Perl 6: More...

Jonathan Worthington

Hannover.pm

Perl Evolves

- Perl 6 is the forthcoming major re-working of the Perl programming language.
- Perl 5 is great!
- Perl 6, when it is ready, aims to be even greater – a tall order.
- Tonight: how Perl 6 aims to provide programmers with more Good Things.

More Huffmanized

- Huffman Coding = Things that are used more often should be shorter.
- Often true in natural language…
  - Frequently used:
    the, a, you, I, me…
  - Rarely used:
    antidisestablishmentarianism
    (Yes, that's an English word. We were trying to compete with German nouns. ☺)

Perl 6: More...

More Huffmanized

- How often did you write this in Perl 5?

```
print "Whatever\n";
```

- In Perl 6: a version of print that puts a new line character on the end for you!

```
say "Whatever";
```

Perl 6: More...

More Huffmanized

- Method calling in Perl 5 used ->

```
$monkey->eat($banana);
```

- In Perl 6 we use the shorter . instead

```
$monkey.eat($banana);
```

- As a bonus, this syntax is more consistent with other OO languages
- Note that concatenation is now ~

More Huffmanized

- How often did you write this in Perl 5?

```
if ($a == 5 || $a == 6 || $a == 7) {
    # Do something.
}
```

- In Perl 6: junctions!

```
if $a == 5 | 6 | 7 {
    # Do something.
}
```

- Note that the parentheses around the condition are no longer needed, either.

More Huffmanized

- How often did you write this in Perl 5?

```
my $contains_five = 0;
for (@a) {
    $contains_five = 1 if $_ == 5;
}
if ($contains_five) {
}
```

- In Perl 6: junctions can be constructed from arrays too

```
if any(@a) == 5 {
}
```

More Huffmanized

- How often did you write this in Perl 5?

```
my $total = 0;
for (@values) {
    $total += $_;
}
```

- In Perl 6: the reduction meta-operator

```
my $total = [+] @values;
```

- Many other uses…

```
my $factorial = [*] 1..$x;
if [<=] @x { # If @x is sorted ascending
}
```

More Orthogonal

- Perl 6 attempts to avoid special cases somewhat by providing more general mechanisms
- Consider `sort` in Perl 5
  - `$a` and `$b` magically exist
  - The real problem: we need an easy and concise way to give a block parameters and get access to them

More Orthogonal

- In Perl 6, we have secondary sigils
- $^whatever is a block parameter
- All block parameter referred to within a block are taken and their names are sorted lexicographically
- The parameters are bound to these variables in lexicographic order

Perl 6: More...

More Orthogonal

- This is how you would sort a list of strings by their length in characters

```
@words .= sort { $^a.chars <=> $^b.chars };
```

- However, this more general mechanism can be used anywhere you want.

```
my $code = {
    say $^x - $^y;
}
$code(2,1); # 1
$code(5,7); # -2
```

Perl 6: More...

More Declarative

- Declarative = just say what you want, not how to do it.
- In Perl 5, handling of parameters passed to subs could be quite a bit of work.
- Perl 6 provides a more declarative syntax.
- The old way is still available.

More Declarative

- First example: a sub that takes three scalar parameters, one optional.
- Perl 5:

```
sub substr {
    die unless @_ == 2 || @_ == 3;
    my ($string, $offset, $length) = @_;
}
```

- Perl 6:

```
sub substr($string, $offset, $length?) {
}
```

Perl 6: More...

More Declarative

- Second example: a variable argument sub with a fixed first parameter.
- Perl 5:

```
sub all_under {
    die unless @_ > 1;
    my ($test, @values) = @_;
}
```

- Perl 6:

```
sub all_under($test, *@values) {
}
```

More Object Oriented

- You can treat everything as an object if you want to.

```
"Hello, world!".say;
$len = $string.chars; # Length in characters
```

- But you don't have to.

```
say "Hello, world!";
$len = chars($string);
```

- File I/O is more OO in Perl 6.

```
my $fh = open ">> quotes.txt";
$fh.say("Vacuums suck!");
```

More Object Oriented

- Classes with methods now clearly separated from modules with subs.

```
class Englishman is Human {
    method drink_tea($cups) {
        for 1..$cups {
            say "I say, that was spiffing!";
        }
    }
}
```

- Notice the new, neater syntax for inheritance

More Object Oriented

- Attributes are now all private; accessor and mutator methods can be generated for you on request.

```
class Englishman is Human {
    has $.name; # Accessor
    has $.ale is rw; # Accessor and mutator
    has @political_views; # Private
}
```

- Method calls now interpolate

```
say "$jeeves.name thinks that Tony Blair " ~
    "is $jeeves.get_view('Blair').";
```

Perl 6: More...

More CPU And Memory Efficient

- You can optionally annotate variables with types.

```
my int $a = 42;
my @list of int = 1..10000;
```

- Using the lower-case int type tells the Perl 6 compiler that it can use a native integer to store the value.
- Can be very fast with a JIT compiler.
- Also more compact in memory.

More Lazy

- Lazy evaluation = on demand.
- We can create infinite lists!

```
my @naturals = 0...; # Or 0..Inf
```

- The computation required to produce an element of the list will only be performed when it is accessed.
- More advanced things are possible:

```
my @evens = map { 2 * $^n } @naturals;
```

More Parallelizable

- Parallelism matters!
  - Need to occupy multiple CPUs to increase performance.
  - The leading edge processors of today have already 2-4 cores.
  - The next generation: even more!
- Most people find parallelism hard
  - Need the language to help us

More Parallelizable

- Hyper-operators let you perform operations element-wise over an array

```
@sums = @a >>+<< @b;
@squares = @a >>**<< 2;
@results = @a>>.some_method();
```

- More than just a short hand for loops
- You are stating that you do not care about the order that the operation is performed on elements, and permitting it to be performed in parallel.

More Parallelizable

- Atomic operations are possible without you having to declare when to take out locks

```
atomic {
    $account1 -= $transfer_amount;
    $account2 += $transfer_amount;
};
```

- Under the hood: software transactional memory
- Helps avoid deadlock

Perl 6: More...

More huffmanized

More orthogonal

More declarative

More object oriented

More CPU and memory efficient

More lazy

More parallelizable

More...

Perl 6: More...

More huffmanized

More orthogonal

More declarative

More object oriented

More CPU and memory efficient

More lazy

More parallelizable

More productive!

Perl 6: More...

More huffmanized

More orthogonal

More declarative

More object oriented

More CPU and memory efficient

More lazy

More parallelizable

More fun!

Perl 6: More...

Danke!

Perl 6: More...

Questions?