Subroutines

Subroutines

- subroutines are blocks of code that can be reused
- start with `sub` and have a name and then are enclosed in code block of `{}`

```
sub a_routine {
    my @args = @_; # the arguments passed in are avaialable as @_;
    print "the arguments are ", join(",", @args), "\n";
}
# here is code to call this subroutine
&a_routine('a','b','c');
```

- Not required to define the subroutine before you use it, so you can writ all your subroutines at the bottom of your file
- Or store in a separate file and bring in with `require`

Subroutine arguments

- Input to a subroutine is always a list, if you want to have some things stay grouped, you must use references.

```perl
#
&evaluate( qw(a b c), qw(Z Y X));

&evaluate( [qw(a b c)], [qw(Z Y X)]);
sub evaluate {
    my @arguments = @_;
    print "1st arg is: $_[0] second arg is $_[1]\n";
    # really you expect it to be used in the second way
    my $dataset1 = $_[0];
    my $dataset2 = $_[0]
}
```

Modules

- Collections of subroutines.
- Allow association of data in structures
- Object-Oriented programming support

Getopt::Long module

Command line operated programs traditionally take their arguments from the command line, for example filenames. Besides arguments, these programs often take command line options as well. Options are not necessary for the program to work, hence the name 'option', but are used to modify its default behavior.

```
$ grep -i 'AACA' DNA.txt > capture.txt
$ make_fake_fasta.pl --length 100
```

script using Getopt::Long

```perl
#!/usr/bin/env perl
use strict;
use warnings;
use Getopt::Long;
my $length = 30;
my $number = 10;
my $help;
GetOptions('l|length:i' => \$length,
           'n|number:i' => \$number,
           'h|help' => \$help);
my $usage = "make_fake_fasta.pl - generate random DNA seqs
Options:
-n <number> the number of sequences to make (default: 10)
-l <length> the length of each sequence (default: 30)
";
die $usage if $help;
my @nucs = qw(A C T G);
for (my $i = 1; $i <= $number; $i++) {
    my $seq;
    for (my $j = 1; $j <= $length; $j++) {
        my $index = int(rand (4));
        my $nuc = $nucs[$index];
        $seq .= $nuc;
    }
    print ">fake$i\n";
    print $seq, "\n";
}
```

List::Util module

```perl
use List::Util qw(shuffle sum);

my @list = (1..12);
my @shuffled = shuffle(@list);
print join(",",@list)," : original\n";
print join(",",@shuffled)," : shuffled\n";

my $sum = sum(@list);
print "total is ", $sum, "\n";
```