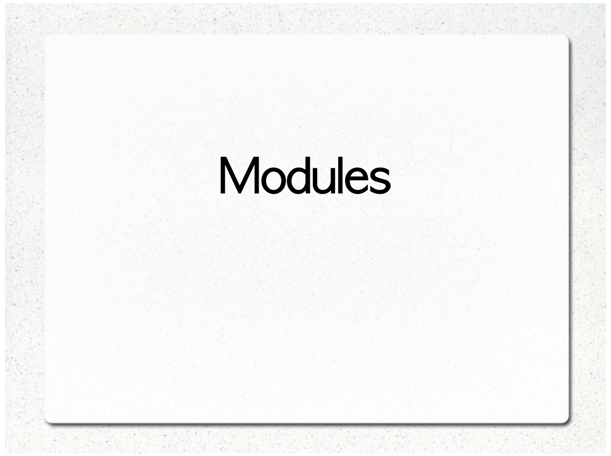
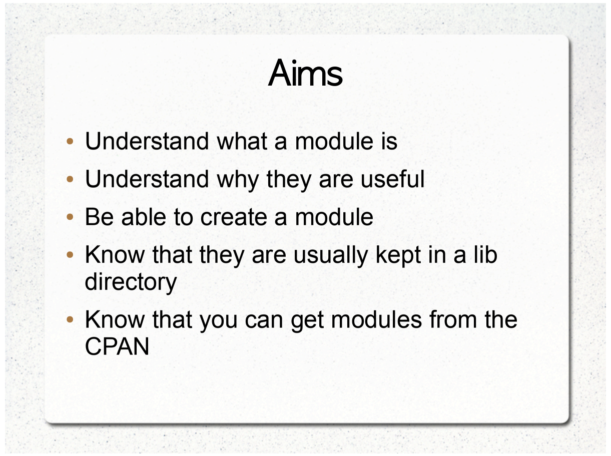
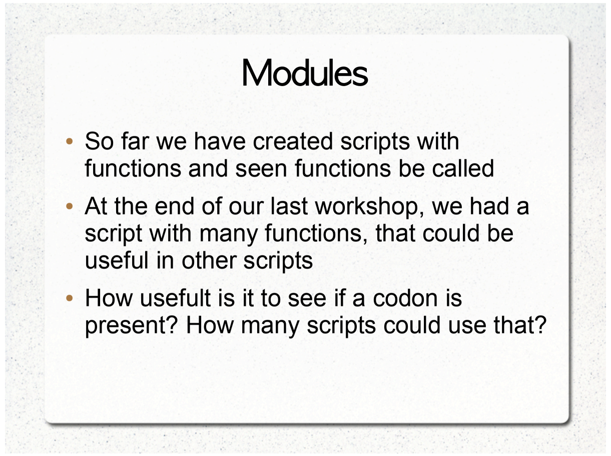


Modules



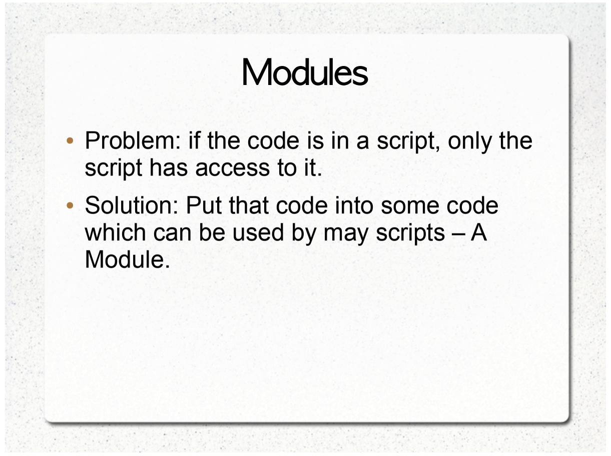
Aims

- Understand what a module is
- Understand why they are useful
- Be able to create a module
- Know that they are usually kept in a lib directory
- Know that you can get modules from the CPAN



