# A Couple Notes on Perl

Birrabenzina

February 10, 2022

## Contents

# 1 Introduction

## 1.1 Script Header and References

The script header must ALWAYS have:

```
#!<PERL PATH>
use warnings;
use strict;
use Data::Dumper;
```

As for bash, there MUST be a shebang, the other three options will (probably) explained later

## 1.2 Acronyms

- PERL: Practical Extraction and Report Language, or Petty Eclectic Rubbish Lister
- CPAN: Comprehensive Perl Archive Network, see `www.cpan.org`
- DWIM: Do What I Mean, the philosophy of perl, basically
- AUTOVIVIFY (n.), AUTOVIVIFICATION (v.): To bring "oneself" to life, part of perl's dwimsiness
- TMTOWTDI: There is More Than One Way To Do It, as for every language

## 1.3 Storage

Perl has 3 storage types

- Scalars
- Arrays
- Hashes

Arrays and hashes are simply more complex scalars

# 2 Scalars

All scalars are preceded by a dollar sign in declaration and call.
A scalar can be a string, a number (int or float), a reference or a filehandle. Perl uses type inference.
Every scalar is declared with the `my` reserved keyword. These are all valid declarations:

```
my $intnum = 69;
my $floatnum = 69.420;
my $chartype = 'a';
my $stringtype = 'Glory to the Programming Republic of Perl!';
my $reftostr = \$stringtype;
```

N.B. Without '`use strict`' and without '`my`', the variable gets initialized either to " " or 0. This is an example of *autovivification*. E.g.

```
my $pie = 3.14;
my $diameter = 42;
my $circumference = $pi * $diameter;
```

Here you get that, since `$pi` isn't declared ANYWHERE, it autovivifies into '0' and the result is 0!. This doesn't happen if you declare at the beginning of the script

```
use warnings;
use strict;
```

# 3 Strings (Scalar)

As we said, scalars can store strings. If you look closely at the previous snippets of code, you don't need to declare the length of the string, perl gets it automatically.

## 3.1 String Literals

String literals MUST be in single or double quotes, the difference being that:

- Single quotes literally give you a "what you see is what you get" value
- Double quotes include some variable interpolation during evaluation.

## 3.2 Functions and Operators on Strings

- `chomp($string)`: removes the last \n char from the string, if it is there
- `$string1.$string2` (concatenation): Concatenates 2 strings together
- `$string x(number)` (repetition): Repeats $string *number* times
- `length($string)`: Gets the length of the string
- `substr($string,$offset,$length)`: Gets a substring of $string, starting *offset* chars and endings after *length* chars
- `split(/REGEXP/,$string,$limit)`: Breaks the string into pieces using regexps. Searching for empty strings divides it in chars
- `join('$separatorstr',$string1,$string2)`: Stitch multiple strings into one
- `qw($string)`: Quotes automatically $string, with single quotes

# 4 Numbers (Scalar)

Perl generally uses floats for storing scalar numbers. If you use integers, it will use integers, i.e.

```
my $funnyint = 69; # scalar => int
my $funnyfloat = 69.420 # scalar => float
```

## 4.1 Numeric Literals

Perl allows: ints, floats, scientific notation, decimal, octal, hex, as:

- Bins, beginning with '0b'
- Hexes, beginning with '0x'
- Octs, beginning with '0'
- Decs, the remaining

e.g.

```
my $highnumber = 3.7e8; #sci-dec
my $octal = 03672673; #oct
my $hexnum = 0xfb214e9; #hex
my $binnum = 0b10000111100101100; #bin
```

## 4.2 Numeric Functions

- `abs(number)`: absolute value
- `int(number)`: casts floats to ints
- `sin, cos, tan`: usual trig functions
- `base**exponent`: exponentiation macro. Use fractional numbers for roots, as usual in maths
- `sqrt(number)`: square root of a (positive) number
- `exp, log`: natural exponentiation and logarithm
- `rand(lim), srand(seed)`: return random numbers. rand() is a pseudorandom number generator (PRNG). Returns numbers $0 \leq$ `rand(num)` $\leq$ `num`. `srand(seed)` seeds rand as in C. For perl 5.004 and higher it's called automatically at the beginning of the execution of the script

Note that when printed, numbers get AUTOMATICALLY converted to strings. If you don't want the default perl version of stringification, you can still use `sprintf(…)`. It can be forced by concatenating the null string to the number as in the following snippet

```
my $notstring = 69; # > 69
my $yesstring = $notstring .= ''; # > '69'
```

### 4.3 `sprintf`

C much?
Syntax:

```
sprintf(format,$list_of_vals);
```

The format is exactly like in C, so

- `%lx`: hex
- `%lo`: oct
- `%lb`: bin
- `%ld`: int
- `%f`: float
- `%e`: scientific notation

### 4.4 Numification

WIP<++>