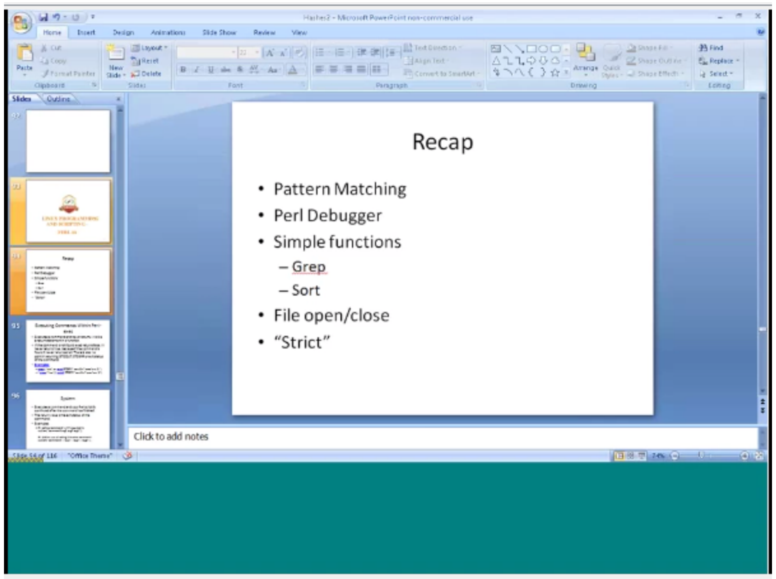
(Refer Slide Time: 00:01)



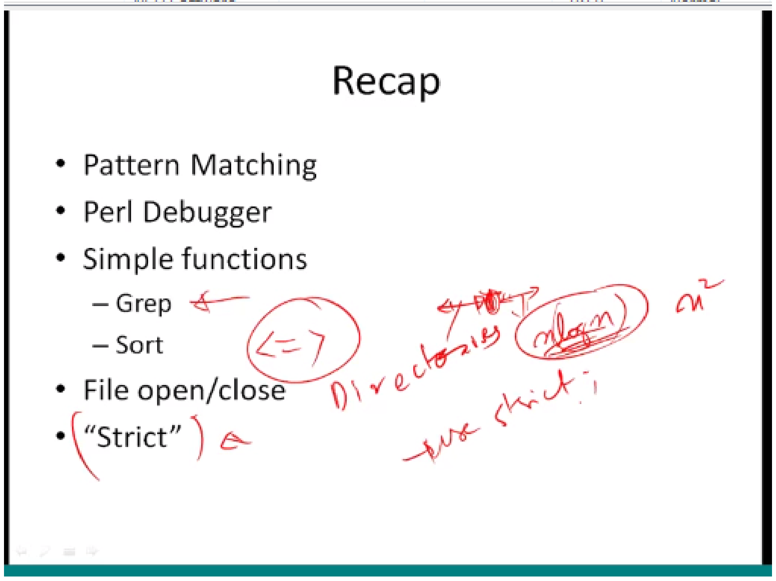
Recap

- Pattern Matching
- Perl Debugger
- Simple functions
  - Grep
  - Sort
- File open/close
- "Strict"

SEER AKADEMI LINUX PROGRAMMING AND SCRIPTING – PERL 10

Hi everyone again welcome to the Linux programming and scripting class, we will be talking about Perl in todays lecture, let us look at what we covered in the last lecture and then we can start from there so let us quickly go to that.

(Refer Slide Time: 00:28)



Recap

- Pattern Matching
- Perl Debugger
- Simple functions
  - Grep
  - Sort
- File open/close
- "Strict"

<=>
Directories
$n\log n$
n^2
use strict;

So in the last lecture we had a lot of things that he covered a lot of the things that new to most of you one thing, we went into like more details about pattern matching I think like I mean this is

one of the key things that you want to understand, in full then we also took it took a look into the Perl debugger the various debugging commands the even through, and I think like I mean that is another thing we want to run into trouble with in equal programs you can involve the Perl debugger.
And then use the Perl debugger forgivable to debug your program that will be like very useful and then we also looked at the functions some of the function, and more in depth manner and one was Grep, the Grep is a function that is used to Grep particular word from a given array of words of given dictionary, and actually like I mean grep is used widely to compare two arrays and get the difference of the array or, compare two arrays and give unique items or the inverse of the differences essentially meaning or the commonality between the two arrays that is the one thing.
And then also like the difference of the array as the output and then these things can be used among simple function we looked at it basically how we can combine the hash table concept or hash array concept, and then use grab to actually grab from that in the Indus essentially, so that is one thing and then the sort actually here we also went into more details here actually like I am in the last class is a new operator on the, the steamboat operator essentially like ,this operator is used as a morning first to do this sorting .

The regular sorting is an alpha sort essentially, so there is a lot of issues this again, so then you have to have a numeric sort and also again how do you do the, numeric sort ascending order and descending order how do you do all those things, we can achieve using the steamboat operator one thing to note here is in Perl the start function by using a quick sort algorithm, so essentially as you know the pick sort algorithm basically like this, starts with any one variable any one element in the array.
And then it tries to sort the elements that are greater than that element after the element and then the elements that are less than their element before the element, and then once it basically like once it anchors that particular element, then it passes these two lists and then again continues the same way the Pacific goes through, so typically we see that it has a complexity of n log n, I think like I mean you will probably like get into all these details once as a real computer science, at least to know here you want to make about this complexity is in this login area.
N login it is kind of it is very good considering this order if it is inspired that means that you actually were going through object in two times this, so I think I mean this is a much controlled behavior so that is what we have using this sort, sorting scallion efficient what I want to say

about that, then we also looked at the file open, file close testing the files with various switches, whether we can read it read the file write the file things like that.

And then we also like in the same place for the breakfast too busy these tests in tests can be done for the directories and then how do we open the file and the file handles things like that we talked about that, and then we also talked about the strict keyword essentially, so this is to make sure that Perl does not come omit any kind of mornings or any other errors that need encompass, because call itself is a tolerant program.

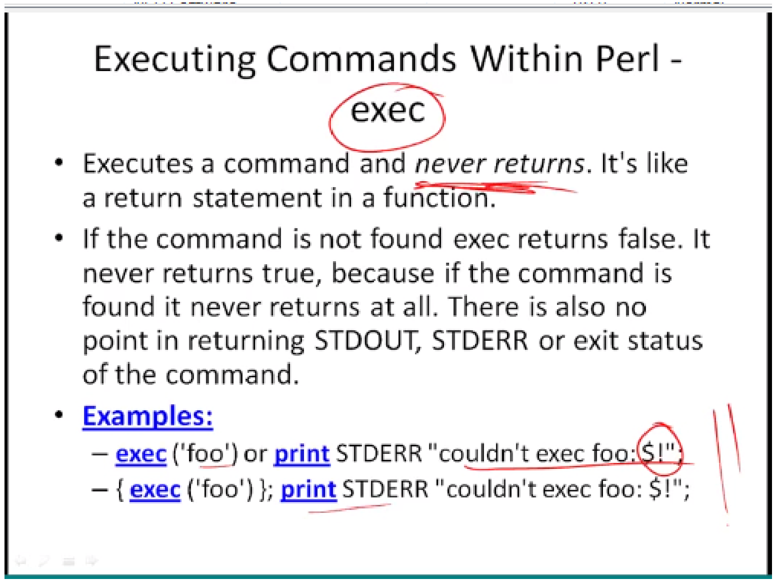
So you know even if you made a mistake but if it is syntactically and semantically correct it just let those things go through so I think structures one keyword that may be using basically, it is the users strict we will see like more and more program folks is using this, so initially like a minute you are doing like simple programs, you move on would not do this work once if you are doing any Perl programming development and strongly OHT use you strict in your programs.

