Recap some of the main concepts

- Scalar variables: Strings and numbers
- Arrays
- Hashes
- remember you can read more and see examples with `perldoc -f METHODNAME` (e.g. put `length` in for METHODNAME) You can also read the perl documentation and intro at perl.org/docs.html

Scalars and Strings

- Scalar variables start with `$`
- Scalar variables can hold a string or number.

```
my $single = 'one';
my $double = 2;
my $triple = 33/11;
```

- Strings can be manipulated with several functions: `reverse`, `lc`, `uc`, `length` `index` to find a substring, `substr` to retrieve a substring
- you can print out a value with `print` or `printf`
- `@array = split($sep,$string)` and `$string = join($sep,@array)` to convert string to array and array to a string

Arrays

- Arrays are used to store multiple values, in an ordered list
- You can access an item in an array with `[]` operator like `$array[2]` to get the value at position `2` in the array. This can be used to retrieve or set a value at that position. For example

```perl
my @array = (1,2,3);
print "array is @array\n";
$array[2] = 44;
print "array is @array\n";
print "The 3rd item in the array is $array[2]\n";
```

- Arrays can be manipulated as well with `push`, `pop`, `shift`, `unshift`
- The function `sort` will return a reordered array
- The function `map` can be used to map all values in an array from one value to another

Array access

- Initialize an array in several ways:

```
my @array = (); #empty array
my @array = (1,2,'seven'); # with values, comma separated
my @array = qw(Kansas Indiana Texas); # quote words
```

- Retrieve items from an array by assigning to variables

```
my @array = qw(First Second Third);
my ($one,$two,$three) = @array;
print "$array[1] == $two\n";
```

- Set items in an array by assigning to variables

```
my @array = qw(First Second Third);
$array[3] = 'Fourth';
print "$array[3]\n";
```

Hashes

- Also called associate arrays or dictionaries, lookup or store a Key and associate it with a value
- Use the `{}` operators to access data in the hash

```perl
my %favs;
$favs{'color'} = 'orange';
$favs{'car'}   = 'ford';
$favs{'fruit'} = 'orange';

print "I like to eat $favs{'fruit'} and drive my
$favs{'color'} $favs{'car'}\n";
```

- You can access the names of the keys with `keys` function.

Looping

- for and foreach loops useful for iterating through set of numbers or an array

```
my @shapes = qw(square triange circle);
foreach my $shape ( @shapes ) {
   print "shape is $shapes\n";
}
for( my $i = 0; $i < scalar @shapes; $i++ ) {
     print "shape is $shapes[$i]\n";
}
```

- Can also loop through a hash of keys and values

```
my %shapes = ('triangle' => 3,
          'square'   => 4,
          'pentagon' => 5,
          'hexagon'  => 6);
# go through the keys, in random (undefined) order
foreach my $shape ( keys %shapes ) {
   print "a $shape has $shapes{$shape} sides\n";
}
# sort by number of sides
foreach my $shape ( sort { $shapes{$a} <=> $shapes{$b} } keys %shapes ) {
   print "a $shape has $shapes{$shape} sides\n";
}
```

Other useful functions

- `rand` – returns a random number greater than or equal to 0 and less than value provided to rand (or 1 if no value is provided). `my $num = rand(5)`
- `int` – return only the integer part of a number
- `scalar` treat something like a scalar – when there is ambiguous context this can be helpful
- `localtime` – return an array of current time in 9 elements (seconds to year), or `my $date = localtime` will return it as text string
- `ord` to convert a symbol (charcter) into ASCII code and `chr` to convert ASCII code number into a symbol (character)

Reading and Writing (I/O or Input/ Output)

- How to get the command-line
- Input data from files
- Read data from a program
- Print data into a file or print data out to a program

Command-line input

- When you run something on the command-line all the arguments are available to perl. Try to write this simple program and call it 'args.pl'

```
print "There are ",scalar @ARGV, " arguments\n";
print "The arguments are: @ARGV\n";
print "The first argument is: $ARGV[0]\n";
```

- Try running the program with this on the command line

  $ perl args.pl arg1 arg2 these_are more args

- The name of the program run is in the `$0` variable

```
print "The program run is $0 the arguments are @ARGV\n";
```

- We will explain some helper modules later that make it easy to get all the command line options easy to parse (see Getopt::Long)

Implicit variable and Defaults

- `$_` is the implicit variable.
- Perl lets you be lazy. If you forget to pass in variable to a function it is assumes you want to use the implicit variable. Sometimes there are also defaults which it will use. For example:

```
$_ = "This is some data";
my @ret = split; # assumes you want to split on white-space,
                 # on the implicit variable
@ret = split(/\s+/,$_); # same as the line above
```

- `$_` is also used when something is returned but you do not assign it to something.
- It is also referred to when you you do not pass in a argument to something

Input/Ouput

Open a file with the `open` function. It takes 2 arguments a filehandle which will be a pointer to the opened file, and a string representing the file to open.

```
open(IN, "input.txt") || die $!;
# read a line in
my $line = <IN>;
# read the whole file
while(<IN>) {
 my ($col1, $col2) = split;
}
close(IN);
```

