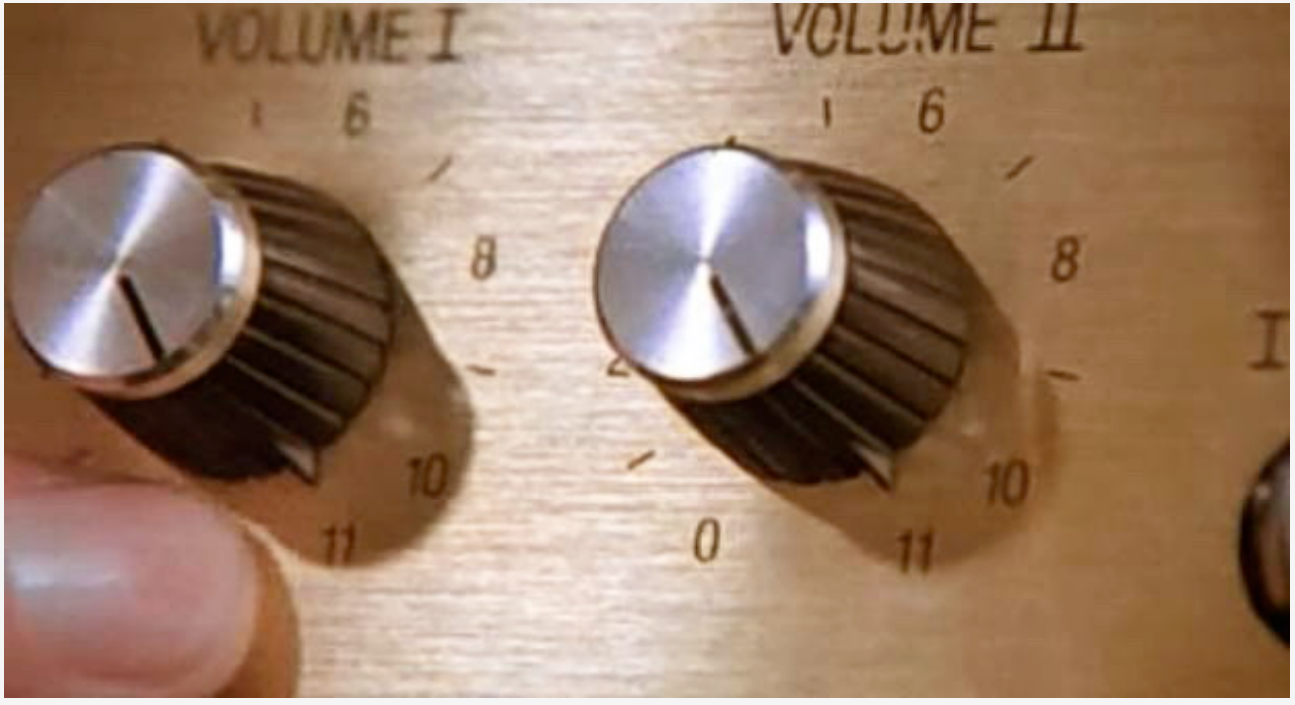
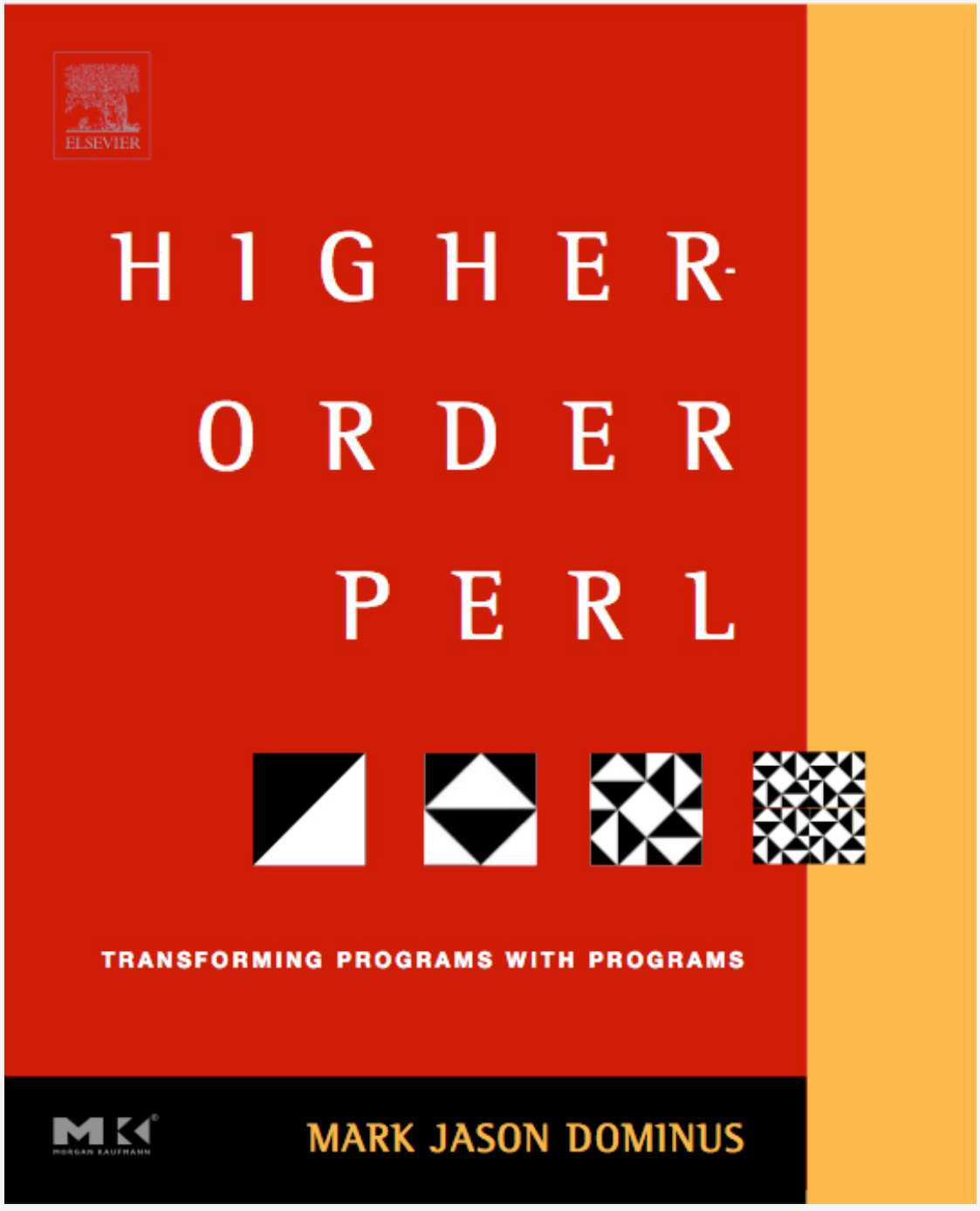
Taking Perl to Eleven

