Perl 6 Pride and Envy

Czech Perl Workshop 2014
2014-05-20
Carl Mäsak

Perl 6 can be thought of as a re-mix of Perl 5:

Keeping what works
Improving what doesn't
Re-imagining Perl with hindsight

Basically,

It's a community project, a work of love and appreciation

Basically,

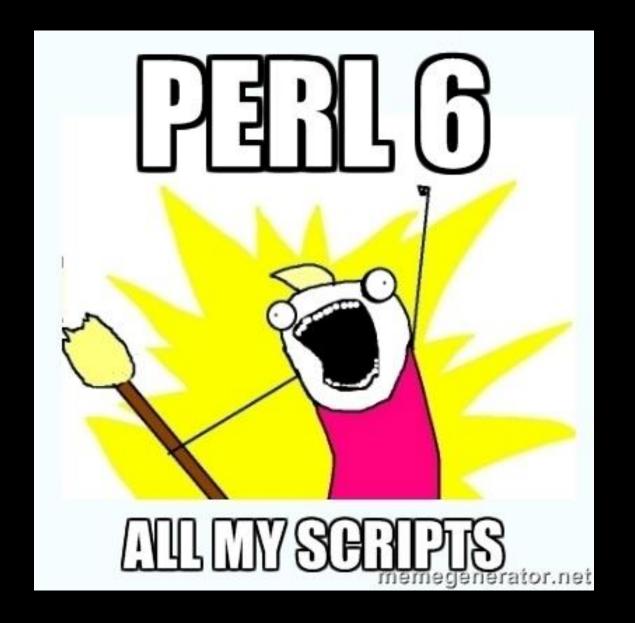
It's fanfiction

We love canon so much that we are writing our own fan work based on it

A fan work can be awesome

It can also be crap

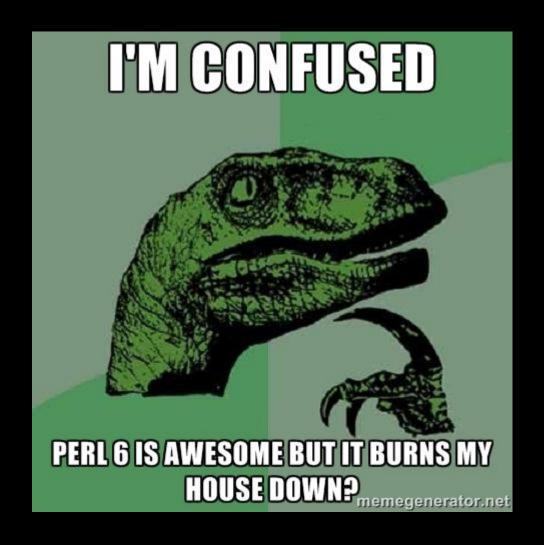
Most of the time, it's a mixture of both



(Perl 6 is awesome)



(just kidding — it's crap)



(just kidding — it's a mixture of both)

In the cases where it's awesome, we feel a sense of prine and want to tell the world how it improves on Perl 5

In the cases where it's crap, we feel a sense of Entry and want to grow up to be more like Perl 5

hi

I'm masak

A Pride

invariant sigils

Sigils that vary with use in Perl were an interesting linguistic experiment.

But the results are back: the confusion isn't worth the benefits.

So Perl 6 makes sigils invariant:

```
my %hash; %hash<foo>; %hash<foo bar>;
```

Perl 6 is more "math-y"

Some have claimed that Perl 6 is full of math, like, all academic and stuff.

It is. I'm proud of that.

```
my $sum = [+] @values;
sub postfix:<!>($n) { [*] 2..$n }
say 5!;
my $dot_product = [+] @v_1 »*« @v_2;
```

Perl 6 is more "math-y"

We should treat more things like values and functions. Values parallelize well. Functions compose and refactor nicely.

functional programming

Where Perl 5 likes to throw itself into a for loop to do things, Perl 6 tends to treat the whole collection of things.

Think map and grep, but with lots more functions like that.

Perl 6 likes to pass functions to things.

smartmatching

Smartmatch is an awesome feature in Perl 6. Perl 5 stole it back early on.

It was never so awesome in Perl 5. Useful yes, but kinda awkward. People called it "psychotic-match".

Why? The type system isn't there.

Task: collect all the letters in a text and print their frequencies.

Perl 5 solution:

```
while (<>) { $count{lc chop}++ while length }
say "$_ => ", $count{$_}//0 for 'a' .. 'z';
```

Let's start by translating the script to Perl 6:

```
my %count is default(0);
for lines() { %count{$_}++ for .comb }
say "$_ => %count{$__}" for 'a' .. 'z';
```

We don't need a for loop, we have hypers:

```
my %count is default(0);
%count{$_}++ for lines».comb;
say "$_ => %count{$_}" for 'a' .. 'z';
```

But why store non-letters and then not count them?

```
my %count is default(0);
%count{$_}++ for lines».comb.grep('a'..'z');
say "$_ => %count{$_}" for %count.keys;
```

But why store non-letters and then not count them?

```
my %count is default(0);
%count{$_}++ for lines».comb.grep('a'..'z');
.say for %count.pairs;
```

Finally, the whole notion of looping and counting things is a bit antiquated when we have the Bag type:

```
.say for lines».comb.grep('a'..'z').Bag.pairs.sort;
```

I would consider this to be an example of Concatenative Programming. Everything is strung together.

lazy lists

The default in Perl 6 tends to be lazy: elements in a sequence get computed on-demand as you require them.

The usual list functions participate in this game, and it turns into a laid-back programming style, where you stop caring if you're generating "too much".

object orientation

In Perl 5, the object-oriented parts were artfully woven into the existing design after the fact.