So I think this is pretty much like what we saw, last the lecture I think I am in the event we like a lot more people in more and more of these areas, but today I want to actually introduce two topics today one is there are some leftover items that I want to talk about, one of them is essentially like how do you run a program within a Perl code, so there are some more new techniques for that and I will explain that, and then the big topic for today is some of the object oriented features of Perl which is actually prominent in the Perl 5.

Which is the fifth generation Perl and this is essentially what all the packages, essentially we will talk about probably how do we use packages, basically you have I mean one thing that made Perl versatile language that exists today, is the ability to actually export and import modules from different sources, so we will see like I mean motivation behind that and then also like how do we will do it in when we talk about the practice, so let me talk about the first ones essentially as I mentioned.

How do we run programs inside pull so say like I mean you want to do externally like a feeding poor directory or something like that inside of Perl how do you do that we will see?

(Refer Slide Time: 08:16)



Executing Commands Within Perl - exec

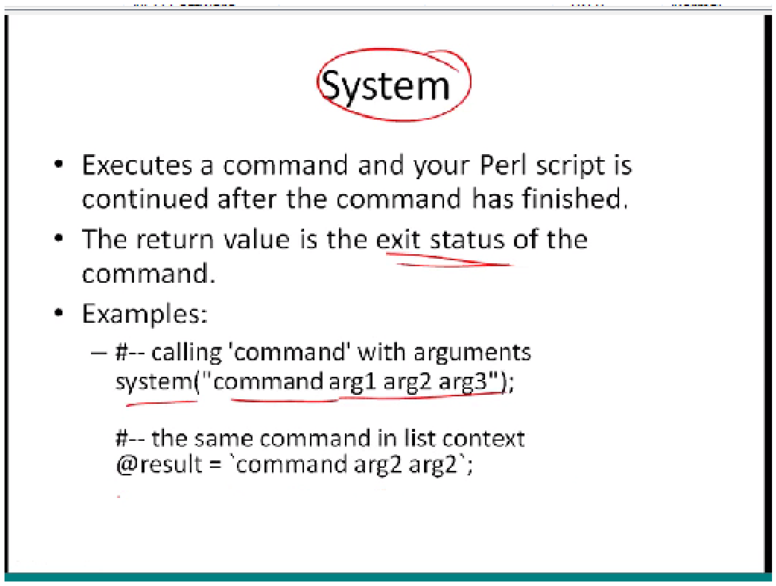
- Executes a command and *never returns*. It's like a return statement in a function.
- If the command is not found exec returns false. It never returns true, because if the command is found it never returns at all. There is also no point in returning STDOUT, STDERR or exit status of the command.
- **Examples:**
  - `exec ('foo') or print STDERR "couldn't exec foo: $!";`
  - `{ exec ('foo') }; print STDERR "couldn't exec foo: $!";`

There are actually three ways of executing commands within Perl the first one is what we call as exec or XX see on exact the exact commands essentially it executes with an add that you provide, but it never returns, so this is the keyword the never returns, it is like a written form statement in a function, if the command is not found the exact returns of false, it never returns true because the demand is found it never returns it on.

There is also no point in executing returning to burning standard out standard error or execute status obstinate so some examples essentially like we see you or print standard error code and execute foo with the dollar bang, I think I gave in this one you already know about this basically the to print the error message number, so gives the concentration error message number for this particular error, the other way to run exec is just enclose the whole statement did in a the collaborative.

And then essentially also if this returns zero then closing the next step and which is it will say that it is not able to execute so exact pretty much like finishes a program, so we would not see one day expect it basically it, it just comes out from that program so this is one method there are places there this gets used for we will talk about it talk about some of the things, but it is one of the useful commands.

(Refer Slide Time: 10:11)



System

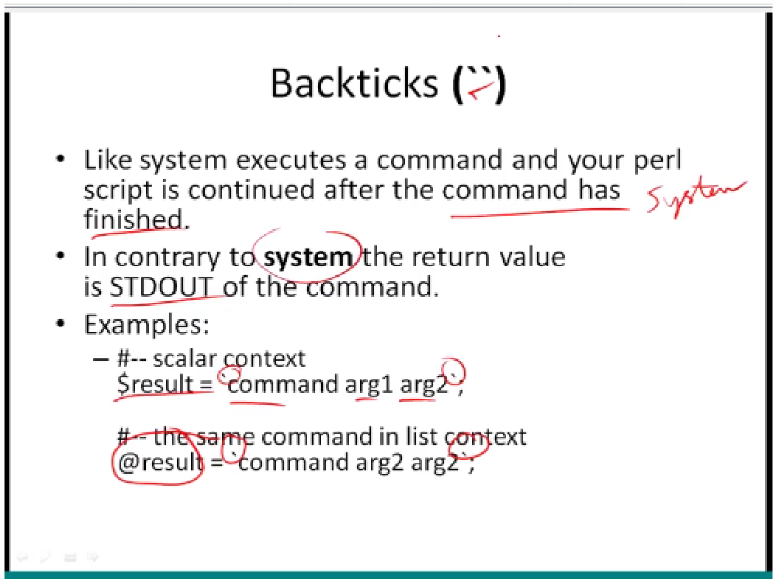
- Executes a command and your Perl script is continued after the command has finished.
- The return value is the exit status of the command.
- Examples:
  - #-- calling 'command' with arguments

```
system("command arg1 arg2 arg3");

#-- the same command in list context
@result = `command arg2 arg2`;
```

The second one is called system in system essentially like specify the command it executes the command and then, once the command finishes basically the first step is continue and then it moves to and the return, value for system command is the exit status of the function itself, so if you are doing a CD and CD you can successful and if it is not one the return value from system is also one, but if that CD was not possible and it returns a 0 then the system command also returns to 0.

Again it is mildly useful but it is not really useful, because as you know then we try to execute commands here we can execute command like CD for example is still to go and take the problem pointer to a different directory, but if something like I mean if you are using an external start around or a final welcome to find certain things how do you actually get those values and then process them further, that is simply not possible with the system comment, so here the example for using the system command is command with arguments and war within the double quotes inside the Perl script.
And then a better way is essentially like including command R 1 R 2 R 3 everything has its own strings so that it is actually better for the system feet and then one loop down under system command.
(Refer Slide Time: 11:55)



Backticks (``)

- Like system executes a command and your perl script is continued after the command has finished. System
- In contrary to **system** the return value is STDOUT of the command.
- Examples:
  - #-- scalar context
    $result = `command arg1 arg2`;

    #-- the same command in list context
    @result = `command arg2 arg2`;

Now the third option that you have is the war is called Backticks, and this is I think like this is very prevalent of one and then a lot of people use this, and essentially this is something that that is you, you will be using in your programs as well, the difference between the Backticks and the system is it is like system, where it executes the command and the poor script is continued after the command is finished, just like the year the system run, but where the difference comes in is essentially the new store line here.

