

# Programming in Perl

*Randy Julian*  
*Lilly Research Laboratories*

## The Crash Course...

- ◆ Getting Perl (CPAN, PPM)
- ◆ Getting/using the ptkdb debugger
- ◆ print "hello world\n"
- ◆ About variables: strings, numbers, arrays
- ◆ Operators and expressions
- ◆ About flow control
- ◆ Introduction to regular expressions
- ◆ Using Perl file I/O
- ◆ Basic subroutines and modules

An assembly of material harvested from various sources

## Getting Perl

### ◆ Windows:

- [www.activestate.com](http://www.activestate.com)
  - ◆ download "ActivePerl" (Either 5.6.x or 5.8.x)
- Run windows installer

### ◆ Linux:

- Probably already installed (standard on most)
- Latest version (5.6.x/5.8) via either RPM (from ActiveState or RedHat) or from source:  
[cpan.perl.org](http://cpan.perl.org)

## Getting the ptkdb debugger

### ◆ Windows:

- `C:\WINDOWS>perl -e "use Devel::ptkdb;"`
  - ◆ If this gives an error, you need to install the debugger
- PPM: Perl Package Manager
  - ◆ `C:\WINDOWS>PPM`
  - ◆ `PPM>install Devel::ptkdb`

### ◆ Linux:

- `$ perl -MCPAN -e 'install "Devel::ptkdb" '`

From: "Perl for C Programmers"

```
00005 fargptr = 0;
00006
00007 $sourceFile = $ARGV[fargptr++];
00008 $outFile = $sourceFile.".out";
00009
00010 # Open the files
00011 open SRCFILE, $sourceFile or die "Cannot open source file";
00012 open OUTFILE, ">$outFile" or die "Cannot open output file";
00013
00014 $line = <SRCFILE>;
00015 while ($line != /##DELTA/) {
00016     $line = <SRCFILE>;
00017 }
00018 @tokens = split("=", $line);
00019 $deltaX = $tokens[1];
00020
00021 $line = <SRCFILE>;
00022 while ($line != /##FIRSTX/) {
00023     $line = <SRCFILE>;
00024 }
00025 @tokens = split("=", $line);
00026 $firstX = $tokens[1];
00027
00028 $line = <SRCFILE>;
00029 while ($line != /##XYDATA/) {
00030     $line = <SRCFILE>;
00031 }
00032
00033 $line = <SRCFILE>;
00034 while ($line != /##END/) {
00035
00036     @tokens = split(/ /, $line);
00037     for( $index = 0; $index <= $#tokens; $index++ )
00038         if( $index == 0 ) {
00039             $x = $tokens[$index];
```

# A Perl Tutorial

Modified from: Nano Gough  
<http://www.computing.dcu.ie/~ngough/perl/tutorial.ppt>

## Perl Tutorial

- Running Perl
- Printing
- Scalar Variables
- Operations and Assignment
- Arrays
- Loops
- Conditionals
- File handling
- Regular Expressions
- Substitution
- Split
- Associative Arrays

## Running Perl

- `#!/usr/local/bin/perl` ( tells the file to run through perl)
- Use `.pl` extension
- `perl programName` (to run the program)
- `perl -d programName` (to run using debugger)
- `perl -w programName` (to run with warnings)

## Printing

#The hash symbol (#) is use to comment lines of code  
; Every statement in perl ends with a semi-colon (;)

```
print "Hello World. I love perl.";
```

```
#prints: Hello World.I love perl.
```

```
print "Hello World\nI love perl\n";
```

```
#prints:
```

```
Hello World.
```

```
I love perl.
```

## Scalar Variables

### Strings/Numbers:

```
$name = "mary";
```

```
$age = 27;
```

```
$income = 1_000_000; # underscores are ignored in numbers
```