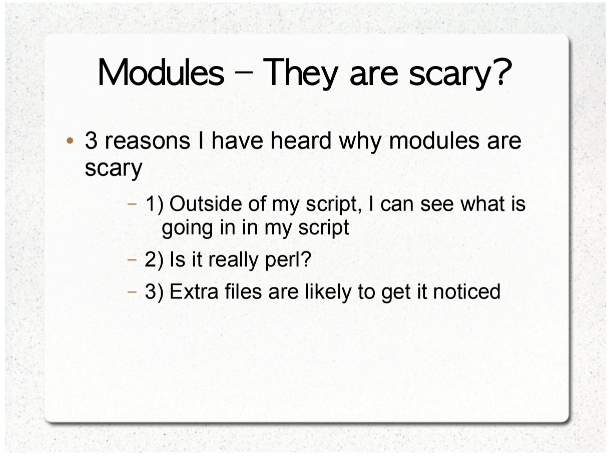
Modules

- So far we have created scripts with functions and seen functions be called
- At the end of our last workshop, we had a script with many functions, that could be useful in other scripts
- How usefult is it to see if a codon is present? How many scripts could use that?



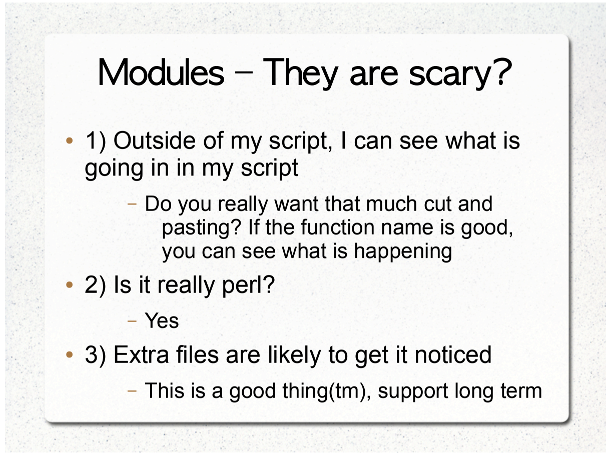
Modules

- Problem: if the code is in a script, only the script has access to it.
- Solution: Put that code into some code which can be used by may scripts – A Module.



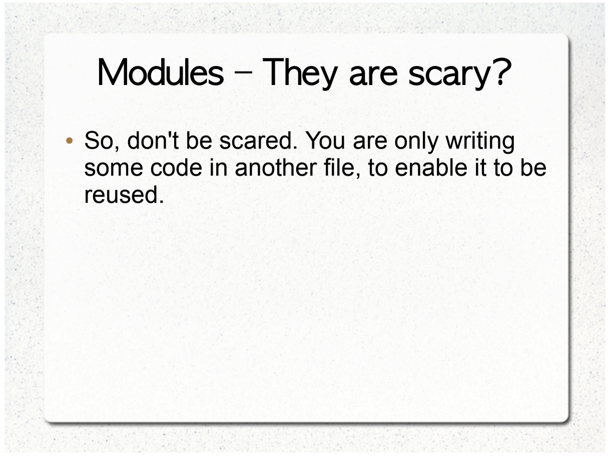
Modules – They are scary?

- 3 reasons I have heard why modules are scary
  - 1) Outside of my script, I can see what is going in in my script
  - 2) Is it really perl?
  - 3) Extra files are likely to get it noticed



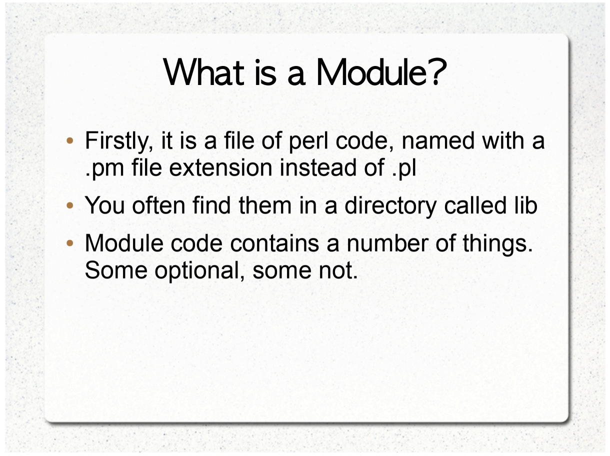
Modules – They are scary?

