of memory basically which are which is stackable you can say like meaning it has a property of last in first out. So that is what this LIFO means basically was not Buster which means that essentially the elements that are added first comes out the last and the last one that is added into the stack, comes out the first and the operation for putting an element in the stack and then removing the element of stack that is a two things. One is pushing the putting the element inside the stack is known as push operator and then when you are removing the element from book stack.

We use a pop operator to remove and here, you see one key thing key difference between this and then wiggler arrays you do not specify. Any kind of in sticks so you just say push at list and then that is assigned to a new element and then. So, basically knows that whatever is last that you added that element is actually like an even. So, the new element is added as the top of the essential and it automatically increments the index but the index is unknown to the user so the user only sees the only one element of stack and then here, is the pop usage of the pop the one basically.

Where we are assigning the pop at list to this variable $ top and again what this does is you do not need to worry about okay. Whether it is in the hundred in there for 150 whatever it is but the top of the element of the last added element is this form pop and then it is put inside this new variable doctor so this is something that we learned.

(Refer Slide Time: 07:29)

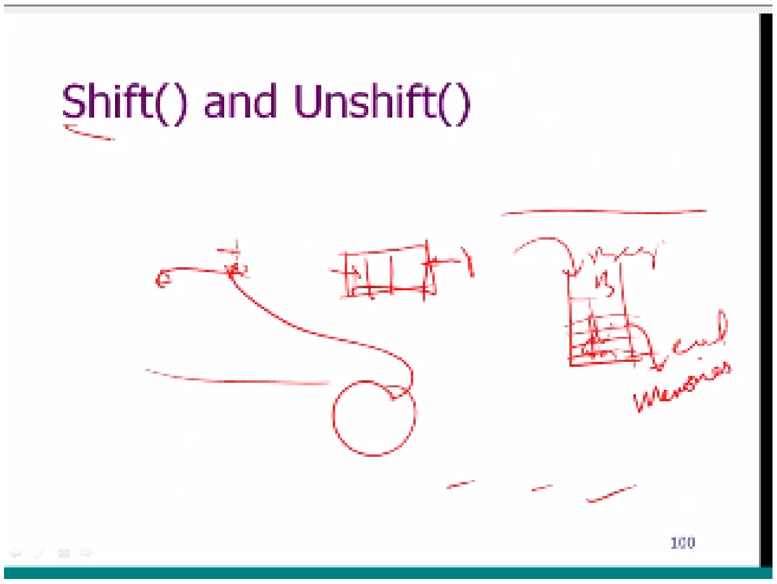


Examples (Stack)

```
$top = pop (@a);
print "[@a]\n";
                        (0,1,2,3,4,5,99,6)
@b = (1, 2, 3);
push (@b, @b);
                            (1,2,3,1,2,3)
push (@b, (7, 8, 9));
                        (1,2,3,1,2,3,7,8,9)
```

98

And then we saw some examples of this form and more and more things basically that being.
(Refer Slide Time: 07:44)

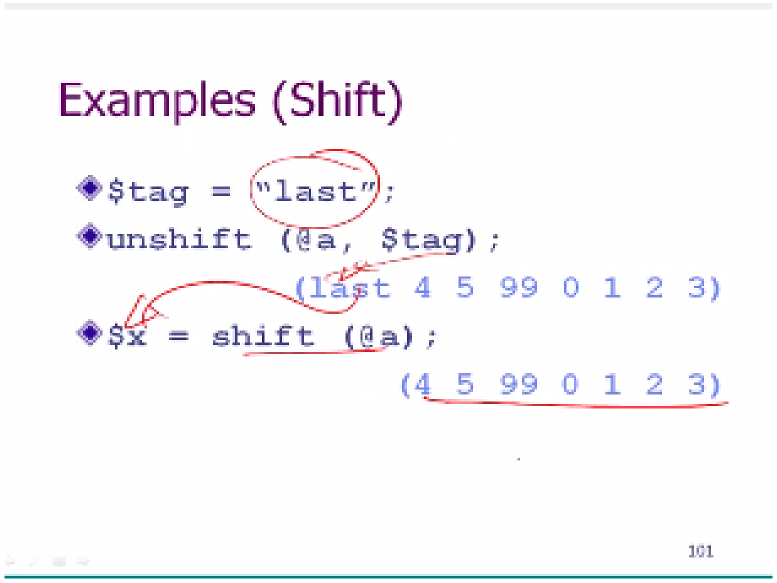


Shift() and Unshift()

100

So then we looked at this shift and un shift again these are some of the key elements essentially which is kind of this one is like the opposite of push and pop very similar. But, it is kind of opposite in the sense that the shift is essentially like to the added the element is added to the lower end of the array instead of the top of the array. So, the last element is in is incremented last index is incremented and then it is added to that. So here, shift means basically like the last one that is worth added that gets popped out. Basically, and then it goes into the array and then the array is kept as is essentially.

And going to the array shifted essentially again it puts back that at the top. So these are some of the useful commands essentially be legend for, when we say like on shift for five actually like them you can see them the 99 is pushed and then the 4 and 5 are added at the end.
(Refer Slide Time: 08:53)



Examples (Shift)

- `$tag = "last";`
- `unshift (@a, $tag);`

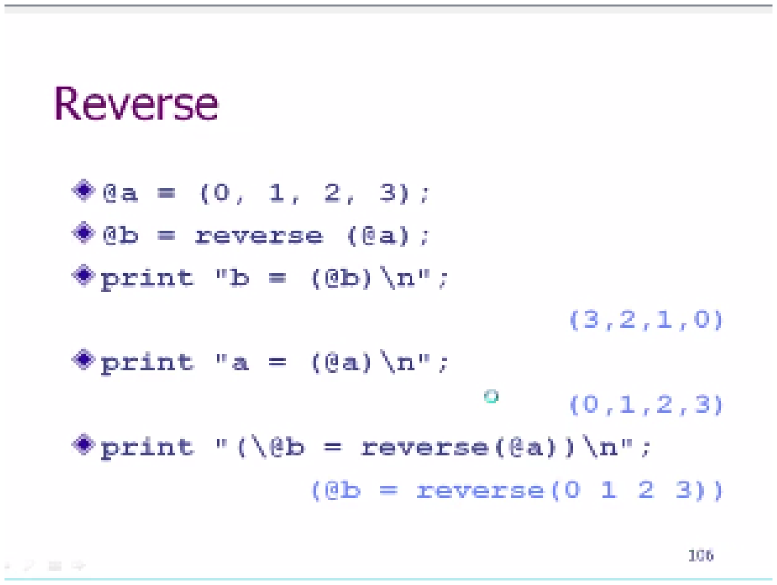
```
(last 4 5 99 0 1 2 3)
```

- `$x = shift (@a);`

```
(4 5 99 0 1 2 3)
```

101

And then we also saw some examples of the shift command here, is some other some more essentially. We are putting the thing inside that and then. We also like shifting there the taller X will get lost as the value and then essentially the remaining array will be intact. So, that is the recap from the last lecture now we are going into the today's lecture. We will continue with the arrays I want introduce a couple of new things to you guys basically. So let us see the first one is the reverse operator.
(Refer Slide Time: 09:45)



Reverse

```
◆@a = (0, 1, 2, 3);
◆@b = reverse (@a);
◆print "b = (@b)\n";
                                (3,2,1,0)
◆print "a = (@a)\n";
                                (0,1,2,3)
◆print "(\@b = reverse(@a))\n";
                  (@b = reverse(0 1 2 3))
```

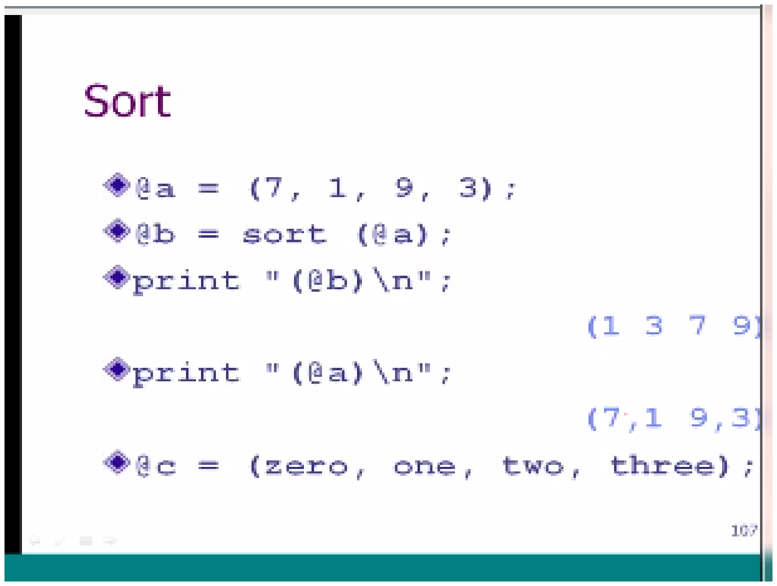
106

Reverse is essentially used to reverse a complete an array, with all the indexes or indices for completely worst. So in this array for example so the taller a is 0, 1, 2, 3. So, this one you can say like it is the $ a 0 element this is $ A one accessory and this one is like $ A three, so now we say the new array at B is reverse of at a one thing you notice is essentially. When we declare a new array. We do not have to declare it up then we can directly make this assignment in the modern Perl actually use for mine or for a moniker for specifying available if it is a good practice to actually.