### Operations and Assignment

```
(* multiplication) (\ division) (- subtraction)
```

```
$a = 1 + 2; # Add 1 and 2 and store in $a
```

```
$a = 5 % 2; # Remainder of 5 divided by 2
```

```
++$a; # Increment $a and then return it
```

```
$a++; # Return $a and then increment it
```

```
--$a; # Decrement $a and then return it
```

```
$a--; # Return $a and then decrement it
```

## Operations and Assignment contd..

```
$a = 5; $b=7;
```

```
$a = $b; # Assign $b to $a ($a=7)
```

```
$a += $b; or $a=$a+b; # Add $b to $a ($a=12)
```

```
$a -= $b; or $a=$a-$b; # Subtract $b from $a ($a=-2)
```

### Concatenation

```
$a = 'Monday'; $b='Tuesday';
```

```
$c=$a." ".$b;
```

```
$c= 'Monday Tuesday'; # Single quote: "don't do anything to string"
```

```
$d= "$a and $b"; # Double quote: "interpret any variables in string"
```

```
print $d; # prints 'Monday and Tuesday';
```

## Arrays

### ▪ Initialize an array/set to null

```
@colors=();
```

### ▪ Functions *push* and *pop*

```
#assign elements to array @colors
```

```
@colors=("red","blue","yellow");
```

```
#use push function to add an element to the end of array
```

```
push(@colors,"green");
```

```
#colors now contains:
```

```
"red","blue","yellow","green"
```

```
#use pop function to remove an element from the end of array
```

```
pop(@colors);
```

```
#colors now contains
```

```
"red", "blue", "yellow"
```

### **#Functions *shift* and *unshift***

```
@colors=("red","blue","yellow");  
$new_el="green";  
#use unshift to append $new_el to start of array  
unshift(@colors, $new_el);  
@colors is now:  
"green","red","blue","yellow"  
  
#use shift to remove an element from the front of array  
shift(@colors);  
@colors is now:  
"red","blue","yellow"
```

### ▪ **Accessing an element of the array**

```
@colors = ("red","blue","yellow");  
print "$colors[0]" #prints: red  
  
$#colors #index of last element of array  
print "$colors[$#colors]"; #prints: yellow  
  
print @colors #prints: redblueyellow  
print "@colors" #print: red blue yellow  
  
$colors = "@colors"; #assigns colors to string  
print $colors; #prints: red blue yellow
```

## Loops

#Loops can be used to iterate through elements of an array

### ▪ foreach Loop

```
foreach $el (@colors)
{
    print "The color is : $el\n";
}
```

#The foreach loop iterates through the array element by #element. In #the first iteration \$el is assigned the value of the first element of #colors (ie; red) etc..

#The result of executing this foreach statement is:

The color is : red

The color is : blue

The color is : yellow

## Testing

### Numbers

`$a == $b` # Is \$a numerically equal to \$b?

# don't use `$a=$b` as this will not compare but just assign \$b to \$a

`$a != $b` # Is \$a numerically unequal to \$b?

`$a < $b / $a > $b` # Is \$a less than/greater than \$b

`$a <= $b / $a >= $b` # Is a less than or equal to/ g.t or eq to \$b

### Strings

`$a eq $b` # Is \$a string-equal to \$b?

`$a ne $b` # Is \$a string-unequal to \$b?

#You can also use logical and, or and not:

`($a && $b)` # Is \$a and \$b true?

`($a || $b)` # Is either \$a or \$b true? !(\$a)

## Loops contd...

### ▪ For Loop

```
for($i=0;$i<=$#colors;$i++)
{
    print "The color is : $colors[$i]\n";
}
```

### ▪ While Loop

```
$i=0;
while($i<=$#colors)
{
    print "$colors[$i]\n";
    $i++;
}
```

## Conditionals

```
#if $a is equal red print the color is red
If($a eq `red`) { print "the color is $a\n";}
#in any other case (if $a not equal to red) print $a is not red
else { print "The color $a is not red\n";}
```

```
#if $a is equal to 1 , add 2 to $a
If($a ==1){ $a = $a+2;}
#elsif $a is equal to 2, add 3 to $a
elsif ($a ==2) {$a = $a+3;}
#in any other case add 1 to $a
else { $a++;}
```

```
#if $a is equal to 1 AND $b is equal to red: print Color 1 is red
If(($a==1)||($b eq `red`)){print "Color $a is $b\n";}
```

