==========
Perl

Note that Perl, unlike Python & Ruby, has no real interactive mode.
(You can get some of the functionality from the Perl debugger, but it executes each line in isolation, which is pretty unsatisfying.)

"There's more than one way to do it."

```
if ($a < $b) { $s = "less"; }
$s = "less" if ($a < $b);
$s = "less" unless ($b >= $a);
```

heavy use of punctuation characters

```
# comment
#! convention              script language identifier
$, @, %, NAKED            scalar, array, hash, filehandle
<..>                      readline of file handle
=~                        pattern match
$_                        default input line and loop index
. and .=                  concatenation
```

Dynamic typing, coercion

```
$a = "4";
print $a . 3 . "\n";      prints 43
print $a + 3 . "\n";      prints 7
```

subroutines

```
sub min {
   my $rtn = shift(@_);          # first argument
      # my gives local lexical scope; @_ is list of arguments
      # local gives dynamic scope
   for my $val (@_) {
      $rtn = $val if ($val < $rtn)
   }
   return $rtn;
}
...
$smallest = min($a, $b, $c, $d, @more_vals);    # args are flattened
```

context

some things behave differently in array and scalar "contexts".

```
@my_array = @_;
$num_args = @_;
```

you can do this yourself:

```
sub abs {
   my @args = @_;
   for (@args) {
      $_ = -$_ if ($_ < 0);     # $_ is a reference;
   }                            # this modifies args in place
   return wantarray ? @args : $args[0];
      # note: ***not*** @args[0]
}
...
print join (", ", abs(-10, 2, -3, 4, -5)), "\n";
print $n = abs(-10, 2, -3, 4, -5), "\n";
```

This prints

```
10, 2, 3, 4, 5
10
```

regular expressions -- discussed in previous lecture

hashes

```
%complements = ("red" => "cyan",
                "green" => "magenta", "blue" => "yellow");
   # NB: => is (almost) an alias for ,
   # (also forces its left operand to be interpreted as a string)
print $complements{"blue"};        # yellow
```

Examples from book

HTML heading extraction (Example 14.23, Fig. 14.4, p. 716)

```
while (<>) {                          # iterate over lines of input
   next if !/<[hH][123]>/;            # jump to next iteration
   while (!/<\/[hH][123]>/) { $_ .= <>; }   # append next line to $_
   s/.*?(<[hH][123]>.*?<\/[hH][123]>)//s;
      # perform minimal matching; capture parenthesized expression in $1
   print $1, "\n";
   redo unless eof;    # continue without reading next line of input
}
```

note:

- #!
- while (<>)
- next, redo
- implicit matching against $_
- update-assignment to $_
- s/// -- could have been written $_ =~ s///
- minimal matching via *?
- character sets in REs: [hH], [123]
- backslash escape of /
- capture with ( )
- trailing s on match allows '.' to match embedded \n

force quit (Example 14.24, Fig. 14.5, p. 719)

```
$#ARGV == 0 || die "usage: $0 pattern\n";
open(PS, "ps -w -w -x -o'pid,command' |");  # 'process status' command
<PS>;                                       # discard header line
while (<PS>) {
   @words = split;                  # parse line into space-separated words
   if (/$ARGV[0]/i && $words[0] ne $$) {
      chomp;                        # delete trailing newline
      print;
      do {
         print "? ";
         $answer = <STDIN>;
      } until $answer =~ /^[yn]/i;
      if ($answer =~ /^y/i) {
         kill 9, $words[0];         # signal 9 in Unix is always fatal
         sleep 1;                   # wait for 'kill' to take effect
         die "unsuccessful; sorry\n" if kill 0, $words[0];
      }                             # kill 0 tests for process existence
   }
}
```

note:

- @ARGV, $#ARGV — latter is last index (one less than length)
- die
- open, file handles
- ps command
  - -w -w — print with unlimited width (wide wide)
  - -x — include processes w/out controlling terminals
  - -o'pid,command' — what to print
- split
- trailing i on match ignores case
- $$ — my process id
- ne (strings) vs != (numbers)
- beginning of line marker: ^ (and eol marker: $)
- built-ins for many common shell commands (kill, sleep)