David Golden • @xdg
Staff Engineer, MongoDB

The Perl Conference 2018

"If we need that extra push…
do you know what we do?"





Functions creating functions

Higher-order programming is going to eleven

Building blocks

Building block 1: anonymous functions

```
my $fcn = sub { ... };
```

```
my $str = $fcn->();
```

```
my $fcn = sub {
  return join(" ", @_)
};


my $str = $fcn->(@words);
```

Building block 2: scope

```
my $separator = " ";

sub joiner {
  return join($separator, @_)
}
```

Building block 3: closure

```
my $separator = " ";

sub joiner {
  return join($separator, @_);  }
}
```

```
sub joiner_maker {
   my $separator = shift;

   return sub {
      return join($separator, @_);
   }
};
```

What happens to `$separator`?

Anonymous function
'closes over'
the lexical variable

```
sub joiner_maker {
   my $separator = shift;

   return sub {
      return join($separator, @_);
   }
};
```

```
my @words = qw/able baker/;

my $spacer = joiner_maker(" ");
my $dasher = joiner_maker("-");
my $banger = joiner_maker("!");

$spacer->(@words); # "able baker"
$dasher->(@words); # "able-baker"
$banger->(@words); # "able!baker"
```

HOP

Functions creating functions

Abstract rules for generating arbitrary data

Why do I want that?

Load testing?

Lots of

Similar, but slightly different things

Start with simple generators...