In Perl 6, they were integrated fully into the language, and they underpin everything in a very real sense.

object orientation

Here's a typical class:

```
class Pair is Enum {
  has $.key;
  has $.value;
}
sub infix: «=>»($key, Mu $value) {
  Pair.new(:$key, :$value);
}
```

Perl 5 is already good at manipulating strings and doing things with regexes. Its toolset is hard to beat.

Perl 6 beats it by elevating string parsing to language parsing. Every parse results in a parse tree. You can build grammars, and act on them.

Here's a typical grammar:

```
grammar HashLang {
   rule TOP { '{' [ <pair> [',' <pair>]* ','? ]? '}' }
   rule pair { <term> '=>' <term> }
   token term { ... }
}
```

Parsing separators is so common that we have the syntax % and %% for it.

```
grammar HashLang {
   rule TOP { '{' <pair>* %% ',' '}' }

   rule pair { <term> '=>' <term> }
   token term { ... }
}
```

% wants a separator between things. %% additionally allows a trailing one.

Another thing that's common is parsing start and end tokens. ~ does that.

```
grammar HashLang {
   rule TOP { '{' ~ '}' <pair>* %% ',' }

   rule pair { <term> '=>' <term> }
   token term { ... }
}
```

new methods

I find these hard to live without nowadays.

```
.pick($n) Like $n things out of a hat
.roll($n) Like $n dice
.uniq Keep non-repeating elems
.min/.max Get extreme values
.classify Collect into bins
.first Find first matching value
```

Task: print the last Friday of each month of a given year. Perl 5:

```
use strict;
use DateTime;
use feature qw( say );
for my $month ( 1..12 ) {
   my $dt = DateTime->last_day_of_month( year => $ARGV[ 0 ],
                                          month => $month );
   while ( $dt->day_of_week != 5 ) {
      $dt->subtract( days => 1 );
   say $dt->ymd ;
```

Perl 6 doesn't have last_day_of_month. So we need to emulate it:

```
for 1..12 -> $month {
   my $d = Date.new(@*ARGS[0], $month, 1)
        .delta(1, month).delta(-1, day);
   while $d.day-of-week != 5 { $d.=delta(-1, days) }
   say $d;
}
```

Something else? Yes, @*ARGS[0] doesn't look so sixish:

```
sub MAIN(Int $year = Date.today.year) {
  for 1..12 -> $month {
    my $d = Date.new($year, $month, 1)
        .delta(1, month).delta(-1, day);
    while $d.day-of-week != 5 { $d.=delta(-1, days) }
    say $d;
  }
}
```

Myself, I would approach this tool not with for loops but with lists of things:

last Friday of each month

But why are we messing with variables when we chain everything?

```
sub MAIN(Int $year = Date.today.year) {
   say ~.value.reverse.first: *.day-of-week == 5
   for classify *.month,
        Date.new("$year-01-01") .. Date.new("$year-12-31");
}
```

last Friday of each month

Or we could write that in the forwards direction, with the help of feeds:

interactive shell

When you write per1, it just sits there:

```
$ perl
^C
```

When you write per16 (or python, or irb), you get an interactive shell:

```
$ per16
> say "OH HAI"
OH HAI
>
```

debugger

In the same vein, you can write per16-debug and get an interactive debugger.

(It's a module, but it's included in Rakudo Star.)

Cool fact: it was a stealth project, made by extending Rakudo, not changing it.

concurrency

The ideas behind concurrency in Perl 6 are finally coming together this year.

Lazy lists are for repeatedly getting something. Supply objects are for repeatedly being given something.

They unify async and events.

refactorability

There's something about Perl 6 that makes it very nice to refactor programs in small incremental steps.

Perl 5 has that too, but not to the same extent. Somehow the features of Perl 6 conspire to make refactoring an extra pleasant experience.

Emmp

performance

Still not fast. Working on it, though.

Moar has been a godsend, and useful work is going on with optimization.

It's reasonable to expect Perl 6 to be not much slower (and sometimes faster) than Perl 5... eventually.

third-party modules

For basically any problem you want to solve, Perl 5 has a module for that.

Perl 6 module count has... three digits. We have some ways to go.

It's a bootstrapping thing. Popularity breeds modules, and vice versa.

nytprof

Devel::NYTProf is just so amazing.

Perl 6 doesn't have anything like that.

Someone needs to write it.

devel-cover

We don't have a Devel::Cover either.

In fact, we're missing many of those tools that reach into the backend and instrument it... except for a debugger.

perl critic

Perl 5 has excellent Perl::Critic capabilities, thanks largely to PPI.

Perl 6... has excellent potential here, but *still* no real way for Perl 6 user code to parse Perl 6 code into an AST and do stuff with it. Hopefully macros will help.

some of the event-y things

When I look at the AnyEvent, POE, and Coro family of modules on CPAN, I also grow a little bit jealous.

We should have something like that. In fact, Perl 6 is still missing a unified event handling story. But we're inching closer with the work on concurrency.

CPAN/PAUSE

The whole Perl 5 module infrastructure, with uploading and testing and smoking and reviews... is impressive.

Perl 6 is still mostly on Github.

We want CPAN and PAUSE. Work is going on in that area.

Conclusion



Everything that's worth doing is worth doing well

Perl 5 and Perl 6 are different

Both have strong points and weak

Perl 5 gets the ecosystem right Perl 6 is a syntax/semantics upgrade

Both should be proud of their strengths and envious about the other's strengths

Perl 5 and Perl 6 will probably never converge fully back into one single language

Neither do I think one of them will ever "win" or obviate the other

But we can certainly remain **one community**, and keep stealing from and inspiring each other

Thank you.