Even, if you are directly using it to specify the array with them my in front. So this is becomes like move and then at be equal to. So this is a good practice whenever, you are using the add V for the first time use the mind. Then Paul knows that actually needs to allocate the memory for this variable. So now let us look at this essentially though when they reverse so we are assigning this as a reverse so now for B 0 that value is 3 and B 1 is 2 and B 3 is actually 0 all have a $ in front in this example we are printing the a corresponding a from 0 through 3 you can see that it is actually length 0 to 3 the values themselves whereas for B the B 0 the response to 3 and then B 3 corresponds to 0.

So, we basically like I mean they to print out this is essentially like I mean specify like I hope I mean you remember all these things already. So, when we escape this character essentially we just print directly like at V and then here, this is literal because it is all inside the code so it would not do this operation it is just prints out the same value but then we when it sees this portion this it converts it into its array and that is 0 1 2 3. So, basically like I mean this is also like ways that you can write out this sentence famous.

(Refer Slide Time: 13:02)



Sort

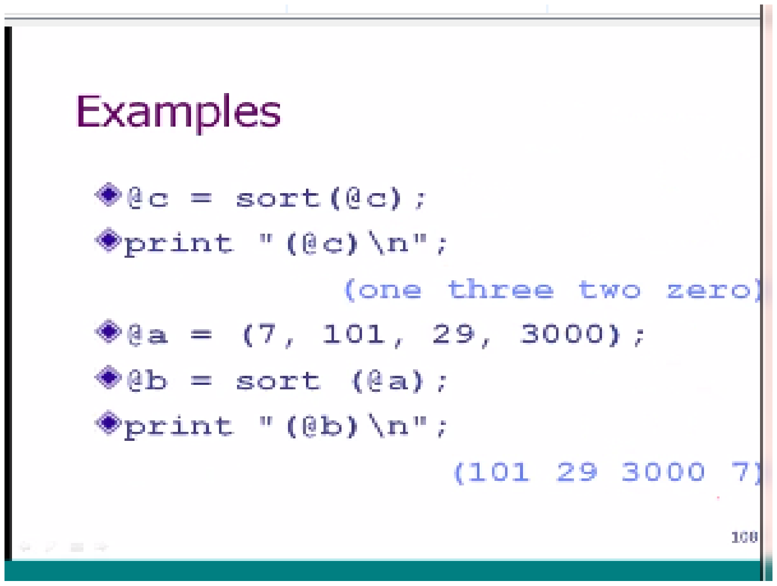
```
@a = (7, 1, 9, 3);
@b = sort (@a);
print "(@b)\n";
                              (1 3 7 9
print "(@a)\n";
                              (7,1 9,3
@c = (zero, one, two, three);
```

107

Okay, and then the other key function is the sort function this is another one that you will be using more often in this in the real world. So here an array is given 7, 1, 9, and 3. And then we fail it back okay, at Bea's sort at A. So, now if we print out that B it prints out 1, 3 7, 9. So you can see that actually it is sorted the in an ascending way. So this is an ascending sort, and then it prints out 1, 2, 3 you know 1, 3, 7, 9 and then the A is actually like an you can see the more food it is kept the same thing basically nothing.

So, the question for you is how do you do descending order sorts think about it. What can we do for descending over for, so one fix solution could be that we can just use this first start the A and then basically like reverse it essentially or you can say like that be equal to reverse sort and A so this basically like, I mean you have got that a first and then put it into rules what happens if you just do the reverse first and then sort it will actually give you the same result right because whatever you are reversing actually does not matter and then basically once you start stable. So now what about the disarray essentially and then what happens in short C.

(Refer Slide Time: 15:06)



Examples

```
◆@c = sort(@c);
◆print "(@c)\n";
                (one three two zero)
◆@a = (7, 101, 29, 3000);
◆@b = sort (@a);
◆print "(@b)\n";
                (101 29 3000 7)
```

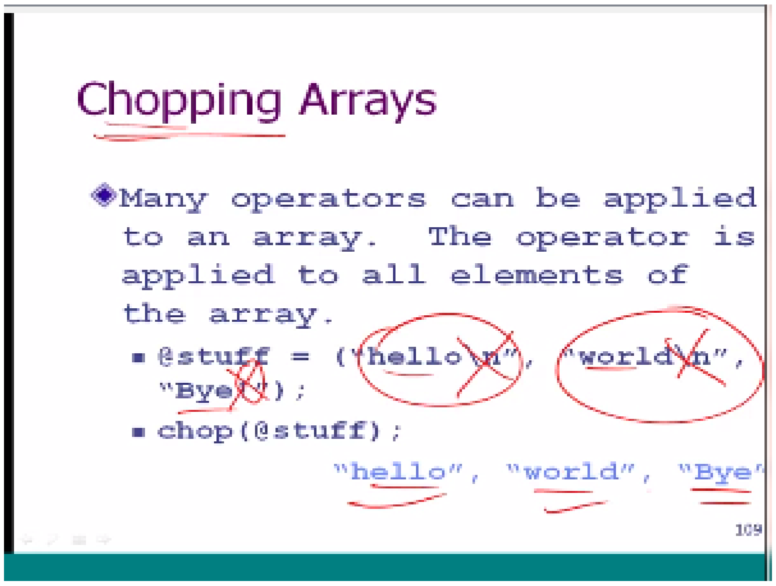
So can you tell me like I mean what happens again this is you should think about this is also like an ascending all up but now it is on the alphabets. So we know that actually like O comes before T and then in this one also the TH comes before TW and then finally this come. So this is 1 this is 2 this is 3 this is 4 or in pearl notation this is 0 this is 1 this is 2 and this. So let us see what happens so here, when we do the sort 1 see that this becomes 0 1 2 and 3. So now we have a another one which is like multiple numbers all have different number of numbers essentially this is 7 this is 101 29 300 and if we do is sort.

What happens essentially like if the sort does not though by the actual value of the number but it is only a little sort? So, in this one this is the first one and this is the first number because the personable so it first starts all the possible. So it comes up with 101 or essentially is 1, 29, 1000 and then 3,000 and then 7.

So the final answer is actually you can see that basically it only did the first number sort the remaining ones it leaves open so to achieve that the real sort what here what we need to do is pretend with zeros for example. So make everything like four-digit number so basically 0 0 0 7 and then 0 2 9 0 0 0 0 2 9 0 1 0 1 and then 3,000. If you do this number then this order will be maintained and then.

What we are getting as a result okay so I think this is an interesting assignment or is an interesting example. As, to how to do the sorting so now let us look at some more functions, essentially like chopping arrays.

(Refer Slide Time: 18:14)



Chopping Arrays

- Many operators can be applied to an array. The operator is applied to all elements of the array.
  - `@stuff = ("hello\n", "world\n", "Bye!");`
  - `chop(@stuff);`

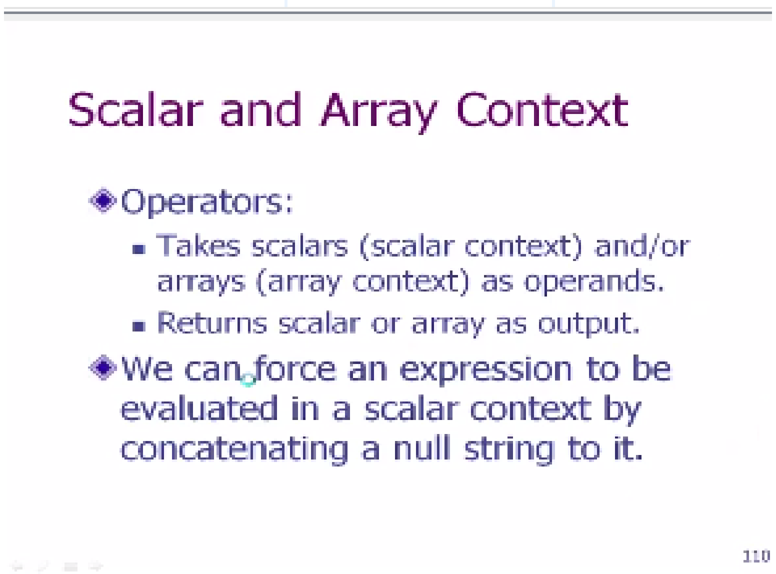
"hello", "world", "Bye"

109

So, we can apply many operators to an array the operator is applied to all the elements of the array. So in this one like I mean so chopping is one of the operations that when we apply this particular operator is applied to all the elements over here, so the net result of this like I mean so if you do a chop on this particular array can anyone tell me like I mean what will be the result. So essentially like the this element this new line gives new line goes away from this element again the new land. What about this one so let us sees so again remember chop removes the last character.

