```
#This page contains several Perl contructs that we will commonly use in the
#NCBI PowerScripting course. In the Perl scripts we will provide, we attempt to
#use these constructs as consistently as possible.


#all text following a "#" in perl is treated as a comment


#File Input


#ARGV[0] is the first commandline parameter, $ARGV[1] is the second.

open (INPUT,"<$ARGV[0]")||die "Aborting. Can't open $ARGV[0].\n";

while(<INPUT>){

               #do something with the current record, the current line by default
               #the current record is always available in the variable $_

chomp $_;      #remove newline from $_
print "$_\n";  #print the current record with newline replaced
}
close(INPUT);

#Parsing a record in perl (by default, the record is a single line)


$delimiter=",";

@fields=split("$delimiter",$_);
                                  #$_ is the current record, $delimeter is a perl
                                  #regular expression that defines the field separator
                                  #in this case, the separator is a comma
                                  #join is the inverse of split
$all_fields=join("$delimiter",@fields);



#Traversing a hash table (associative array)


#In Perl, hashes are arrays indexed by arbitrary strings called keys, each of which
#points to a single data element.
#Hash names begin with the "%" character. However, when referencing a hash value,
#the dollar sign precedes the name and the key goes in curly brackets "{}": $hash{$key}

foreach $index (keys %hash){

                              #print each key and value in hash table called "hash" separated by a
print "$index\t$hash{$index}\n";


}

#Working with Arrays in Perl


#array names in perl begin with the "@" character

$bigdata="all,of,GenBank,and,RefSeq,in|a|big|file";

#Split also allows multiple delimiters enclosed in square brackets "[]"

@bigdata_array=split(/[,|]/,$bigdata);
```

```
foreach $bigitem (@bigdata_array){

#The =~ operator tests for the presence of the regular expression, in this case "a", in $bigitem.
#Push appends $bigitem to the end of the array @smalldata.

if ($bigitem =~ /a/){push (@smalldata,$bigitem);}


}

print @smalldata;
print join("\n",@smalldata);

 #Accessing the Internet Using Perl


#For retrieving data from a url post in perl, we will use the LWP module "get" function

use LWP::Simple; #supports the "get" function

$baseurl="http://eutils.ncbi.nlm.nih.gov/entrez/eutils/";
$eutil="esearch.fcgi?";
$parameters="db=nucleotide&term=human[orgn]+AND+jak3";
$url=$baseurl.$eutil.$parameters;

$raw=get($url);

print $raw;

#eutility output is given in XML by default
#we can parse the XML output in $raw using the following construction

#split $raw into an array of lines, @lines
@lines=split(/^/,$raw);


foreach $line (@lines){
#if a line contains an XML ID tag, the id, a string of digits, is captured and
#pushed onto an array called @ids

if ($line=~/<ID>(\d+)<\/ID>/){push(@ids,$1);}
}

 #Using Subroutines in Perl


#parameters are pased to a subroutine using a parameter hash table

$params{"term"}="mouse[orgn]";

%results=egquery(%params);

sub egquery {

#Parameters passed to a subroutine are stored in the special array @_

local %params=@_;

#Do something here with the parameters, putting the results in %results
#Then pass the results back to the subroutine call

return(%results);

}
```