Perl Training

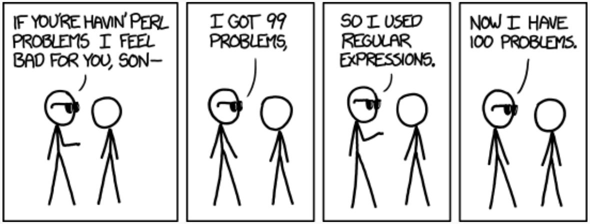
a journey to the most exotic language on Earth

Ettore di Giacinto

Material available @ `https://github.com/mudler/perl_training`

SUSE

Table of contents


IF YOU'RE HAVIN' PERL PROBLEMS I FEEL BAD FOR YOU, SON—
I GOT 99 PROBLEMS,
SO I USED REGULAR EXPRESSIONS.
NOW I HAVE 100 PROBLEMS.

Ultra-fast Perl overview

Pros:

- Huge library archive - CPAN
- Extremely flexible language
- Performs quite well to be interpreted
- Lots of functionalities

but...cons:

- TIMTOWTDI - good/bad thing
- Lot of caveats
- Difficult to deal when we want optimizations
- Not all things from CPAN are good
- Lots of functionalities which use should be discouraged! (I'm looking at you, *attributes* .. *base*)

The Life Cycle of a Perl Program

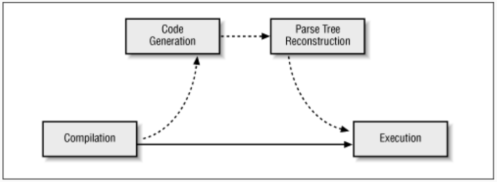
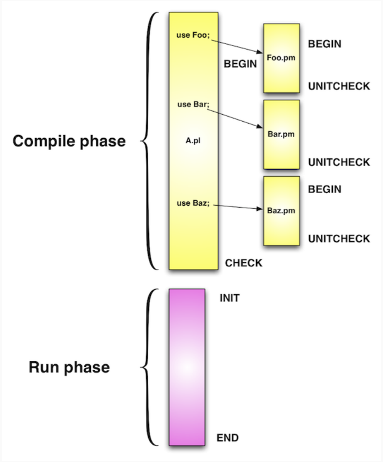


Figure 1: from https://docstore.mik.ua/orelly/perl3/prog/ch18_01.htm

- The Compilation - Tree building (**use** and **no** declarations, Lexical declaration with no assigment. BEGIN are executed in FIFO, later interpreter is called again to re-evaluate CHECK blocks in LIFO)
- The Code Generation Phase (optional) - if CHECK blocks where specified, Perl will generate intermediate C code (or Bytecode) compiling them so your machine can execute that image directly
- The Parse Tree Reconstruction Phase (optional) - if Bytecode, then Perl needs to reconstruct the Parse tree before being able to execute
- Execution - The interpreter takes the parse tree and execute it.

Perl Objects - Compiletime vs Runtime



Perl is well known for..

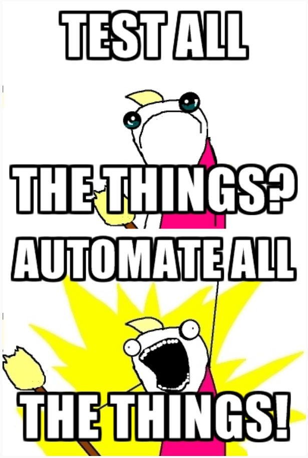
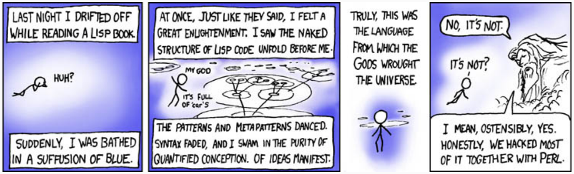