So from here, this exclamation mark extra mode. So we only get bun so if you do like it chop $ stuff this is what we will get we get the hello and you get the world and then you will pick up back you only get the new lines from the first two and then the exclamation character from the next one.

(Refer Slide Time: 19:39)

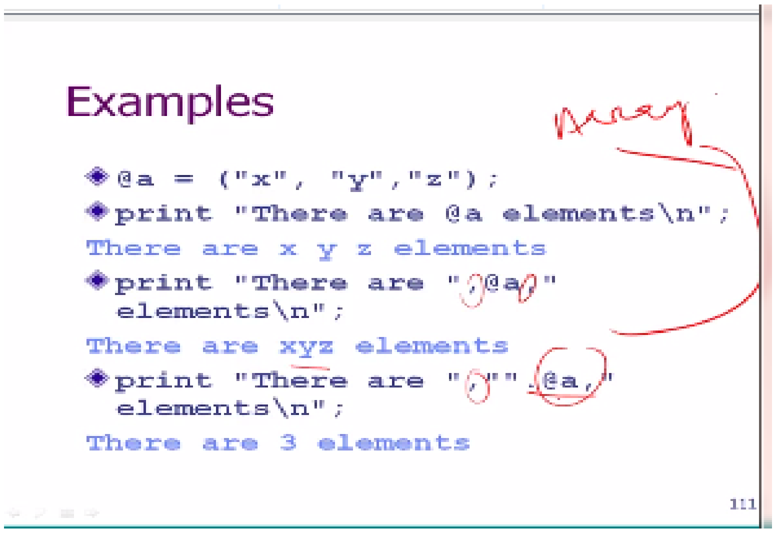


Scalar and Array Context

- Operators:
  - Takes scalars (scalar context) and/or arrays (array context) as operands.
  - Returns scalar or array as output.
- We can force an expression to be evaluated in a scalar context by concatenating a null string to it.

110

So again as I mentioned like I mean the you know earlier the various operators have these various contexts essentially the Taylor and Iowa context. So the it takes both scalar as well as erases a plant it can return scalar and arrays of output if we want an expression to be evaluated in a scalar context we can force that by using a null string to it so what with what we mean by that let us look at some of these examples.
(Refer Slide Time: 20:29)



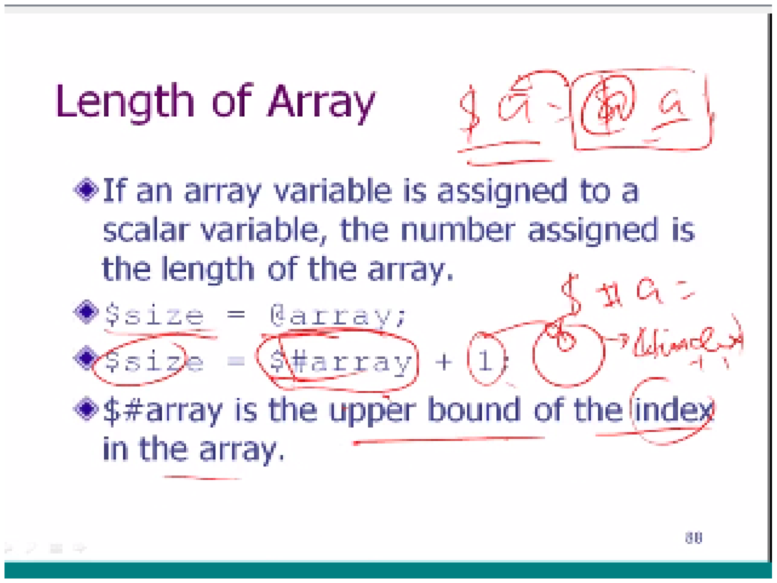
Examples

```
@a = ("x", "y","z");
print "There are @a elements\n";
There are x y z elements
print "There are ",@a," elements\n";
There are xyz elements
print "There are ","".@a," elements\n";
There are 3 elements
```

111

So here is an array is given as XY I agree and then we print out these elements basically dispense or the same thing because a XY and Z. Now, we say essentially like okay how many elements are there. So again if this prints out let me include the comma operator basically like it just you know concatenated everything and then prints out.

Now there is the next one. Which is the concatenating comma to the array now what happens? So here, essentially it only prints the number of elements which is now it is no longer an array context. So this is all like array context now this in this example will become scale abundance.
So again we go back to the original rule so in for the array is essentially like a mini game like and we can have the scalar context or the array context and the operators essentially like and info they can force. We can make them evaluate as a better scalar context ordinary context by this concatenating and null string to the array. So in this one we saw that actually once we give a null string basically in this case like I am in a comma which is a no creature that is an operator and we are concatenating that to the this one to this array.
And there now it is evaluated as four scalar and actually gives the number of elements in the array and so goes to the literally printing all the labels. So here we entered out all the elements whereas here it princess prints out just those the number of elements. We also saw like an operator to print out the number of elements in an array I hope you remember that I am just going back to give you that information this was covered in the last lecture we talked about athletes even before all these things Yeah right length of an array.
(Refer Slide Time: 23:28)



Length of Array

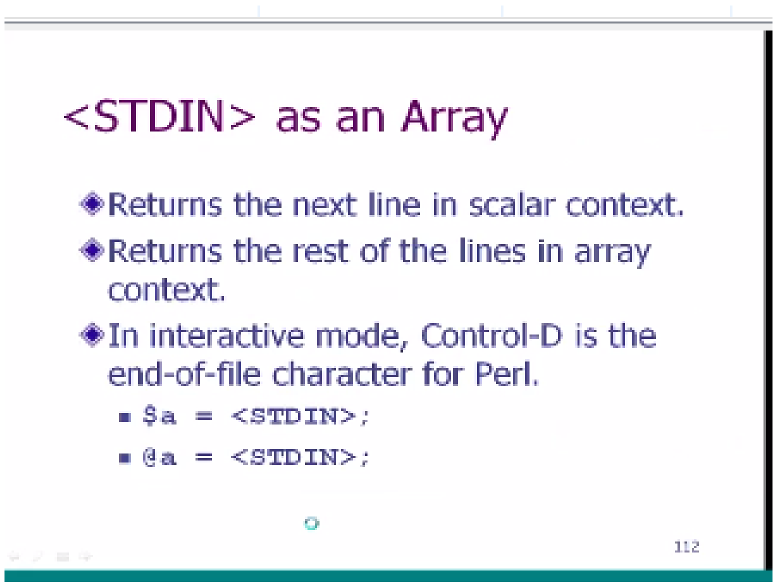
- If an array variable is assigned to a scalar variable, the number assigned is the length of the array.
- `$size = @array;`
- `$size = $#array + 1`
- `$#array` is the upper bound of the index in the array.

88

$a = @a
$#a =
defined
-1

So this is another way to get the array length essentially. Which is the $ hash array name basically or here we also mentioned that the $ size is $ A. So when we specify the $ actually ligament that changes this context into a scalar context and then basically like I mean we get the link of the area. So I hope you remember that. So that is how we are printing that and here we actually like printed out that with this is a essentially like pretty much what we did at this point

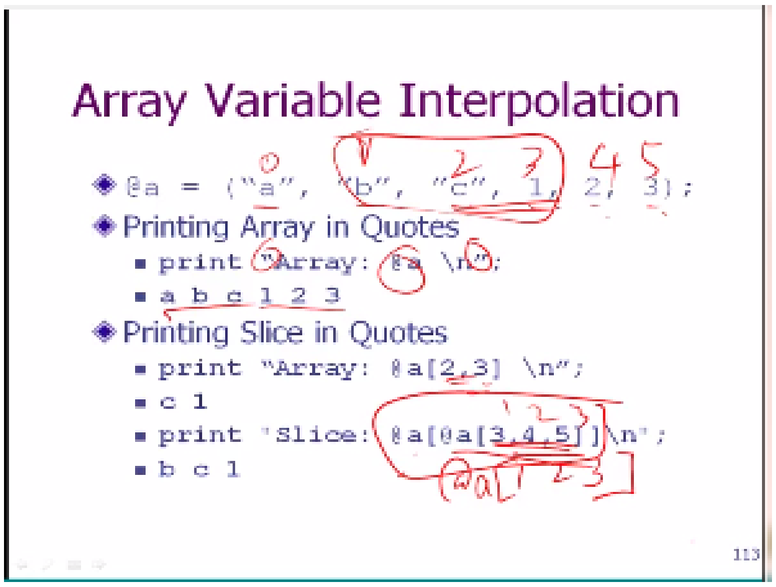
which is we are concatenating with a null string so feed here and then that prints out the actual size.
So think of this like I mean we actually did this in this last class and now essentially make we're just formalizing how we did it by describing this context essentially so now let us go back you so now let us look up look at the some of the things that we have been learning earlier which is a CD I n as you know ligament that is the input to the array from the terminal so far we have been learning that as some scalar context where we actually used it to input one value at a time now let us see like I mean how we can use it to use it in a array context and then see how can we use it in the array fashion.
(Refer Slide Time: 26:14)


