Pathologically Eclectic Rubbish Lister

Author: Reuben Francis Cornel

1 Perl Design Philosophy

- PERL is an acronym for Practical Extraction and Report Language. But I guess the title is a rough translation of that.
- PERL was developed by Larry Wall, in 1983, because he felt lazy to use other tools.
- PERL today gets is shape due to the various anthropological and linguistic studies which have affected its founder in the past.

2 Running PERL Programs

- A PERL program can be run in two ways.
  - **Command Line:** You invoke the PERL interpreter and the file name which has the script.

    `$ perl` *filename*
  - **Standalone Script:** This program is run directly of the command line. But you have to call the PERL interpreter in the interpreter line.

    `$ chmod +x` *filename*
    `$ ./`*filename*

3 Basic Syntax Overview

- All statments in PERL end with a semicolon. For example

```
print "Hello World";
```

- Comments start with a `#` and this extends to the end of the line.

```
#This is a comment
```

- Whitespaces are irrelevant in statements, but not inside quotes.

4 PERL Variable Naming Syntax

- The first character of the variable name tells you the type of variable.
- The rest of the variable name is a user defined. This can start with an underscore or a letter, and then can be followed by any letters, underscores or digits.
- The case of the variables is significant.

5 Scalars

- Scalar data is the most basic datatype in perl. It can either be a number or a string. We shall look at how strings and numbers are represented before actually moving on to use them in scalar variables.

5.1 String Representation

- We start by discussing about string literals in Perl. Perl provides us with two ways to represent string literals.
  - **Single Quoted Strings**: Here most special characters are not interpreted. Variable values are not substituted. Only the "'" and the back slash character have to be escaped
  - If you want a new line character in single quotes just type it in.
  - **Double Quoted Strings**: These are string literals which are delimited by ". The advantage of using these are you can interpolate the values into a string.
  - All special characters have to be escaped if you want them to be displayed when you use double quotes. These characters can be escaped using the backslash character.

5.2 Number Representation

- Representation of numbers in perl is very much like the number represenations in C. Unlike strings, numbers don't have to be quoted.
- Apart from the normal representations. Perl also allows you to represent numbers in exponential format. For example .005 can be represented as 5E-3

5.3 Scalar Variables

- In perl both numeric and string values can be stored in scalar variables.
- Scalar Variables are created just by assigning values to names. But it must be remember that **all scalars are prefixed by a $**.
- Scalar variable names are case sensitive.
- But what if a variable is accessed but no value has been assigned to it?
- As you can see this could be a source of bugs. If you want to find out undefined variables, you have to enable perl's warning messages. This is done by adding `-w` to the command line.
- We can check if a variable is defined or undefined using the `defined` function call. This returns 1 if the scalar is defined else a blank string represented by "".

5.4 The Default Variable

- The default variable in perl is an implicit scalar variable which is defined by the perl interpreter automatically.
- It obtains its values based on the context of the situtation. This can become a cause for a lot of misunderstanding.
- **Please do not use this variable if you expect other people to read your code**
- The default variable can be accessed using $_.

6 Operators

- Perl's scalar operators are almost similar to operators provided by most programming languages. The table below summarises the mathematical operators.

Operator	Meaning
Mathematical Operators	
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
**	Exponentiation
++	Auto increment
--	Auto decrement

- Comparing scalars in perl is as simple as comparing values in any other language. But since there is automatic conversion between strings and numbers you must tell perl which type of comparison you want to use. The table below assumes that the comparision you are going to perform is of the format `$left` <OP> `$right`, where <OP> is the operator.

Operation	**Numeric Version**	**String Version**	**Returns**
less than	<	lt	1 iff. $left is less than $right
less than or equal to	<=	le	1 iff. $left is less than or equal to $right
greater than	>	gt	1 iff. $left is greater than $right
greater than or equal to	>=	ge	1 iff. $left is greater than or equal to $right
equal to	==	eq	1 iff. $left is the same as $right
not equal to	!=	ne	1 iff. $left is not the same as $right
compare	<=>	cmp	-1 iff. $left is less than $right, 0 iff. $left is equal to $right 1 iff. $left is greater than $right

- Perl has two string operators, one for concatenation, one for string duplication.
  - The "." operator is the string concatention operator.
  - "x" is the duplication operator.