- 1) Outside of my script, I can see what is going in in my script
  - Do you really want that much cut and pasting? If the function name is good, you can see what is happening
- 2) Is it really perl?
  - Yes
- 3) Extra files are likely to get it noticed
  - This is a good thing(tm), support long term



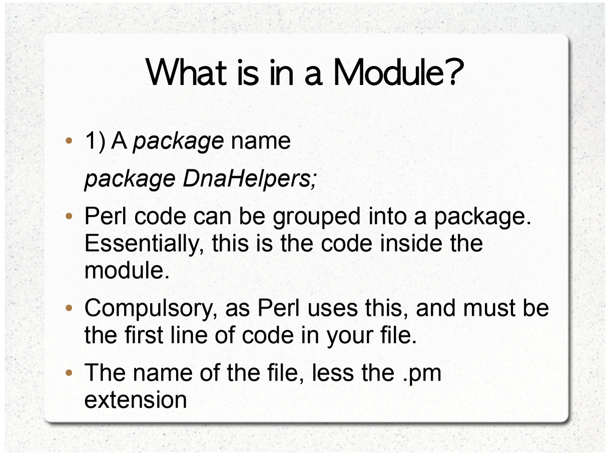
Modules – They are scary?

- So, don't be scared. You are only writing some code in another file, to enable it to be reused.



What is a Module?

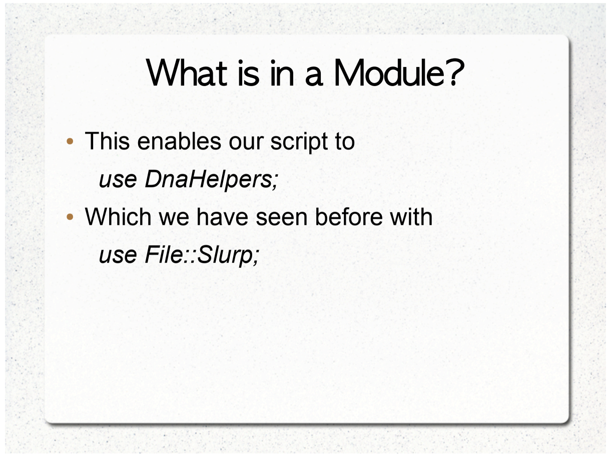
- Firstly, it is a file of perl code, named with a .pm file extension instead of .pl
- You often find them in a directory called lib
- Module code contains a number of things. Some optional, some not.



What is in a Module?

- 1) A *package* name

  package DnaHelpers;
- Perl code can be grouped into a package. Essentially, this is the code inside the module.
- Compulsory, as Perl uses this, and must be the first line of code in your file.
- The name of the file, less the .pm extension

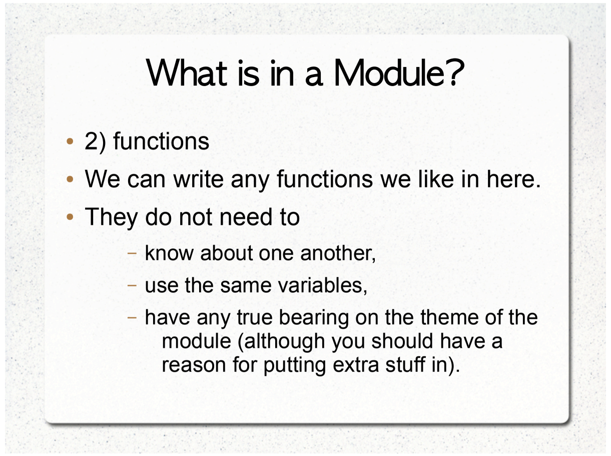


What is in a Module?

