# Some Useful Perl Functions

Concatenation Operator: **.**

- This operator allows you to 'join' things together.

Syntax example:

```
$v = "encyc" . "lopedia";
$v2 = "Hello" . "world";
$v3 = "Hello " . "again.";
```

$v is now "encyclopedia"
$v2 is now "Helloworld"
$v3 is now "Hello world" (note the space!)

- You can also join variables together with . operator

Syntax example:

```
$v = "It's";
$v2 = "almost";
$v3 = "over!";

$v4 = $v . " " . $v2 . " " . $v3 . "Smile!";
```

$v4 is now "It's almost over!"

- it can also be used to append to the end of an already existing string.

```
$v = "attggg";
$v .= "CCCAAT";
```

- using this operator, $v now contains: attgggCCCAAT

## The length( ) function

- this function returns the length of the string placed in between ( )

```
#!/usr/bin/perl -w

$v = "aaatttcccggg";
$len = length($v);
print "\nv = $v\n";
print "length of v is: " . $len . "\n";
exit;
```

Output:

```
v = aaatttcccggg
length of v is: 12
```

## Initializing / Erasing variable Data

```
$v = "atttcggcaa";
$v = "";    ## now '$v' is empty
```

For arrays:

```
@ary1 = (); ## initializing an empty array
@ary1 = (2,4,5);
@ary1 = (); ## now @ary1 is empty again
```

Same goes for hashes:

```
%hash1 = (); ## initializing / emptying
```

# Working with Files

## 1. Opening and Closing a file in perl

A. Syntax for opening a file in perl:

```
open(FILEHANDLE, "filename") or die "Can't open filename\n";
```

**open** → This is the perl function used to open a file.

**FILEHANDLE** → - The string you use to refer to the file throughout your perl script

- This can be any text you like (CAPTIALS are optional but recommended, spaces not allowed).

**"filename"** → - The actual name of the file (example: sequences.fa, somefile.txt, etc…),

- The double quotes are required!

**or die ("…")** → - This tells the program to quit if it cannot open or find a file called 'filename'

- This part of the syntax is optional but recommended.

- You can put any string you like between the ( )'s

## B. Syntax for closing a file in perl:

```
close(FILEHANDLE);
```

- Files can be opened in different 'modes':

```
open(FILEHANDLE, "filename"); ## open file in read mode
open(FILEHANDLE, "<filename"); ## open file in read mode, explicitly
open(FILEHANDLE, ">filename"); ## create the file and write to it
open(FILEHANDLE, ">>filename"); ## append to the end of an already
                                ##  existing file
```

# 2. Reading Data from a file in perl

A. Assigning a line from a file to a variable:

```
$var1 = <FILEHANDLE>;
```

- This assigns the next line from FILEHANDLE to '$var1'.

- The entire line is assigned to $var1 include the new line character (\n) at the end of the line and any other "unseen" characters.

- You can remove the new line (\n) character using the `chomp()` command.

B. Assigning a file to an array:

- You can iteratively take a line from a file and assign it to an array:

```
while($line = <FILEHANDLE>) ## this will stop iterating when it
                            ## reaches the end of file
{
  push(@ary1, $line); ## the current contents of '$line' are added
                      ## to '@ary1'
}
```

# File Reading in Action…

Sample Program:

```
#!/usr/bin/perl -w

## This script will read in a FASTA format file
##  and print each line of the file to the screen.

## Here we are opening the file
open(FASTA, "<randSeq.fa") or die "I can't open
randSeq.fa!\n";

## Here we are reading in the file line-by-line.
## Each line of the file is assigned to the variable $line
while($line = <FASTA>)
{
  print $line;  ## print to the screen.
}

close(FASTA); ## close the file when you are done.
exit;
```

This is what the file looks like:

```
>rand_1
GATGTTGAATTAACTTACGGA
ACTGCAACGCAGAGAATACCC
ATGGATT
>rand_2
GGAAGTTGTGTGTTCCGCTAC
CTGGCGCATTAGTGCGTAAGC
TTTAACG
>rand_3
CGGCCGACAGCGGCACAATCC
TGCCGGGGGCGCGCTGTGCCA
GAAGTGC
..more sequences..
```

What's the Output?

```
>rand_1
GATGTTGAATTAACTTACGGAACTGCAACGCAGAGAATACCCATGGATT
>rand_2
GGAAGTTGTGTGTTCCGCTACCTGGCGCATTAGTGCGTAAGCTTTAACG
>rand_3
CGGCCGACAGCGGCACAATCCTGCCGGGGGCGCGCTGTGCCAGAAGTGC
>rand_4
CCGCGGCTTTACCAGGGCAACAGCAGCATGTACTACCCCGGTCTCGTGC
```

