Leftover - reading files from directories

- To open and read names of files in a directory you need two functions
- `opendir` to open a directory, works like open does for files (only you can only read a directory) and `readdir` to get a list of files in a directory

```
1 opendir(DIR,"mydir") || die "cannot open dir: $!"
2 foreach my $file ( readdir(DIR) ) {
3     print "file is $file\n";
4     if( $file eq 'special.dat' ) {
5         open(IN, "mydir/$file") || die $!;
6         while(<IN>) {
7         }
8     }
9 }
```

Leftover - making / deleting a directory

- `mkdir` will create a directory through perl
- `rmdir` will remove an empty directory in perl

References

Reference are ways to refer to a complicated data structures as a single, scalar value. This lets one pass around multiple arrays and they stay separate. We also primarily use reference to store multiple things in a slot in an array.

- Reference to an array is done with `\` or `[]`
- Reference to a hash is done with `\` or `{}`

For example this lets one pass around multiple arrays and they aren't flattened into one. Consider this code.

```
1 my @array1 = qw(A B C D);
2 my @array2 = qw(W X Y Z);
3 my @array3 = (@array1, @array2);
4
5 print join(",", @array3), "\n";
```

Storing multiple items in an Array

```
[ 0 | 1 | 2 ]
  |   |
  |   -------------------------
 { apple => 'red' ,            |
   turtle => 'green' }      { bannana => 'yellow',
                              pig     => 'pink' }
```

Often we use this approach to store multiple things for a single key in hash too. What if a gene has mutiple protein domains, function, or other information you wanted to store for it?

Array of Arrays

```
 1 my @matrix = ( [qw( 1 0 0 0) ],
 2                [qw( 0 1 0 0) ],
 3                [qw( 0 0 1 0) ],
 4                [qw( 0 0 0 1) ] );
 5
 6 for my $row ( @matrix ) {
 7     print join(",", @$row), "\n";
 8 }
 9
10 push @matrix, [qw(1 1 1 1)];
```

Array of Hashes

- can have arrays made up of Hashes

```
 1 my @favorites = ( {
 2 'name'  => 'worm_1',
 3 'fruit' => 'apple',
 4 'size'  => 12 });
 5 push @favorites, { 'name' => 'worm_2',
 6 'fruit' => 'pear',
 7 'size'  => 10};
 8
 9 for my $record ( @favorites) {
10     print "name is ", $record->{'name'}, " and size is ",
11     $record->{'size'}, "\n";
12 }
```

Hash of Arrays

- The pockets of the roulette wheel are numbered from 1 to 36.
- In number ranges from 1 to 10 and 19 to 28, odd numbers are red and even are black.
- In ranges from 11 to 18 and 29 to 36, odd numbers are black and even are red.

```
 1 my %roulette = ( 'Black' => [],
 2                  'Red'   => [] );
 3 foreach my $num ( 1..10, 19..28 ) { # .. operator just lists all numbers in a range
 4     if( $num % 2 == 0 ) { # even
 5         push @{$roulette{'Black'}}, $num;
 6     } else {
 7         push @{$roulette{'Red'}}, $num;
 8     }
 9 }
10 foreach my $num ( 11..18, 29..36 ) {
11     if( $num % 2 == 0 ) { # even
12         push @{$roulette{'Red'}}, $num;
13     } else { push @{$roulette{'Black'}}, $num; }
14 }
15 for my $color ( keys %roulette ) {
16     print "$color: ",join(",", @{$roulette{$color}}),"\n";
17 }
```

Hashes of Hashes

- can make hashes that contain hashes. This makes for convient data structures once you get the hang of it.

```
1 my %tickets;
2 $tickets{'section 1'}->{'row 2'}->{'seat A'} = '$10';
3
4 my $genes; # make this a hash reference
5 $genes->{'p53'}->{'length'} = 300;
6 $genes->{'p53'}->{'exons'} = 6;
```

Combing hashes of hashes

http://courses.stajich.org/public/gen220/data/Nc3H.expr.tab http://courses.stajich.org/public/gen220/data/Ncrassa_OR74A_InterproDomains.tab

```
 1 my $url = 'http://courses.stajich.org/public/gen220/data/Ncrassa_OR74A_InterproDomains.ta
 2 open(my $fh => "GET $url |") || die $!;
 3 my %genes;
 4 while(<$fh>) {
 5     my ($gene,$domain, $domain_name, $start,$end,$score) = split;
 6 # store an array as the value for each key by making it a reference to an array
 7 # using the @{$genes{$gene}} which is forcing what is the value
 8 # to be an array reference. Then we use push to add something to
 9 # this array
10 # Because perl will automatically initialize the value, based on the context
11 # we DON'T need to do this, but it is what is happening under the hood
12 # if this is the first time accessing this key
13 # $genes{$gene} = [];
14  push @{$genes{$gene}}, $domain_name;
15 }
16 # now unpack to print this out
17 for my $gene ( keys %genes ) {
18     my @domains = @{$genes{$gene}};
19     print join("\t", $gene, join(",", @domains)), "\n";
20 }
```

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 `{}`

```
1 sub a_routine {
2     my @args = @_; # the arguments passed in are avaialable as @_;
3     print "the arguments are ", join(",", @args), "\n";
4 }
5 # here is code to call this subroutine
6 &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

- Always a list, if you want to have some things stay grouped, you must use references.

```
1 &evaluate( qw(a b c), qw(Z Y X));
2 sub evaluate {
3     my (@list) = @_;
4
5 }
```