The Perl Advent Calendar 2017 Redux

Mark Fowler

The Perl Conference Europe 2018

http://perladvent.org/2017

History

- For the last **18 years** there's been a Perl Advent Calendar
- A new article on Perl for each day of advent
- This year, like the first few years, I wrote all the articles myself.
- **Next year: http://cfp.perladvent.org/**

http://perladvent.org/2017

About This Talk

- **Brief** summary of the 2017 advent calendar

- I’ve given several similar talks over the years at Perl conferences

http://perladvent.org/2017

Talking Quickly

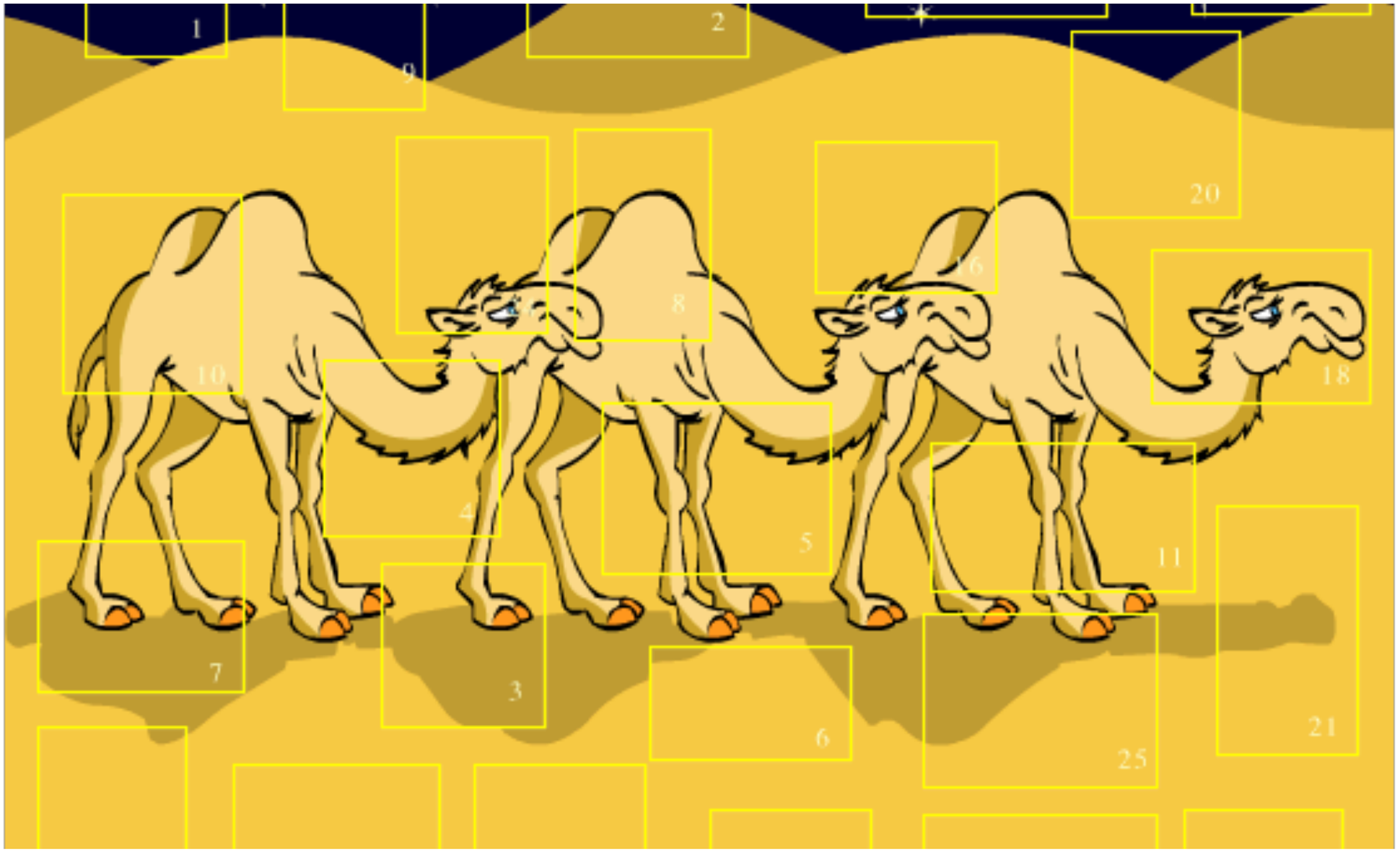
$$\frac{50\text{min} - 1\text{min} - 1\text{min}}{24\text{ days}} = 2\text{ min per day only}$$

- I'm going to talk *really really quick* and *skim over all the details*

- The idea here is to **expose you to the concepts**

- Please **READ THE FULL ARTICLE** for details. They're entertaining and contain semi-humorous stories of Santa's elves

http://perladvent.org/2017



Emoji

\N{…}, etc

Emoji

Character 127877, FATHER CHRISTMAS.

Emoji


```
use open qw( :encoding(UTF-8) :std );
print "\x{1F385} thanks you for all your hard work tonight";
```

Emoji

```
use open qw( :encoding(UTF-8) :std );
print "\x{1F385} thanks you for all your hard work tonight";
```

Emoji

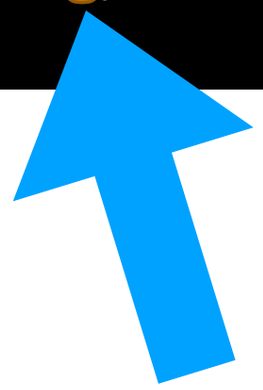

```
use open qw( :encoding(UTF-8) :std );
print "\x{1F385} thanks you for all your hard work tonight";
```

Emoji

```
use utf8;
use open qw( :encoding(UTF-8) :std );
print "🤦 thanks you for all your hard work tonight";
```

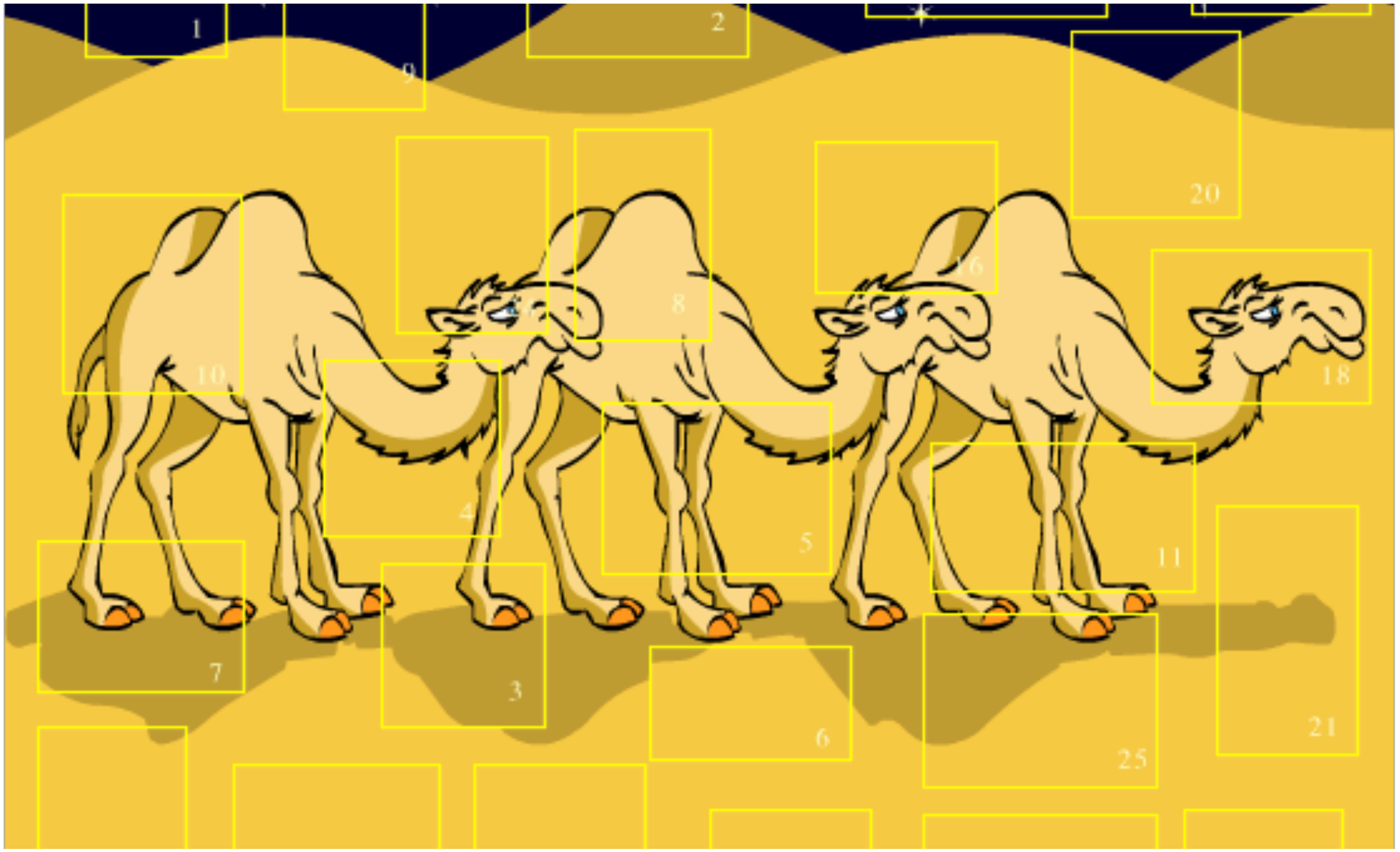
Emoji


```
use utf8;
use open qw( :encoding(UTF-8) :std );
print "🤶 thanks you for all your hard work tonight";
```



Emoji


```
use open qw( :encoding(UTF-8) :std );
print "\N{FATHER CHRISTMAS} thanks you for all your …";
```