Table of contents



LAST NIGHT I DRIFTED OFF WHILE READING A LISP BOOK.
HUH?
SUDDENLY, I WAS BATHED IN A SUFFUSION OF BLUE.
AT ONCE, JUST LIKE THEY SAID, I FELT A GREAT ENLIGHTENMENT. I SAW THE NAKED STRUCTURE OF LISP CODE UNFOLD BEFORE ME.
MY GOD
IT'S FULL OF 'car'S
THE PATTERNS AND METAPATTERNS DANCED. SYNTAX FADED, AND I SWAM IN THE PURITY OF QUANTIFIED CONCEPTION. OF IDEAS MANIFEST.
TRULY, THIS WAS THE LANGUAGE FROM WHICH THE GODS WROUGHT THE UNIVERSE.
NO, IT'S NOT.
IT'S NOT?
I MEAN, OSTENSIBLY, YES. HONESTLY, WE HACKED MOST OF IT TOGETHER WITH PERL.

Perl Data structures - Overview

Perl is a pecularial language, and have different kinds of data structures from other languages, and are prefixed with sigils (**$**, **@**, **%**, **&**) :

- SCALAR - **$**
- ARRAY - **@**
- HASH - **%**
- CODE - **&**
- GLOB / Symbols - the meta-data type.

Perl Data structures - Overview

```
my $var = "foo"; $var = "1"; $var = 1 # Always scalar

my @var = ('var'); # array

my %var = (bla => boo); # hash

sub { 1 } # code

*var = \"bar" # Glob - ignore it for now
```

it is only the context holding them together ...

Table of contents

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Perl - Context

Context is the only pivot of Perl. Everything is subject to context: operators enforces context to operands, and expressions are evaluated by their context. So if we are forgetting about context, we are doing something wrong for sure.

```
sub array_or_string { wantarray ? qw(1 2 3) : 3 }

my @array  = array_or_string(); # yields 1 2 3 - list context
my $string = array_or_string(); # gives 3 - scalar context
```

Perl - Context

There are mainly 3 types of context in Perl:

- Scalar context
- List context
- void context

Void context is a special case of Scalar context.

Perl - Context

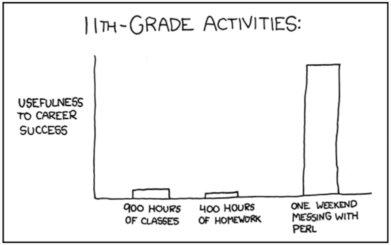
There are mainly 3 types of context in Perl:

- Scalar context
- List context
- void context

Void context is a special case of Scalar context.

```perl
sub array_or_scalar_or_void {
       !defined wantarray ? print "Void context\n"
      : wantarray         ? qw(1 2 3)
      :                     3;
}

my @array  = array_or_scalar_or_void(); # yields 1 2 3 - list context
my ($one, $two)  = array_or_scalar_or_void();
my ($one, $two, undef)  = array_or_scalar_or_void();
my $three = array_or_scalar_or_void(); # returns 3 - scalar context

array_or_scalar_or_void; # will print "Void context".
```

Table of contents



Pointers

YES! we have pointers! ...or something that looks similar to them

...not like Python ...

Pointers in Perl - References

References can be accessed by using '\'

```
my $arrayref  = \@array;
my $hashref   = \%hash;
my $scalarref = \$var;
```

Note, you can use '\' also to get reference of inline-declared variables:

```
my $scalarref = \"still valid";
```

Pointers in Perl - References

Dereferencing is the act of getting the real value out of the reference.
As always in Perl, **TIMTOWTDI**.

```
my $hashref = { foo => 'bar' } # Reference to inline declared hash

print ${$hashref}{foo}."\n";   # Will print 'bar'
print $$hashref{foo}."\n";     # this too
print $hashref->{foo}."\n";    # this as well
```

There is no general rule for deferencing, but as Perl is context dependent, as a rule of thumb you usually force a sigil context to dereference the variable type.