- This enables our script to

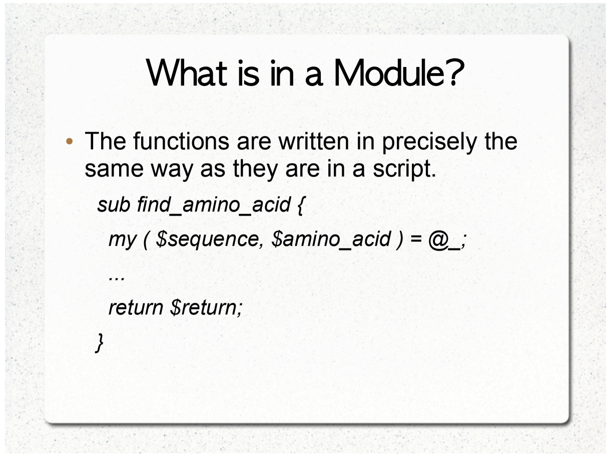
  use DnaHelpers;
- Which we have seen before with

  use File::Slurp;



What is in a Module?

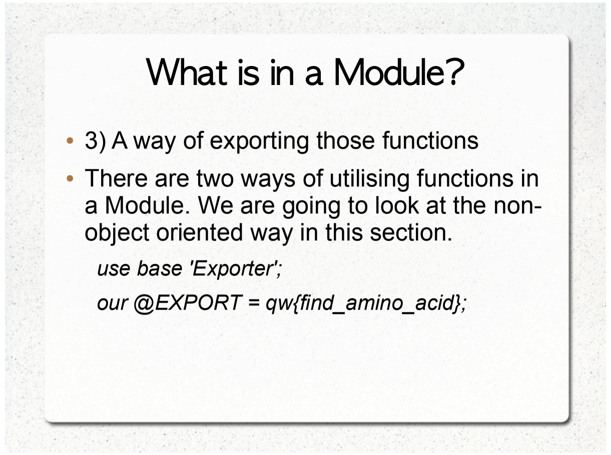
- 2) functions
- We can write any functions we like in here.
- They do not need to
  - know about one another,
  - use the same variables,
  - have any true bearing on the theme of the module (although you should have a reason for putting extra stuff in).



What is in a Module?

- The functions are written in precisely the same way as they are in a script.

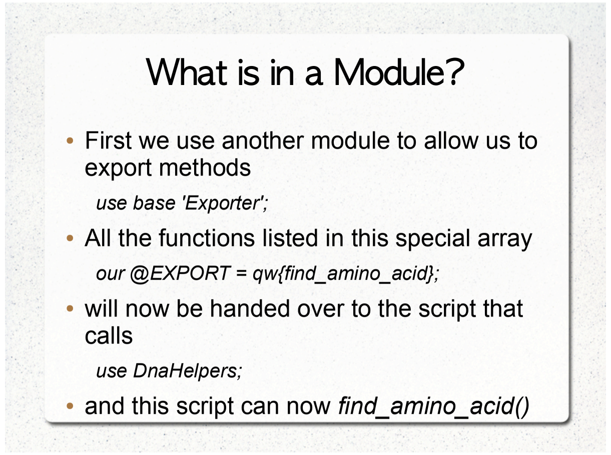
```
sub find_amino_acid {
  my ( $sequence, $amino_acid ) = @_;
  ...
  return $return;
}
```



What is in a Module?

- 3) A way of exporting those functions
- There are two ways of utilising functions in a Module. We are going to look at the non-object oriented way in this section.

```
use base 'Exporter';
our @EXPORT = qw{find_amino_acid};
```



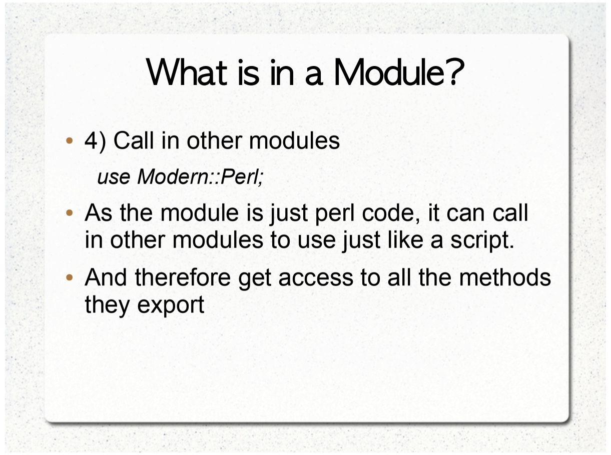
What is in a Module?