Around the world with Emoji

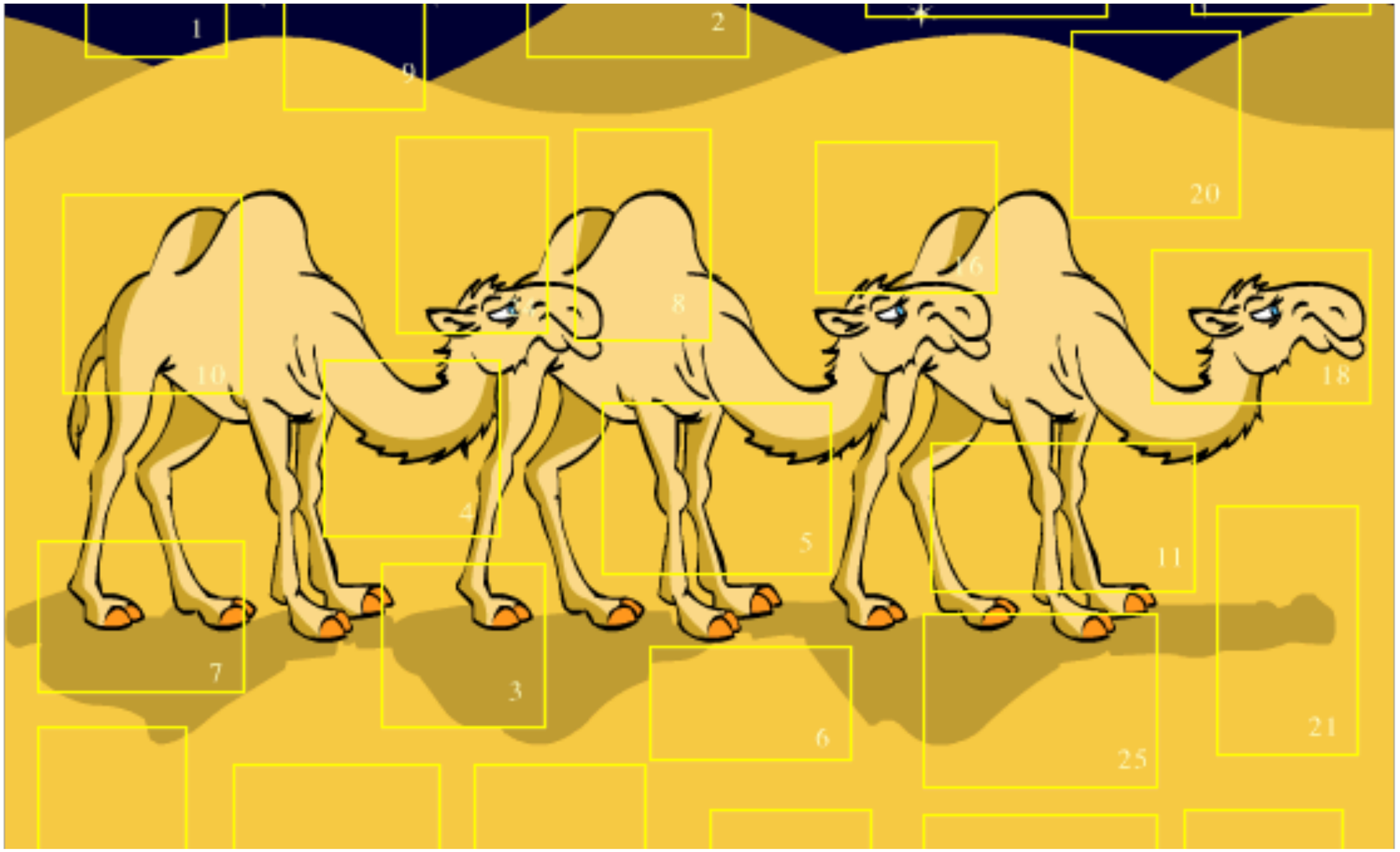
Multi-char glyphs, Emoji::NationalFlag

Multi-Char Emoji

```
print "\N{REGIONAL INDICATOR SYMBOL LETTER U}";
print "\N{REGIONAL INDICATOR SYMBOL LETTER S}";
```

Multi-Char Emoji

```
use Emoji::NationalFlag qw( code2flag );
print code2flag( $country_code );
```



Context Matters

Context, DBIx::Class searching, Perl::Critic

Context Matters

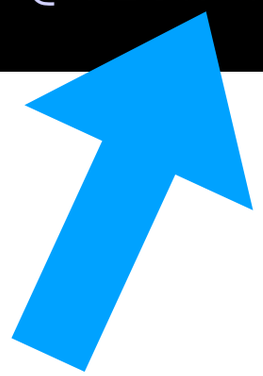

```
my $rs = $self->_schema->resultset('Child');

my $result_set_object = $rs->search({ naughty => 0 });
my @children          = $rs->search({ naughty => 0 });
```

Context Matters


```
my $rs = $self->_schema->resultset('Child');

my $result_set_object = $rs->search({ naughty => 0 });
my @children          = $rs->search({ naughty => 0 });
```

Context Matters


```
my $rs = $self->_schema->resultset('Child');

my $result_set_object = $rs->search({ naughty => 0 });
my @children          = $rs->search({ naughty => 0 });
```



Context Matters

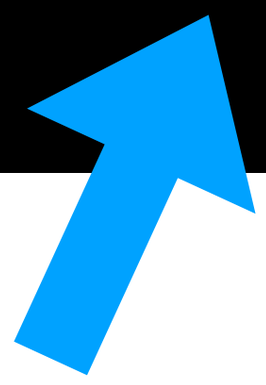

```perl
my $rs = $self->_schema->resultset('Child');

my $result_set_object = $rs->search({ naughty => 0 });
my @children          = $rs->search({ naughty => 0 });
```

```perl
has '_result_sets' => (
    is => 'ro',
    lazy => 1,
    builder => '_build_result_sets',
);

sub _build_result_sets {
    my $self = shift;
    my $rs = $self->_schema->resultset('Child');
    return {
        naughty => $rs->search( naughty => 1 ),
        nice    => $rs->search( naughty => 0 ),
    };
}
```

Context Matters


```perl
my $rs = $self->_schema->resultset('Child');

my $result_set_object = $rs->search({ naughty => 0 });
my @children          = $rs->search({ naughty => 0 });
```

```perl
has '_result_sets' => (
    is => 'ro',
    lazy => 1,
    builder => '_build_result_sets',
);

sub _build_result_sets {
    my $self = shift;
    my $rs = $self->_schema->resultset('Child');
    return {
        naughty => $rs->search( naughty => 1 ),
        nice    => $rs->search( naughty => 0 ),
    };
}
```



Context Matters


```perl
my $rs = $self->_schema->resultset('Child');

my $result_set_object = $rs->search({ naughty => 0 });
my @children          = $rs->search({ naughty => 0 });
```

```perl
has '_result_sets' => (
    is => 'ro',
    lazy => 1,
    builder => '_build_result_sets',
);

sub _build_result_sets {
    my $self = shift;
    my $rs = $self->_schema->resultset('Child');
    return {
        naughty => $naughty[0],
        $naughty[1] => $naughty[2],
        ...
        nice    => $nice[0],
        $nice[1] => $nice[2],
        ...
    };
}
```

Context Matters


```perl
my $rs = $self->_schema->resultset('Child');

my $result_set_object = $rs->search({ naughty => 0 });
my @children          = $rs->search({ naughty => 0 });
```

```perl
has '_result_sets' => (
    is => 'ro',
    lazy => 1,
    builder => '_build_result_sets',
);

sub _build_result_sets {
    my $self = shift;
    my $rs = $self->_schema->resultset('Child');
    return {
        naughty => $rs->search( naughty => 1 ),
        nice    => $rs->search( naughty => 0 ),
    };
}
```

Context Matters

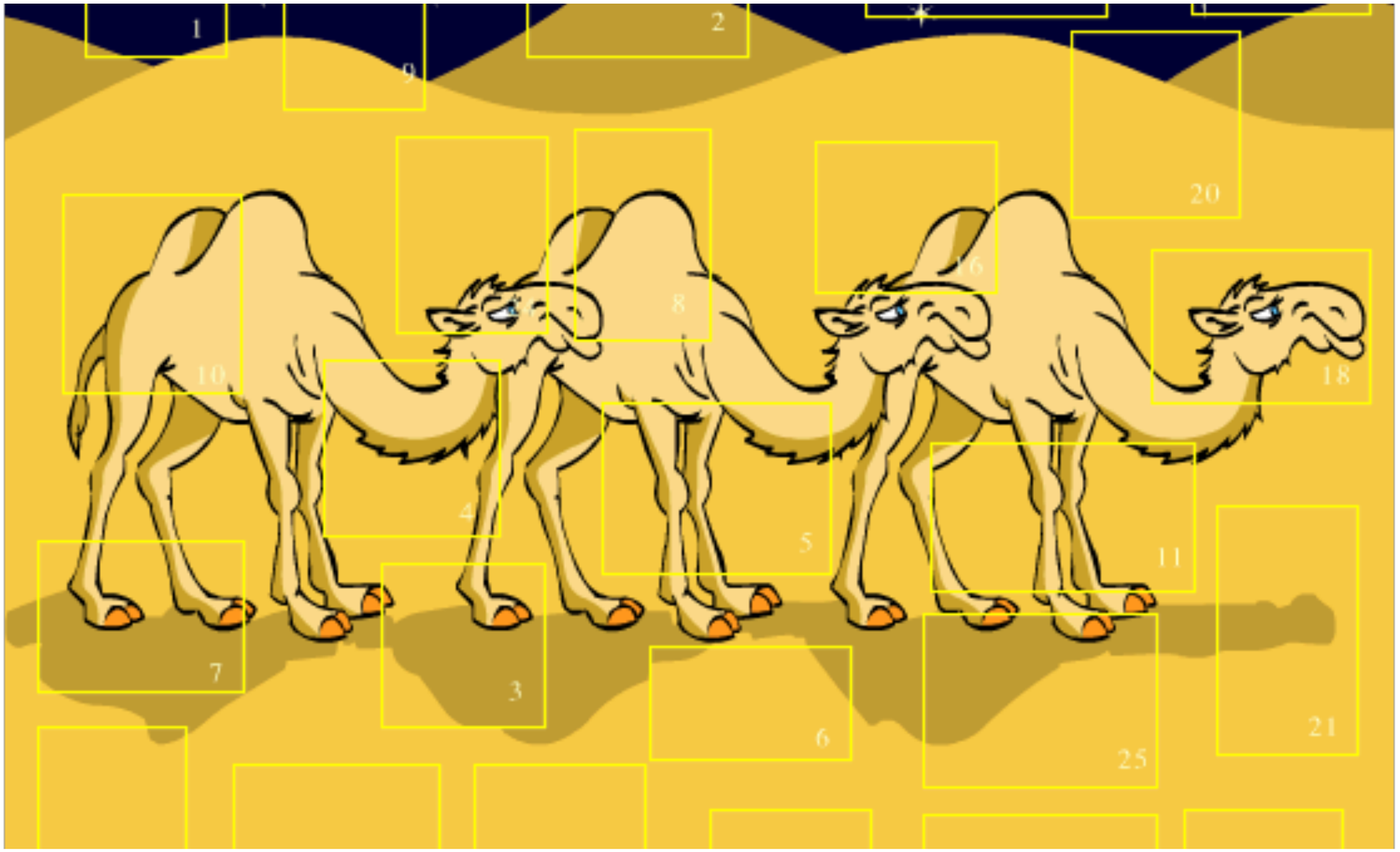

```perl
my $rs = $self->_schema->resultset('Child');

my $result_set_object = $rs->search({ naughty => 0 });
my @children          = $rs->search({ naughty => 0 });
```

```perl
has '_result_sets' => (
    is => 'ro',
    lazy => 1,
    builder => '_build_result_sets',
);

sub _build_result_sets {
    my $self = shift;
    my $rs = $self->_schema->resultset('Child');
    return {
        naughty => $rs->search_rs( naughty => 1 ),
        nice    => $rs->search_rs( naughty => 0 ),
    };
}
```

Context Matters

Perl::Critic

```
package Perl::Critic::Policy::NorthPole::ProhibitSearchMethod;

use strict;
use warnings;

use Perl::Critic::Utils qw( :severities is_method_call );
use base 'Perl::Critic::Policy';

sub default_severity     { return $SEVERITY_HIGHEST }
sub default_themes       { return qw( northpole ) }

# Don't support any parameters in the config file
sub supported_parameters { return () }

# only call 'violates' for elements that are Words (this speeds things up)
sub applies_to           { return 'PPI::Token::Word' }

# examine each PPI::Token::Word in the source code, and violate if it's
# any call to the method 'search'
sub violates {
    my $self    = shift;
    my $element = shift;

    return unless $element->content eq 'search';
    return unless is_method_call($element);

    return $self->violation(
        'Use of the search() method is prohibited',
        'Not allowed',
        $element,
    )
}
```