Generate an array of integers

```
# 100 random ints from 0 to 255

sub make_int_array {
  my @array;
  push @array, int(rand(256))
    for 1..100;
  return \@array;
}
```

Ick! Constants!

```
# 100 random ints from 0 to 255

sub make_int_array {
  my @array;
  push @array, int(rand(256))
    for 1..100;
  return \@array;
}
```

```
# N random ints from 0 to X

sub make_int_array {
   my ($n, $x) = @_;
   my @array;
   push @array, int(rand($x+1))
      for 1..$n;
   return \@array;
}
```

Generate an array of discrete choices

```
# N strings drawn from discrete set

sub make_discrete_array {
   my ($n, @list) = @_;
   my $s = @list;
   my @array;
   push @array, $list[int(rand($s))]
      for 1..$n;
   return \@array;
}
```

```
my $ref = make_discrete_array(
   100,
   'Larry Wall',
   'Damian Conway',
   'Ricardo Signes',
   'Sawyer X',
   ...
);
```

Repeat for every kind of array you need?!?

NO!

Generalize array generation

```
# N elements provided by a function

sub make_array {
   my ($n, $f) = @_;
   my @array;
   push @array, $f->()
      for 1..$n;
   return \@array;
}
```

```
# N elements provided by a function

sub make_array {
   my ($n, $f) = @_;
   return
      [ map { $f->() } 1..$n ];
}
```

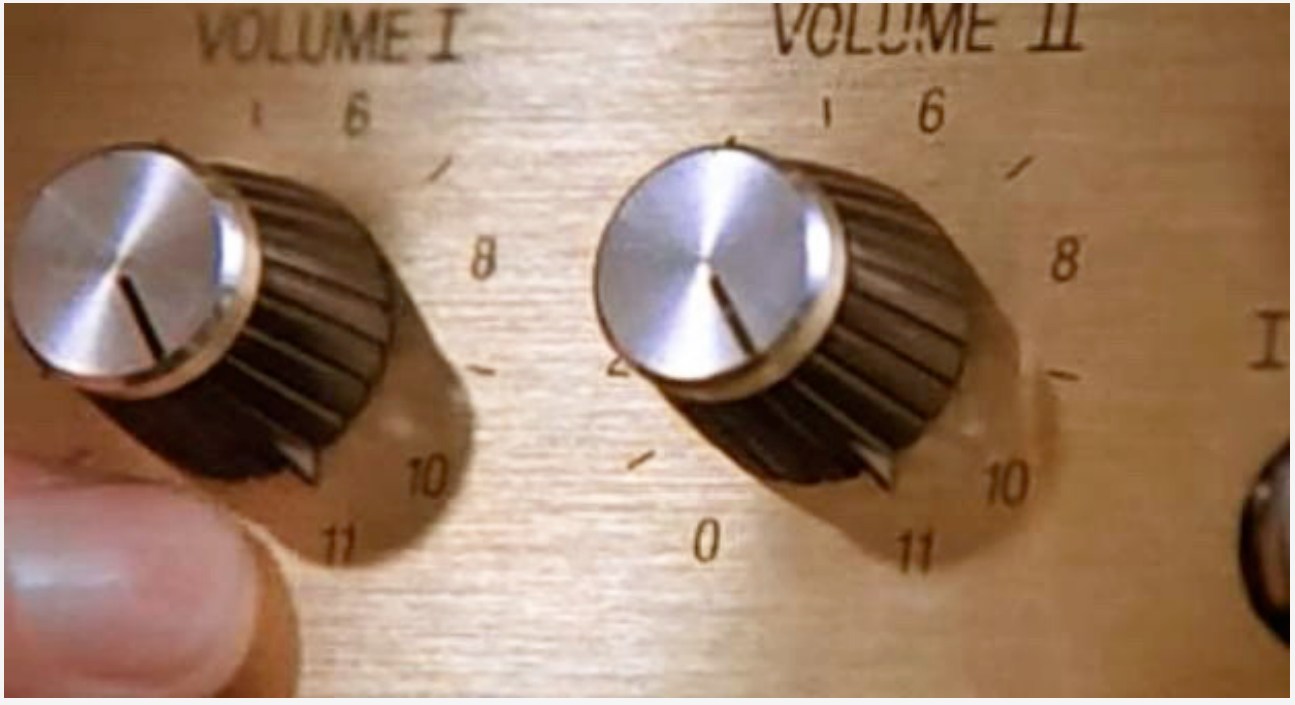
```
# 100 random ints from 0 to 255

make_array(
  100, sub { int(rand(256) }
);
```

```
# 100 discrete choices

my @list = qw(...);
my $s = @list;

make_array(
  100, sub { $list[int(rand($s))] }
);
```

Ick! Constants!