<STDIN> as an Array

- Returns the next line in scalar context.
- Returns the rest of the lines in array context.
- In interactive mode, Control-D is the end-of-file character for Perl.
  - `$a = <STDIN>;`
  - `@a = <STDIN>;`

112

So in the scalar context it returns the next line this we saw actually we are when we studied about the scalars. Whereas in an array context actually returns the rest of the lines so in an interactive mode we need to use the control D is that is as the end of the fun in the pipe character for Pearl. So how do we use it is using these two notations for them to a $ A equal to standard in this is a scalar context. And if you use an ampersand a to the standard in this become scenario context so as you know that essentially like, I mean basically like what you open it to is the how it determines whether it is a scalar context or is it a um it is a array context you so now let us look at some examples also of how we can use it.
(Refer Slide Time: 27:54)



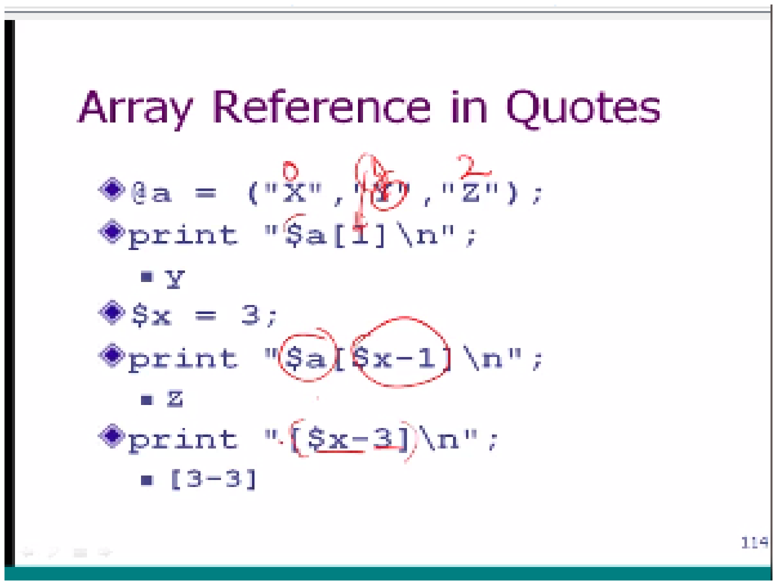
Array Variable Interpolation

- @a = ("a", "b", "c", 1, 2, 3);
- Printing Array in Quotes
  - print "Array: @a \n";
  - a b c 1 2 3
- Printing Slice in Quotes
  - print "Array: @a[2,3] \n";
  - c 1
  - print 'Slice: @a[@a[3,4,5]]\n';
  - b c 1

113

And before that we will let us look at how what is the array variable interpolation essentially. So first of all in the print we already saw that actually we are going wrong even within the double quotes arrays can be interpolated and it can print out the values. So here, when we specify like I mean this is like you can think of you can see that actually it is not a just a one is a numerical array or as of cigarettes it is a combination of both and it well actually cannot distinguish between those I mean it fits them as the same. So it is very versatile in that sense so that is why I can print out ABC and 1, 2, 3.
Then we use the slice essentially like Amoco here it is two three so you know that this is index 0 1 2 3 or 5 so now what we print out is CN 1 which is the correct thing about index 2 and 3 so now when we do this is much more trickier now the slice of at a and then this 3 4 5 so now what should I print out so 3 4 5 we know that it is actually like it is 1 2 and 3. So when? we do another add basically like I mean that is needs to print out this one two and three which is DC and 1 so let us see what it prints out so that is the it prints out DC 1 so again I want to reiterate this one so 3 4 5 so first it is basically like at a the three four five evaluates to one two three and then it prints out what is 1 2 3 which is BC 1.
So you can see that actually like I mean in this case actually it is it is actually interpolating it twice basically like a minute sir first interpolating first set of values and then it goes into the second set of values and in this case like I mean if you make a mistake and say like 1 2 3 or something like that first one as we know like I mean then which is sent out nothing because those

ABC are not of real values and you know like a slice function we cannot have non numeric values. So now let us see like how the array reference or for panning ports.
(Refer Slide Time: 30:54)



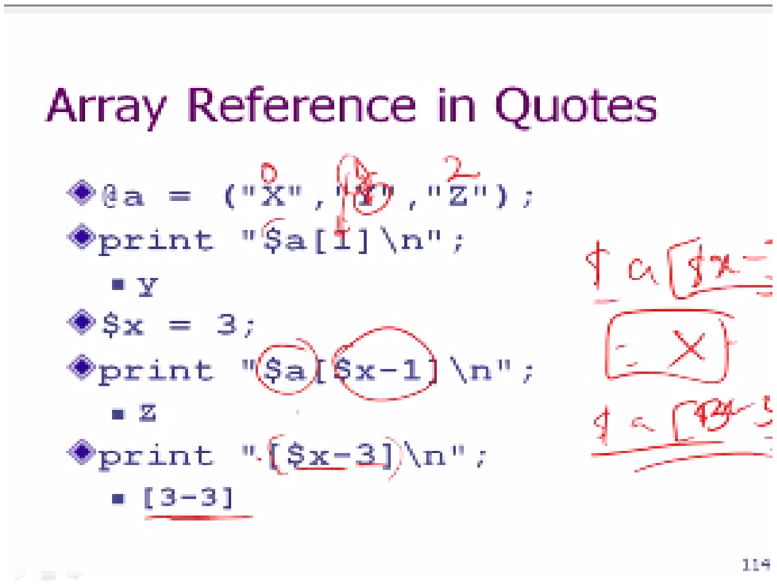
Array Reference in Quotes

```
◆@a = ("X","Y","Z");
◆print "$a[1]\n";
   ▪ Y
◆$x = 3;
◆print "$a[$x-1]\n";
   ▪ Z
◆print "[$x-3]\n";
   ▪ [3-3]
```

114

So in this case like again within the code this is actually interpolated. So A1 is again I will mark it mark this 1 + 0 1 & 2 so you can see that actually in this case it will print out the white because one corresponds to one so one corresponds to Y so the Y will get printed out now let's see ligament in another case X is equal to $ X we are depending - 3 and then $ X - 1 so what do you think this should print out so in this case again this is first evaluated and then it goes and interpolates for the array value.

So in this case it should print out the V which is what it does now when we do this one alone what does it print any answer it directly prints $ 3-3 because see now it knows that actually it is a indexed context but you know there is nothing there here so it will print out direct labels so now we have a another question if you want to print out this like say in kind of expression what do we do.
(Refer Slide Time: 32:40)



Array Reference in Quotes

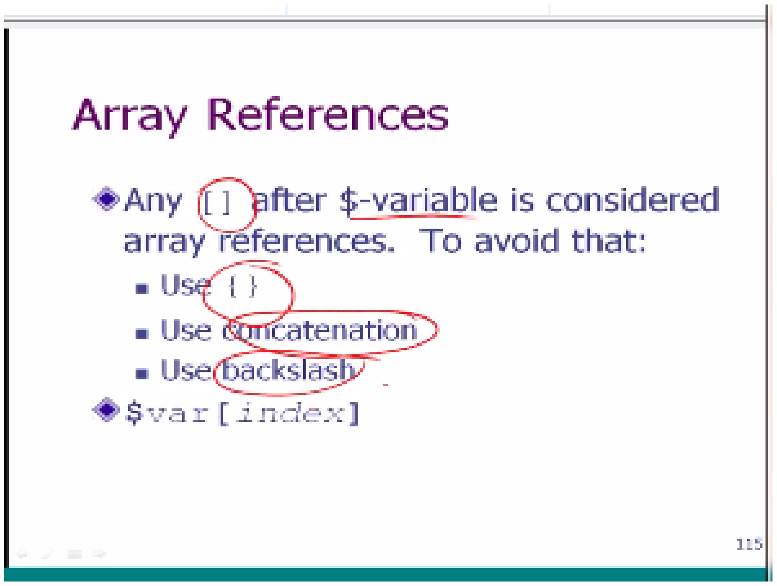
```
◆@a = ("X","Y","Z");
◆print "$a[1]\n";
  ▪ Y
◆$x = 3;
◆print "$a[$x-1]\n";
  ▪ Z
◆print "[$x-3]\n";
  ▪ [3-3]
```

114

So one thing to note is essentially any this square brackets after A $ variable is always considered as an array reference. So if we put like $ A $ X - 3 it cannot understand that okay if you want on a printout literally like I mean this way it you cannot print out because it doesn't know that okay this needs to be printed out.

