Perl Tutorial

based on a tutorial by Nano Gough
http://www.computing.dcu.ie/~ngough/perl/tutorial.ppt

Why Perl?

- Perl is
  - Pathologically Eclectic Rubbish Lister
  - the duct tape of the internet
  - the Swiss-army chain saw of UNIX
- good at
  - text processing
  - rapid development
  - flexibility
  - operating system stuff
    - in particular UNIX/LINUX
  - code re-use
    - CPAN:large repository of re-usable modules
- bad at
  - numeric processing
  - debugging
  - efficiency

The Three Virtues of a Good Programmer
(not necessarily of a good Student)

- LAZINESS:
  - The quality that makes you go to great effort to reduce overall energy expenditure.
  - makes you want to re-use other people's code
- IMPATIENCE:
  - The anger you feel when the computer is being lazy.
  - makes you get things done quickly (rapid prototyping) and efficiently (optimize code)
- HUBRIS:
  - Excessive pride.
  - makes you want to show off (code sharing) and write (and maintain) programs that other people won't want to say bad things about.

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

Examples:

$name = ‘mary’;

$age = 27;

- scalars store a single value
- regardless of type (there are no types)
- scalar variables always start with a '$'

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';
$d= $a . ' and ' . $b;
$d=‘Monday and Tuesday’;
```

Interpolation

```
# double quotations may include vars
$c= “$a $b”;
# c is now 'Monday Tuesday';
$d= “$a and $b”;
# $d is now 'Monday and Tuesday';
```

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
```

There are no Boolean values

- false are
  - the empty string ''
  - the number 0
  - undefined value undef
  - empty list
- everything else is true

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)
```

Conditionals

```
#if $a is equal red print the colour is red
If($a eq ‘red’) { print “the colour is $a\n”;}
#in any other case (if $a not equal to red) print $a is not red
else { print “The colour $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 $b is equal to red: print Colour 1 is red
If(($a==1) || ($b eq ‘red’)){print “Colour $a is $b\n”;}
```

Arrays

- **Initialize an array/set to null**

```
@colours=();
```

- **Functions *push* and *pop***

```
#assign elements to array @colours
@colours=("red","blue","yellow");
#use push function to add an element to the end of array
push(@colours,"green");
#colours now contains:
"red","blue","yellow","green"
#use pop function to remove an element from the end of array
pop(@colours);
#colours now contains
"red", "blue", "yellow"
```

#Functions *shift* and *unshift*

```
@colours=(“red”,”blue”,”yellow”);
$new_el=“green”;
#use unshift to append $new_el to start of array
unshift(@colours, $new_el);
```

@colours is now:

“green”,“red”,”blue”,”yellow”

```
#use shift to remove an element from the front of array
shift(@colours);
```

@colours is now:

“red”,”blue”,”yellow”

▪ Accessing an element of the array

```
@colours = (“red”,”blue”,”yellow”);
print “$colours[0]”;     #prints: red

#  $#colours points to index of last element of array @colours
print “$colours[$#colours]; #prints: yellow

  print  @colours;           #prints: redblueyellow
  print “@colours”;          #print: red blue yellow

  $colours = "@colours"; #assigns colours to string
  print $colours;              #prints: red blue yellow
```

Loops

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

- **Foreach Loop**

```
foreach $el (@colours)
{
        print “The colour 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 #colours (ie; red) etc..

#The result of executing this foreach statement is:

```
The colour is : red
The colour is : blue
The colour is : yellow
```

Loops contd…

- **For Loop**

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

- **While Loop**

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

Can also be written as:
`$i < @colours`

Explanation:

In a "scalar context" (whenever the parser expects a scalar) an array is interpreted as the number of elements contained in it

Split

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

$example = “My name is Nano Gough”;

@name=split(/\s+/,$example);

@name = “My”, “name”, “is”, “Nano”, “Gough”
```

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

$name = “Nano:Gough”;

($first_name, $surname)=split(/\:/,$name);

$first_name = “Nano”;

$surname = “Gough”;
```

Associative arrays / hashes

The elements of associative arrays have keys with associated values

- **Initialize**

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

- **Assign elements**

```
$Mygrades{‘english’}=80;
$Mygrades{‘irish’}=70;
$Mygrades{‘maths’}=50;
```

- **Printing**

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

Prints:

```
english => 80
irish => 70
maths => 50
```

File handling

- **Opening a file**

$filename =“MyFile.txt”;

open(FILE,"/users/capg/ngough/perl/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,”…..);

- 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)

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 full stop, but remove the digit from the beginning of these lines

```
while(<FILE>)
{
        $line=$_;
        chop($line);    # removes a newline at the end of the line
        if($line =~ /^\d+(.*\.)$/)
        {print “$1\n”;}
}
```

- ^\d+ (specifies that $line must begin with one or more digits)
- () are used for grouping and remembering parts of the RE
- .* This digit can be followed by any character any no. of times
- \. This is followed by a full stop (The slash is included to despecialise the ‘.’)
- $. This specifies that the previous character (‘.’) must be the last on the line
- $1 contains anything that has matched between the first pair of ()

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; (.*) = “people went to the concert”;

Perl groups the elements specified by (.*) together and assigns it a default variable name : $1;

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 the first instance of london (lc) in $sentence to London (uc);
$sentence =~ s/london/London/;

#replaces all instances (because of g) of red in $sentence to blue
$sentence =~ s/red/blue/g;
```

Example

```
$sentence= “the red and white dress”;
$sentence =~ s/red/blue;
# $sentence is now = “the blue and white dress”
```

Some on-line Perl Tutorials:

http://www.comp.leeds.ac.uk/Perl/start.html

http://archive.ncsa.uiuc.edu/General/Training/PerlIntro/

http://www.pageresource.com/cgirec/index2.htm

Text books:

Perl cookbook; Tom Christiansen and Nathan Torkington

Programming Perl; Larry Wall, Tom Christiansen, and Randal L Schwartz