Here(doc) it is Merry Christmas

Heredocs, New 5.26 heredoc syntax

Heredoc Syntax


```
sub number_of_children {
    my $self             = shift;
    my $naughty_or_nice = shift;

    my $sql = "SELECT count(*)\n" .
              "  FROM children\n" .
              "  WHER naughty_or_nice = \$1";

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self           = shift;
    my $naughty_or_nice = shift;

    my $sql = "SELECT count(*)\n" .
              "  FROM children\n" .
              "  WHER naughty_or_nice = \$1";

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self            = shift;
    my $naughty_or_nice = shift;

    my $sql = <<END
       SELECT count(*)
         FROM children
         WHER naughty_or_nice = \$1
END

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self            = shift;
    my $naughty_or_nice = shift;

    my $sql = <<SQL
       SELECT count(*)
         FROM children
         WHER naughty_or_nice = \$1
SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self             = shift;
    my $naughty_or_nice  = shift;

    my $sql = <<SQL
       SELECT count(*)
         FROM children
         WHER naughty_or_nice = \$1
SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self            = shift;
    my $naughty_or_nice = shift;

    my $sql = <<SQL
       SELECT count(*)
         FROM children
         WHER naughty_or_nice = \$1
SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self            = shift;
    my $naughty_or_nice = shift;

    my $sql = <<SQL
       SELECT count(*)
         FROM children
        WHERE naughty_or_nice = \$1
SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self             = shift;
    my $naughty_or_nice = shift;

    my $sql = <<SQL
       SELECT count(*)
         FROM children
        WHERE naughty_or_nice = \$1
SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self                  sh
    my $naughty_or_nice       rt;

    my $sql = <<'SQL'
       SELECT count(*)
         FROM children
        WHERE naughty_or_nice = $1
SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self             = shift;
    my $naughty_or_nice = shift;

    my $sql = <<'SQL'
       SELECT count(*)
         FROM children
        WHERE naughty_or_nice = $1
SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax

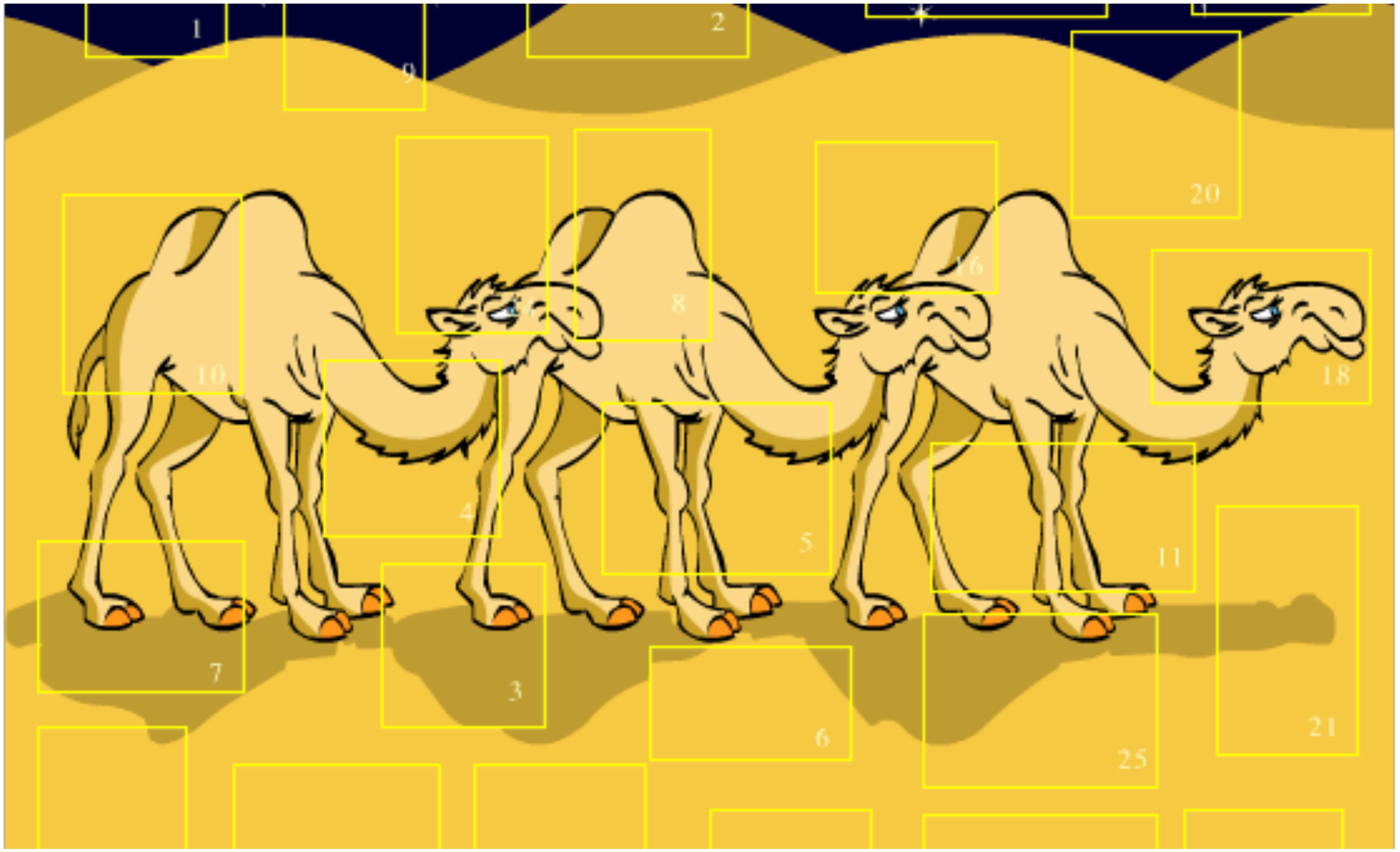

```
sub number_of_children {
    my $self            = shift;
    my $naughty_or_nice = shift;

    my $sql = <<'SQL'
       SELECT count(*)
         FROM children
        WHERE naughty_or_nice = $1
SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self
    my $naughty_or_nic      ift;

    my $sql = <<~'SQL'
       SELECT count(*)
         FROM children
        WHERE naughty_or_nice = $1
    SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```

Heredoc Syntax


```
sub number_of_children {
    my $self            = shift;
    my $naughty_or_nice = shift;

    my $sql = <<~'SQL'
       SELECT count(*)
         FROM children
        WHERE naughty_or_nice = $1
    SQL

    my $dbh = $self->database_handle;
    return $dbh->selectall_arrayref($sql,{},$naughty_or_nice)->[0];
}
```



Santa Baby(cart)

Babycart syntax

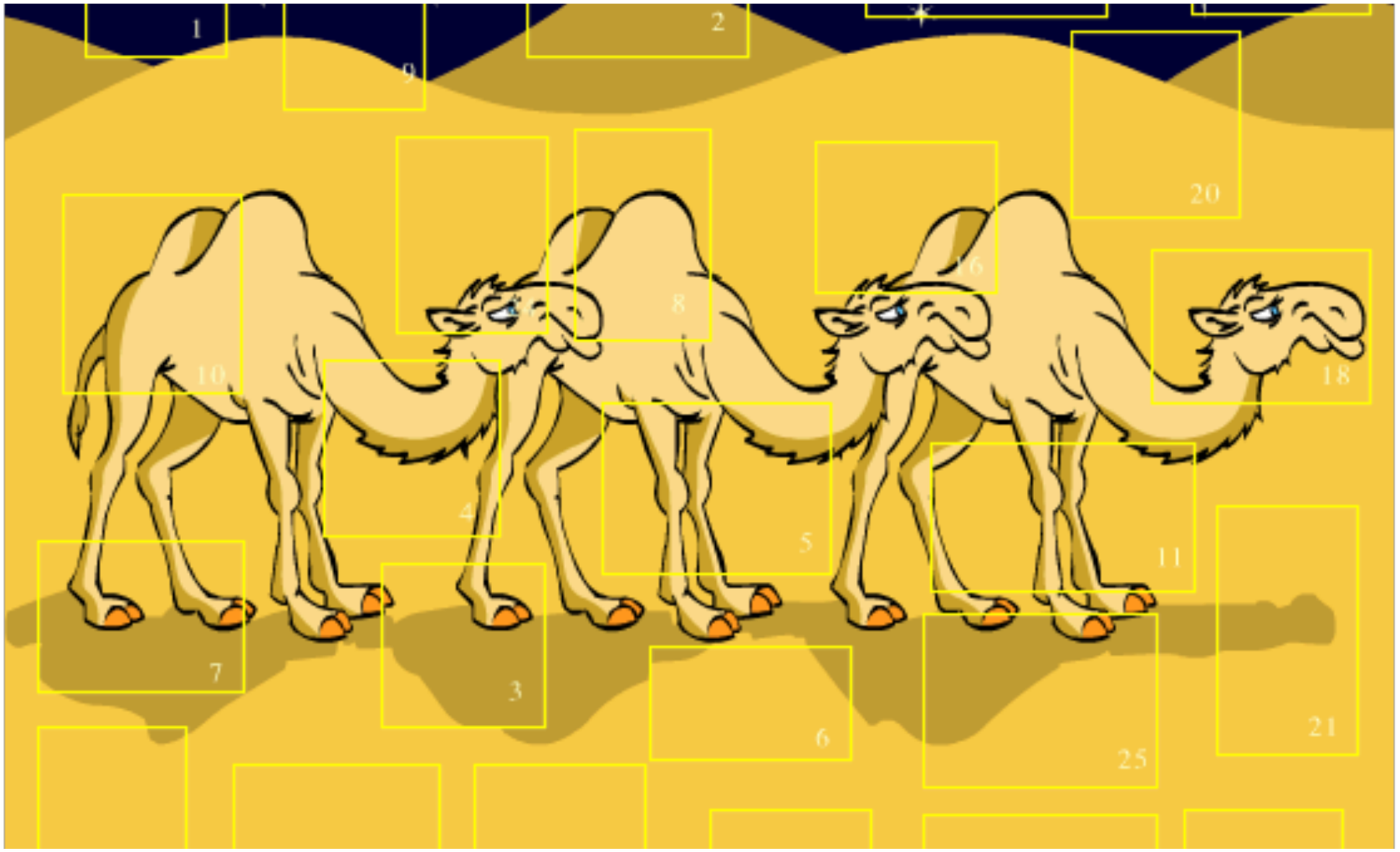
Santa Baby(cart)

5

```
print "Hello @{[ $self->name ]}";
```

Santa Baby(cart)


```
sub find_records_over ($self, $min) {
    return $self->database_handle
                ->selectall_arrayref(<<~"SQL",{},$min);
          SELECT id, @{[ $self->fieldname ]}
            FROM @{[ $self->tablename ]}
           WHERE @{[ $self->fieldname ]} = ?
    SQL
}
```