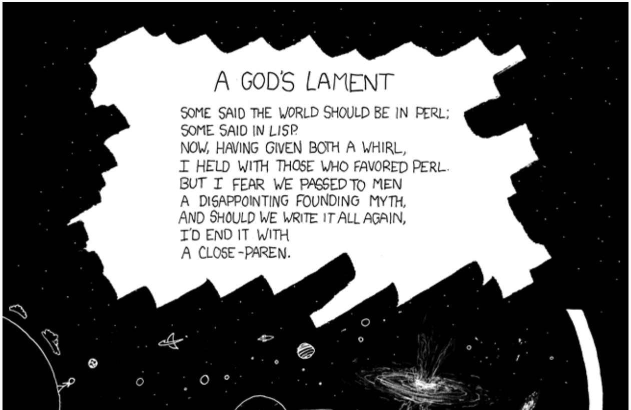
```
my $arrayref = [qw( one two tree )]; # Reference to inline array
print join(" ", @{$arrayref})."\n";  # Will print 'one two tree'

my $scalarref = \"foo"; # Reference to inline scalar
print $$scalarref."\n";  # Will print 'one two tree'
```

Pointers in Perl - References and Context

A good example is Hash slicing. Which is just forcing dereferencing a hash into an array of their values, so you can perform operations directly on the hash. It can be seen as a combination of forcing array dereferencing on part of the hash while imposing list context.

```
my $hashref = { test => { 1 => 1, 2 => 2, 3 => 3} };
@{$hashref->{test}}{qw(1 2 3)} = qw(4 5 6);
# ^^^ context forced to array.

my %test = (1 => 1, 2 => 2, 3 => 3);
# ^^^ context forced to hash
@test{qw(1 2 3)} = qw(4 5 6);

# Stolen from OpenQA :)
@{$job->{settings}}{keys %$worker_settings} = values %$worker_settings;↩
      # BAD!
# Can you tell why this example *works* but it is a bad practice?
```

Table of contents


A GOD'S LAMENT
SOME SAID THE WORLD SHOULD BE IN PERL;
SOME SAID IN LISP.
NOW, HAVING GIVEN BOTH A WHIRL,
I HELD WITH THOSE WHO FAVORED PERL.
BUT I FEAR WE PASSED TO MEN
A DISAPPOINTING FOUNDING MYTH,
AND SHOULD WE WRITE IT ALL AGAIN,
I'D END IT WITH
A CLOSE-PAREN.

Perl Objects - Introduction

Perl has a large number of libraries giving out Object system in a OOTB fashion:

- Moo
- Moose
- MOP
- Mojo::Base
- Class::* (Rabbit hole)
- even I had my own implementation based off on Mojo::Base
- ...you name it

Perl Objects - Introduction

What is MOP?

- Meta-Object Protocol
- API on top of Object
- Abstraction to normal Object model, yielding to the Reflective property

Why Perl have so many object implementation, and why everything is so complicated?

- TIMTOWTDI lead to a lot of implementations
- Every developer grabbed ”something new” from other OOP Model
- Unsatisfaction with general state of OOP in Perl
- Official MOP proposals have been attempted already to land in Perl 5, but never happened.

Perl Objects - Introduction

OOP State in Perl, key points:

- Packages != Objects
- Packages can bring Objects to you
- Basic OOP in Perl allows you to construct your own OOP Model
- Perl is not hiding nothing to you
- DO NOT create Packages that are named all lowercase (reserved to perl pragmas), or all uppercase (Built-in types)

Perl Objects - Introduction

Everything starts with the **bless** keyword.

```
bless REF,CLASSNAME
```

From Perldoc:

”This function tells the **thingy** referenced by REF that it is now an object in the CLASSNAME package” - Thus everything can be “Objectified” :)

Perl Objects - Introduction

Bless is just marking the reference belonging to a Package, thus inheriting the functions (Note: **new()** constructor is just a convention).