This way so in this case it returns just X which is not what we want we want to see the heaven below a $ X or 3-3 something like this so to prevent that from printing this X and actually like the stopping the interpolation of the array variable itself. We have to use either this or a concatenation or backslash like I mean all three are okay basically to do this one.

(Refer Slide Time: 33:58)



Array References

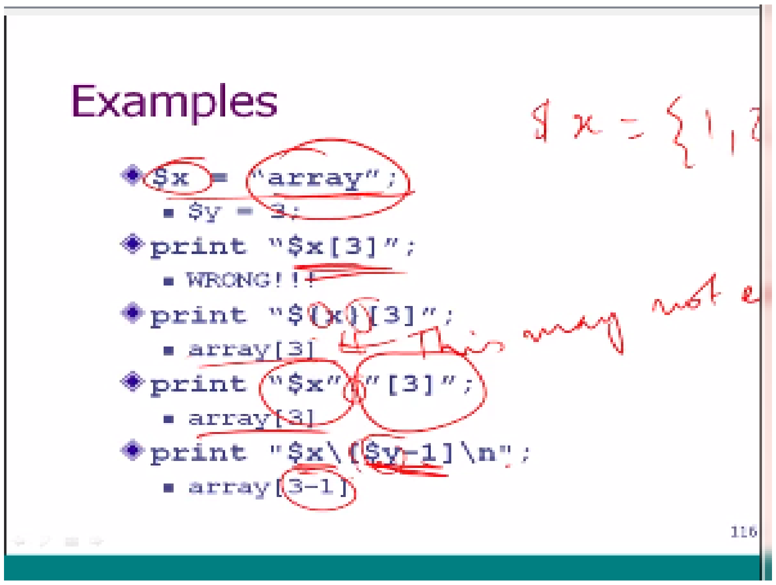
- Any [ ] after $-variable is considered array references. To avoid that:
  - Use { }
  - Use concatenation
  - Use backslash
- `$var[index]`

115

So what do we mean essentially legging so in this using the brackets essentially so you can specify seeing the $ bracket a and then use the card the square brackets and this is one way to actually avoid the contagion and print little terms basically so in this case like a Google say and then this is evaluated as one separate entity and that is not confused with $ a this kind of an evaluation so this is one way to do it the other way to do it will be they color a and then you concatenate with it $ over X - 3 so the concatenation operation again basically like avoids that confusion and make sure.

That we have the we will get the desired effect and then thirdly is basically to use the backlash which is like going on a then back escape the square bracket $ X – 3 also escape with another Scrabble so all clear can be used otherwise it is just treated as for the variables within box so let us look at some examples here.

(Refer Slide Time: 35:45)



Examples

```
◆ $x = "array";
   ▪ $y = 3;
◆ print "$x[3]";
   ▪ WRONG!!
◆ print "${x}[3]";
   ▪ array[3]
◆ print "$x"."[3]";
   ▪ array[3]
◆ print "$x\[$y-1]\n";
   ▪ array[3-1]
```

$x = {1,2

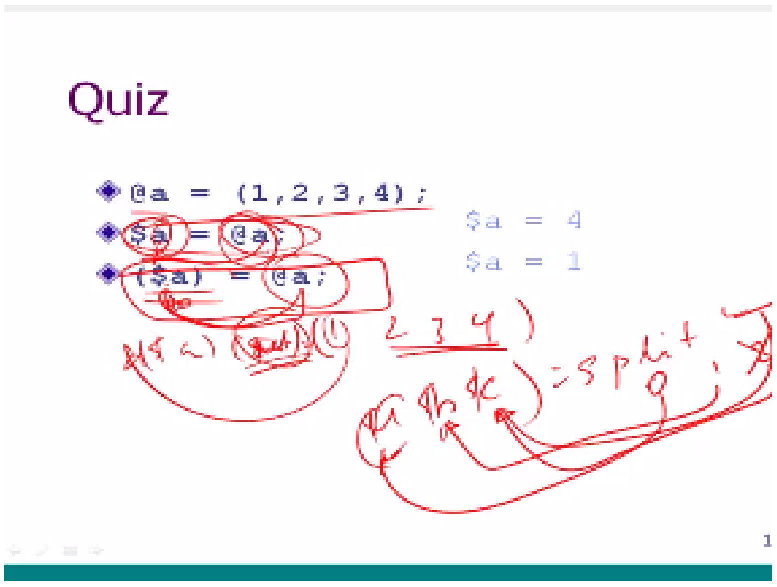
This may not e

116

X is an array and Y is $ three so if you do a $ X and in square bracket $ three this is actually like wrong because there is it is the single element array so again you know that basically a string is an array of characters but then you specify like this it is a single element 1 and the also like I mean in this one this variable is treated as a scalar variable so there is no array context to it so it comes from so to make sure that we can print this way what we need to do is essentially do this so not to use the brackets so in this case is repent as array element 3 so again this printing S element.

We even though this may not exist other way to do it is little X concatenated with the colony so that is also prints up as the same every number three and then the third one that we saw was with

escape the various characters so we can use $ x escape the square brackets and then $ y1 and then this one this will print array to array literally like $ three maximum because it only evaluates this would not evaluate this function.
Because there is the d square brackets are risky so it just puts this and be these ones at the same time if there was such an array of $ x equals one two and three so four of them so in this kind of case basically if you do not escape this these miss ones then it will just print out to first evaluate this 3 - 1.
Which is 2 so this is 0 0 1 2 and so we posed in prints slide it just lets three then it looks it so keep in mind this one this will help you with a lot of bugs avoiding out of bugs essentially for I think this is one of the key things that we will learn in this class also like I mean there are some you some other types of arrays that we will be learning basically in your courses so I am not going to go into like a lot of details at this point.
I think like I have been right now like we have a lot more knowledge than what we started with which we can actually like now really go into more challenging doing some challenging work.
(Refer Slide Time: 39:46)



Quiz

```
@a = (1,2,3,4);
$a = @a;                 $a = 4
($a) = @a;               $a = 1
```

So here, is one quick quiz for you so at a is defined as a four elements array what is $ a we could specify $ able to add a what does that mean and if you specify parent this is $ a equals at a four so I am sure that you may have guessed it right away this is something that we covered in the last lecture so at a denotes the number of elements basically solids.

The $ a equal to a day this assignment it assigns the number of elements to but this one so are converting the array context into a scale or context so the answer is value is 4 but what about this the second essentially here.
It is actually like now it combines that so what does that mean basically so when we specify this one this actually means the first element of this array is a stein $2 a so this is something that we use not a lot of times basically like so it is this so if you specify like I mean we can I say that this is 4 that is 1 2 3 4 or and then basically the first element is captured in other a and then the remaining is like this and then that is this 2 3 4 and we will see like.

I mean what are the some of the so pull actually like a sense this number is variables automatically. So in the next lecture or I mean in the future lectures we will see.
Like I mean how do we depth this the main system because this is something that is for a very commonly used operation even like I mean you can say in an say we will we will say basically like an grade any line and then split on something and then we can say like okay assign it to ABC in this content all the way one of your boxing what this means is basically like.
I mean the first element the zeros element is assigned to and then the one turn is a time to be and then to be sent to see and in this case like I mean if - the result is not there and what rest is assigned to see so this is a useful operation so try to remember this.
We will be also like learning more about this and then so this is pretty much like end of a one dimensional array lecture so I hope like I mean this is useful and right now we talked about just single dimensional arrays we will go into more complex data structures which are the multi dimensional array essentially multi-dimensional arrays also we can use , so lists and lists of lists to represent these multi-dimensional arrays we will learn about that again static assignment still would not get like lot more memory offensive in nature.
So we will typically avoid that but essentially like I mean we will see how lists are done in Perl in the next class.
So I think that is pretty much done so let me just do a recap of what we learnt today again we started with some other new elements in the array with this basically like reverse function so the reverse function is essentially like I mean it is an operator on an array and we get the value as essentially the indexes being rivers in the reverse order.
So if you have a ten element array when we do a reverse operation it still returns a ten elementary but the indexes are exactly opposite so the 0th element will become the ninth element and then the first element will become the eighth element and so and so all the way and that is