North Pole Safety Precautions

Building your own strictures

Safety Precaution


```
use strict;
use warnings;
```

Safety Precaution


```
no indirect;
```

Safety Precaution


```
no multidimensional;
```

Safety Precaution


```
use autodie qw( :all );
```

Safety Precaution


```
use utf8;
```

Safety Precaution


```
use feature 'say';
```

Safety Precaution


```
use experimental qw( signatures );
```

Safety Precaution

6

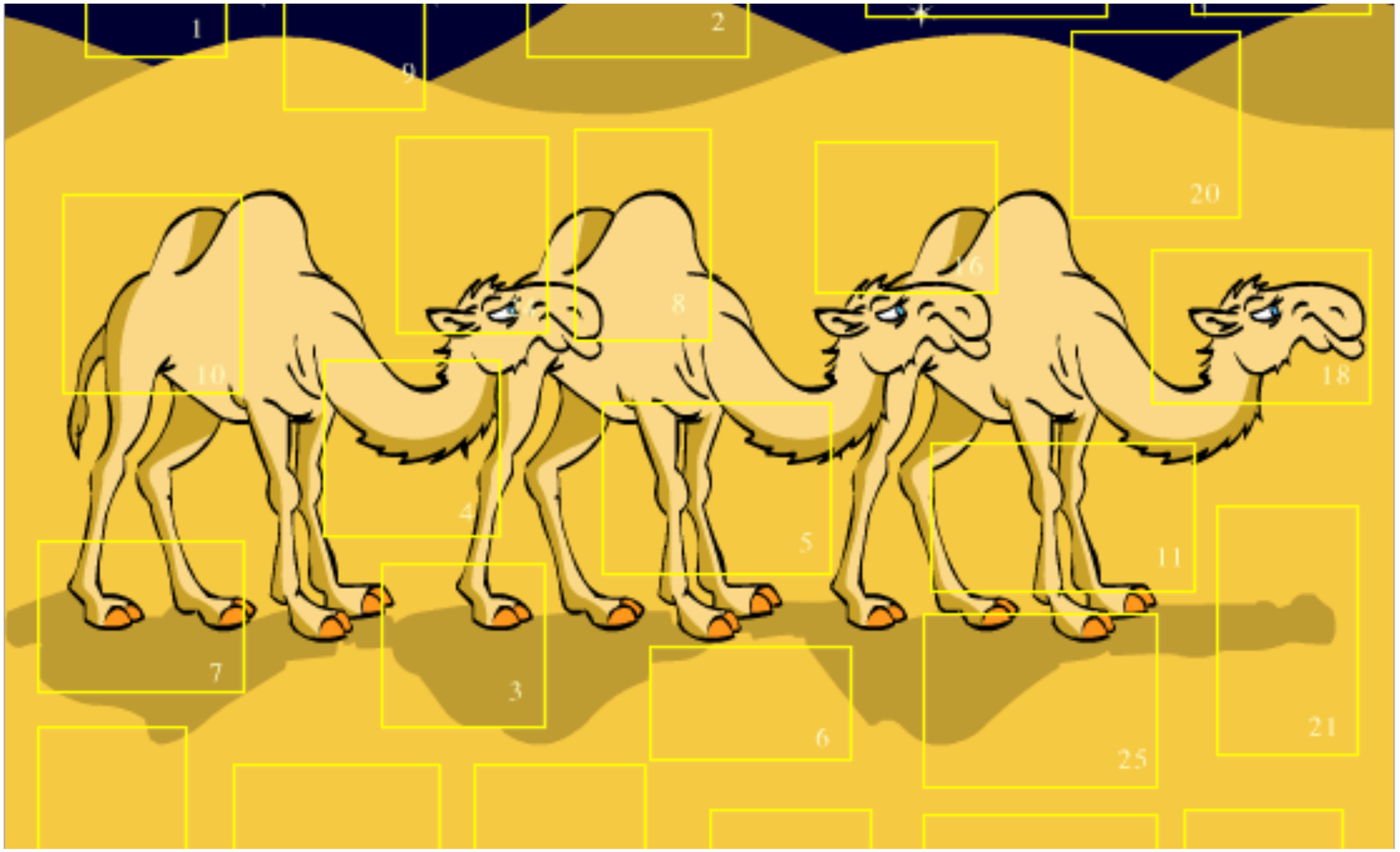
Safety Precaution


```
use NorthPole::OurPerl;
```

Safety Precaution

Import::Into

```
sub import {
    my $caller_level = 1;

    strict->import::into($caller_level);
    warnings->import::into($caller_level);

    my @experiments = qw(
        lexical_subs
        postderef
        signatures
    );
    experimental->import::into( $caller_level, @experiments );
    ...
}
```



Choices, choices, so many choices!

Term::Choose

Choices


```
Mark@travis:/tmp$ perl reindeer.pl
```

```
use Term::Choose qw( choose );
my $reindeer = choose([qw(
    Ruldolph
    Dasher Dancer Prancer Vixen
    Comet Cupid Dunder Blixem
)], { prompt => 'Who is your favourite reindeer?' }) or exit;
say "Cool I like $reindeer too!";
```

Choices


```
Mark@travis:/tmp$ perl reindeer.pl
```

```
use Term::Choose qw( choose );
my $reindeer = choose([qw(
    Ruldolph
    Dasher Dancer Prancer Vixen
    Comet Cupid Dunder Blixem
)], { prompt => 'Who is your favourite reindeer?' }) or exit;
say "Cool I like $reindeer too!";
```

Choices


```
Mark@travis:/tmp$ perl films.pl
```

```
use Term::Choose qw( choose );
my $film = choose(
   \@films,
   { prompt => 'What Christmas movie shall I rent?' },
) or exit;
say "Renting $film";
```

Choices


```
Mark@travis:/tmp$ perl films.pl
```

```
use Term::Choose qw( choose );
my $film = choose(
   \@films,
   { prompt => 'What Christmas movie shall I rent?' },
) or exit;
say "Renting $film";
```

Choices

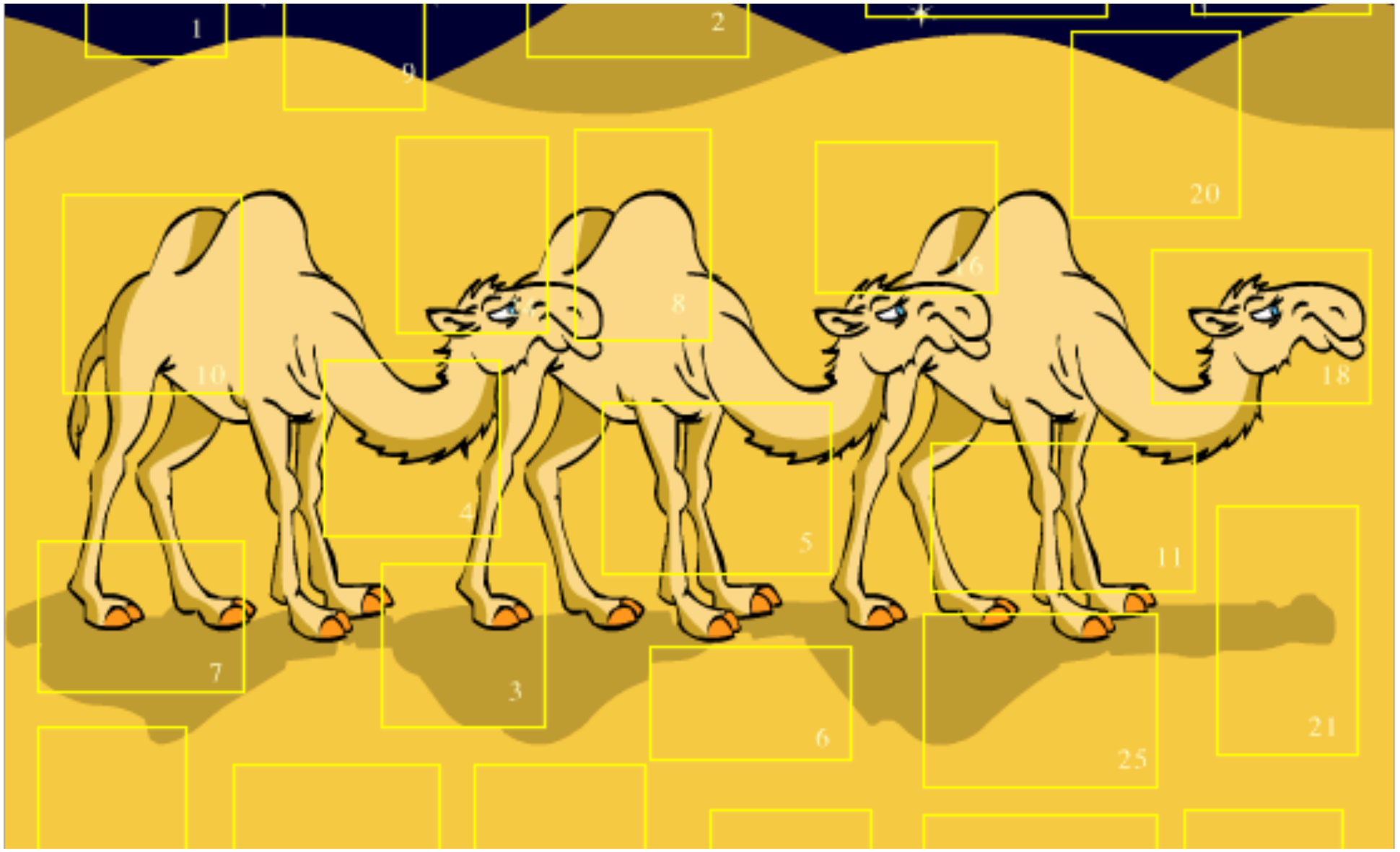

```
Mark@travis:/tmp$ perl wrapping.pl
```

```
use Term::Choose qw( choose );
my @options = choose([
    'with wrapping paper',
    'with a bow on it',
    'with ribbons',
    'with a gift tag',
], { prompt => 'How should I wrap up the present?' });
say "Wrapping up the present with $_" for @options;
```

Choices


```
Mark@travis:/tmp$ perl wrapping.pl
```

```
use Term::Choose qw( choose );
my @options = choose([
    'with wrapping paper',
    'with a bow on it',
    'with ribbons',
    'with a gift tag',
], { prompt => 'How should I wrap up the present?' });
say "Wrapping up the present with $_" for @options;
```



Constantly Merry

Const::Fast

Constantly Merry

- Some things we want to be variable

```
my $mince_pies_eaten = 0;
$mince_pies_eaten++;  # yum!
$mince_pies_eaten++;  # yum! yum!
```

Constantly Merry

- Some things we want to be variable

```
my $mince_pies_eaten = 0;
$mince_pies_eaten++;  # yum!
$mince_pies_eaten++;  # yum! yum!
```

- But others we want to be constant:

```
my $WISE_MEN = 3;
```

Constantly Merry

- Some things we want to be variable

```
my $mince_pies_eaten = 0;
$mince_pies_eaten++;  # yum!
$mince_pies_eaten++;  # yum! yum!
```

- But others we want to be constant:

```
my $WISE_MEN = 3;
```

```
$WISE_MEN++;
```

Constantly Merry

- Some things we want to be variable

```
my $mince_pies_eaten = 0;
$mince_pies_eaten++;  # yum!
$mince_pies_eaten++;  # yum! yum!
```

- But others we want to be constant:

```
my $WISE_MEN = 3;
```

Constantly Merry


```
sub WISE_MEN() { return 3 }
```

Constantly Merry


```
sub WISE_MEN() { return 3 }
```

```
use Lingua::EN::Numbers::Ordinate qw( ordinate );

sub WISE_MEN() { 3 }

for (1..WISE_MEN) {
    say "Getting gift from ".ordinate($_)." wise man";
}
```

Constantly Merry


```
sub WISE_MEN() { return 3 }
```

```
use Lingua::EN::Numbers::Ordinate qw( ordinate );

sub WISE_MEN() { 3 }

for (1..WISE_MEN) {
    say "Getting gift from ".ordinate($_)." wise man";
}
```

Constantly Merry


```
sub WISE_MEN() { return 3 }
```

```
use Lingua::EN::Numbers::Ordinate qw( ordinate );

sub WISE_MEN() { 3 }

for (1..WISE_MEN) {
    say "Getting gift from ".ordinate($_)." wise man";
}
```

```
shell$ perl -Mfeature=say -MO=Deparse script.pl
    sub WISE_MEN () {
        3;
    }
    use Lingua::EN::Numbers::Ordinate ('ordinate');
    use feature 'say';
    foreach $_ (1 .. 3) {
        say 'Getting gift from ' . &ordinate($_) . ' wise man';
    }
```

Constantly Merry


```
sub WISE_MEN() { return 3 }
```

```
use Lingua::EN::Numbers::Ordinate qw( ordinate );

sub WISE_MEN() { 3 }

for (1..WISE_MEN) {
    say "Getting gift from ".ordinate($_)." wise man";
}
```

```
shell$ perl -Mfeature=say -MO=Deparse script.pl
    sub WISE_MEN () {
        3;
    }
    use Lingua::EN::Numbers::Ordinate ('ordinate');
    use feature 'say';
    foreach $_ (1 .. 3) {
        say 'Getting gift from ' . &ordinate($_) . ' wise man';
    }
```

Constantly Merry


```
use Lingua::EN::Numbers::Ordinate ('ordinate');

sub MONTH_OF_XMAS() { ordinate(25) };  # 25th
sub DATE_OF_XMAS()  { ordinate(12) };  # 12th

print 'On the '.DATE_OF_XMAS.' day of the '.MONTH_OF_XMAS.' month...';
```

Constantly Merry

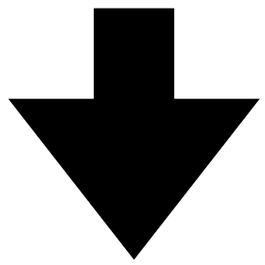

```
use Lingua::EN::Numbers::Ordinate ('ordinate');

use constant MONTH_OF_XMAS => ordinate(25);
use constant DATE_OF_XMAS  => ordinate(12);

print 'On the '.DATE_OF_XMAS.' day of the '.MONTH_OF_XMAS.' month...';
```