### Some other string comparison operators:

eq - equality

ne - not equal

lt - less than

le - less than equal to

gt - greater than

ge - greater than equal to

### File handling

#### ▪ Opening a file

```
$filename = "MyFile.txt";  
open(FILE, "/data/MyFile.txt") || die ("Cannot open file MyFile : $!\n");
```

**File:** Filehandle for MyFile.txt

**Die:** If the file cannot be opened for reading the program will 'die' (ie quit execution) and the reason for this will be returned in **#!**

The above file has been opened for reading : `open(FILE, "filename");`

▪ To open a file for writing: `open(FILE, ">OutFile.txt");`

Outfile.txt will be overwritten each time the program is executed

▪ To open a file for appending: `open(FILE, ">>Append.txt");`

▪ Close File: `close(FILE);`

### File processing

```
#open input file for reading
open(IN,"InFile.txt")|| die "Can't open file...$!\n";
#open output file for writing
open(OUT,>"OutFile.txt")|| die "Cant open file...$!\n";

while(<IN>) #while there are still lines in InFile.txt
{
    $line=$_; #read in the lines one at a time
    chop($line); #remove end of line character
    #if $line meets conditional print to OutFile.txt
    if($line eq "Number 7")
    {
        print OUT "$line\n"; } #endif
}#endWhile
close(IN); close(OUT); #close Files
```

### Regular expressions

#A regular expression is contained in slashes, and matching occurs with the =~ operator.

#The following expression is true if the string *the* appears in variable \$sentence.

```
$sentence =~ /the/
```

#The RE is case sensitive, so if \$sentence = "The quick brown fox"; then the above match will be false.

```
$sentence !~/the/ (True) because the (lower case) is not in $sentence
```

#To eliminate case use *i*

```
$sentence =~!/the/i; (True) because case has been eliminated with i
```

**These Special characters can be used to match the following:**

. # Any single character except a newline  
^ # The beginning of the line or string  
\$ # The end of the line or string  
\* # Zero or more of the last character  
+ # One or more of the last character  
? # Zero or one of the last character

\s+ (matches one or more spaces)  
\d+ (matches one or more digits)  
\t (matches a tab)  
\n (matches a new line)  
\b (matches a word boundary)

test.txt:

```
open(FILE,"test.txt");  
while(<FILE>)  
{  
    $line=$_;  
    chop($line);  
    if($line !~ /this/i)  
    {  
        print "$line\n";  
    }  
}  
close(FILE)
```

```
#  
100 This matches.  
200 This does not  
That does not.
```

Output:

```
#  
That does not.
```

### An Example using RE's

**TASK :** We have a file containing lines in different formats. We want to pick out the lines which start with a digit and end in a period.

```
open(FILE,"test.txt");
while(<FILE>)
{
    $line=$_;
    chop($line);
    if($line =~ /^\\d+(.*)\\.$/ )
    {
        print "$line\\n";
    }
}
close(FILE)
```

- `^\\d+` (specifies that \$line must begin with a digit)
- `(.*)` This digit can be followed by any character any no. of times
- `\\.`  This is followed by a period (The slash is included to make the `'.'` literal )
- `$`. This specifies that the previous character `('.')` must be the last on the line

```
open(FILE,"test.txt");
while(<FILE>)
{
    $line=$_;
    chop($line);
    if($line =~ /^\\d+(.*)\\.$/ )
    {
        print "$line\\n";
    }
}
close(FILE)
```

test.txt:

```
#
100 This matches.
200 This does not
That does not.
```

Output: 100 This matches.