the reverse of nature then we also looked at the spot essentially like sort is an ascending order sort.
And it does not it is not a numerical spot per se it is really just an alphabetical sort but we can use it as a numerical sort give it sufficient number of numbers or make all the numbers same length then we can make it as an insult and we saw actually like . I mean if we do this seven one nine three then it will turn proper way and then we also looked at how to do an descending odd sort which is like nine seven three one how to get that those numbers and for that we use both the function reverse and the fourth.
And then it can be evaluated together and we will get the required or required effect and then the even any string arrays can be sorted and since the start function for based on that before the string arrays will also be sorted in the with model as in this case.
Basic is to be 1 3 to 0 cell by number so when you start like length with multiple length numbers you have to make sure you have to take care because it will still sort based on the first number so that you would get the desired effect then we looked at the chopping the array essentially like chop is when you use chopped as an operator those of that operator is applied to all the elements of the array.
So here when you do the top of ad star every element is getting chopped with the last the last character so or hello and world just the newline those gets thrown away and I think by the exclamation mark also then we formalize the concept of the scalar and the array context the first one basically.
We saw as to how to represent the array itself in a scalar than the anari context go in an array context essentially elegantly this let it be and then you get the answer as just X Y Z the trace elements are those so elements whereas if we concatenate the array with a null character then we will get the number of array of the scalar context which is the number of elements inside the Dalek and so we get like an interaction number.
Then we also looked at the stain as an array because again every variable will have the scale of context and or added context so if you specify like $ A equal to standard M then that assumes that it is a scale of context and it returns the next line.
Whereas if you do an at a equal to standard I n then it keeps getting line after line until we shift control D so that is an in the array context then we looked at how arrays will be interpolated in various situations mainly it to print arrays in a imports essentially ligament we need to specify the codes and then will be specified area inside.
The court that array will be evaluated and then four slices the same thing basic things like this will be evaluated the ablated within the put and then it turns out the answer appropriately light

also can have like the multiple evaluation system we are giving for you can evaluate it once and then further evaluations will be similar kind of things basically so you can have, so this is this is something like.

I mean to keep in mind because we will learn about these kind of things in the multi dimensional context as well so in this case essentially like the at three four five as a slice results in the value one two and three which again is a slice and then those values are the same which is PC and more so that is another thing that we learnt in the context related or out into play and then now let us look at that the other interpolation.
Since this is another thing that we want so when we try to interpolate arrays inside the code essentially Legume.
So, it is evaluated and then we completely evaluated and then we get that result so for example in this one but why is the result and in this one because the Z S also and then the other thing that we also noticed was when we do this kind of just the square brackets without any annexation front it literally prints out everything.
So it since of the square brackets it prints out $ X which is actually three and then it also prints out this minus sign so 3 - 3 so this entire thing is getting printed out so that leads to an interesting conversation basically once we have the array reference itself.
Then how do we how do we use that or how do we distinguish between an array reference and non array variable so the three ways to distinguish that is this hole here one two and three the first one is to use this the brackets essentially so that is what we mentioned missing anything $ a is a scalar variable.
We use $ in brackets a to distinguish it between a scalar variable and then array with an index and then use like $s x moves three so and then the second method is using the concatenation so we instead of e saying that $ a and then like this we use the concatenation with the next one.
So here, essentially like ending focus is treated as one little video so whatever this variable may be evaluated this is this represented as is this one and then all of X evaluated. With the screen mode so epileptic and then finally the using the backslash or an escape character this is this one this which is used to escape.
These control values cinching for example, here square brackets can be used in other contexts essentially for representing the area index but once we escaped it then a literal value is used which is just the square brackets, so you think that we can actually get around mistaking the $ variable fallen and context.

So this is something that we also want and give there is an example that we talked about one thing to note here is when we have the $ x as a scale of context which is from here we use it in our context pearl does not like it and error out so here it explains all the three methods that we I talked about essentially moved about in all ways and then finally like we also saw like a small quiz to distinguish between a scale of context and an area context for example.
In this one it is a scalar context this is the example that we saw earlier for this print out the number of women for this ad which is the same as for and then the other thing that is relevant is form is using the parentheses to actually get values from gallery or a tea which is very essentially.
So from here when we obtain that $4 a what we get is essentially the first element on the at a which is element Oh first one dancer is E4 and for the second one dancer is the other eight to one so I think like that is pretty much it for today once again thanks for listening and thanks for all the things okay.
I will see you next time thanks bye Oh you hi everyone again welcome to welcome to this lecture this is again the I want to remind you about the class that we are talking about is Linux programming and scripting today we are going to continue our lecture on goal we covered a lot of things about Perl.
So far we started with just doing some really simple programming Perl then we went into more kind of like we started learning about the data structure he lent a couple of them one is the main data structure that that is pretty much the workhorse of Perl is the scalars we learned about scalars and then we went into the arrays actually in between we also learnt about strings.
We touched upon it I think like I mean there is more two strings that we will be seeing in the future classes in couple of future classes but now we will be we are talking about arrays and I want to continue our discussion on more a and today I will be finishing up with arrays.
So that you have a good understanding of how to represent our ratings how to use arrays in your programs and how to manipulate various data using arrays so all these things are the key things be learning around objectives from this session.
So before I start with the today's lecture I want to give you a brief recap of the what we learned in the last lecture so let's begin with that discussion you so the last feature we first stopped started talking about the range operator I hope you remember the range operator which is the two dots which represents essentially a range of numbers and it has two contexts.

That we talked about one was irate context and the other one with a la Congo and what is the difference between those two that also p1 was good basically in the scalar context it also has a

Boolean nature so that it actually returns a true or false value so that is one thing that we talked about then we went into a racing more detail form we understood how to specify an array and how the array literal works.
Basically what are the rules governing it in terms of putting a $ in front and then it needs to be in a square bracket that you need to specify and then you are a itself with the elements of super fighting the regular brackets.
Basically in the moment we can specify each and every element inside that array and then work on that and then they are addressed using this form the index indices are index this is subscribe in square bracket then we also talked about the slice I just wanted to remind you about slice let's just quickly look into the slice operator.
So the slice is essentially we it is a shorthand representation of multiple elements of an array so here this is something that we talked about basically the add sign or the ampersand sign with the name and then we can specify multiple elements form with comma separated once.
And it is initially like I mean that specified with all the ways elements of that array and we also noted some of the rules governing the slice basically follow the indices may not be increasing and in this is need not be different basically the same indices also constitutes the slice then we looked at the stack essentially stack as I mentioned basically the set of memory basically which are which is stackable.
You can say like meaning it has a property of last in first out so that is what this LIFO means basically was not Buster which means that essentially the elements that are added first comes out the last and the last one that is added into the stack comes out the first and the operation for putting an element in the stack and then removing the element of stack that is a two things one is pushing the putting the element inside.
The stack is known as push operator and then when you are removing the element from book stack we use a pop operator to remove and here you see one key thing key difference between this and then wiggler arrays you do not specify any kind of in sticks, so you just say push at list and then that is assigned to a new element and then so basically knows that whatever is last that you added that element is actually like an even.

So the new element is added as the top of the essential and it automatically increments the index but the index is unknown to the user so the user only sees the only one element of stack and then here is the pop usage of the pop the one.
Basically where we are assigning the pop at list to this variable $ top and again what this does is you do not need to worry about okay whether it is in the hundred in there for 150 whatever it is

but the top of the element of the last added element is this form pop and then it is put inside this new variable doctor so this is something that we learned and then we saw some examples of this form and more and more things basically that being.
So then we looked at this shift and un shift again these are some of the key elements essentially which is kind of this one is like the opposite of push and pop very similar but it is kind of opposite in the sense that the shift is essentially like to the added the element is added to the lower end of the array instead of the top of the array.
So the last element is in is incremented last index is incremented and then it is added to that so here shift means basically like the last one that is worth added that gets popped out basically and then it goes into the array and then the array is kept as is essentially we go home and then un shifted enchi Lille again it puts back that at the top.
So these are some of the useful commands essentially be legend for when we say like on shift for five actually like them you can see them the 99 is pushed and then the four and five are added at the end and then we also saw some examples of the shift command here is some other some more essentially. We are putting the thing inside that.
And then we also like shifting there the taller X will get lost as the value and then essentially the remaining array will be intact so that is the recap from the last lecture now we are going into the today's lecture we will continue with the arrays I want introduce a couple of new things to you guys basically so let us see the first one is the reverse operator reverse is essentially used to reverse a complete an array with all the indexes or indices for completely worst so in this array for example so the taller a is 0 1 2 3.
So this one you can say like it is the $ a 0 element this is $ a one accessory and this one is like $ a three so now we say the new array at B is reverse of at a one thing you notice is essentially when we declare a new array we do not have to declare it up then we can directly make this assignment.