- First we use another module to allow us to export methods

  use base 'Exporter';
- All the functions listed in this special array

  our @EXPORT = qw{find_amino_acid};
- will now be handed over to the script that calls

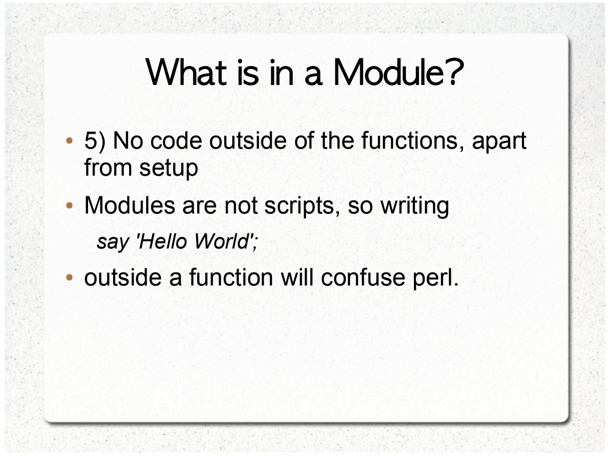
  use DnaHelpers;
- and this script can now *find_amino_acid()*



What is in a Module?

- 4) Call in other modules

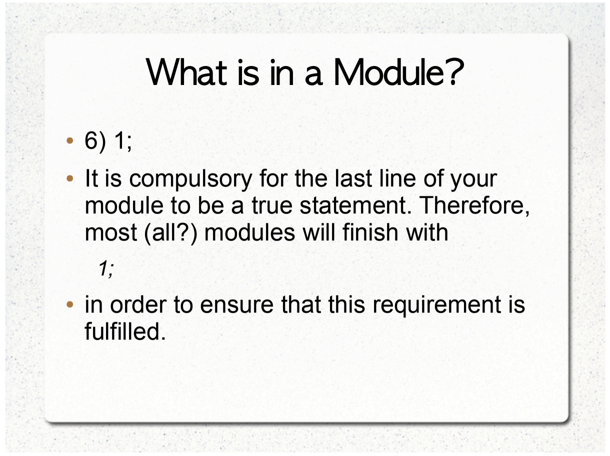
  use Modern::Perl;
- As the module is just perl code, it can call in other modules to use just like a script.
- And therefore get access to all the methods they export



What is in a Module?

- 5) No code outside of the functions, apart from setup
- Modules are not scripts, so writing

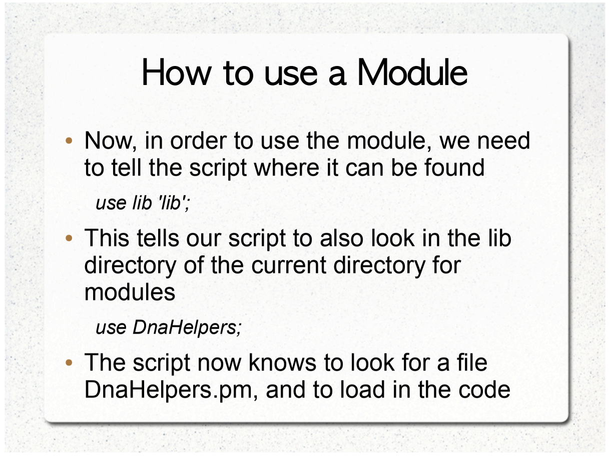
  say 'Hello World';
- outside a function will confuse perl.



What is in a Module?

- 6) 1;
- It is compulsory for the last line of your module to be a true statement. Therefore, most (all?) modules will finish with

  1;
- in order to ensure that this requirement is fulfilled.