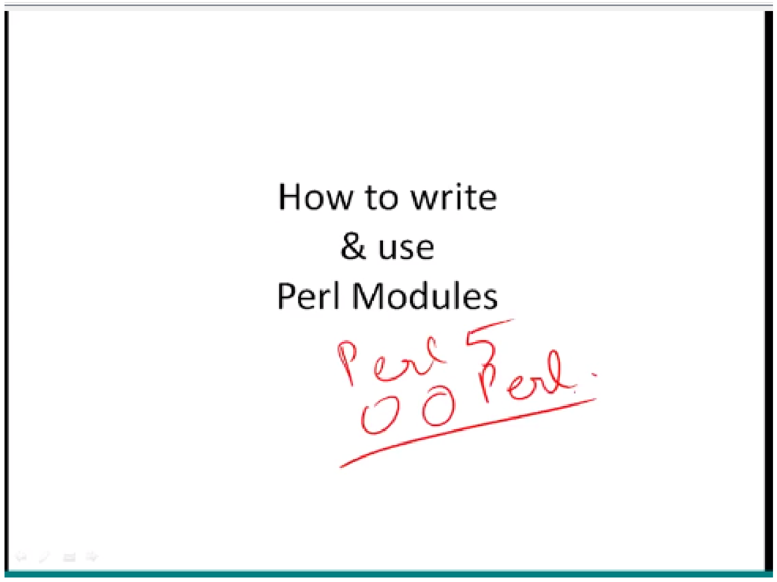
There contrary to the system the return value is standard out of the command so if you do the back pick it is not the status but whatever goes into the standard out becomes the return value, so here we can say like I mean in our context down dollar result is command or they are one or two and then you notice the back pick here, here so this command that gets executed and essentially will come along whatever the command says return value stored in a car scalar, into the dollar results.
And then if you want if you if the command actually returns a list then all you got to do is you can store it as an array with result the name and then execute the command in the Backticks mode and then whatever comes up is stored as an array this is what was the result of that command, so this is very useful when you are doing a sort external stuff for some for some reason and then actually using it and throwing it into the results so.
So again to recap on this one exec essentially like I mean it basically execute the command never return, so it is useful but it is not that useful the system essentially in again this is the next best thing their system actually returns be the exit status of the program of the command, rather than

actually any, any other thing base give it is an output of the command boom it does not produce that it only produces the exit status.

So here if you want to use this you may have to store the values in temporary file and then basically like again we back into the Perl into Perl and then you can use it, so like I mean a system command is typically used to generate like I mean large amount data, which will be used at some point of time later in the program, so that the system and then finally the Backtick commands the Backtick commands essentially just like the system they execute the command, and the Perl script continues after the execution is finished.

But the difference is that actually now you can pass on whatever the output of the particular command back to the program that initiates this call, so it can be passed to those things and then you can do for the processing of all, so the Backticks is like a very useful I will say like I mean more often more, more practical and then Backticks commands and the system command, so now that we covered this let us see the next one this.

(Refer Slide Time: 16:01)



How to write

& use

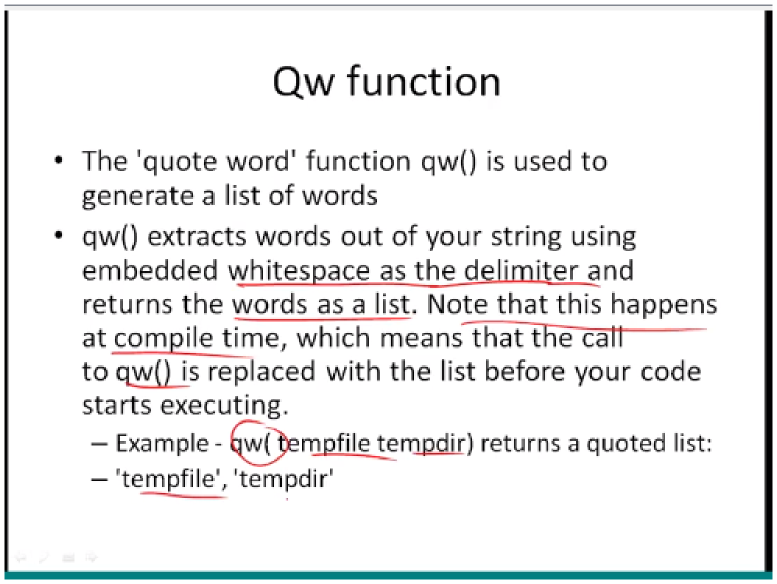
Perl Modules

Perl 5

OO Perl.

How to write and use Perl modules, again remember that this is basically this is Perl five which is some people call as object oriented Perl but I, I think I like the key thing here is though we are developing programs, how do we get it into another core program and how we can foster like a collaborative demand of there is called systems, in fact today lot of companies whether it is hardware or software are using Perl, as the vehicle I mean use doing Perl programming, and the way that they do it is collectively they are doing it, and then more in order to make the system work we need these modules the concept of modules and how do we actually write and use them.

(Refer Slide Time: 17:03)



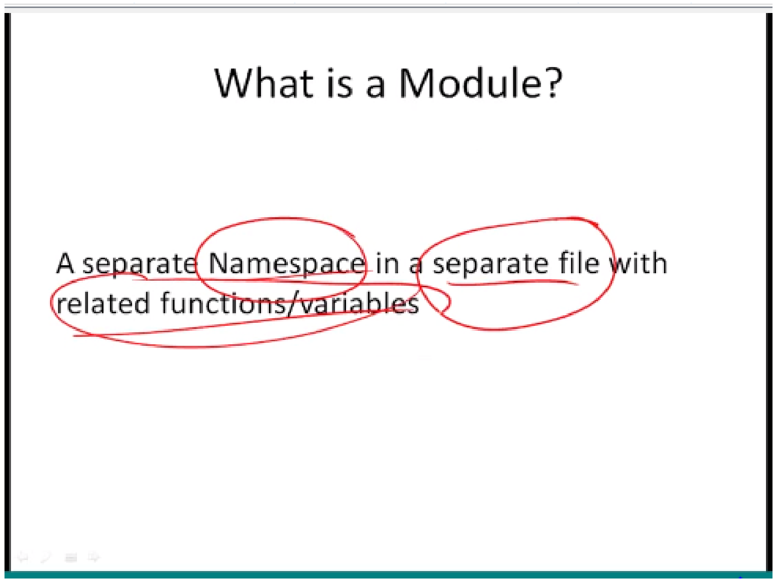
Qw function

- The 'quote word' function qw() is used to generate a list of words
- qw() extracts words out of your string using embedded whitespace as the delimiter and returns the words as a list. Note that this happens at compile time, which means that the call to qw() is replaced with the list before your code starts executing.
  - Example - qw(tempfile tempdir) returns a quoted list:
  - 'tempfile', 'tempdir'

So just before that I wanted to introduce you one small function which is called the QW function it is called code word, it is used to generate lists of words the QW actually extract words out of the string using embedded whitespace as the delimiter, so this is important you can actually like to change this delimiter to any other delimiters that you can read up on that term of this command and then it returns words as a list essentially.

So and then it also one thing that note to note one thing to know before it does not it happens at compile time so between that a call to the WQ is replaced with the list itself before the code starts executing, so here one simple example is code word M file tender it returns ten file and tender as two different values.

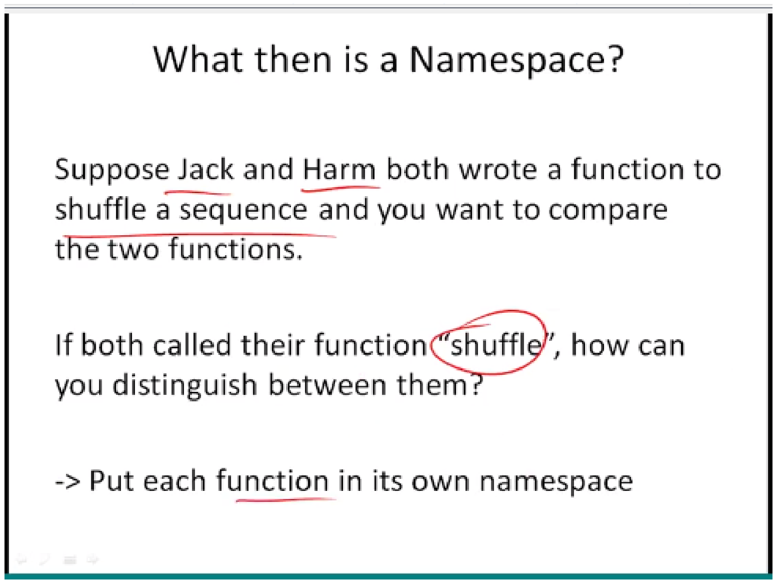
(Refer Slide Time: 18:12)



What is a Module?