Constantly Merry


```
print 'On the '.DATE_OF_XMAS.' day of the '.MONTH_OF_XMAS.' month...';
```

Constantly Merry


```
print 'On the '.DATE_OF_XMAS.' day of the '.MONTH_OF_XMAS.' month...';
```



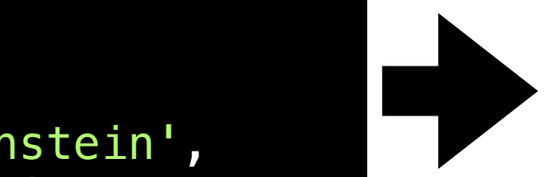
```
print "On the $DATE_OF_XMAS day of the $MONTH_OF_XMAS month...";
```

Constantly Merry


```
use constant WISE1 => 'Melchior';
use constant WISE2 => 'Caspar';
use constant WISE3 => 'Balthazar'

my %gifts = (
    WISE1() => 'Gold',
    WISE2() => 'Frankenstein',
    WISE3() => 'Muwhur?',
);

my $au = $gifts{ WISE1() };
```

Constantly Merry

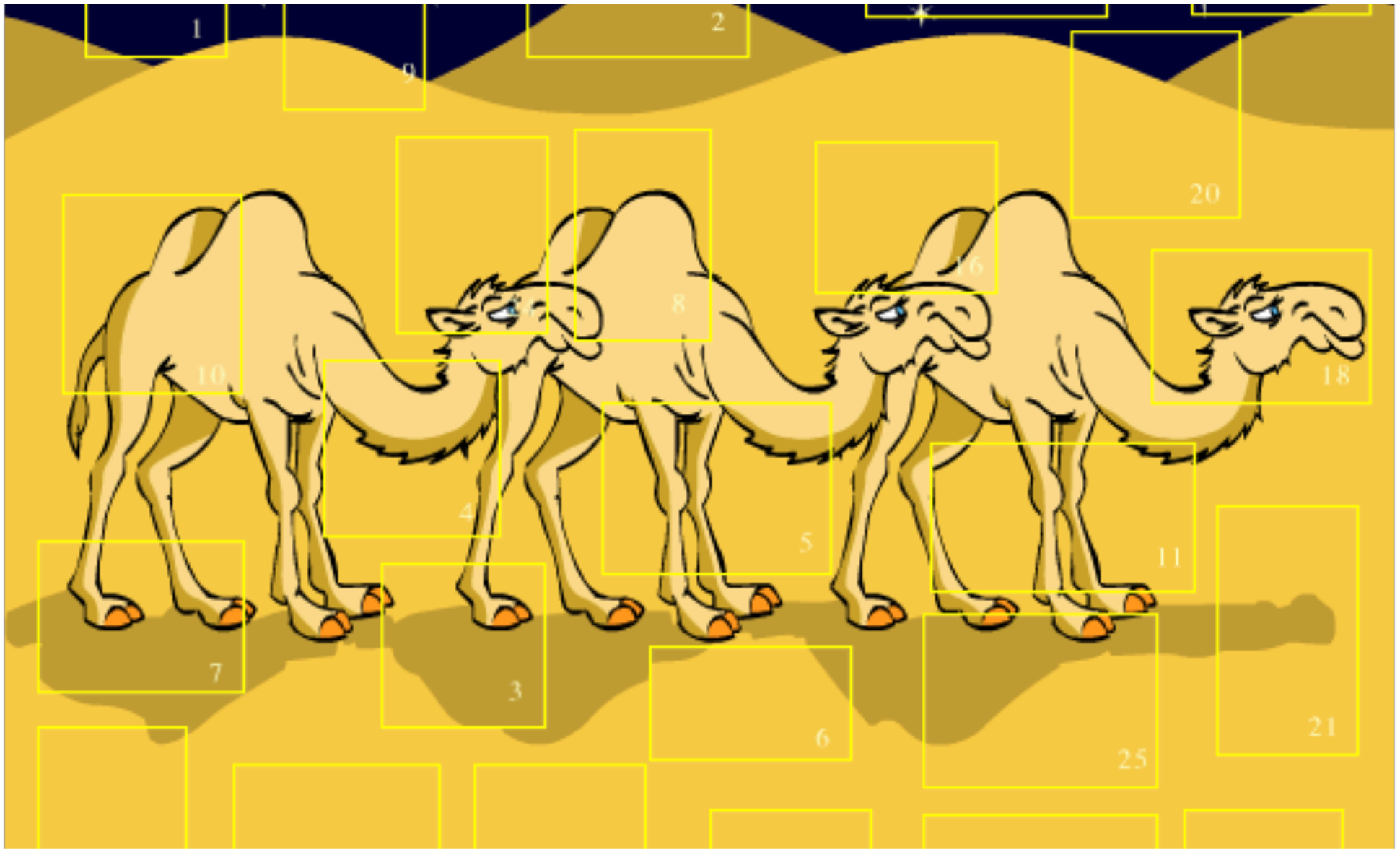

```
use constant WISE1 => 'Melchior';
use constant WISE2 => 'Caspar';
use constant WISE3 => 'Balthazar'

my %gifts = (
    WISE1() => 'Gold',
    WISE2() => 'Frankenstein',
    WISE3() => 'Muwhur?',
);

my $au = $gifts{ WISE1() };
```



```
my %gifts = (
    $WISE1 => 'Gold',
    $WISE2 => 'Frankenstein',
    $WISE3 => 'Muwhur?',
);

my $au = $gifts{ $WISE1 };
```

Constantly Merry


```
use Const::Fast qw( const );

const my $WISE1 => 'Melchior';
const my $WISE2 => 'Caspar';
const my $WISE3 => 'Balthazar'

my %gifts = (
    $WISE1 => 'Gold',
    $WISE2 => 'Frankenstein',
    $WISE3 => 'Muwhur?',
);

my $au = $gifts{ $WISE1 };

say "$WISE2 brought $gifts{ $WISE2 }"
```



Matching the NaughtyNice Formula

Named Regex Patterns
Recursive Decent Parsers in Regexes

NaughtyNice

- You can ***DEFINE*** named blocks in Perl Regexes

```
say "Matches" if $string =~ m{
    (?(DEFINE)
        (?<wise> Melchior | Caspar | Balthazar )
        (?<gift> gold | frankincense | myrrh )
    )

    (?:
        (?&wise) [ ] (brought|gave) [ ](?&gift) |
        (?&gift) [ ] was [ ] (brought|given) [ ] by [ ] (?&wise)
    )
}x;
```

- “Mechior brought gold”
- “gold was brought by Mechior”

NaughtyNice

- You can ***DEFINE*** named blocks in Perl Regexes

```
say "Matches" if $string =~ m{
    (?(DEFINE)
        (?<wise> Melchior | Caspar | Balthazar )
        (?<gift> gold | frankincense | myrrh )
    )

    (?:
        (?&wise) [ ] (brought|gave) [ ](?&gift) |
        (?&gift) [ ] was [ ] (brought|given) [ ] by [ ] (?&wise)
    )
}x;
```

- “Mechior brought gold”
- “gold was brought by Mechior”

NaughtyNice

- You can ***DEFINE*** named blocks in Perl Regexes

```
say "Matches" if $string =~ m{
    (?(DEFINE)
        (?<wise> Melchior | Caspar | Balthazar )
        (?<gift> gold | frankincense | myrrh )
    )

    (?
        (?&wise) [ ] (brought|gave) [ ](?&gift) |
        (?&gift) [ ] was [ ] (brought|given) [ ] by [ ] (?&wise)
    )
}x;
```

- “Mechior brought gold”
- “gold was brought by Mechior”

NaughtyNice

- You can ***DEFINE*** named blocks in Perl Regexes

```
say "Matches" if $string =~ m{
    (?(DEFINE)
        (?<wise> Melchior | Caspar | Balthazar )
        (?<gift> gold | frankincense | myrrh )
    )

    (?:
        (?&wise) [ ] (brought|gave) [ ](?&gift) |
        (?&gift) [ ] was [ ] (brought|given) [ ] by [ ] (?&wise)
    )
}x;
```

- “Mechior ought gold”
- “gold was brought by Mechior”

NaughtyNice

- Take a BNF

```
<letter>     ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" |
                 "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" |
                 "u" | "v" | "w" | "x" | "y" | "z"
<unders>     ::= <letter> | "_" | <letter> <unders> | "_" <unders>
<variable>   ::= <letter> | <letter> <unders>

<digit>      ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<startdigit> ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<digits>     ::= <digit> | <digit> <digits>
<number>     ::= <startdigit> | <startdigit> <digits>

<oper>       ::= "+" | "-" | "*" | "/"
<bracket>    ::= "(" <expression> ")"
<thing>      ::= <number> | <variable> | <bracket>
<expression> ::= <thing> | <thing> <oper> <expression>
```

NaughtyNice

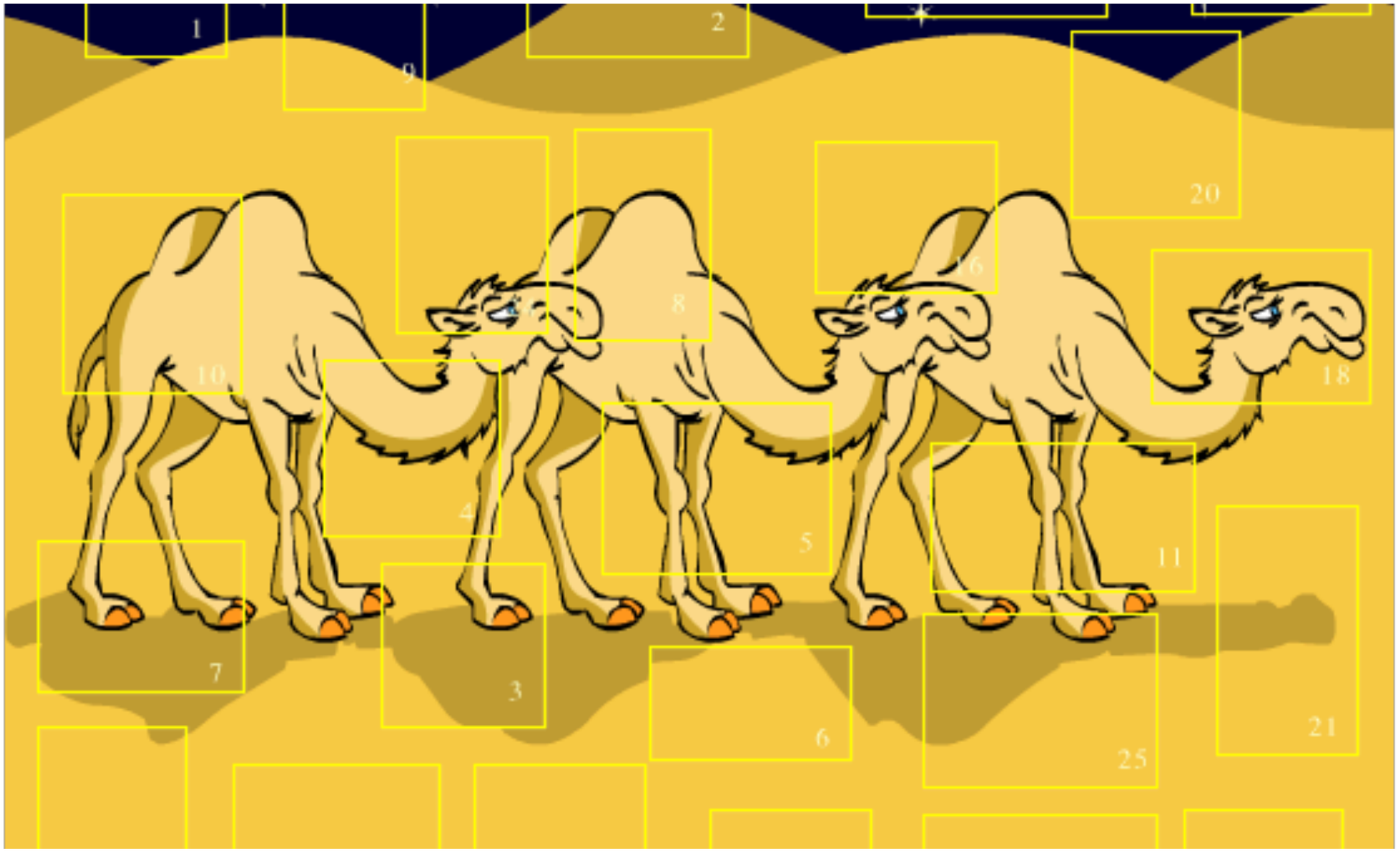
- Take a BNF….simplify it…

```
<variable>   ::= qr/[a-z][a-z_]*/
<number>     ::= qr/[1-9][0-9]*/
<oper>       ::= qr/[+*/-]/
<bracket>    ::= "(" <expression> ")"
<thing>      ::= <number> | <variable> | <bracket>
<expression> ::= <thing> | <thing> <oper> <expression>
```

