

Perl 6

Edward Higgins

2019-03-04

Perl 6

A language for the 21st Century

[Originally presented in Vim,]
[best viewed in a full-screen terminal at 70x20 characters]

In the beginning...

In the beginning...

C

sh

awk

sed

grep

In the beginning...

```
C      \  
sh     \  
awk    :=> Perl 1.0  
sed    /  
grep  /
```

(1987) Perl 1.0

(1988) Perl 2.0

(1989) Perl 3.0

(1991) Perl 4.0

(1994) Perl 5.0

	Perl 5		Perl 6
	-----		-----
(2000)	Perl 5.6		Design process announced
(2002)	Perl 5.8		Dev. on Parrot VM
(2009)	Perl 5.10		Dev. moved to Rakudo VM
(2015)	Perl 5.22		Perl 6.c (First stable release)
(2018)	Perl 5.28		Perl 6.d

The Perl Philosophy

Natural language inspired

- Flexible, malleable, expressive
- Dialects are okay
- The language grows with you
- Context is important

The Perl Philosophy

TIMTOWTDI (There's more than one way to do it)

DWIM (Do what I mean)

Easy things should be easy,
hard things should be possible

Give the user enough rope to
shoot themselves in the foot

The Perl 6 Philosophy

Easy things should stay easy,
hard things should get easier

No arbitrary limits

Similar things should look similar

Visual distinctions are important

-Ofun

#

Variables

```
my $scalar = 3.14;
```

```
my @array = (1, 2, 3);
```

```
my %hash = first-name => "Larry",  
           surname     => "Wall" ;
```

#

Typed Variables

```
my Rat $scalar = 3.14;
```

```
my Int @array = (1, 2, 3);
```

```
my Str %hash = first-name => "Larry",  
                surname   => "Wall" ;
```

#

More Variables

```
my Num $scalar = pi;
```

```
my Int @array = (1, 2, 3);
```

```
my Str %hash;
```

```
%hash<first-name> = "Larry";
```

```
%hash<surname>    = "Wall" ;
```

#

Rational numbers

```
say 0.1.Rat + 0.2.Rat == 0.3.Rat;
```

#

Floating point numbers

```
say 0.1.Num + 0.2.Num == 0.3.Num;
```

#

Numbers?

```
say 0.1 + 0.2 == 0.3;
```


#

if-then-else

```
my $age = 3;

if $age < 18 {
    say "You can't drink in the UK yet";
} elsif 18 <= $age <= 20 {
    say "You can't drink in the USA yet";
} else {
    say "You can drink (almost) anywhere!";
}
```

#

given-when

```
my UInt $age = 19;
```

```
given $age {
```

```
    when 0..17 { say "You can't drink in the UK yet" }
```

```
    when 18..20 { say "You can't drink in the USA yet" }
```

```
    default { say "You can drink (almost) anywhere!" }
```

```
}
```

#

for loops

```
for (0 ... 10) -> $x {  
  print "$x ";  
}
```

#

for loops

```
for (0 ... ^ 10) -> $x {  
  print "$x ";  
}
```

#

for loops

```
for ^10 -> $x {  
  print "$x ";  
}
```

#

Sequences

```
my @integers = 1 ... *;
```

```
say @integers[20];
```

#

Sequences

```
my @evens = 2, 4 ... *;
```

```
say @evens[~10];
```

#

Sequences

```
my @powers = 2, {2 * $_} ... *;
```

```
say @powers[10];
```


#

Sequences

```
my @powers = 2, 4, 8 ... *;
```

```
say @powers[10];
```

#

Sequences

```
my @fibonacci = 0, 1, 1 ... *;
```

```
say @fibonacci[10];
```

#

Sequences

```
my @fibonacci = 0, 1, { $\$^x + \$^y$ } ... *;
```

```
say @fibonacci[ $\wedge$ 10];
```

#

Metaoperators

```
my @a = ( 1, 2, 3);
```

```
my @b = (10, 20, 30);
```

```
say @a <<+>> @b;
```

```
say 2 <<*<< @a;
```

```
say [lcm] @a;
```

#

Hyperoperators

```
my @a = (1,2,3);
```

```
my @b = <a b c>;
```

```
say @a X @b;
```

```
say @a Z @b;
```

#

Hyperoperators

```
my @a = (1,2,3);
```

```
my @b = <a b c>;
```

```
say @a X~ @b;
```

```
say @a Z~ @b;
```

Object Oriented

```
class Dog {  
  
}
```

Object Oriented

```
class Dog {  
  has Str $.name is required;  
  has Int $.age;  
  
  method bark(Str $target) {  
    say "$!name barked at $target";  
  }  
}
```


Object Oriented

```
class Dog {  
  has Str $.name is required;  
  has Int $.age;  
  
  method bark(Str $target) {  
    say "$!name barked at $target";  
  }  
}  
  
my Dog $fido = Dog.new(name => "Fido");  
  
$fido.bark("the audience");
```

#

Functional

```
sub add($a, $b) { return $a + $b }
```

```
sub make_add($b) {  
    return sub ($a) { add($a, $b) }  
}
```

```
my &add_3 = make_add(3);
```

```
say add_3(2);
```

#

Functional

```
my &add_3 = * + 3;
```

```
say add_3(2);
```

#

Multi-dispatch

```
multi sub sort(@list where *.elems < 2) {  
    return @list;  
}  
  
multi sub sort(@list where *.elems >= 2) {  
    my $pivot = @list[0];  
    my @before = @list[1 .. *].grep(* before $pivot);  
    my @after = @list[1 .. *].grep(* !before $pivot);  
  
    return flat sort(@before), $pivot, sort(@after);  
}  
  
say sort (14, 1, 61, 25, 8);
```

#

Regular Expressions

```
my $string = "length = 5";
```

```
$string ~~ /length \s* "=" \s* (\d+)/;
```

```
say $0.Int;
```

#

Regular Expressions

```
my $string = "length = 5";
```

```
$string ~~ /length  
          \s* "="  
          \s* $<length> = (\d+) /;
```

```
say $<length>;
```

#

Regular Expressions

```
grammar Params {  
    rule TOP      { <assignment>+ }  
  
    rule assignment { <param> "=" <value> ";" }  
  
    token param     { <.\alpha>+ }  
  
    token value     { <.\digit>+ }  
}
```

```
my $string = "length = 5; width = 20; time = 4;";
```

```
my $parsed = Params.parse($string);
```

```
say $parsed;
```

#

Concurrency

```
my $promise = start {  
  my $x = 0;  
  for 1 ... 10 -> $i { $x += $i }  
  $x;  
}  
  
my $result = await $promise;  
  
say $result;
```


#

Concurrency

```
my $veg_supplier = Supplier.new;

my $feedback = supply {
  whenever $veg_supplier.Supply {
    emit("We've got veg: " ~ $_);
  };
}

$feedback.tap( -> $str { say "$str" });

$veg_supplier.emit("Radish");
$veg_supplier.emit("Lettuce");
$veg_supplier.emit("Tomato");
```

Useful Links

The Perl 6 documentation

<https://docs.perl6.org>

Rakudo, a Perl 6 implementation

<https://rakudo.org>

Online REPL

<https://glot.io/new/perl6>

Rosetta Code

<https://rosettacode.org>