A separate Namespace in a separate file with related functions/variables

So now so this is important in the next section when we talk about modules, because ligament most of modules we write it as functions and basically you for code word function a lot, so let us look at first of all what is a model? so the module is essentially a separate namespace in a separate file, with related functions and variables, so there are three things here one is a separate namespace, another one is separate file and then there are related functions and variable
So how do we do that we will see basically?
(Refer Slide Time: 19:06)



What then is a Namespace?

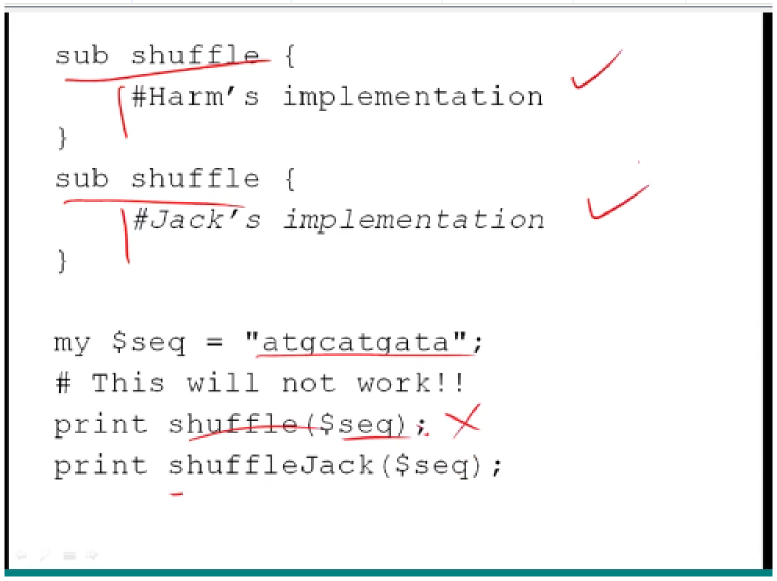
Suppose Jack and Harm both wrote a function to shuffle a sequence and you want to compare the two functions.

If both called their function "shuffle", how can you distinguish between them?

-> Put each function in its own namespace

So what is a namespace the namespace provides a certain context to the functional variable so essentially you can think of namespace as any kind of means itself so my hammer, Marys lamb, Gulliver's Travels Dutch cheese, French bread, your car, so you can think of this as like this these

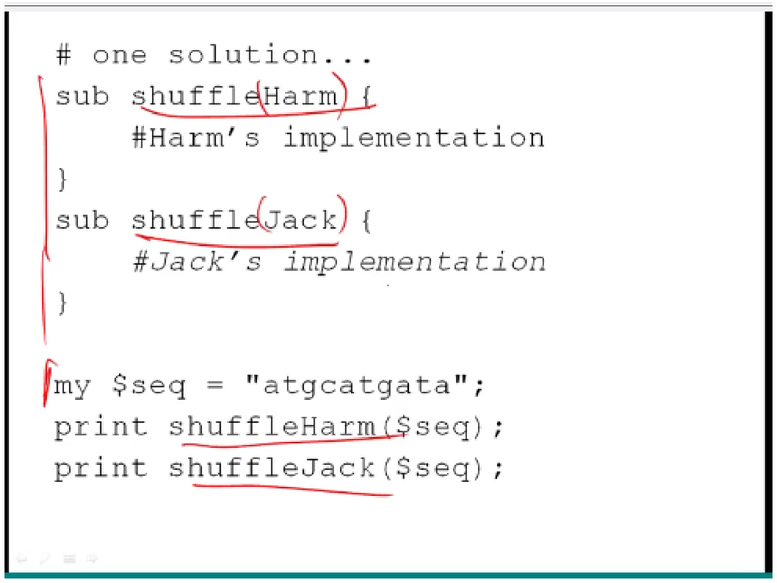
pronouns here because you are wrong various possessive here with you these are all the namespace that now that is basically like I mean that provides the context to the B function.
So if go in short essentially like and say like two people writing some function you can say like A is a sort or B Is a sort, so we will see more about the that context essentially, so here one example I have like two engineers jack and harm, they both wrote a function to shuffle a sequence, and you want to compare the two functions they both called this function as shuffle now how do you distinguish between them, so the way to distinguish between branding put the function in its own namespace.
(Refer Slide Time: 20:47)



```
sub shuffle {
    #Harm's implementation
}
sub shuffle {
    #Jack's implementation
}

my $seq = "atgcatgata";
# This will not work!!
print shuffle($seq);
print shuffleJack($seq);
```

So now we call the first one as jack or harm so there is a sub shuffle and this is Harms implementation, and then the sub shuttle which is Jack's implementation, if you if you put basically they like I mean this is the sequence that we need to do, and then if you say like I am in shuffle SEQ this will not work because it does not know like I am going okay bitch function to use again same thing it does not work so we need to distinguish between this and this how do we do that?
(Refer Slide Time: 21:26)



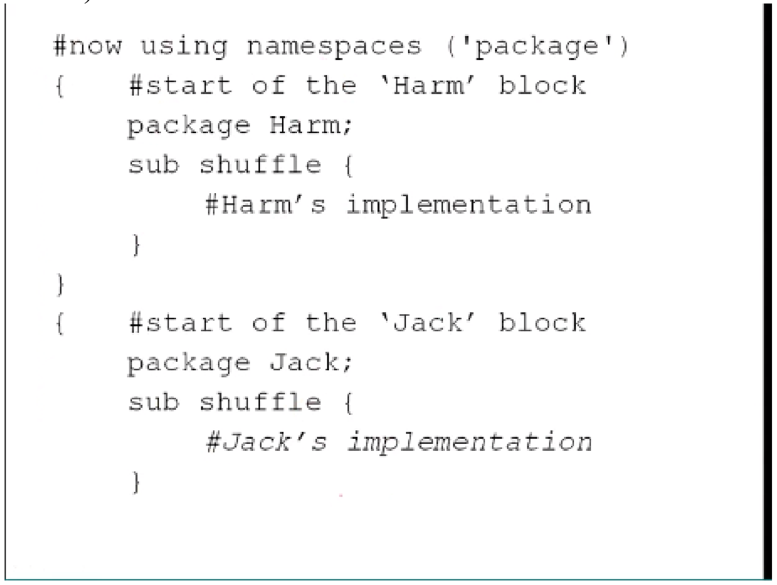
```
# one solution...
sub shuffleHarm {
    #Harm's implementation
}
sub shuffleJack {
    #Jack's implementation
}

my $seq = "atgcatgata";
print shuffleHarm($seq);
print shuffleJack($seq);
```

So again it is all the same basically like I am in the top one and then what the sequence that we want to shuffle is also the same only thing is right now we name the functions as shuffle harm and shuffle Jack, so you notice here that actually the mobile names are different, we add this additional element to it yeah this is well and good but what happens basically, when we you know we have to say somebody has to tell the others essentially like a more like if say like jack is the one that is the del of the shuffle second time.

Now Jack needs to at least how needs to tell Jack that hey Jack I am using the name shuffle harm, so you should use shuffle Jack or everyone should have these kind of mean arbitrary, this is kind of arbitrary infancy basically.

