Do you have any questions about assignment #6?
Regular Expressions are a really big deal in Perl. They are a significant part of why so many people use Perl.

Regular expressions can be used with several types of operations.

- Matching – put the matching expression between matched symbols, typically `//`.
- Substituting – The normal format is `s///` where after the first `/` you put the expression to match and after the second `/` is what to replace it with. Put a `g` at the end to substitute multiple. An `i` to ignore case.
- Transcription – replace chars using `tr///`. 
■ By default a regular expression will happen on the variable $_. To make it happen on something else use the binding operator, =~.
The syntax of calling a subroutine is very much like the syntax of calling a function in C. We give the name of the subroutine followed by an argument list in parentheses.

In Perl the parentheses are optional in some cases. (Definitely when there are no arguments.)

Older Perl implementations required a & in front of the subroutine name. Only a few usages would require that now.

Subroutines can be called before they are defined in Perl.
A subroutine in Perl looks like the following:

- `sub name {
  statements
}

Note the lack of an argument list. Instead, the variable `@_` will contain the arguments passed in.

The `my` statement can be used to declare local variables. This is also how we get values out of `@_`. Note that Perl allows list assignments for this purpose.

A return statement allows you to return values from a subroutine.
By default, Perl passes arguments by value.
The argument list is a list of scalars. Passing things other than scalars leads to flattening.
You can pass a reference by putting \ in front of the variable you are passing at the point of the call.
When using a reference in the subroutine you have to prepend the type of what you are pulling out in front of the variable name. So you get things like $$i, @$a, %$h, or $$a[5].
To make it so that Perl will force you to declare variable put the following line at the top of your program.
  • use strict;

It is also a good idea to add the following line at the top of your code.
  • use warnings;
Command-Line Arguments

- When you run a Perl program, any arguments specified on the command line are put in a variable called @ARGV.
- As a bonus, the $0 variable stores the name of the program.
For subroutines that you will want to reuse often it can be helpful to put them into a separate file.

It is common to name these files with .pm and the last line of the file needs to be 1;

Inside your other programs the use keyword allows your code to use your module.
If you run Perl with the -d option it will go into an interactive debugging mode.

You can force this by adding it to the #! at the top of the program or using perl -d from command line.

The command q will stop the debugger. h and h h give help.
We have a quiz on Perl next class.
Assignment #6 is due on Friday.