(And `sub { . . . }`)

Let's go one higher!



Generalize the value generating functions

```
sub int_maker {
   my ( $min, $max ) = @_;
   my $range = $max - $min + 1;
   return sub {
      $min + int(rand($range));
   };
}

# random integer from 3 to 6
$f = int_maker(3,6)
$i = $f->(); # e.g. 4
```

```
sub pick_maker {
   my (@list) = @_;
   my $s = @list;
   return sub {
      $list[ int(rand($s)) ]
   };
}

# random pick from a list of names
$f = pick_maker(@names)
$n = $f->(); # e.g. "Larry"
```

Higher-order value generation

Generators replace anonymous functions

```
# 100 random ints from 0 to 255
make_array(
  100, sub { int(rand(256) }
);

# 100 discrete choices
my @list = qw(...);
my $s = @list;
make_array(
  100, sub { $list[int(rand($s))] }
);
```

```
# 100 random ints from 0 to 255
make_array(
  100, int_maker(0,255)
);

# 100 discrete choices
my @list = qw(...);

make_array(
  100, pick_maker(@list)
);
```

Simpler & Expressive

```
# 100 random ints from 0 to 255
make_array(
  100, sub { int(rand(256) }
);

# 100 discrete choices
my @list = qw(...);
my $s = @list;
make_array(
  100, sub { $list[int(rand($s))] }
);
```

```
# 100 random ints from 0 to 255
make_array(
  100, int_maker(0,255)
);

# 100 discrete choices
my @list = qw(...);

make_array(
  100, pick_maker(@list)
);
```

We have higher-order value generators

Why stop there?

Higher order array generation!

```
sub make_array {
   my ($n, $f) = @_;

   return [
      map { $f->() } 1..$n
   ];
}
```

```
sub make_array {
   my ($n, $f) = @_;

      return [
         map { $f->() } 1..$n
      ];
}
```

```
sub array_maker {
   my ($n, $f) = @_;
   return sub {
      return [
         map { $f->() } 1..$n
      ];
   };
}
```

```
# old way
$r = make_array(
  100, int_maker(0,255)
);
```

```
# old way
$r = make_array(
  100, int_maker(0,255)
);

# higher-order way
$f = array_maker(
  100, int_maker(0,255)
);

$r = $f->();
```

Which lets us do this…

```
# array of array of integers

$f = array_maker(
  100,
  array_maker(
    100, int_maker(0,255)
  )
);

$r = $f->();
```

Or even this...

```
# AoAoA: 100x100 RGB values?

$f = array_maker(
  100,
  array_maker(
    100,
    array_maker(
      3, int_maker(0,255)
    )
  )
);
```

Arrays of random size?

Generalize $n?

```
# array of array of integers

$f = array_maker(
   sub { 100 },
   array_maker(
      sub { 100 }, int_maker(0,255)
   )
);

$r = $f->();
```

Ick!

```
# transformation helper function

sub _x {
   my $v = shift;
   my $is_code = ref($v) eq 'CODE';
   return $is_code ? $v->() : $v;
}
```

```
sub array_maker {
   my ($n, $f) = @_;
   return sub {
      return [
         map { _x($f) } 1.._x($n)
      ];
   };
}
```

One function
Many outcomes

```
# constant size, constant value

$f = array_maker(
  $n,
  $v
);



$r = $f->();
```

```
# constant size, random values

$f = array_maker(
  $n,
  pick_maker(@options)
);



$r = $f->();
```

```
# random size, random values

$f = array_maker(
  int_maker(3,6),
  pick_maker(@options)
);



$r = $f->();
```

```
# nested random AoA

$f = array_maker(
   int_maker(3,6),
   array_maker(
      int_maker(10,20),
      pick_maker(@options)
   )
);

$r = $f->();
```