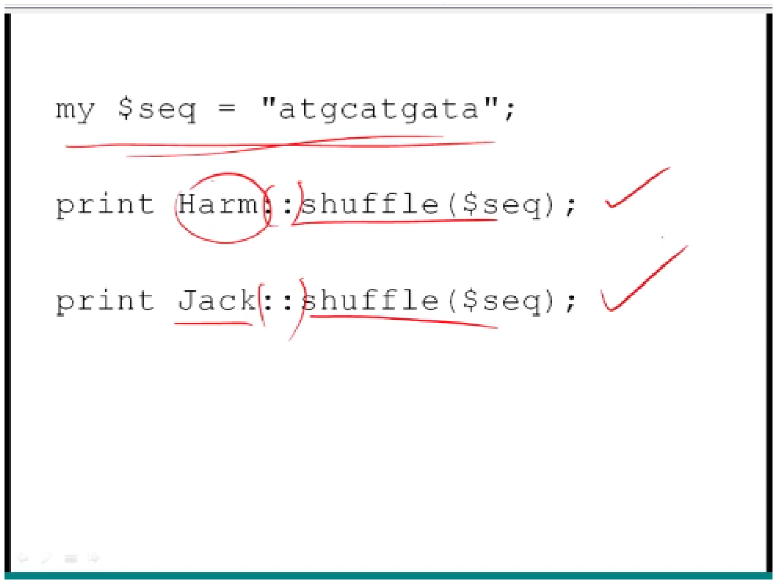
(Refer Slide Time: 22:26)



```
#now using namespaces ('package')
{    #start of the 'Harm' block
     package Harm;
     sub shuffle {
          #Harm's implementation
     }
}
{    #start of the 'Jack' block
     package Jack;
     sub shuffle {
          #Jack's implementation
     }
```

So to avoid this essentially we use the package ascension, so what we say is we will call we will have the packages armed and then inside, that we will still use the same name which is shuffle, and then that is followed by the harmful presentation and then we will have Jack block and with the package called Jack, and then it also has the same thing the as a function which is the shuffle and then this is the Jack limitation so now we define these two things but how do we use them basically .

(Refer Slide Time: 23:06)

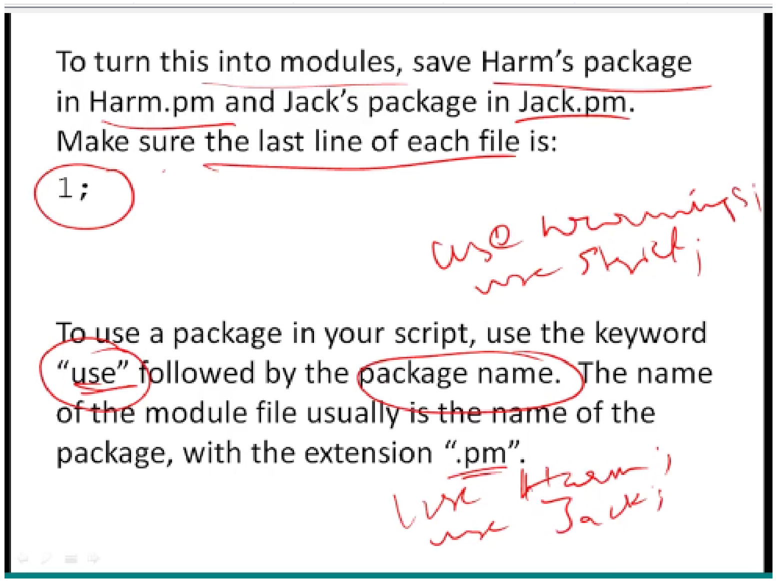


```
my $seq = "atgcatgata";

print Harm::shuffle($seq);

print Jack::shuffle($seq);
```

So the way to use it is like again plugging in the sequence is the same here we use harm as the package followed by this column, column and then the function name here again the same thing for the deck it is basically Jack :, : and then the function these two will work basically and then you will get like two different answers based on how each one implemented these two functions.
(Refer Slide Time: 23:35)


To turn this into modules, save Harm's package in Harm.pm and Jack's package in Jack.pm. Make sure the last line of each file is:

```
1;
```

use warnings;
use strict;

To use a package in your script, use the keyword "use" followed by the package name. The name of the module file usually is the name of the package, with the extension ".pm".

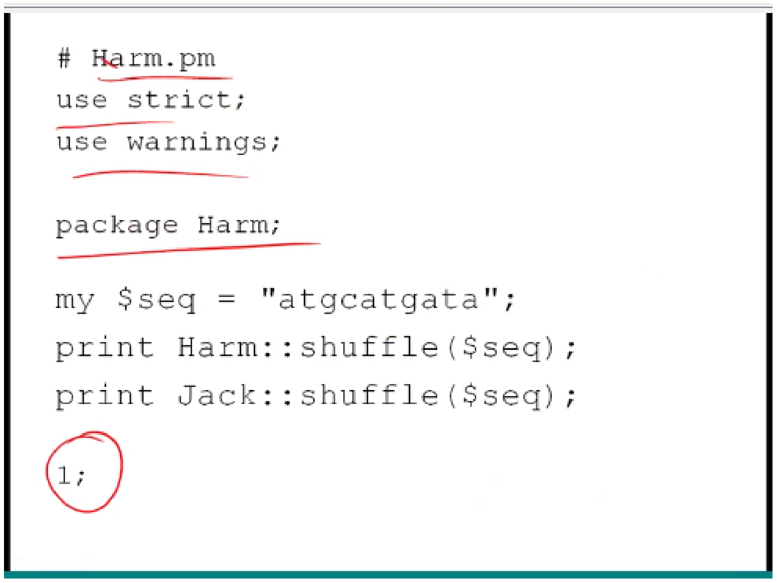
use Harm;
use Jack;

So this is the package that we can see basically like and so the package is essentially like m is something that is form, used inside the same program so essentially like harm sequence, and now if you want to compare these two packages basically you need to put these two packages copy from harm say go and put it into your program copy from Jack's on the shuffle and then put it in the program and then we need to write the basically like then we can write the whole program to work with that, in its time kind of cumbersome basically like having for one thing that we want to do is we do not want to even depend on harm or Jack in order to develop our program so.

So for that we need we need to turn these packages into modules so to turn this into modules we say harms package as harm dot p.m. and Jack's packages Jack got p.m. but one small change that we make we put this one semi, I mean one semicolon as the last line of equal so then so then they are packages, so then I Harm can check this module I mean these packages are converted into modules now the harm can check in this module and then Jack and set separately checking this module, one then the end somewhere basically.

So in order to use the these modules in our script basically what we need to do is we need to use the keyword use followed by the package, name I think I given you already saw the use of form over use essentially warning you strict, these are all the various ways of even using these and then the name of the module pile usually the name of the package, basically with the extension P and so essentially like I mean so we just say the use arm and then use Jack in this case.

(Refer Slide Time: 26:05)



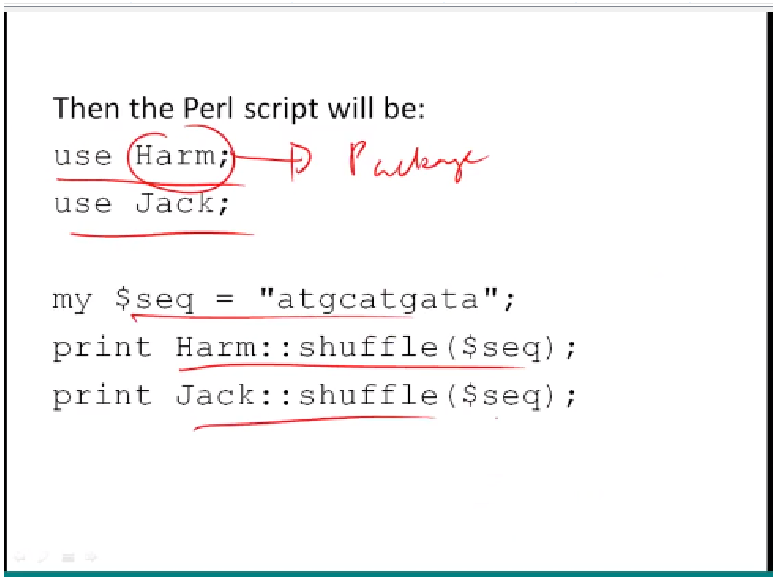
```
# Harm.pm
use strict;
use warnings;

package Harm;

my $seq = "atgcatgata";
print Harm::shuffle($seq);
print Jack::shuffle($seq);

1;
```