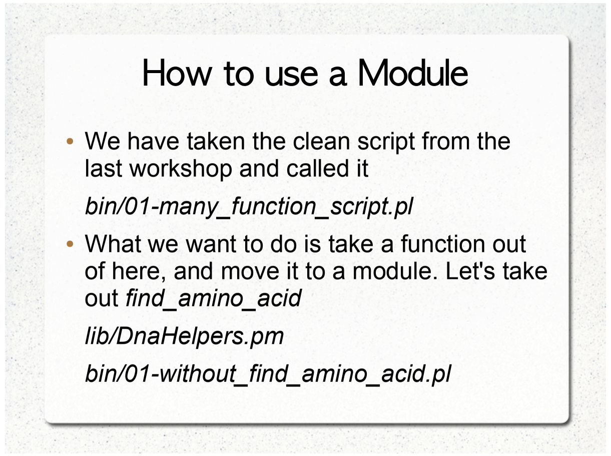


How to use a Module

- Now, in order to use the module, we need to tell the script where it can be found

  use lib 'lib';
- This tells our script to also look in the lib directory of the current directory for modules

  use DnaHelpers;
- The script now knows to look for a file DnaHelpers.pm, and to load in the code



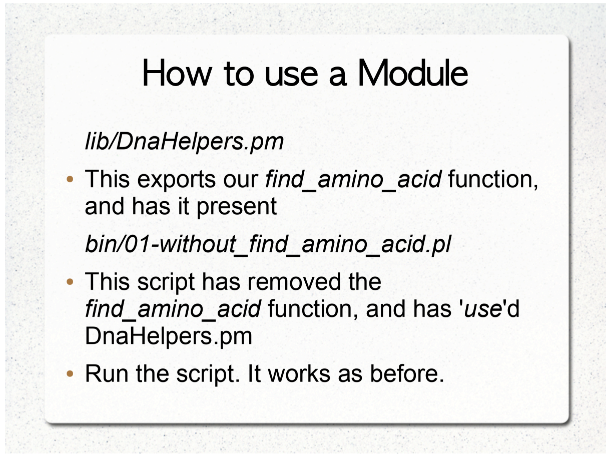
How to use a Module

- We have taken the clean script from the last workshop and called it

  bin/01-many_function_script.pl
- What we want to do is take a function out of here, and move it to a module. Let's take out *find_amino_acid*

  lib/DnaHelpers.pm

  bin/01-without_find_amino_acid.pl

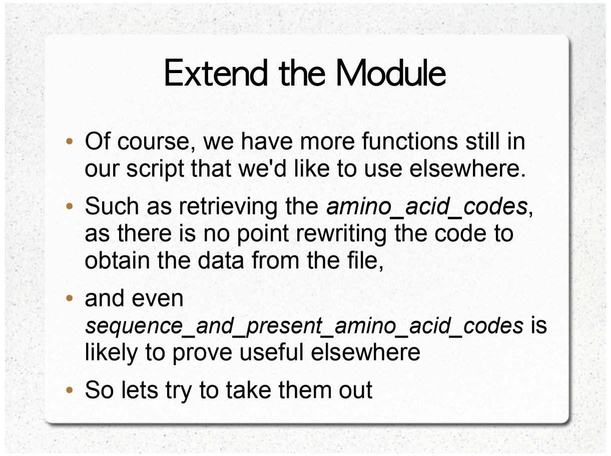


How to use a Module

lib/DnaHelpers.pm

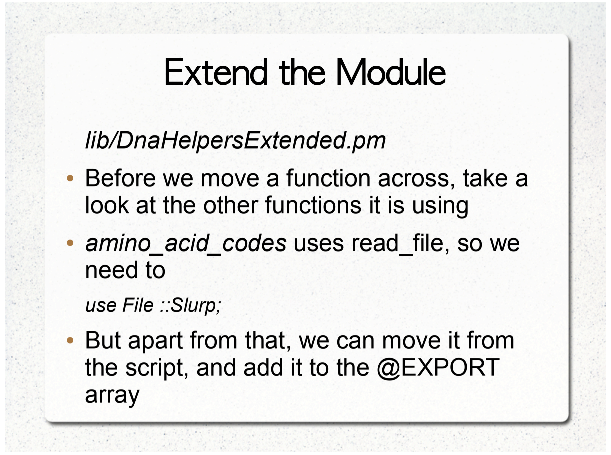
- This exports our *find_amino_acid* function, and has it present

  bin/01-without_find_amino_acid.pl
- This script has removed the *find_amino_acid* function, and has '*use*'d DnaHelpers.pm
- Run the script. It works as before.