7 Arrays

- Arrays in perl are conceptually closer to the lists of LISP, but syntax used is like the syntax used by C.
- Perl arrays are dynamic. That is they grow and shrink as required.
- Perl arrays are accessible just like C arrays. So you can subscript anywhere directly. As a result you have advantages of a dynamic list as well as a static array.
- Since items of a list are scalars, there is no distinction between numeric and string values.

7.1 List Literals

- Like scalar literals it is also possible to write list literals in your code. Ofcourse inserting string literals into these list literal require proper quoting.
- There are two primary ways to quote list literals
  - () - This is the basic way to write list literals. When writing elements using this all string have to be double/single quoted. Numbers as usual don't have to be quoted.
  - **qw** - This is called the *quoting operator*. This operator has a slightly funny syntax. The quoting operator is followed by a "stop character". It will keep on eating elements in a list till it reaches the next "stop character". The advantage of this is you do not need to quote strings in any additional way since `qw` does it for you.

Example	**Meaning**
();	# this list has no elements; the empty list
qw//;	# another empty list
("a", "b", "c", 1, 2, 3);	# a list with six elements
qw/hello world how are you today/;	# another list with six elements

  - Finally we have a slight misfit here, its the ".." operator. This operator helps build list from initial values.

Example	**Meaning**
(1 .. 100);	# a list of 100 elements: the numbers from 1 to 100
('A' .. 'Z');	# a list of 26 elements: the uppercase letters From A to Z
('01' .. '31');	# a list of 31 elements: all possible days of a month

7.2 Array Variables

- Variable which have list literals assigned to them are called array variables. Array variables are represented by preceding "@" symbol.
- You can also concatenate a number of lists simply by listing them together and using comma as the delimiter
- When you create a new array, a set of scalar variables get created automatically, first of all for a list of n elements. There are scalar variables $array[0], $array[1], ... ,$array[n-1] that contain the first, second, ... nth variable elements in the array.
- Another scalar variable is associated with an array variable, `@array`, is `$#array`. This gives the number of elements of the array. This scalar like any other scalar can be modified, using appropriate operations. But make sure, that you don't set it to -1, this would delete your list.

7.3 Working with Arrays

7.3.1 Slicing Arrays

- Say for some reason you are hungry and want a section of cake what would you do? cut a slice of the cake isn't it.
- In the same way when you want to create a list which has a subsection of elements of a bigger list, you slice the bigger list to get a piece of it.
- The [] operators help you slice an array and get a section of with, ofcourse that is generally with the help of the .. operator.

7.3.2 Array Functions

- There are a set of functions that work on arrays, the most basic ones are listed below.
- **push**: Pushes an element at the top of the list.
- **pop**: Pops an element from the top of the list.
- **unshift**: Adds an element at the end of the list
- **shift**: Removes an element from the end of the list.
- **split**: Splits a string into an array based on a specified seperator.
- **join**: Join an array into a single string based on seperator.

7.4 Array Miscellany

7.4.1 Context

- It may have occurred to you by now that in certain places we can use a list, and in other places we can use a scalar. Perl knows this as well, and decides which is permitted by something called a context.
- There are two types of contexts
  - List Context
  - Scalar Context
- Perl knows which context is to be used when that is why we can use variables in different places and get different results.

```
@things = qw/a few of my favorite/;
$count = @things;
@moreThings = @things;
```

7.4.2 Array Interpolation

- An array can be interpolated with in a double quoted string, the result is same as if any other scalar would have been interpolated.

7.4.3 The Default Array

- This creature is similar to the default variable, again this is a context based variable, hence avoid using it when you can. Unlike other variable, if you have to access a subscript of the default array you would have to access the subscript as shown $_[n]. Where n is the subscript number.

8 Control Structures

8.1 Truth Values in Perl

- Almost all control structure work on boolean values returned by expressions. But which of these values are true and which are false?
- Everything in perl is true except.
  - The strings “” (the empty string) and “0” (the string containing only the character, 0), or any string expression that evaluates to either “” (the empty string) or “0”.
  - any numeric expression that evaluates to a numeric 0.
  - any value that is not defined (i.e., equivalent to undef).
- Try out the below exercise

Expression	Boolean value
0	
0.0	
””	
”0”	
”0.0”	
undef	
42 - (6 * 7)	
”0.0” + 0.0	
”foo”	

8.2 If Construct

- This is the simple if construct provided by most programming languages. Its syntax is as shown.

```
if (expression) {
  execute this code
} elsif (another_expression) {
  execute this code
} else {
  execute this code
}
```

8.3 While Construct

- The syntax of the while construct is shown below.

```
while (expression) {
execute this code;
}
```

8.4 Do While Construct

- The syntax of the while construct is shown below.

```
do {
   execute this code
   } while (expression);
```

8.5 For Construct

- The syntax of the for construct is shown below.

```
for(Initial_Statement; expression; Increment_Statement) {
  execute this code
}
```

8.6 Foreach Construct

- The foreach structure takes a scalar, a list and a block, and executes the block of code,setting the scalar to each value in the list, one at a time.