Data::Fake

Goal: create lots of fake super-hero records

```
{
  name      => 'The Tick',
  battlecry => 'Spoon!',
  birthday  => '1973-07-16',
  friends   => [
    'Arthur',
    'Die Fledermaus',
    'American Maid',
  ],
}
```

Data::Fake

Higher-order
structured data

```
use Data::Fake qw/Core Names Text Dates/;

my $hero_factory = fake_hash(
  {
    name      => fake_name(),
    battlecry => fake_sentences(1),
    birthday  => fake_past_datetime("%Y-%m-%d"),
    friends   => fake_array(fake_int(2,4),fake_name()),
  }
);
```

Declarative

```
{
  name      => 'The Tick',
  battlecry => 'Spoon!',
  birthday  => '1973-07-16',
  friends   => [
    'Arthur',
    'Die Fledermaus',
    'American Maid',
  ],
}
```

```
{
  name      => fake_name(),
  battlecry => fake_sentences(1),
  birthday  => fake_past_datetime("%Y-%m-%d"),
  friends   => fake_array(fake_int(2,4),fake_name()),
}
```

```
use Data::Fake qw/Core Names Text Dates/;

my $hero_factory = fake_hash(
  {
    name      => fake_name(),
    battlecry => fake_sentences(1),
    birthday  => fake_past_datetime("%Y-%m-%d"),
    friends   => fake_array(fake_int(2,4),fake_name()),
  }
);
```

```
use Data::Fake qw/Core Names Text Dates/;

my $hero_factory = fake_hash(
  {
    name      => fake_name(),
    battlecry => fake_sentences(1),
    birthday  => fake_past_datetime("%Y-%m-%d"),
    friends   => fake_array(fake_int(2,4),fake_name()),
  }
);
```

```
use Data::Fake qw/Core Names Text Dates/;

my $hero_factory = fake_hash(
  {
    name       => fake_name(),
    battlecry => fake_sentences(1),
    birthday   => fake_past_datetime("%Y-%m-%d"),
    friends    => fake_array(fake_int(2,4),fake_name()),
  }
);
```

```
use Data::Fake qw/Core Names Text Dates/;

my $hero_factory = fake_hash(
  {
    name      => fake_name(),
    battlecry => fake_sentences(1),
    birthday  => fake_past_datetime("%Y-%m-%d"),
    friends   => fake_array(fake_int(2,4),fake_name()),
  }
);
```

```
use Data::Fake qw/Core Names Text Dates/;

my $hero_factory = fake_hash(
  {
    name      => fake_name(),
    battlecry => fake_sentences(1),
    birthday  => fake_past_datetime("%Y-%m-%d"),
    friends   => fake_array(fake_int(2,4),fake_name()),
  }
);
```

Sample output

```
# $hero_factory->()

{
    'name'      => 'Angel Deon Rush',
    'battlecry' => 'Et ratione ipsam provident.'
    'birthday'  => '1997-04-17',
    'friends'   => [
        'Josie Vivian Greer',
        'Kaylee Nyla Thompson',
        'Eliana Alejandra Velazquez',
        'Jamie Ryan Bond'
    ],
}
```

```
# $hero_factory->()

{
    'name'      => 'Wade Stetson Johnson'
    'battlecry' => 'Omnis exercitationem inventore ab.',
    'birthday'  => '1983-02-20',
    'friends'   => [
        'Carter Alaysia Bryan',
        'Christina Milan Hyde',
        'Ivanna Kyla Fox',
        'Jocelynn Aisha Fuentes'
    ],
}
```

```
# $hero_factory->()

{
    'name'      => 'Alison Ayla Monroe'
    'battlecry' => 'Aliquid et vel fuga.',
    'birthday'  => '1996-02-21',
    'friends'   => [
        'Amalia Hailey Johnston',
        'Ashton Messiah Frederick',
        'Aleena Genesis Holloway'
    ],
}
```

Lots of

Similar, but slightly different things

Recap

- Closures → anonymous functions with state
- HOP → functions generating functions
- Array generation → generalized behaviors
- Data::Fake → declarative structured data

Take your Perl to eleven!