# Is everyone still with me?




http://www.dilbert.com/comics/dilbert/archive/dilbert-20030619.html

# 3. Writing Data to a file in perl

- In order to write to a file it first must be created (if it doesn't already exists) and it must be opened in one of the two 'write' modes for a file:

```
open(FILEHANDLE, ">filename"); ## create
open(FILEHANDLE, ">>filename"); ## append
```

-Syntax for writing to a file:

```
print FILEHANDLE "what you want to write to the file";
```

# File Writing in Action…

```
#!/usr/bin/perl -w

## This script will read in a FASTA format file
##  and write the first 3 lines of it to a file.

## Here we are opening the file for reading
open(FASTA, "<randSeq.fa") or die "I can't open randSeq.fa!\n";

## Here we open a new file in WRITE mode
open(OUTPUT, ">output.txt") or die "Can't create output file\n";

$i = 0; ## used as a counter later on

## Here we are reading in the file line-by-line.
## Each line of the file is assigned to the variable $line
while($i < 3)
{
  $line = <FASTA>; ## read in a line from FASTA and assign it to $line

  print OUTPUT "$line"; ## write the data in 'line' to OUTPUT file
  $i++; ## increment the counter

}

close(FASTA); ## close the file when you are done.
close(OUTPUT); ## close the output file
exit;
```

# The Fasta file begin read:

```
>rand_1
GATGTTGAATTAACTTACGGAACTGCAACGCAGAGAATACC
CATGGATT
>rand_2
GGAAGTTGTGTGTTCCGCTACCTGGCGCATTAGTGCGTAAG
CTTTAACG
>rand_3
CGGCCGACAGCGGCACAATCCTGCCGGGGGCGCGCTGTGCC
AGAAGTGC
>rand_4
CCGCGGCTTTACCAGGGCAACAGCAGCATGTACTACCCCGG
TCTCGTGC
>rand_5
GTTAGCGCTTTGAAACTGCCAGTCTATGCCGAGGTAGTTAT
CTAGGTTC
>rand_6
TGAAATGAGGCCGGTTCCAGTTTGGCGCTGCGCGAAATTGG
CGACGGTA
```

# What's the Output?

```
>rand_1
GATGTTGAATTAACTTACGGAACTGCAACGCAGA
GAATACCCATGGATT
>rand_2
```

# Reading in user input from the keyboard

Example:

```
#!/usr/bin/perl -w

print "Please enter your name: ";
$n = <>;

print "Your name is: $n";
```

STDIN denotes Standard Input, which is usually the terminal from which the program was invoked (ie: the keyboard is the source of the input).

While the Perl program is running, each time the term <> is encountered, the program stops and waits for the user to enter some characters followed by a newline (Return). The entered strings will then become the value of <>.

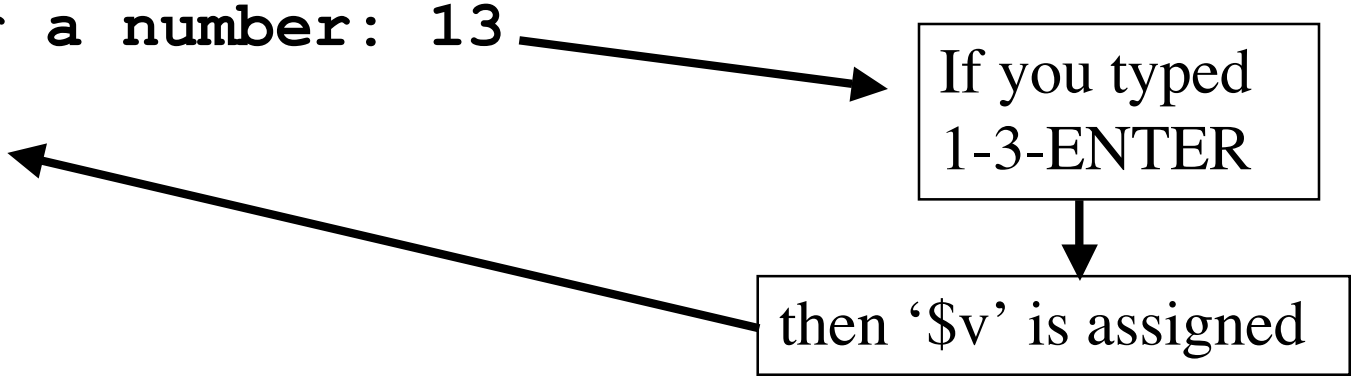
If <> is assigned to a scalar variable (like $n above) then the scalar is assigned the user's input.
You may also see this written as: **`$n = <STDIN>;`**

Note that when entering information from the keyboard. The newline character (\n) is always added to the input when you hit the RETURN/ENTER button!

```
Please enter a number:

Please enter a number: 13

$v = "13\n";
```




"13\n" ≠ 13

To fix this, use **`chomp()`** command to remove '\n'

```
chomp($v);
```

Now: **`$v = 13;`**