Extend the Module

- Of course, we have more functions still in our script that we'd like to use elsewhere.
- Such as retrieving the *amino_acid_codes*, as there is no point rewriting the code to obtain the data from the file,
- and even *sequence_and_present_amino_acid_codes* is likely to prove useful elsewhere
- So lets try to take them out

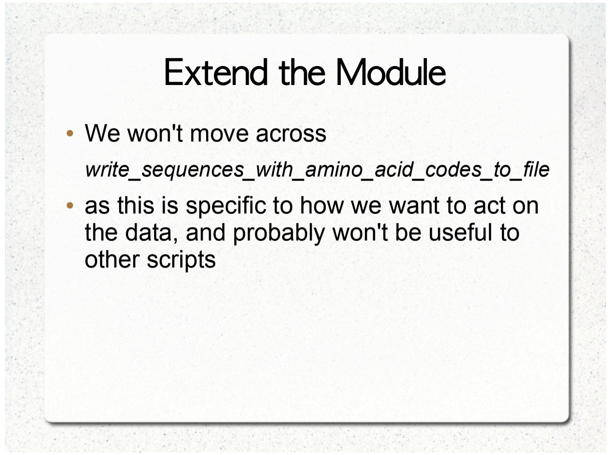


Extend the Module

lib/DnaHelpersExtended.pm

- Before we move a function across, take a look at the other functions it is using
- *amino_acid_codes* uses read_file, so we need to

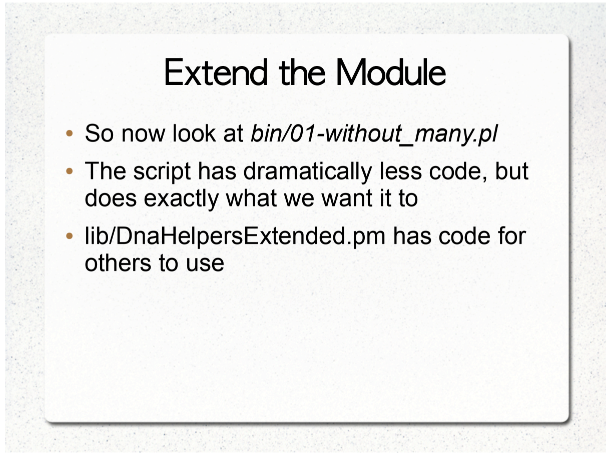
  use File ::Slurp;
- But apart from that, we can move it from the script, and add it to the @EXPORT array



Extend the Module

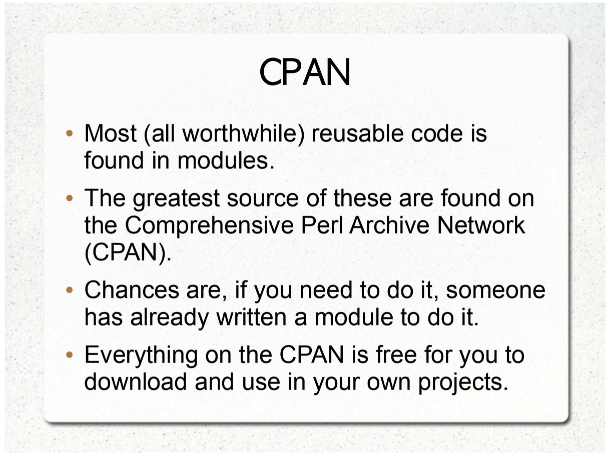
- We won't move across

  write_sequences_with_amino_acid_codes_to_file
- as this is specific to how we want to act on the data, and probably won't be useful to other scripts