NaughtyNice

- Take a BNF….simplify it…and turn it right into a Regex

```
say "Matches" if $string =~ m{
  (?(DEFINE)
    (?<variable>    [a-z][_a-z]* )
    (?<number>      [1-9][0-9]* )
    (?<oper>        [+*/-] )
    (?<bracket>     [(] (?&expression) [)] )
    (?<thing>       (?&number) | (?&variable) | (?&bracket) )
    (?<expression>  (?&thing) | (?&thing) (?&oper) (?&expression) )
  )

  \A (?&expression) \z
}x;
```



Evaluating the NaughtyNice Formula

Eval::Closure

NaughtyNice 2

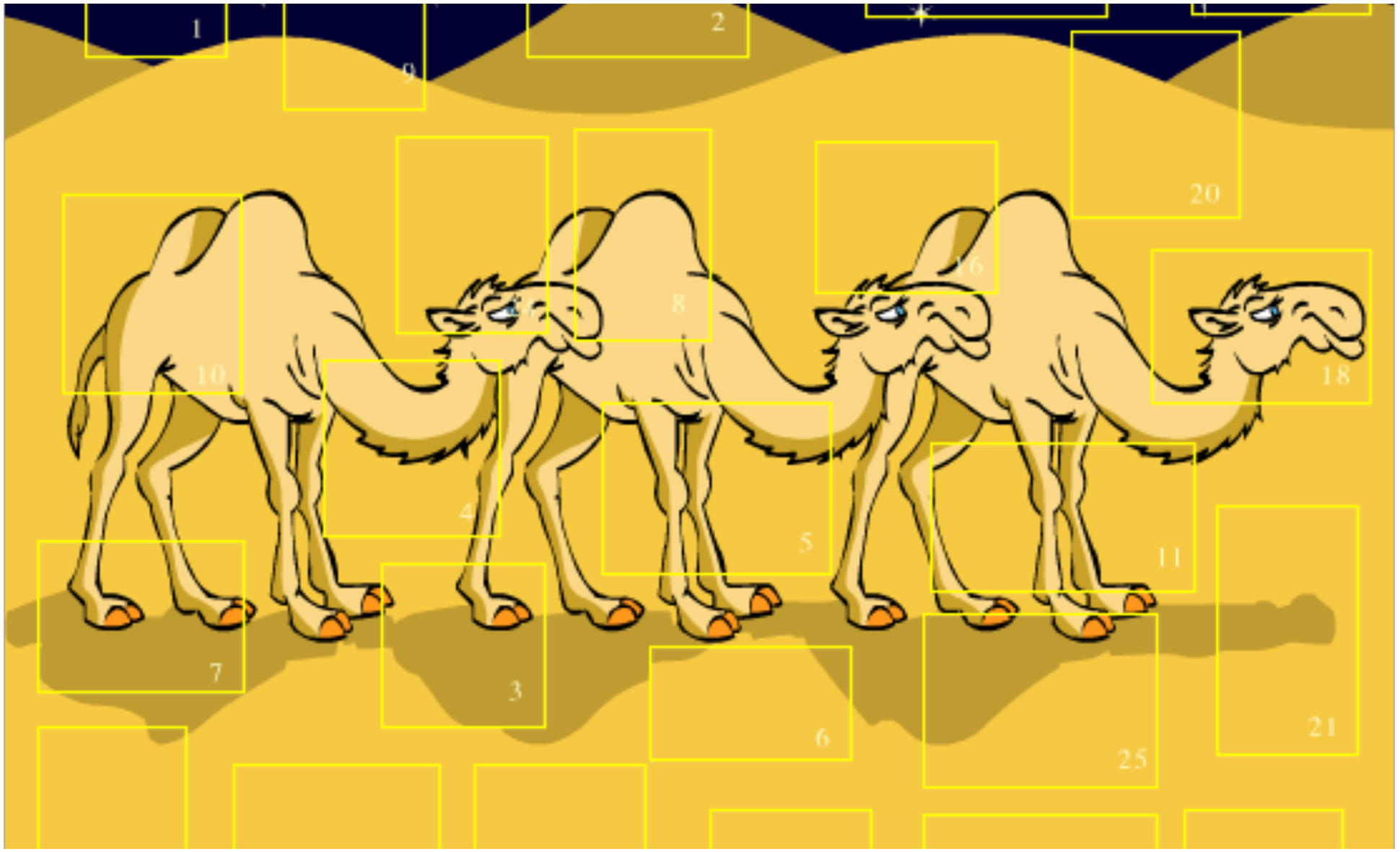

```
sub create_function ($self, $user_code) {
    my $splines = $self->reticulate_splines;
    return eval "sub { $user_code }";
}
```

- Code generation often uses string eval to create new compiled code
- In the above example the $user_code can *see* $splines
- But it can **ALSO** see **$self**

NaughtyNice 2


```perl
use Eval::Closure qw( eval_closure );

sub create_function ($self, $user_code) {
    return eval_closure(
        source => "sub { $user_code }",
        environment => {
            ‘$splines' => $self->reticulate_splines,
        },
    );
}
```

- Using eval_closure hides all but the passed in values!
- Moose uses this a lot



Tiny Path Handling

Path::Tiny

Tiny Paths

- **Path::Tiny** is a *tiny* file abstraction

```
use Path::Tiny qw( path );

my $dir                  = '/Users/Mark/shopping_list';
my $with_trailing_slash = "$dir/";
my $file                 = 'mia_birthday.txt';

# WRONG: prints "/Users/Mark/shopping_listmia_birthday.txt"
say "$dir$file";

# WRONG: prints "/Users/Mark/shopping_list//mia_birthday.txt"
say "$with_trailing_slash/$file";

# RIGHT: both prints "/Users/Mark/shopping_list/mia_birthday.txt"
say path($dir, $file);
say path($with_trailing_slash, $file);
```

Tiny Paths


```
my $fh = path('/tmp/foo')->openr;
```

Tiny Paths


```
my $fh = path(‘/tmp/foo')->openw;
```

Tiny Paths


```
my $fh = path('/tmp/foo')->openw_utf8;
```

Tiny Paths

- Automatic **flock**ing when writing / appending!

```
$xmas_list->append_utf8("A \N{BIRD} for the \N{PEAR}\N{DECIDUOUS TREE}");
```

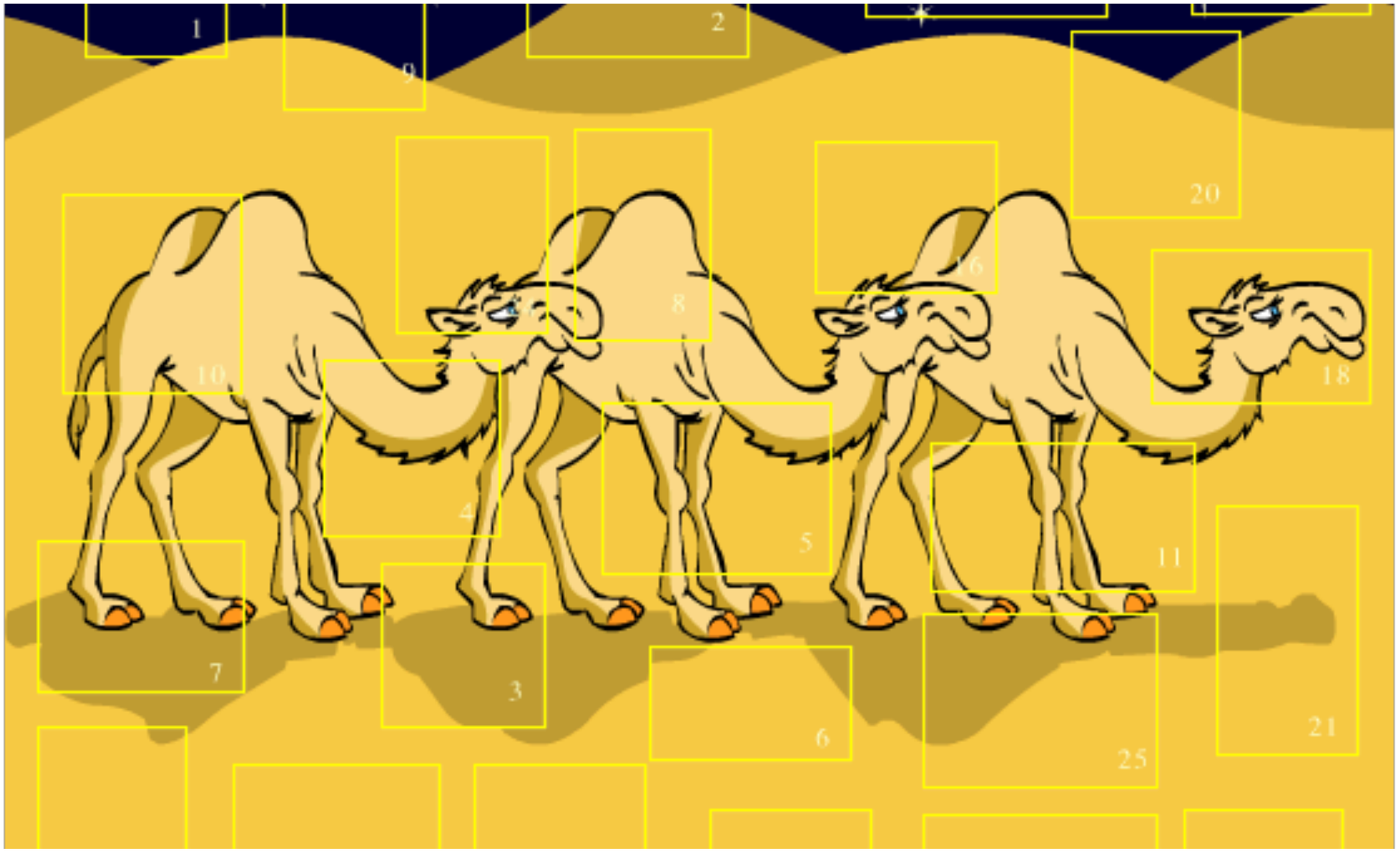
A 🐦 for the 🍐🌳

Tiny Paths

- Altering each line of a file with callback

```
my $counter = 1;
$xmas_list->edit_lines_utf8(sub {
    $_ = "$counter. $_";
    $counter++;
});
```

1. A 🐦 for the 🍐🌳
2. Two 📞 🐦



A Christmas Memory

Devel::MAT

Xmas Memory

Who’s Been Eating All My Memory?