In the modern Perl actually use for mine or for a moniker for specifying available if it is a good practice to actually even if you are directly using it to specify the array with them the my in front so this is this becomes like move and then at be equal to so this is a good practice whenever you're using the add V for the first time use the mind then Paul knows that actually needs to allocate the memory for this variable.
So now let us look at this essentially though when they reverse so we are assigning this as a reverse so now for B 0 that value is 3 and B 1 is 2 and B 3 is actually 0 all have a $ in front in this example we are printing the a corresponding a from 0 through 3.

You can see that it is actually length 0 to 3 the values themselves whereas for B the B 0 the response to 3 and then B 3 corresponds to 0 so we basically like I mean they to print out this is essentially like I mean specify like.
I hope I mean you remember all these things already so when we escape this character essentially we just print directly like at V and then here this is literal because it is all inside the code so it would not do this operation it is just prints out the same value but then we when it sees this portion this it converts it into its array and that is 0 1 2 3 so basically like I mean this is also like ways that you can write out this sentence okay and then the other key function is the start function.
This is another one that you will be using more often in this in the real world so here an array is given seven one nine three and then we fail it back okay at Bea's sort at a so now if we print out that B it prints out one three seven nine so you can see that actually it is sorted there in an ascending way so this is an ascending sort.
And then it prints out one two three you know one three seven and then the A is actually like an you can see the more food it is kept the same thing basically nothing so the question for you is how do you do a descending order sort think about it what can we do for descending over four so one fix solution could be that we can just use this first start the A.
And then basically like reverse it essentially or you can say like that be equal to reverse sort and a so this basically like I mean you have got that a first and then put it into rules what happens if you just do the reverse first and then sort it will actually give you the same result right because whatever you mare reversing actually does not matter and then basically once you start stable.

So now what about the disarray essentially and then what happens in short fee so can you tell me like I mean what happens again this is you should think about this is also like an ascending all up but now it is on the alphabets, so we know that actually like Oh comes before tea and then in this one also TH comes before TW and then finally this come.
So this is 1 this is 2 this is 3 this is 4 or in pearl notation this is 0 this is 1 this is 2 and this is 3 so let us see what happens so here when we do the sort 1 see that this becomes 0 1 2 & 3 you so now we have a another one which is like multiple numbers all have different number of numbers essentially this is 7 this is 101 39 300 and if we do is sort what happens essentially like if the sort does not go by the actual value of the number but it is only a little sort so in this one this is the first one and this is the first number because the personable so it first starts all the possible so it comes up with 101 or essentially.

It is 129 thousand and then 3,000 and then 7 so the final answer is actually you can see that basically it only did the first number sort the remaining ones it leaves open so to achieve that the real sort what here what we need to do is pretend with zeros for example ,so make everything like four-digit number so busy row 0 0 7 and then 0 2 9 0 0 0 0 2 9 0 1 0 1 and then 3,000 if you do this number.
Then this order will be maintained and then what we are getting as a result okay, so I think this is an interesting assignment or is an interesting example as to how to do the sorting so now let us look at some more functions essentially like chopping erase so we can apply many operators to an array the operator is applied to all the elements of the array.
So in this one like I mean so chopping is one of the operations that when we apply this particular operator is applied to all the elements over here so the net result of this like I mean so if you do a chop on this particular array can anyone tell me like I mean what will be the result so essentially like the this element this new line gives new line goes away from this element again the new land.
What about this one so let us see so again remember chop removes the last character so from here this exclamation mark extra mode so we only get bun so if you do like it chop $ stuff this is what we will get we get and you get the world and then you will pick up back you only get the new lines from the first two then array the exclamation character from the next one.
So again as I mentioned like I mean the you know earlier the various operators have these various contexts essentially the Taylor and Iowa context so the it takes both scalar as well as erases a plant it can return scalar and arrays of output if we want an expression to be evaluated in a scalar context we can force that by using a null string to it so what with what we mean by that let us look at some of these examples.
So here is an array is given as XY I agree and then we print out these elements basically dispense or the same thing because a XY and z now we say essentially like okay how many elements are there.
So again if this prints out let me include the comma operator basically like it just you know concatenated everything and then prints out now there is the next one which is the concatenating comma to the array now what happens so here essentially it only prints the number of elements which is now it is no longer an array context so this is all like array context now this in this example will become a scale abundance.
So again we go back to the original rule so in for the array is essentially like a mini game like and we can have the scalar context or the erase comedy context and the operators essentially like

and info they can force we can make them evaluate as a better scalar context ordinary context by this concatenating.
And null string to the memory so in this one we saw that actually once we give a null string basically in this case like I'm in a comma which is a no creature that is an operator and we are concatenating that to the this one to this array and there now it is evaluated as four scalar and actually gives the number of elements in the array and so goes to the literally printing all the labels so here we entered out all the elements.
Whereas here it princess prints out just those the number of elements we also saw like an operator to print out the number of elements in an array I hope you remember that I'm just going back to give you that information this was covered in the last lecture we talked about athletes even before all these things Yeah right linked of an array so this is another way to get the array length essentially which is the $ hash array name basically or here.
We also mentioned that the $ size is $ a so when we specify the $ actually ligament that changes this context into a scalar context and then basically like I mean we get the link of the area so I hope you remember that so that is how we are printing that and here we actually like printed out that with this is a essentially like pretty much what we did at this point which is we are concatenating with a null string so feed here and then that prints out the actual size so think of this like I mean we actually did this in this last class and now essentially make we're just formalizing how we did it by describing this context essentially.
So now let us go back you so now let us look up look at the some of the things that we have been learning earlier which is a CD I n as you know ligament that is the input to the array from the terminal so far we have been learning that as some scalar context where we actually used it to input one value at a time now let us see like.
I mean how we can use it to use it in a array context and then see how can we use it in the array fashion you Oh so in the scalar context it returns the next line this we saw actually we are when we studied about the scalars whereas in an array context actually returns the rest of the lines so in an interactive mode we need to use the control D is that is as the end of the fun in the pipe character for Pearl.
So how do we use it is using these two notation for them to a $ a equal to standard in this is a scalar context and if you use an ampersand a to the standard in this become scenario context so as you know that essentially like, I mean basically like what you open it to is the how it determines whether it is a scalar context or is it a um it is a array context you so now let us look at some examples.

Also of how we can use it and before that we will let us look at how what is the array variable interpolation essentially so first of all in the print we already saw that actually we're going wrong even within the double quotes arrays can be interpolated and it can print out the values.
So here when we specify like I mean this is like you can think of you can see that actually it is not a just a one is a numerical array or as of secrets it is a combination of both and it well actually cannot distinguish between those I mean it fits them as the same so it is very versatile in that sense.
So that is why I can print out ABC and one two three then we use the slice essentially like Amoco here it is two three so you know that this is index 0 1 2 3 or 5 so now what we print out is CN 1 which is the correct thing about index 2 and 3 so now when we do this is much more trickier now the slice of at a and then this 3 4 5 so now what should I print out so 3 4 5.
We know that it is actually like it is 1 2 & 3 so when we do another add basically like I mean that is needs to print out this one two and three which is DC and 1 so let us see what it prints out so that is the it prints out DC 1 so again I want to reiterate this one so 3 4 5 so first it is basically like at a the three four five evaluates to one two three and then it prints out what is 1 2 3 which is BC 1 so you can see that actually like I mean in this case actually it is actually interpolating it twice basically like a minute sir first interpolating first set of values.
And then it goes into the second set of values and in this case like I mean if you make a mistake and say like 1 2 3 or something like that first one as we know like I mean then which is sent out nothing because those ABC are not of real values and you know like a slice function we cannot have non numeric values you so now let us see like how the array reference or for panning ports.
So in this case like again within the code this is actually interpolated so a1 is again I will mark it mark this 1 plus 0 1 & 2 so you can see that actually in this case it will print out the white because one corresponds to one so one corresponds to Y so the Y will get printed out now let us see ligament in another case X is equal to $ X we are depending - 3 and then $ X – 1.
So what do you think this should print out so in this case again this is first evaluated and then it goes and interpolates for the array value so in this case it should print out the V which is what it does now when we do this one alone what does it print any answer it directly prints $ 3-3 because see now it knows that actually it is a indexed context but you know there is nothing there here so it will print out direct labels.
So now we have a another question if you want to print out this like say in kind of expression what do we do so one thing to note is essentially any this square brackets after a $ variable is always considered as an array reference so if we put like $ a $ x - 3 it cannot understand.

