Cool Things in Perl 6

brian d foy
brian@stonehenge.com

May 3, 2008

Stonehenge

- I'm not a Perl 6 contributor
- Not about the implementions
- Not about new syntax for old things
- About new features not in Perl 5
- Stuff that makes me want Perl 6

Stonehenge

Caveats

- **Cribbed from the Synopses**

  http://feather.perl6.nl/syn/

- **Some of this might not work yet**

Stuff I want

- Most languages can do the job
- But how much code does it take?
- And where does that code live?
- What's a primitive and what's built-in?

In this talk

- Junctions
- New list techniques
- Meta operators

Junctions

Stonehenge

- A junction is a single value that is equivalent to multiple values
- Useful with comparisons
- Parallelizable
- Short circuitable

Any

- | or any()

```
if $x == 1 | 2 | 3 { ... }
if $x eq any( q:w( a b c ) )
    {...}
```

- Mutable

```
(1 | 2 | 3 ) + 1;  # 2 | 3 | 4
```

Stonehenge

All or one

- & or all()

```
if $x > ( $i & $j & $k ) {...}
if $x > all( $i, $j, $k ) {...}
```

- ^ or one()

```
if $x == ($i ^ $j ^ $k) {...}
if $x == one($i, $j,$k) {...}
```

Stonehenge

none

- **none()**

```
 if $x eq none( $s, $t, $u )
      {...}
```

Easy lists

Stonehenge

Fancy ranges

- Lists can be unbounded

```
0 .. *
```

- Not consecutive

```
0 .. 100 :by(3)
```

Exclusive ranges

- Exclusive lists

```
1^..^10  # 2,3,4,5,6,7,8,9
```

- 0 up to one less

```
^5  # 0,1,2,3,4
```

Stonehenge

Multiple lists

- Zip lists to iterate over them together

```
for zip(@a, @b) -> $a, $b {
    say "Got $a and $b" }
```

- Stops at shortest list

Feed operators

- Directs output to a "sink"

```
@in ==> map {...} ==> @out
@out <== map {...} <== @in
```

- Source is lazy
- Allows parallelization

Multiple sources

- Stack multiple sources with ==>>
- Looks ahead for sink

```
source1() ==>>
source2() ==>>
source3() ==>>
sink();
```

Stonehenge

Meta operators

Superpowers

- Give normal operators super powers
- Make common operations even easier
- Remove messy looping monkey code

Stonehenge

Five types

- Assignment
- Negated relational
- Hyper
- Reduction
- Cross

Stonehenge

Assignment

- Binary assignment like C and Perl 5
- Normal assignment

```
$count = 5;

$count = $count + 1;

$count += 1;
```

- Mostly with scalar operators in Perl 5

More operators

- More operators (instead of builtins)
- The , operator to make a list

```
@array = 1, 2, 3;
```

- Binary assignment is a push

```
@array ,= 4, 5, 6
```

Stonehenge

Negated relational

- Put a ! in front of a comparator

```
if $version !== 6 { # or !=
    say "How are we here?" }

if $version !> 5 {
    say "Here again?!" }
```

- Think "*isn't* greater than"

Stonehenge

Hyperoperators

- Obviates looping for single operations
- Applies operation to each element

```
@numbers >>++;

@negatives >>-;
```

- Can do either way

```
@negatives = -<<@positives;
```

Stonehenge

List on list

- Surround an operator with angle brackets (no extra spaces)

```
>>op<<          <<op>>

>>op>>          <<op<<
```

- Makes new list
- Also with french quotes

```
»op«  «op»  »op»  «op«
```

Stonehenge

>>op<<

- List on the left and right
- One element from each for result

```
(1,2,3) >>+<< (4,5,6)    # 5,7,9
```

- Intersection of hash

```
%foo >>+<< %bar
```

Hypergwimmery

- Guess What I Mean (GWIM)
- Pointing one way GWIMs on that side
- One side is "shaped" differently

```
(1,2,3) >>**>> 2    # 1,4,9
```

- Doesn't matter which side

```
'.jpg' <<~<< q:w(a b) # a.jpg b.jpg
@numbers >>max>> 2
```

Stonehenge

Doublegwimmery

- Which side needs shaping?
- Point all arrows outward
- Perl guesses

```
@a <<+>> @b
```

Stonehenge

Reduction

- Finally, a built-in reduce

```
my $summerial = [+] @numbers;
my $factorial = [*] @numbers;
my $ascends = [<] @numbers;
```

Stonehenge

Pseudo reduction

- Keep the intermediate results with \op

```
[\+] ^4;   # (0, 1, 3, 6);
```

- Produce a triangle list

```
[\,] ^4
# ([0],[0,1],[0,1,2], [0,1,2,3]);
```

Cross operator

- Make tuples with X

```
q:w( a b ) X ( 1, 2 )
# (a, 1),(a,2),(b,1), (b,2)
```

Stonehenge

Hypercross

- Perform the operation on all tuples

```
(1,2) X~X q:w(a b)
# 1a, 1b, 2a, 2b
```

Questions

Stonehenge