Xmas Memory

Devel::MAT

```
shell$ pmat script.pl.pmat
Perl memory dumpfile from perl 5.22.0
Heap contains 2366 objects
pmat>
```

Xmas Memory

Devel::MAT

```
shell$ pmat script.pl.pmat
Perl memory dumpfile from perl 5.22.0
Heap contains 2366 objects
pmat> help
callstack - Display the call stack
count     - Count the various kinds of SV
elems     - List the elements of an ARRAY SV
find      - List SVs matching given criteria
help      - Display a list of available commands
identify  - Identify an SV by its referrers
io        - Commands working with IO SVs
largest   - Find the largest SVs by size
roots     - Display a list of the root SVs
show      - Show information about a given SV
sizes     - Summarize object and byte counts across different SV types
symbols   - Display a list of the symbol table
values    - List the values of a HASH-like SV
```

Xmas Memory


```
#!/usr/bin/perl

use strict;
use warnings;

my $self = "happy";
$self x= 1024*1024*1024;  # 5 Gigs!
```

Xmas Memory


```perl
#!/usr/bin/perl

use strict;
use warnings;

my $self = "happy";
$self x= 1024*1024*1024;  # 5 Gigs!

die;
```

Xmas Memory


```
#!/usr/bin/perl

use strict;
use warnings;

my $self = "happy";
$self x= 1024*1024*1024;  # 5 Gigs!

die;
```

```
shell$ perl -MDevel::MAT::Dumper=-dump_at_DIE script.pl
Dumping to /Users/Cedar/test/script.pl.pmat because of DIE
Died at script.pl line 9.
```

Xmas Memory


```
pmat> largest
   SCALAR(PV) at 0x7fab7d016a60: 5.0 GiB
   HASH(540) at 0x7fab7d002e90=strtab: 33.6 KiB
   INVLIST() at 0x7fab7e01bf88: 9.9 KiB
   INVLIST() at 0x7fab7e01c018: 9.8 KiB
   STASH(84) at 0x7fab7d0030d0=defstash: 3.1 KiB
   others: 228.5 KiB
```

Xmas Memory


```
pmat> largest
   SCALAR(PV) at 0x7fab7d016a60: 5.0 GiB
   HASH(540) at 0x7fab7d002e90=strtab: 33.6 KiB
   INVLIST() at 0x7fab7e01bf88: 9.9 KiB
   INVLIST() at 0x7fab7e01c018: 9.8 KiB
   STASH(84) at 0x7fab7d0030d0=defstash: 3.1 KiB
   others: 228.5 KiB
```

Xmas Memory


```
pmat> largest
   SCALAR(PV) at 0x7fab7d016a60: 5.0 GiB
   HASH(540) at 0x7fab7d002e90=strtab: 33.6 KiB
   INVLIST() at 0x7fab7e01bf88: 9.9 KiB
   INVLIST() at 0x7fab7e01c018: 9.8 KiB
   STASH(84) at 0x7fab7d0030d0=defstash: 3.1 KiB
   others: 228.5 KiB
```

```
pmat> identify 0x7fab7d016a60
    SCALAR(PV) at 0x7fab7d016a60 is:
    \-the lexical $self at depth 1 of CODE() at 0x7fab7d003478=main_cv
     \-the main code
```

Xmas Memory


```
sub santas_workshop {
    my $phrase = "happy";

    my $self = {
        attributes => {
            mood => scalar($phrase x (1024*1024*1024)),
            height => 'short',
        },
        name => 'Cedar Greentrifle'
    };

    die();
}
```

```
pmat> identify 0x7f9f9b011860
    SCALAR(PV) at 0x7f9f9b011860 is:
    \-value {mood} of HASH(1) at 0x7f9f9b003250, which is:
      \-(via RV) value {attributes} of HASH(1) at 0x7f9f9b011878, which is:
        \-(via RV) the lexical $self at depth 1 of CODE(PP) at 0x7f9f9b016a78, which is:
          \-the symbol '&main::santas_workshop'
```

Xmas Memory


```perl
#!/usr/bin/perl

use strict;
use warnings;

sub expand {
    my $count = shift() - 1;
    return "Merry Christmas" if $count == 0;
    return {
        map { $_ => expand($count) } 1..10
    };
}

my $big_data_structure = expand(6);
die();
```

Xmas Memory


```
Perl memory dumpfile from perl 5.22.0
 Heap contains 124628 objects
 pmat> largest
 HASH(548) at 0x7fb053002e90=strtab: 33.9 KiB
 INVLIST() at 0x7fb053085f88: 9.9 KiB
 INVLIST() at 0x7fb053086018: 9.8 KiB
 STASH(85) at 0x7fb0530030d0=defstash: 3.2 KiB
 HASH(70) at 0x7fb0530217c8: 2.7 KiB
 others: 10.4 MiB
```

Xmas Memory


```
pmat> largest --owned
 CODE() at 0x7fb053003478=main_cv: 10.2 MiB: of which
  |   PAD(3) at 0x7fb053003490: 10.2 MiB: of which
  |    |   REF() at 0x7fb053016ad8: 10.2 MiB
  |    |   HASH(10) at 0x7fb053011950: 10.2 MiB
  |    |   others: 10.2 MiB
  |   SCALAR(UV) at 0x7fb053016a60: 24 bytes
 STASH(85) at 0x7fb0530030d0=defstash: 70.4 KiB: of which
  |   GLOB(%*) at 0x7fb053017918: 22.6 KiB: of which
  |    |   STASH(2) at 0x7fb053021678: 22.4 KiB
  |    |   GLOB(%*) at 0x7fb053034d48: 22.1 KiB
  |    |   others: 22.1 KiB
  |   GLOB(%*) at 0x7fb053080b20: 14.6 KiB: of which
  |    |   STASH(39) at 0x7fb053080b38: 14.5 KiB
  |    |   GLOB(&*) at 0x7fb055015de0: 2.4 KiB
  |    |   others: 12.7 KiB
  |   GLOB(%*) at 0x7fb0530809e8: 3.9 KiB: of which
  |    |   STASH(3) at 0x7fb053080a60: 3.8 KiB
  |    |   GLOB(&*) at 0x7fb053080ad8: 3.3 KiB
  |    |   others: 3.3 KiB
  |   others: 26.1 KiB
 HASH(548) at 0x7fb053002e90=strtab: 33.9 KiB
 REF() at 0x7fb053085fb8: 10.0 KiB: of which
```

Xmas Memory


```
pmat> largest --owned
 CODE() at 0x7fb053003478=main_cv: 10.2 MiB: of which
  |   PAD(3) at 0x7fb053003490: 10.2 MiB: of which
  |    |   REF() at 0x7fb053016ad8: 10.2 MiB
  |    |   HASH(10) at 0x7fb053011950: 10.2 MiB
  |    |   others: 10.2 MiB
  |   SCALAR(UV) at 0x7fb053016a60: 24 bytes
STASH(85) at 0x7fb0530030d0=defstash: 70.4 KiB: of which
  |   GLOB(%*) at 0x7fb053017918: 22.6 KiB: of which
  |    |   STASH(2) at 0x7fb053021678: 22.4 KiB
  |    |   GLOB(%*) at 0x7fb053034d48: 22.1 KiB
  |    |   others: 22.1 KiB
  |   GLOB(%*) at 0x7fb053080b20: 14.6 KiB: of which
  |    |   STASH(39) at 0x7fb053080b38: 14.5 KiB
  |    |   GLOB(&*) at 0x7fb055015de0: 2.4 KiB
  |    |   others: 12.7 KiB
  |   GLOB(%*) at 0x7fb0530809e8: 3.9 KiB: of which
  |    |   STASH(3) at 0x7fb053080a60: 3.8 KiB
  |    |   GLOB(&*) at 0x7fb053080ad8: 3.3 KiB
  |    |   others: 3.3 KiB
  |   others: 26.1 KiB
HASH(548) at 0x7fb053002e90=strtab: 33.9 KiB
REF() at 0x7fb053085fb8: 10.0 KiB: of which
```

Xmas Memory


```
pmat> largest --owned
 CODE() at 0x7fb053003478=main_cv: 10.2 MiB: of which
  |   PAD(3) at 0x7fb053003490: 10.2 MiB: of which
  |    |   REF() at 0x7fb053016ad8: 10.2 MiB
  |    |   HASH(10) at 0x7fb053011950: 10.2 MiB
  |    |   others: 10.2 MiB
```

```
pmat> identify 0x7fb053016ad8
    REF() at 0x7fb053016ad8 is:
    \-the lexical $big_data_structure at depth 1 of CODE() at
                                          0x7fb053003478=main_cv, which is:
      -\the main code
```

Xmas Memory


```
my $greeting;
$greeting = {
    data => 'Merry Christmas',
    link => \$greeting,
};
```

Xmas Memory

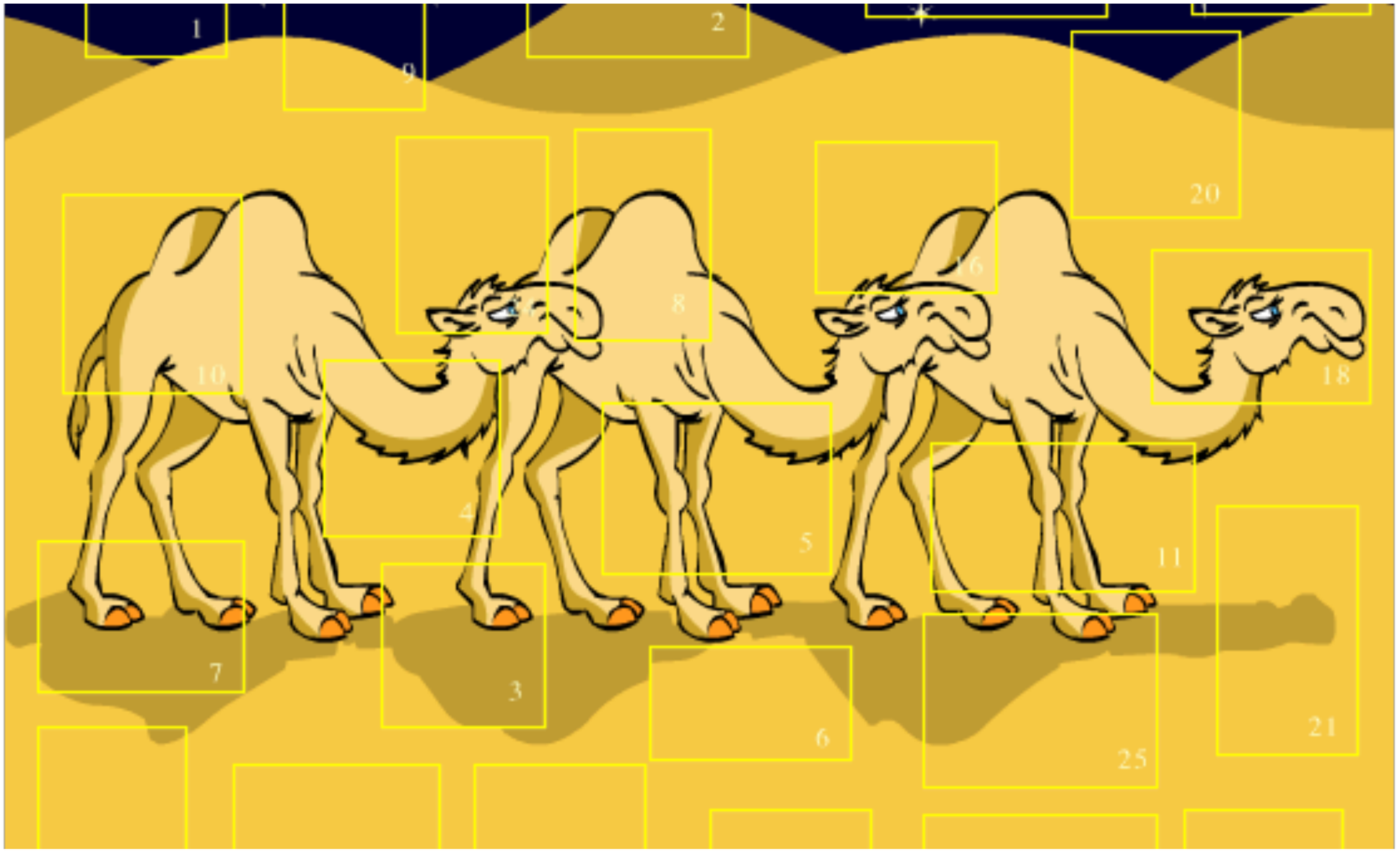

```
my $greeting;
$greeting = {
    data => 'Merry Christmas',
    link => \$greeting,
};
```

```
pmat> find pv --eq 'Merry Christmas'
    SCALAR(PV) at 0x7fa2d56e3330: "Merry Christmas"
    SCALAR(PV) at 0x7fa2d54fa2d0: "Merry Christmas"
    SCALAR(PV) at 0x7fa2d57a0588: "Merry Christmas"
    SCALAR(PV) at 0x7fa2d5052890: "Merry Christmas"
    SCALAR(PV) at 0x7fa2d3901790: "Merry Christmas"
    SCALAR(PV) at 0x7fa2d29b26a0: "Merry Christmas"
    SCALAR(PV) at 0x7fa2d518f440: "Merry Christmas”
```