So here we let us see like how we do this basically so the harm dot p.m. itself the way that we will write is we use the same you straight and warning, we call it as a package harm, and then we write the shuffle then you can see that the one semicolon is there.
(Refer Slide Time: 26:27)

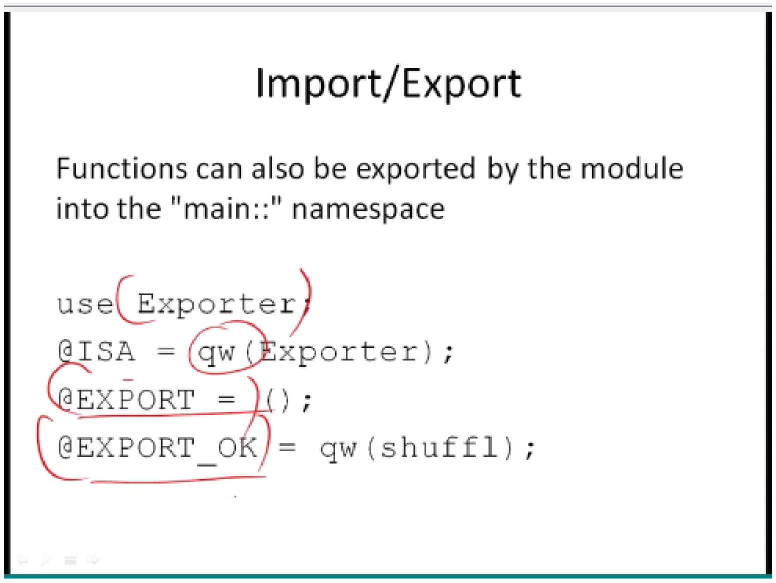


Then the Perl script will be:

```
use Harm;  --> Package
use Jack;

my $seq = "atgcatgata";
print Harm::shuffle($seq);
print Jack::shuffle($seq);
```

Now in the Perl script the way that we can do it is basically use arm and use Jack these are all the package names, essentially because the five itself is stored with arm dot p.m. so this is the packages, we kind of imported this packages from these locations, now we give the same thing and then more the same exact command that we saw, and then this should produce right expected.
(Refer Slide Time: 27:00)



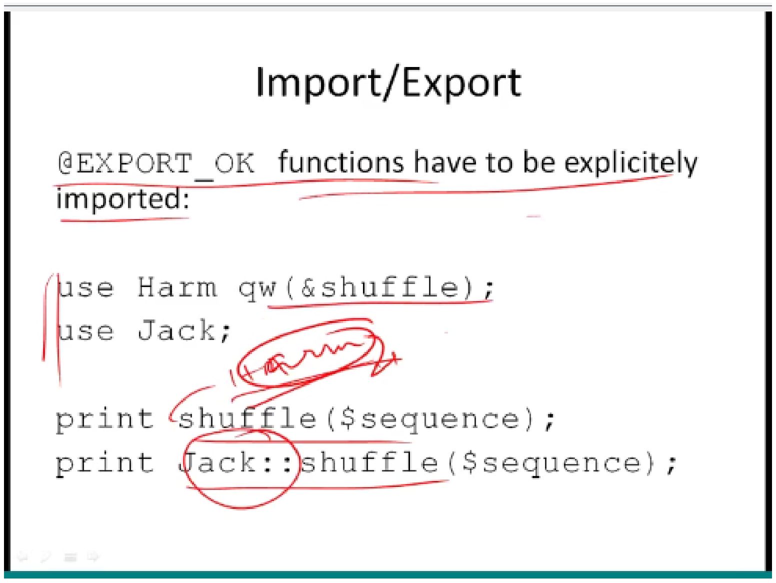
Import/Export

Functions can also be exported by the module into the "main::" namespace

```
use Exporter;
@ISA = qw(Exporter);
@EXPORT = ();
@EXPORT_OK = qw(shuffl);
```

So we can also export functions by the modules into the main namespace so here we basically like I mean we use what is called an exporter? and then essentially like I specify, you know using the codeword the exporter into this array and then export is nothing and basically like this is another variable, these are the two preset variables that we will see what they represent in the next sections.

(Refer Slide Time: 27:43)



Import/Export

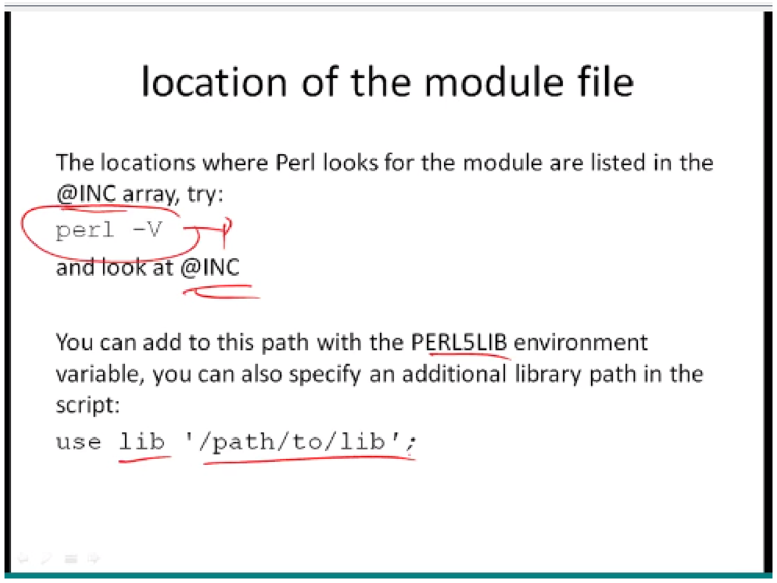
@EXPORT_OK functions have to be explicitely imported:

```
use Harm qw(&shuffle);
use Jack;

print shuffle($sequence);
print Jack::shuffle($sequence);
```

So the export okay functions have to be explicitly imported basically, so essentially like that until now what this means is like I mean once we add this in, in front of our programs essentially that becomes the modules of the functions that can be exported, so actually there is an e missing here and say so, so we say basically like for shuffle function the export is okay for that particular

function, and then that is our so that is what we mean here basically functions have to be explicitely imported.
So here essentially like everything the same thing basically and then we say like shuffle sequence and then we will say like Jack shuffle sequence, so see here that actually like we do not need to specify harm anymore here, that the reason is basically like I mean go you cannot we do not need to put the arm huh because this is something that is what is being exported from this function, so for the harm actually like we are using this basically like it is directly exported into this main namespace.
So you do not have to specifically assign a jack namespace of that particular shuffle so wherever you call shuffle it is assumed that it is basically coming from harm okay.
(Refer Slide Time: 29:36)



location of the module file

The locations where Perl looks for the module are listed in the @INC array, try:

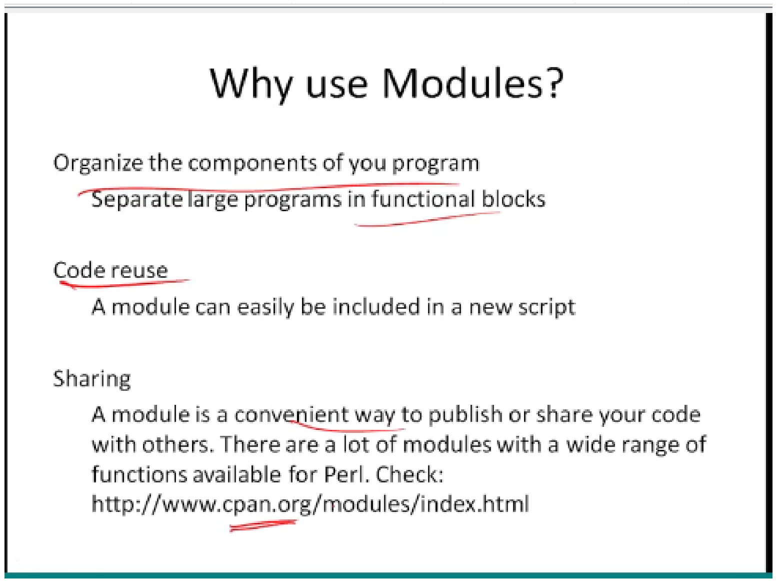
```
perl -V
```

and look at @INC

You can add to this path with the PERL5LIB environment variable, you can also specify an additional library path in the script:

```
use lib '/path/to/lib';
```

You so the location of the full module is the other thing basically that we want to talk about essentially, it is the essentially like a minute so specified in this add INC array, so once you have the Perl minus V which is the verbal open then you can actually like print this @ INC, where the wrapper looks for modules in your path, so and then you can also add this cat with the Perl 5 live in random variable which is used life and then path to the libraries.
(Refer Slide Time: 30:31)



Why use Modules?