Filehandles

Filehandles can also be stored in variables

```
my $fh;
open($fh => "gene.dat") || die $!;
while(<$fh>) {
 print $_;
}
#I like to use this in one line
open(my $fh2 => "gene2.dat") || die $!;
while(<$fh2>) {
 print $_;
}
```

Writing to a file

`open` is also used to open a file for printing out to.

```perl
open(OUT => ">outputfile.txt") || die $!;
print OUT join("\t", qw(NAME RANK TOWN)), "\n";
print OUT join("\t", qw(GRIFFITH SHERIFF MAYBERRY)), "\n";
print OUT join("\t", qw(RAWLS DEPUTY BALTIMORE)), "\n";
close(OUT);

open(my $fh => ">outfile2.txt") || die $!;
print $fh "A data report\n";
print $fh "This is also a report line\n";
```

Data embedded in a script

```
while(<DATA>) {
 my ($col1,$col2) = split(/\s+/,$_);
}

__DATA__
Color  Size Model
red    10   Jumbo
yellow 8    Large
pink   2    Mini
```

Practice

Write a script that will open and print out the first 5 lines of a file. The name of the file to open should be passed in on the command line as the first argument.

```
use strict;
use warnings;

# get the name of the file from the cmdline

# open the file

# read a line at a time of the file
while( ... ) {
# print the line out
# and stop after 5 lines
}
```

Pipes for processes

You can combine operations that are on the command line with the `|` operator in UNIX. This can also be used in Perl when specifying an `open` command to actually run a program and have the output sent back to the Perl program. This can be used to obtain data from a program. For example here we run the grep command to find lines in a file that have the string 'gene'. Only those lines will be returned and thus be seen by the Perl program.

```perl
open(IN,"grep 'gene' gene.dat | ") || die $!;
while(my $line = <IN>) {
 print "line is $line\n";
}
```

Or it can be used to send data to a program. For example this script will print out data to a program which will then compress the data. Notice how the pipe comes at the beginning because we plan to send data into the program.

```perl
open(my $fh => "| gzip -c > file.gz") || die $!;
print "hello there\n";
print "can you see this?\n";
```

Now look at the file in your directory. It is compressed - you can read it with `more` and see it is binary file. However you can read it with `zcat` or you can uncompress it with `gunzip`.

Pipe trick

Can use it to open a compressed file on the fly.

```
open(my $fh => "zcat file.gz |") || die $!;
while(<$fh>) {
 # process data in a file that was compressed, without making a new copy of the file as un
}
```

Read data from the web with cmdline

```perl
my $url = 'http://s.fungidb.org/1es5REC';
# -S option will not print any statistics
open(my $fh => "curl -S '$url' |") || die $!;
while(<$fh>) {
  print $_;
}

open($fh => "GET '$url' |") || die $!;
while(<$fh>) {
  print $_;
}
```

- the shortened link was to NCBI: http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=163644330&retmode=text&rettype=fasta to retrieve a sequence in FASTA

Let's try this together

Login to biocluster, Download data files. Data are in this http://courses.stajich.org/public/gen220/data/ which represent some time points in growth for a fungus.

http://courses.stajich.org/public/gen220/data/Nc20H.expr.tab http://courses.stajich.org/public/gen220/data/Nc3H.expr.tab

```
 wget http://courses.stajich.org/public/gen220/data/Nc3H.expr.tab
(or on biocluster)
/shared/gen220/data_files/expression/Nc3H.expr.tab
/shared/gen220/data_files/expression/Nc20H.expr.tab
```

Write a script to read in the Nc20H and Nc3H data into a hash (one hash for each datafile). Store in the hash the gene name (the 1st column) and the FPKM data. Each gene will appear once in each file.

- Print out a new file which has the gene name, the expression in 3H and the expression in 20H.
- Extra - print it out so that it is sorted by the 3HR timepoint

A solution

```perl
use strict;
use warnings;
my (%expr3H,%expr20H);
open(my $fh => 'Nc3H.expr.tab') || die $!;
while(<$fh>) {
 my @row = split("\t",$_);
 next if $row[0] eq 'gene_id'; # skip when it is the header line
 $expr3H{$row[0]} = $row[5];
}

open($fh => 'Nc20H.expr.tab') || die $!;
while(<$fh>) {
 my @row = split("\t",$_);
 next if $row[0] eq 'gene_id'; # skip when it is the header line
 $expr20H{$row[0]} = $row[5];
}

open(my $outfh => ">Combined.tab") || die $!;
my $gene;
for $gene ( keys %expr3H) {
 print $outfh join("\t", $gene, $expr3H{$gene}, $expr20H{$gene}), "\n";
}

open($outfh => ">Combined_sorted.tab") || die $!;
for my $gene ( sort { $expr3H{$b} <=> $expr3H{$a} } keys %expr3H) {
 print $outfh join("\t", $gene, $expr3H{$gene}, $expr20H{$gene}), "\n";
}
```