9 Associative Arrays(Hashes)

- The associative array more popularly known as the hash tables is the third natively supported datatype of perl.
- Unlike scalars and arrays, we don't have literals for hashes.
- A hash is demarkated by a % symbol preceding the identifier.
- Unlike arrays hashes can be subscripted by arbitary scalar values, we use { } to subscript values.

9.1 Hash Functions

9.1.1 Keys and Values

- The `keys` function gives a list of all keys in a hash
- The `values` function gives a list of all values in a hash

9.1.2 Each

- This returns a key-value pair from the hash.

10 Regular Expressions In Perl

- Perl supports most of the regular expression syntaxes supported by UNIX. It has full support for Basic, Interval and Tagged Regular expressions.

10.1 Regular Expression Operators

- =˜This operator is used check if a given pattern exists in a string. The pattern to the searched is given in the string which is surrounded by two forward slashes.
- When we use tagged regular expressions, the values which are extracted from the string are stored in scalar variables which denoted by $1, $2 .... $n, where n is the number of the tag.

10.2 Regular Expression Functions

- The `s` function. This function performs the job on a scalar variable what sed perform on a line, in simple words substitution.
- The `tr` function. This function performs the job done by the `tr` command.

11 File Handling

11.1 Opening a file for reading

- For opening a file Perl provides the `open` function. The syntax of the function is as shown below.

  `open (`*`File Handle, Filename`*`)`

- A file is opened for reading as shown below. INFILE here is the file handle. This file handle will be used by functions to read and write to a file.

```
open (INFILE, ‘‘/home/reuben/tempfile’’);
```

- A file is opened for writing with shell like operators >, >>, having their usual meanings.

```
open (OUTFILE, ‘‘>/home/reuben/tempfile’’); #Overwrite
open (OUTFILE, ‘‘>>/home/reuben/tempfile’’); #Append
```

- To read from a file handle do the following.

```
$variable=<FILEHANDLE>;
```

- To write to a file do the following.

  `print FILEHANDLE` *`scalar`*`;`

- By default, the standard input file handle is represented by STDIN. Therefore if you want to read anything from the input all you have to do is. But sometimes the word STDIN might be missing but it will still mean the same thing.

```
$variable=<STDIN>;
```

- The EOF is signified by blank string.
- Filehandles can be closed using the `close` command. Syntax is as follows `close FILEHANDLE;`

12 Subroutines

- Until now, all the Perl programs that we have written have simply a set of instructions, line by line. Like any good language, Perl allows one to write modular code. To do this, at the very least, the language must allow the programmer to set aside subroutines of code that can be reused. Perl, of course, provides this feature.

12.1 Defining Subroutines

- Defining a subroutine is quite easy. You use the keyword sub,

  followed by the name of your subroutine, followed by a code block.

  `sub` *function name*{ *C*ode to be executed }

- When calling the function the function call may or may not be prefixed with a "&" symbol.
- Arguments to functions are passed using the default array. There are 2 ways to extract these arguments.
  - Use the `shift` function on the default array.
  - Try getting the values from the default array directly.
- The return keyword is used to return arguments. You can use return to return any possible legal value.

13 Perldoc

- A complete installation of perl comes with its inbuilt documentation, this is called Perl Documentation or PerlDoc in short.
- Most linux systems come with a copy of Perldoc installed. To start perldoc, all you have to do is type `perldoc perltoc` at the command line.
- You can select a section of the documentation by typing `perldoc` *section name*.

14 Conclusion

- Perl in it self is a HUGE langauge. Trying to cover it is quite a challenge.
- The things which have been ignored in this presentation are Perl's features for network communication and object orientation, apart from that a whole lot of packages.
- The advantage of using perl is it provides a way for both the novice and the guru to write code in more than one way, hence true to its slogan "There is more than one way to do it."