Random Bits of Perl

None of this stuff is worthy of it's own lecture, but it's all a bunch of things you should learn to use Perl well

eval

- Evaluate Perl code

```
$code = "print ('Hello World\n')";
eval $code; #prints Hello World
```

- trap errors
  - using block notation, if there are any run-time errors, eval returns undef and sets error in $@
- either block notation {…} or expression notation "…" is allowed
  - block syntax checked at compile time, so faster. Expression checked at run-time.
  - If no errors, executes code as though you had actually typed the code in your Perl script.
    - both forms set errors in $@.

Back-ticks

- Third kind of quoted string
- Executes an external program and returns the output

```
@files = `ls *.html`;
```

- Your script will run "`ls *.html`" from the command prompt, and store the output of that program in @files
- Back-ticks do interpolation just like double quotes
- Note that this is not the same as the `system` function
  - `system` returns return value of the called program, not the output

Quoting operators

- You may get strings that end up looking messy because of all the quotes:

```
$s = "He said \"She said \"They
  said \"This is bad.\"\"\"";
```

- can use the quoting operator qq// to 'choose your own quotes'.
  - Much like choosing your own delimiters in pattern matching

```
$s = "He said qq/She said qq(They
  said qq[This is bad])/"
```

  - You could argue this doesn't look much better
    - You could be right.

Many quoting operators

- In each case below, you can use any non-alpha-numeric character for the delimiter, just as you can with m// and s///
- q// - single quoted string
- qq// - double quoted string
  - does interpolation
- qx// – back-ticked string
- qw// - quote words
  - qw/Mon Tue Wed/ → ("Mon", "Tue", "Wed")
- qr// - quote regular expression
  - evaluate string as though it's a RegExp
  - Then use result in an actual pattern match
  - mostly for efficiency – don't worry about this one too much

map

- map expression list
  - evaluate expression (or a block) for each value of list. Sets $_ to each value of list, much like a foreach loop
  - Returns a list of all results of the expression

```
@words = map {split ' ' } @lines;
```

  - Set $_ to first member of @lines, run `split ' '` (split acts on $_ if no arg provided), push results into @words, set $_ to next member of @lines, repeat.

```
@times = qw/morning afternoon evening night/;
@greetings = map "Good $_\n", @times;
```

@greetings → ("Good morning", "Good afternoon", Good evening", "Good night")

grep

- Similar to map (but not exact)
- returns a list of all members of the original list for which evaluation was true.
  - (map returns list of all return values of each evaluation)
- Typically used to pick out lines you want to keep

```
@code = grep !/^\s*#/, @all_lines;
```

  - removes all lines beginning with comments
  - Assigns $_ to each member of @all_lines, then evaluates the pattern match. If pattern match is true, the $_ is added to @code

do

- yeah, yeah, this should have been included with the looping constructs. My bad.
- Similar to C/C++ do/while construct.
- Execute entire block following do once. If block followed by a `while` modifier, check the conditional, and then redo the loop.
- Major Difference: in Perl, `do` is not a looping structure. Therefore, cannot use `next`, `last`, or `redo`.