That okay if you want on a printout literally like I mean this way it you cannot print out because it does not know that okay this needs to be printed out this way so in this case it returns just X which is not what we want we want to see the heaven below a $ X or 3-3 something like this so to prevent that from printing this X.
And actually like the stopping the interpolation of the array variable itself we have to use either this or a concatenation or backslash like I mean all three are okay basically to do this one so what do we mean essentially legging so in this using the brackets essentially so you can specify seeing the $ bracket a and then use the card the square brackets and this is one way to actually avoid the contagion and print little terms basically.

So in this case like a Google say and then this is evaluated as one separate entity and that is not confused with $ a this kind of an evaluation so this is one way to do it the other way to do it will be they color a and then you concatenate with it $ over X minus three so the concatenation operation again basically like avoids that confusion and make sure.
That we have the we will get the desired effect and then thirdly is basically to use the backlash which is like going on a then back escape the square bracket $ X - 3 also escape with another Scrabble so all clear can be used otherwise it is just treated as for the variables within box so let us look at some examples here X is an array and Y is $ 3.
So if you do a $ X and in square bracket $ three this is actually like wrong because there is it is the single element array so again you know that basically a string is an array of characters but then you specify like this it is a single element 1 and the also like I mean in this one this variable is treated as a scalar variable.
So there is no array context to it so it comes from so to make sure that we can print this way what we need to do is essentially do this so not to use the brackets so in this case is repent as array element 3 so again this printing s element we even though this may not exist other way to do it is little X concatenated with the colony so that is also prints up as the same every number three and then the third one that we saw was with escape the various characters.
So we can use $ X escape the square brackets and then $ Y1 and then this one this will print array to array literally like $ three maximum because it only evaluates this won't evaluate this function because there is the d square brackets are risky, so it just puts this and be these ones at the same time if there was such an array of $ x equals one two and three.
So four of them so in this kind of case basically if you do not escape this these miss ones then it will just print out to first evaluate this 3 - 1 which is 2 so this is 0 0 1 2 and so we posed in prints

slide it just lets three then it looks it so keep in mind this one this will help you with a lot of bugs avoiding out of bugs essentially.
For I think this is one of the key things that we will learn in this class also like I mean there are some you some other types of arrays that we will be learning basically in your courses so I'm not going to go into like a lot of details at this point. I think like I have been right now like we have a lot more knowledge than what we started with which we can actually like now really go into more challenging doing some challenging work.

So here is one quick quiz for you so at a is defined as a four elements array what is $ a we could specify $ able to add a what does that mean and if you specify parent this is $ a equals at a four so I am sure that you may have guessed it right away this is something.
That we covered in the last lecture so at a denotes the number of elements basically solids the $ a equal to a day this assignment it assigns the number of elements to but this one so are converting the array context into a scale or context so the answer is value is 4 but what about this the second essentially here it is actually like now it combines that so what does that mean basically so when we specify this one this actually means the first element of this array is a stein $2 A so this is something that we use not a lot of times basically like. So it is this so if you specify like I mean we can I say that this is 4 that is 1 2 3 4 or and then basically the first element is captured.
In other a and then the remaining is like this and then that is this 2 3 4 and we will see like I mean what are the some of the so pull actually like a sense this number is variables automatically so in the next lecture or I mean in the future lectures we will see like I mean how do we depth this the main system because this is something that is for a very commonly used operation even like I mean you can say in an say.
We will we will say basically like an grade any line and then split on something and then we can say like okay assign it to ABC in this content all the way one of your boxing what this means is basically like I mean the first element the zero the element.
Is assigned to and then the one turn is a time to be and then to be sent to see and in this case like I mean if - the result is not there and what rest is assigned to see so this is a useful operation so try to remember this we will be also like learning more about this and then so this is pretty much like end of a one dimensional array lecture.
So I hope like I mean this is useful and right now we talked about just single dimensional arrays we will go into more complex data structures which are the multi dimensional array essentially multi-dimensional arrays also we can use so lists and lists of lists to represent these multi-dimensional arrays.

We will learn about that again static assignment still would not get like lot more memory offensive in nature so we will typically avoid that but essentially like I mean we will see how lists are done in Perl in the next class, so I think that is pretty much done, so let me just do a recap of what we learnt today again we started with some other new elements in the array with this basically like reverse function.
So the reverse function is essentially like I mean it is an operator on an array and we get the value as essentially the indexes being rivers in the reverse order so if you have a ten element array when we do a reverse operation it still returns a ten elementary but the indexes are exactly opposite.
So the 0th element will become the ninth element and then the first element will become the eighth element and so and so all the way and that is the reverse of nature then we also looked at the spot essentially like sort is an ascending order sort and it does not it is not a numerical spot per se it is really just an alphabetical sort , but we can use it as a numerical sort give it sufficient number of numbers or make all the numbers same length then we can make it as an insult and we saw actually like I mean if we do this seven one nine three then it will turn proper way and then.
We also looked at how to do an descending odd sort which is like nine seven three one how to get that those numbers and for that we use both the function reverse and the fourth and then it can be evaluated together and we will get the required or required effect and then the even any string arrays can be sorted and since the start function for based on.
That before the string arrays will also be sorted in the mouth with model as in this case basic is to be 1 3 to 0 cell by number, so when you start like length with multiple length numbers you have to make sure you have to take care because it will still sort based on the first number so that you would not get the desired effect then we looked at the chopping the array.
Essentially like chop is when you use chopped as an operator those of that operator is applied to all the elements of the array ,so here when you do the top of ad star every element is getting chopped with the last the last character.
So or hello and world just the newline those gets thrown away and I think by the exclamation mark also then we formalize the concept of the scalar and the array context the first one basically we saw as to how to represent the array itself in a scalar than the anari context go in an array context essentially elegantly this let it be and then you get the answer as just X Y Z the trace elements are those.

So elements whereas if we concatenate the array with a null character then we will get the number of array of the scalar context which is the number of elements inside the Dalek and so

we get like an interaction number, then we also looked at the stein as an array because again every variable will have the scale of context and or added context, so if you specify like $ a equal to standard M.
Then that assumes that it is a scale of context and it returns the next line whereas if you do an at a equal to standard I n then it keeps getting line after line until we shift control D so that is an in the array context then we looked at how arrays will be interpolated in various situations mainly it to print arrays in a imports essentially ligament.
We need to specify the codes and then will be specified area inside the court that array will be evaluated and then four slices the same thing basic things like this will be evaluated the ablated within the put and then it turns out, the answer appropriately light also can have like the multiple evaluation system, we are giving for you can evaluate it once and then further evaluations will be similar kind of things basically.
So you can have so this is this is something like I mean to keep in mind because we will learn about these kind of things in the multi dimensional context as well so in this case essentially like the at three four five as a slice results in the value one two and three which again is a slice and then those values are the same which is PC and more so that is another thing that.
We learnt in the context related or out into play and then now let us look at that the other interpolation since this is another thing that we want so when we try to interpolate arrays inside the code essentially Legume, so it is evaluated and then we completely evaluated and then we get that result so for example in this one.
But why is the result and in this one because the Z S also and then the other thing that we also noticed was when we do this kind of just the square brackets without any annexation front it literally prints out everything so it since of the square brackets it prints out $ X which is actually three and then it also prints out this – sign.
So three minus three so this entire thing is getting printed out so that leads to an interesting conversation basically once we have the array reference itself then how do we how do we use that or how do we distinguish between an array reference and non array variable so the three ways to distinguish that is this hole.

Here one two1, 2 and 3 the first one is to use this the brackets essentially so that is what we mentioned missing anything $ a is a scalar variable we use $ in brackets a to distinguish it between a scalar variable and then array with an index and then use like $ S X moves 3.
So and then the second method is using the concatenation so we instead of E saying that $ A and then like this we use the concatenation with the next one , so here essentially like ending focus is

treated as one little video so whatever this variable may be evaluated this is this represented as is this one and then all of X evaluated with the screen mode.
So epileptic and then finally the using the backslash or an escape character this is this one this which is used to escape these control values cinching for example here square brackets can be used in other contexts essentially for representing the area index but once.
We escaped it then a literal value is used which is just the square brackets so you think that we can actually get around mistaking the $ variable fallen and context so this is something that we also want and give there is an example that we talked about one thing to note here, is when we have the $ x as a scale of context which is from here we use it in our context pearl does not like it and error out.
So here it explains all the three methods that we I talked about essentially moved about in all ways and then finally like we also saw like a small quiz to distinguish between a scale of context and an area context for example in this one it is a scalar context.
This is the example that we saw earlier for this print out the number of women for this ad which is the same as for and then the other thing that is relevant is form is using the parentheses to actually get values from gallery or a tea which is very essentially, so from here when we obtain that $ 4 A what we get is essentially the first element on the at a which is element Oh first one dancer is E 4 and for the second one dancer is the other eight to one.
So I think like that is pretty much it for today once again thanks for listening and thanks for all the things um okay I will see you next time thanks bye.