Organize the components of you program

Separate large programs in functional blocks

Code reuse

A module can easily be included in a new script

Sharing

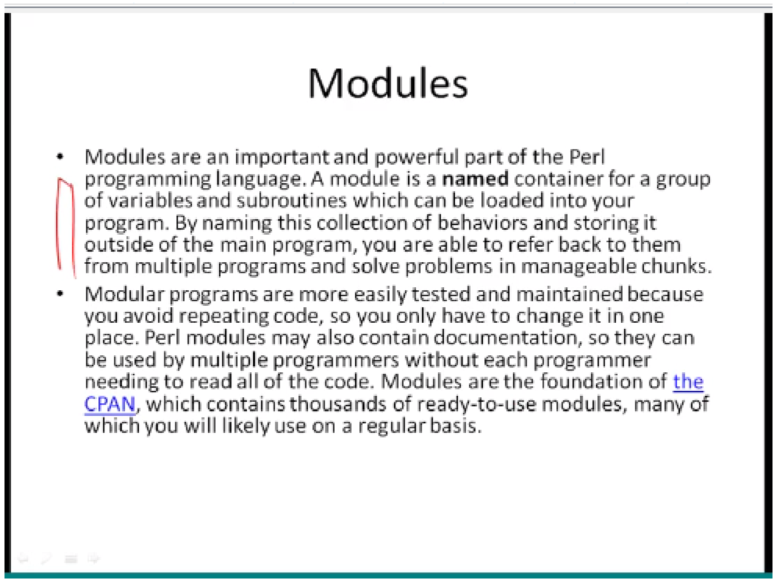
A module is a convenient way to publish or share your code with others. There are a lot of modules with a wide range of functions available for Perl. Check: http://www.cpan.org/modules/index.html

So now the question is like why we use modules? so as I mentioned basically like there are there is, one reason which is the sharing and core development of various programs, but let us look at this essentially so the number one reason is essentially to organize the contents of the program, so what we can do is when we have make large programs we can develop some of the components as a bottoms up approach where we write the essential functions.

And then those functions can be stored in various file and essentially like in use and export mechanism or modules to export that into the main, main program, so this is one reason the second reason is the code reuse there we can use the same once we write a particular sort function and if we know that that is the most efficient way of going home.

We can use that over and over again in other scripts and we do not have developed this at all and you use the code from one to the other and then the third main motivation for using the modules, is it is a convenient way to publish or share the code with others so there are a lot of modules and we are wide range of functions are available over, so this is one website that I want you to go and check for the various modules in Perl.

(Refer Slide Time: 32:11)



Modules

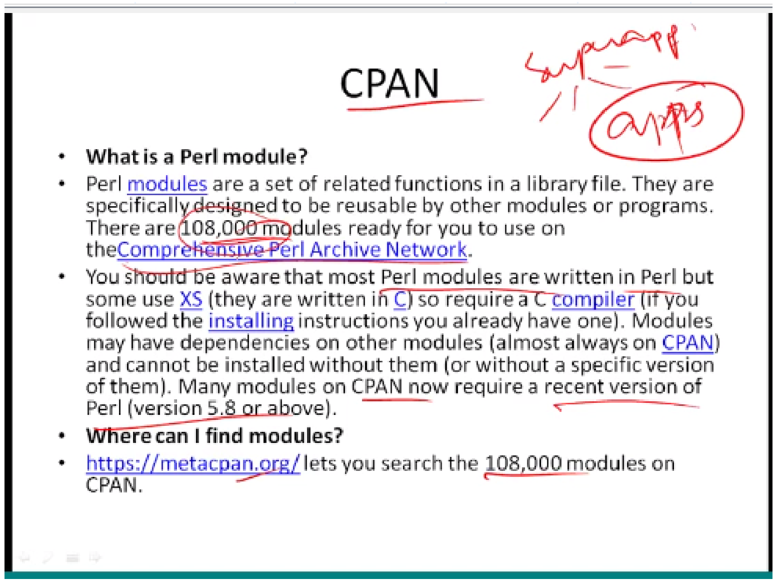
- Modules are an important and powerful part of the Perl programming language. A module is a **named** container for a group of variables and subroutines which can be loaded into your program. By naming this collection of behaviors and storing it outside of the main program, you are able to refer back to them from multiple programs and solve problems in manageable chunks.
- Modular programs are more easily tested and maintained because you avoid repeating code, so you only have to change it in one place. Perl modules may also contain documentation, so they can be used by multiple programmers without each programmer needing to read all of the code. Modules are the foundation of the CPAN, which contains thousands of ready-to-use modules, many of which you will likely use on a regular basis.

Some more details on the modules, modules are an important and powerful part of programming language, the module is a named container for a group of variables and subroutines which can be loaded into your program, by naming this collection of behaviors and storing it outside the main program you are able to refer back to them for from multiple programs, and solve from solve from problems in manageable chunks, as I said this is essentially like promoting the three use within.

The modular programs are easily tested and maintained because you can actually test just the modules and maintain them and you can avoid repeating the code, and then the, the mainly main reason why we can avoid repeating the code is also because, we need to change any kind of bugs in only one place and then all the others automatically gets changed and then the Perl modules can may also contain documentation, they can be used by multiple program without each programmer needing to read all the code.

So essentially it promotes multi user people to work on programs simultaneously and it also promotes developing a big environment, which does several things collection of tools and techniques basically for solving a given problem, and they can be generated by that and then the modules or foundation of the Cpan it contains thousands of ready to use models, many of which you will likely use on a regular basis so Perl insulation the Cpan model also comes and then you can use them for your programming purposes so I want you to take a look at this Cpan the website is given will be this one so now let us look at what is Cpan?.

(Refer Slide Time: 34:23)



CPAN

Superapp
apps

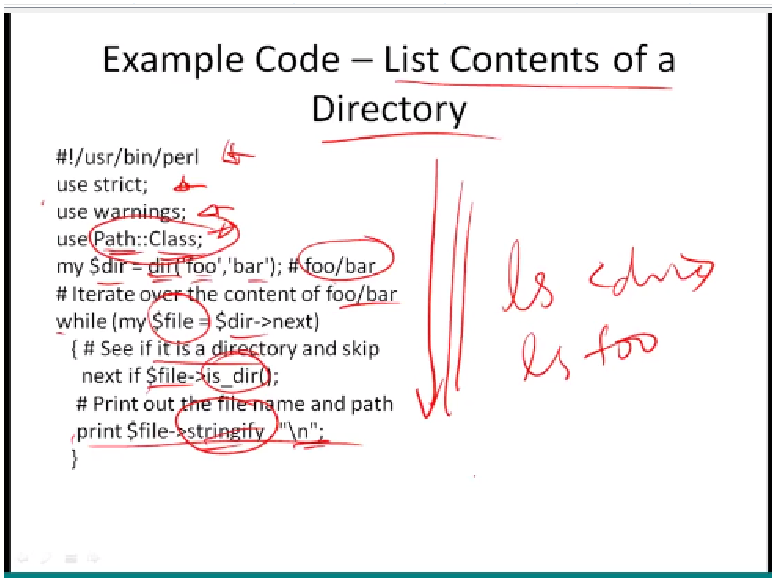
- **What is a Perl module?**
- Perl modules are a set of related functions in a library file. They are specifically designed to be reusable by other modules or programs. There are 108,000 modules ready for you to use on the Comprehensive Perl Archive Network.
- You should be aware that most Perl modules are written in Perl but some use XS (they are written in C) so require a C compiler (if you followed the installing instructions you already have one). Modules may have dependencies on other modules (almost always on CPAN) and cannot be installed without them (or without a specific version of them). Many modules on CPAN now require a recent version of Perl (version 5.8 or above).
- **Where can I find modules?**
- https://metacpan.org/ lets you search the 108,000 modules on CPAN.