### RE's contd

- [a-z] (matches any lower case letter)
- [a-zA-z] (matches any letter)

In the previous example a line was matched under the following condition:

```
if($line =~ /^(\d+)(.*)\./)
```

**The RE would match the line:** 10 people went to the concert.

(\d+) = 10; This is assigned to default variable \$1

(.\*) = "people went to the concert"; This is assigned to \$2

Perl groups the elements specified by () together and assigns it a default variable: \$1,\$2...etc.

```
print "$1\n"; # prints : people went to the concert
```

### Substitution

**#substitution is a useful facility in perl which can be used to replace one element with another**

#replaces all instances of lafayette (lc) in \$sentence to Lafayette (uc);

```
$sentence =~ s/lafayette/Lafayette/;
```

#replaces all instances of red in \$sentence to blue

```
$sentence =~ s/red/blue/;
```

#### Example

```
$sentence = "the red and white dress";
```

```
$sentence =~ s/red/blue/;
```

```
$sentence is now = "the blue and white dress"
```

## Split

**#split is a useful function : splits up a string and puts it on an #array**

```
$example = "Luke I am your father";  
@name=split(/\s+/, $example);  
@name = "Luke", "I", "am", "your", "father"
```

**#using split you can also assign elements to variables**

```
$name = "Luke:Skywalker";  
($first_name, $surname)=split(/\:/, $name);  
$first_name = "Luke";  
$surname="Skywalker";
```

## Associative arrays / hashes

The elements of associative arrays have keys with associated values

### •Initialize

```
%Mygrades=();
```

### •Assign elements

```
$Mygrades{'CHM696'}=90;  
$Mygrades{'CHM621'}=70;  
$Mygrades{'CHM624'}=50;
```

### •Printing

```
while(($key,$value)=each(%Mygrades))  
{print "$key => $value\n";}
```

*ptkdb has trouble with each()*

Prints:

```
CHM696 => 90  
CHM621 => 70  
CHM624 => 50
```

## Plain Old Documentation (POD)

```
# The lines from here to the =cut are part of
# Perl's internal documentation system. If you
# want view the documentation use the
# command:
#   pod2text <script>
#
=pod

=head1 NAME

test_ave.pl - Test the moving average program

=head1 SYNOPSIS

    perl test_ave.pl

=head1 DESCRIPTION

The I<test_ave.pl> reads a series of numbers
from I<num.txt>
and print a moving average spanning three data
points.

=head1 EXAMPLES
```

Sample run:

```
perl test_ave.pl
Read 64 items
Moving average
4.596666666666667
4.606666666666667
4.64
4.676666666666667
4.71
...
```

```
=head1 AUTHOR

Steve Oualline,
E<lt>oualline@www.oualline.comE<gt>.

=head1 COPYRIGHT

Copyright 2002 Steve Oualline.
This program is distributed under the GPL.

=cut
```

## Sub-routines

```
my @raw_data = ();
my $flag = read_data("num.txt");
if (not $flag) {
    die("Could not read file");
}
```

```
sub read_data($)
{
    my $file = shift; # The file to read
    open DATA_FILE, "<$file" or return (undef);
    @raw_data = <DATA_FILE>;
    chomp(@raw_data);
    close(DATA_FILE);
    return (1); # Success
}
```

```

sub moving_average($)
{
  my $increment = shift; # Increment for average
  my $index;           # Current item for average
  my @result;         # Averaged data

  for ($index = 0;
      $index <= $#raw_data - $increment;
      $index++) {
    my $total = sum(@raw_data[$index..$index + $increment - 1]);
    push (@result, $total / $increment);
  }
  return(@result);
}

```

here a selection of  
the @raw\_data  
array is passed to  
sum() using the  
.. operator

```

sub sum(@)
{
  my @array = @_;      # The array to sum
  my $the_sum = 0;     # Result so far

  foreach my $element (@array) {
    $the_sum += $element;
  }
  return ($the_sum);
}

```