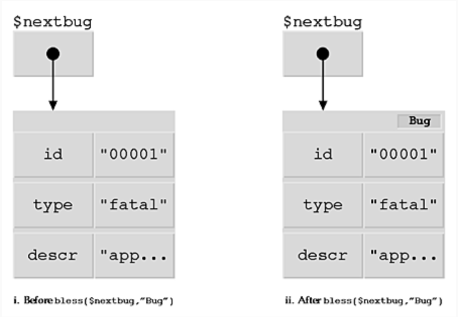


Figure 2: from: Damian Conway's Bless My Referents

That means that a function of a package can return a blessed reference of the type of the Package (thus inheriting functions).

Perl Objects - Introduction

Example Package that offer it's Objectified veriant:

```perl
{
package Foo;
sub new() { bless {}, 'Foo' }
sub newBar() { bless {}, 'Bar' }
1;
};
{
package Bar;

sub printer { shift; print "@_\n" }

!!42;
};
my $foo = Foo->new();
my $bar = Foo->newBar();
my $bar2 = $foo->newBar();
Foo->newBar()->printer("Hello");
```

Tips

- Get used to read the code from right to left.

Package modules

What makes an object inherit methods from another one?

ISA

Eeach package contains a special array called **ISA**. The **ISA** array contains a list of that class's parent classes, if any. This array is examined when Perl does method resolution, which we will cover later.

ALL CAPS

This frame uses the `allcaps` titleformat.

Potential Problems

This titleformat is not as problematic as the `allsmallcaps` format, but basically suffers from the same deficiencies. So please have a look at the documentation if you want to use it.

Table of contents

Typography

```
The theme provides sensible defaults to
\emph{emphasize} text, \alert{accent} parts
or show \textbf{bold} results.
```

becomes

The theme provides sensible defaults to *emphasize* text, accent parts or show **bold** results.

Font feature test

- Regular
- *Italic*
- SmallCaps
- **Bold**
- ***Bold Italic***
- **Bold SmallCaps**
- `Monospace`
- *`Monospace Italic`*
- **`Monospace Bold`**
- ***`Monospace Bold Italic`***

Lists

Items

- Milk
- Eggs
- Potatos

Enumerations

1. First,
2. Second and
3. Last.

Descriptions

PowerPoint	Meeh.
Beamer	Yeeeha.

Animation

- This is important

Animation

- This is important
- Now this

Animation

- This is important
- Now this
- And now this

Animation

- This is really important
- Now this
- And now this

Figures

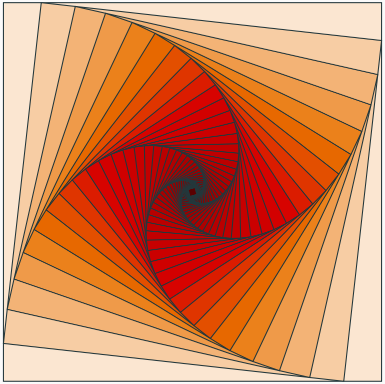


Figure 3: Rotated square from texample.net.

Tables

Table 1: Largest cities in the world (source: Wikipedia)

City	Population
Mexico City	20,116,842
Shanghai	19,210,000
Peking	15,796,450
Istanbul	14,160,467

Blocks

Three different block environments are pre-defined and may be styled with an optional background color.

Default

Block content.

Alert

Block content.

Example

Block content.

Default

Block content.

Alert

Block content.

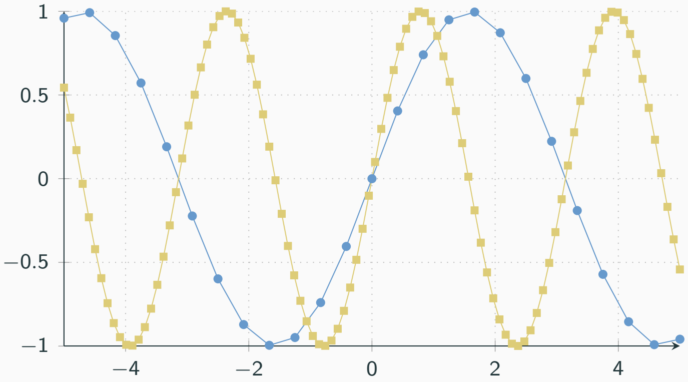
Example

Block content.