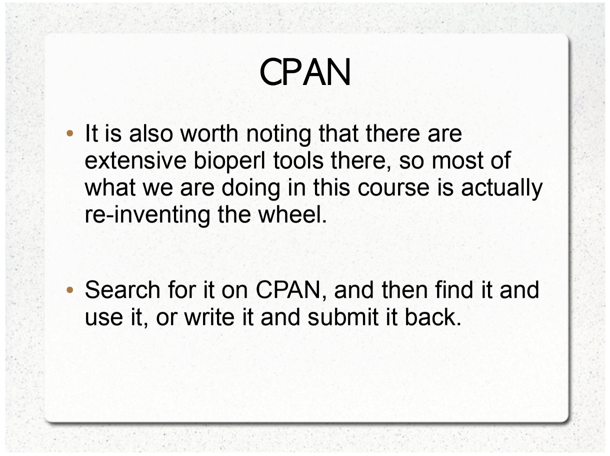
Extend the Module

- So now look at *bin/01-without_many.pl*
- The script has dramatically less code, but does exactly what we want it to
- lib/DnaHelpersExtended.pm has code for others to use



CPAN

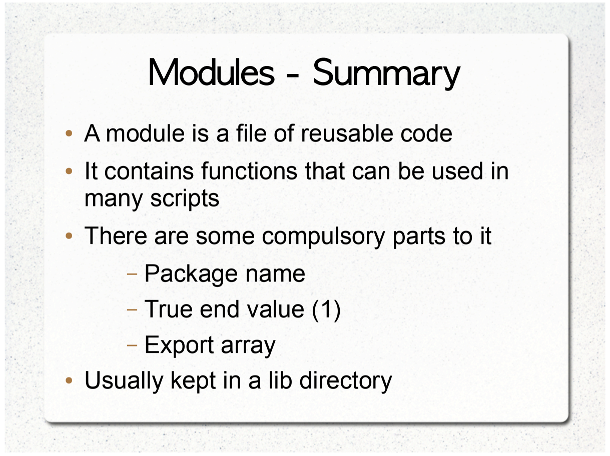
- Most (all worthwhile) reusable code is found in modules.
- The greatest source of these are found on the Comprehensive Perl Archive Network (CPAN).
- Chances are, if you need to do it, someone has already written a module to do it.
- Everything on the CPAN is free for you to download and use in your own projects.



CPAN

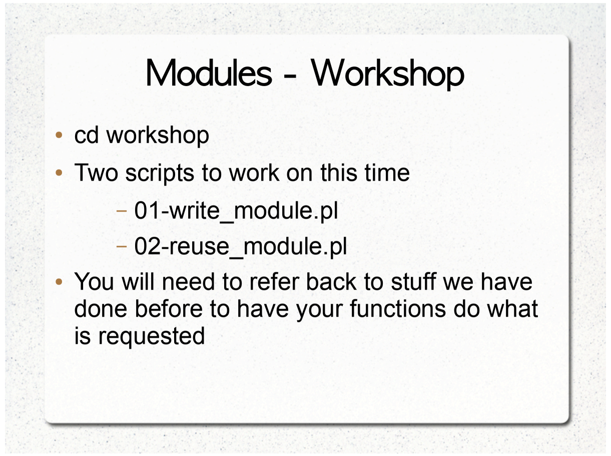
- It is also worth noting that there are extensive bioperl tools there, so most of what we are doing in this course is actually re-inventing the wheel.

- Search for it on CPAN, and then find it and use it, or write it and submit it back.



Modules - Summary

- A module is a file of reusable code
- It contains functions that can be used in many scripts
- There are some compulsory parts to it
  - Package name
  - True end value (1)
  - Export array
- Usually kept in a lib directory



Modules - Workshop

- cd workshop
- Two scripts to work on this time
  - 01-write_module.pl
  - 02-reuse_module.pl
- You will need to refer back to stuff we have done before to have your functions do what is requested