So before that like I mean I mentioned Perl modules they are set of related functions and in the library file they are specifically designed to be reusable by the other body program currently there are about 108,000 modules but readily available for your use, so what is Cpan? Cpan is this particular thing basically it is a comprehensive Perl Archive network I want you to go and like look for it because the Cpan function Zip and modules are widely used and pretty much it comes with every pole installation your installation you have.

And one thing to also notice most Perl modules are written in Perl, but come use some excess and they are they require Cpan, modules may also have dependencies on other modules from almost always on Cpan and cannot be installed without them, so or we need without a specific version with learning form, so and then the Cpan itself in the modules in Cpan they require which is 5.8, so this is another location is not anemic Meta Cpan Meta Cpan and the core.

Where you can search the 108 thousand modules I mentioned on see that think of these modules as now your apps only thing is like these apps are actually connected to each other, so that anyone can write a super app by just meshing these apps into that Supra so essentially looking at the end then so this clip and you can think of it as though iTunes for goal and that I have 100 ink modules the only thousand models three for you to develop the new apps.

(Refer Slide Time: 36:43)



Example Code – List Contents of a Directory

```
#!/usr/bin/perl
use strict;
use warnings;
use Path::Class;
my $dir = dir('foo','bar'); # foo/bar
# Iterate over the content of foo/bar
while (my $file = $dir->next)
  { # See if it is a directory and skip
    next if $file->is_dir();
   # Print out the file name and path
   print $file->stringify . "\n";
  }
```

So I think I mean this gives you like a good overview essentially now we will look at the one code segment basically, we have one today here essentially Lego trying to list the contents of a directory, so the first statement you know basically like a Google pearl use straight use warnings, so here we are actually now exporting and module essentially called path and then basically net which contains the package class.

So here essentially ligament defining the directory basically, so full bar essentially that goes by full slash bar you can have like many much minimally more and then they are all like we are opening it at a directory were calling this direction, and now we want to iterate over the contents of the full slash bar basically we are saying yeah this particular scalar variable is taller the next essentially the next of file, first we see basically like if it is a directly then you skip basically, so we do a is the test essentially on this particular file and essentially we break the loop and then go to the next one.

If it is if the distance becomes true if the test is there this is not true then what we do is we basically son we print that particular line into the as a string and with the newline character, so the string if I is another function that is specified in the this one basically which essentially like takes this content of the file which is the full slash bars first, first one and then it did it creates a string out of the whole characters and then it prints out that particular string.

So I think I gave this is a very simple program as you can see like a memo so this is you can think of this as the LS command essentially so LS and directory game you get all this foo you get this particular welcome and you can see then one two three four five six seven eight nine would within less than ten lines you can actually make this command work, so this is how powerful you can see the programs and then how powerful it can get.

I think let me just recap this whole session basically for you, so we started by talking about how to execute commands inside of pool using the three main methods one is the exec which is not very useful because they execute the command and native comes off ,so you can put it as a like last command this would not send an email or something you can make secure the email command or something, then we also looked at system the system come a system specification essentially it executes the command.
But it continues from where it is for where it locked off and until the Perl script itself is finished, but the return code is actually the exit status of the command it is nothing it is not the output of the command but just a good status of the command is true or false, so you have to use it with a pinch of salt here, basically if you do not want any output coming from the particular command to be further process, then you can use a system called okay.
And then finally the Backtick essentially is essentially like a Backtick is the way that we can actually capture the results or capture the output of this particular command into various variables, and this can be used within the pulse of picture, so there are a couple of examples that we quoted here think of this disagreement and have it in a scalar context or and list of scalar context, so these things you can do then the next one that you talked about was how to write and you pull modules.
This is one of the advanced topics essentially language is also normal object-oriented Perl or Perl 5, one of the function that we talked about which is key to all the things is the two QW or the code word and this is used to extract words out of the string, and using the embedded white space and the delimiter and it returns the words as well, so a simple command is shown here something, and so then we went into the packages and the modules essentially the modules essentially is the separate names facing accepted file with related functions and available.
So we wanted to understand what the namespaces and the namespace is something that provides context to the functions and variables, so if the distinguish between is the start function B start function, and then we explained the namespace with an example essentially like where we have two, two individuals Jack and harm who trying to develop the same program or shuffle and how to be distinguished with them between one.

So if you just put the function name with those two implementation as four functions that does not work, so one way to do the come up with a solution is basically embed the name into the function for shuffle harm and shuffle Jack, we can do this what this is again cumbersome and they she can convey all the it is, it is a very subjective so you cannot really implement this, so in

order to find the happy medium what we said was basically in recent concept of a package there we start the harm block with what is called the package specification will package keyword and then harm.
And then we write as searchable and then for the Jack's basically that also starting with at a Jack and then if shuffle and then once we use it giving this a harm on Collin and Jack Collin, so this is this is the way how we can you know import those packages or programmable, so but the package can be turned into modules, by just using the one followed by a semicolon at the end of the program, or the last line of the program should masses and then if you want to use the package in the script we use the keyword use followed by the package name.
And then we do not have to specify the extension p.m. yet so here basically what we use it we do the define the hundred p.m. with one semicolon, and then we use it inside the thing they should be this harm give like this keyword use harm to make sure that this package is exported, so another way to do it is it is any leg anymore you can actually export them to so this is a simple export essentially we will use Harm and Jack but you can also give it a main context essentially or encoding to the main works namespace.
So that we can avoid every time typing this some name followed by column will have name the main come in the mains context, to do that we use an exporter so we specify use exporter, and then we basically like specifically these commands, and essentially the we say that export okay is shuffle, and this actually gives the, the command actually let them in how to do, so we say basically use harm and then again the wave rules as the subroutine essentially like for once.
We say that this is a subroutine and with this package that one we don't have to specify the harm anymore basically because it is it is also matches the expo coping and then we continue with that process, then we talked about the locations essentially like on how to look at the libraries, and then we talked about the why we use models again the three reasons that we outlined are one is to organize the components of your own model, your own programs then to promote the code reuse where you develop once and share it and then use it in terms.

And then sharing the code with others essentially this is the key thing while how the collaborative would not be one a moment can be developed and these are all like the functions are all available in Cpan dot org which is an archetype essentially, the Cpan itself stands for the comprehensive pull our team network and currently it has about 100,000 modules and one thing to note is like not all the modules for within in Perl some of them are written in the excess from they also require a compiler.

And then we also want to note that the modules themselves basically all other modules and so essentially like we need to install the whole thing would not get all the models, and then the recent version of Perl is that is another thing that we need for accessing Cpan taken modules, so the person Perl version has to be five point eight or ever okay, so I think and then we event into this one example where we just wanted to list the contents of a directory it does not check other things or does not have any other things basically this is kind of leggings and LS – L.

So it does not check whether there are any like dot files inside that or any other further checks on one only like directly fit avoid and if only prints the all the regular files, so even L LS minus L we prints all the directories and everything this only prints out there is no files, so I think this is pretty much it for Perl from the next class onwards.

We will start the tickle programming language I hope this is this was enjoyable have some quizzes coming up the PA will organize some of the quizzes and tests for Perl so thank you very much okay bye.