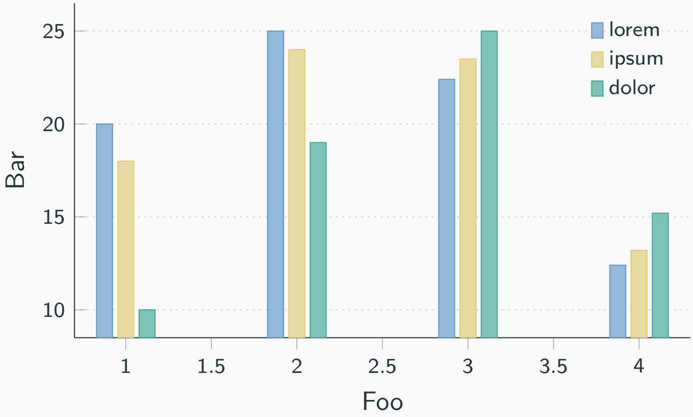
Math

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n$$

Line plots



Bar charts



Quotes

Veni, Vidi, Vici

Frame footer

metropolis defines a custom beamer template to add a text to the footer. It can be set via

```
\setbeamertemplate{frame footer}{My custom footer}
```

References

Some references to showcase [allowframebreaks] [**?**, **?**, **?**, **?**, **?**]

Table of contents

Summary

Get the source of this theme and the demo presentation from

`github.com/matze/mtheme`

Questions?

Backup slides

Sometimes, it is useful to add slides at the end of your presentation to refer to during audience questions.

The best way to do this is to include the `appendixnumberbeamer` package in your preamble and call `\appendix` before your backup slides.

metropolis will automatically turn off slide numbering and progress bars for slides in the appendix.

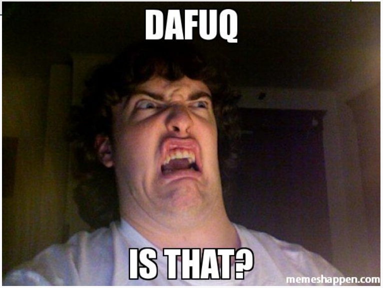
Table of contents

Globs - Automa(g/t)ic assignment

```
*var = \"test";

no strict 'refs';

print ${*{"var"}{SCALAR}}."\n";
```

Globs - Automa(g/t)ic assignment

```
*var = \"test";

no strict 'refs';

print ${*{"var"}{SCALAR}}."\n";
```



Perl - Context

Being able to manipulate and force context, allows us to have few tricks:

```
my $count = ()= @array; # Forces list context
my $count = scalar @array; # Forces scalar context
my $one = (qw(1 2 3))[0]; # Force list context and retrieve an element ↩
     in array

my $string = '42 is the right answer';
my $back_to_number =0+ $string;  # 42 (Venus operator)
$string--; # $string now is 41
my $bangbang     = !!$string;  # 1

@{[ qw(1 2 3) ]}; # Baby cart operator
```

Perl - Context - extra

And always thanks to Perl context fun, we can create arrays or hashes based on variable options easily:

```
my $dog = 1; # Try to flip them!
my $cat = 1;
my @array = (
    ('wof') x !!($dog),
    ('meow') x !!($cat)
);

$dog = 1;
$cat = 1;
my %hash = (
    (wof => $dog) x !!($dog),
    (meow => $cat) x !!($cat)
);
```

x is the string multiplicator, and the double bang (**!!**) reduces the expression in the right to a boolean.

References i