Xmas Memory


```
my $greeting;
$greeting = {
    data => 'Merry Christmas',
    link => \$greeting,
};
```

```
pmat> find pv --eq 'data'
    SCALAR(PV) at 0x7fa2d2003340: "data"

    pmat> identify 0x7fa2d2003340
    SCALAR(PV) at 0x7fa2d2003340 is:
    \-a constant of CODE() at 0x7fa2d2003478=main_cv, which is:
      \-the main code
```



It's All There in the Numbers!

charnames::viacode, Unicode::UCD

All in the Numbers

32

All in the Numbers


```
use charnames qw(:full);
my @chars = split //, “ᔕᓇ";
say charnames::viacode(ord $_) foreach @chars;
```

All in the Numbers


```
use charnames qw(:full);
my @chars = split //, "১২";
say charnames::viacode(ord $_) foreach @chars;
```

```
BENGALI DIGIT ONE
BENGALI DIGIT TWO
```

All in the Numbers

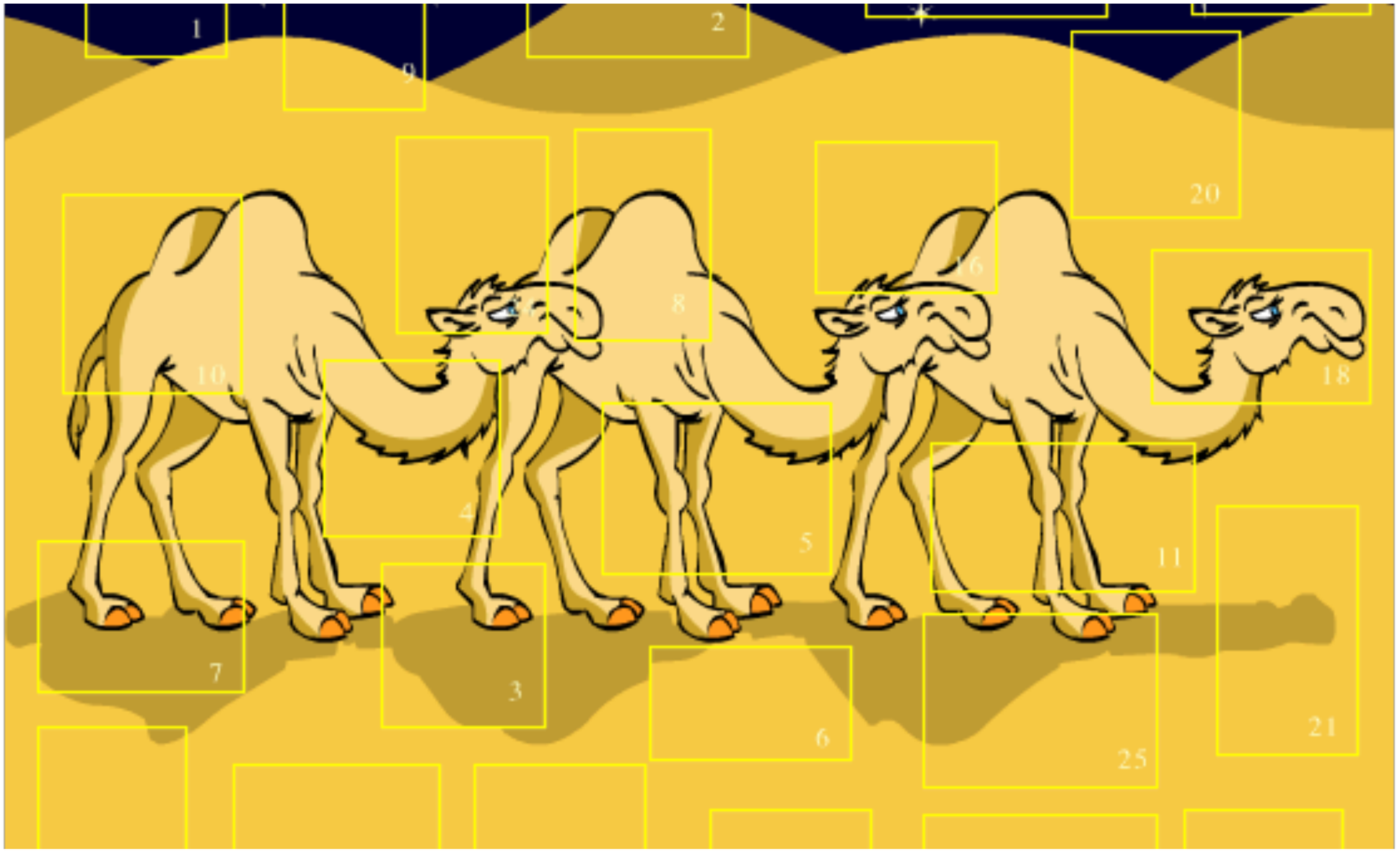

```
use charnames qw(:full);
my @chars = split //, “১২";
say charnames::viacode(ord $_) foreach @chars;
```

```
BENGALI DIGIT ONE
BENGALI DIGIT TWO
```

১২ ⇝ 12

All in the Numbers


```
use Unicode::UCD qw(num);
say num("ᏁᏆ");  # prints "12"
```



Prereqing around the Christmas Tree

Hooking @INC, Perl::PrereqScanner,
Perl::PrereqScanner::Lite, Perl::PrereqScanner::NotSoLite

Prerequisites


```
package oreLoadingDebugging;
unshift @INC, sub { say $_[1]; return };
1;
```

Prerequisites


```
package oreLoadingDebugging;
unshift @INC, sub { say $_[1]; return };
1;
```

```
perl –MoreLoadingDebugging –c myscript.pl
```

Prerequisites


```
package oreLoadingDebugging;
unshift @INC, sub { say $_[1]; return };
1;
```

```
perl -MoreLoadingDebugging -c myscript.pl
```

- Problems:
  - Doesn't find run time dependencies
  - Totally unsafe
  - Must compile
  - Filename cleanup is tedious
  - Indirect dependencies

Prerequisites

- Instead of **executing** the source code we **parse it**
  - Perl::PrereqScanner
  - Perl::PrereqScanner::Lite
  - Perl::PrereqScanner::NotSoLite

Prerequisites


```perl
use Perl::PrereqScanner;
use JSON::PP qw(encode_json);
my $scanner = Perl::PrereqScanner->new;
my $requirements = $scanner->scan_file('myscript.pl');
print encode_json( $requirements->as_string_hash );
```

Perl::PrereqScanner

- Works with Traditional Perl, Moose
- Based on PPI

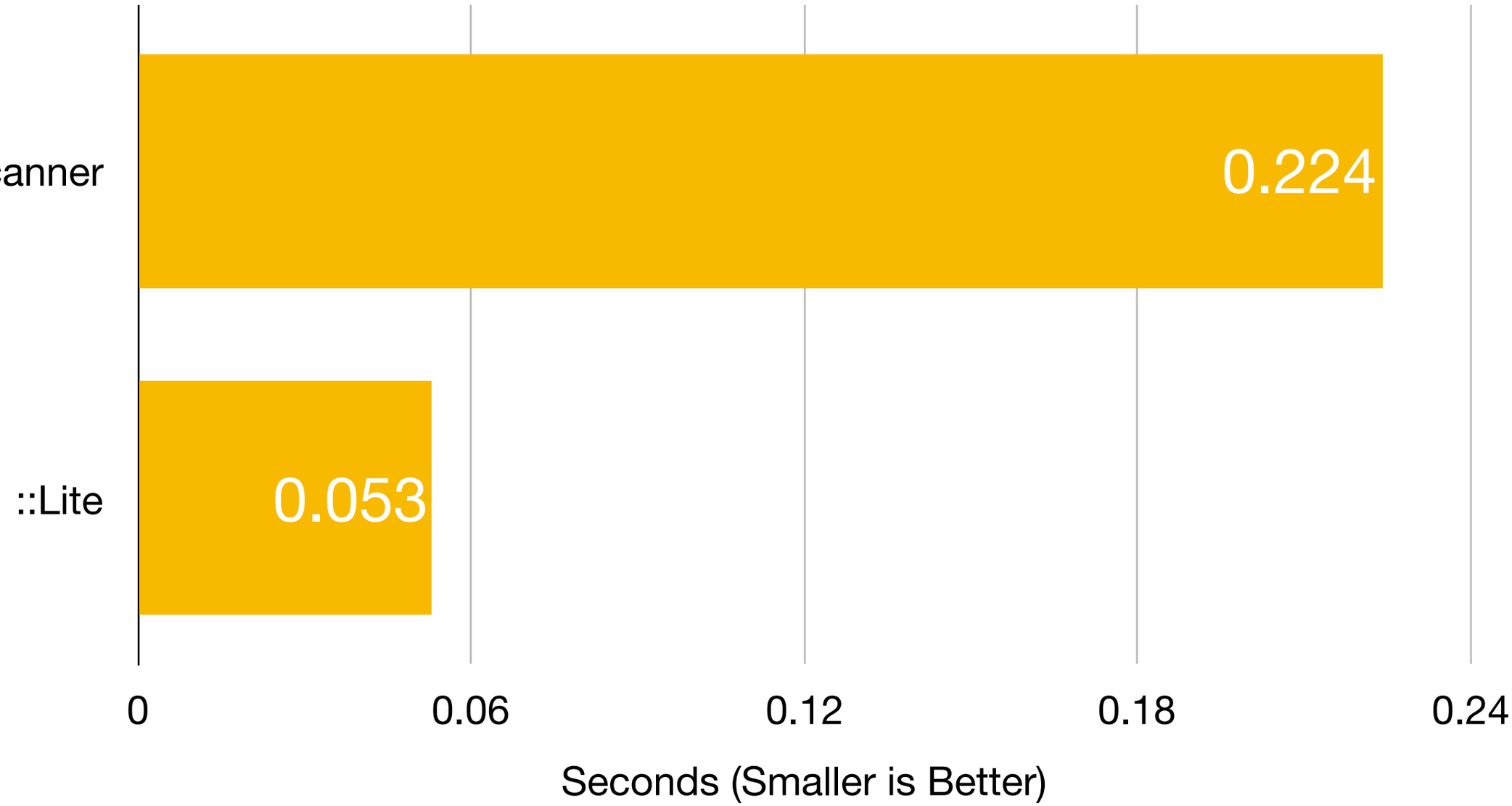
Prerequisites


```perl
use Perl::PrereqScanner::Lite;
use JSON::PP qw(encode_json);
my $scanner = Perl::PrereqScanner::Lite->new;
my $requirements = $scanner->scan_file('myscript.pl');
print encode_json( $requirements->as_string_hash );
```

Perl::PrereqScanner::Lite

- Works with Traditional Perl, Moose
- Faster XS

Prerequisites

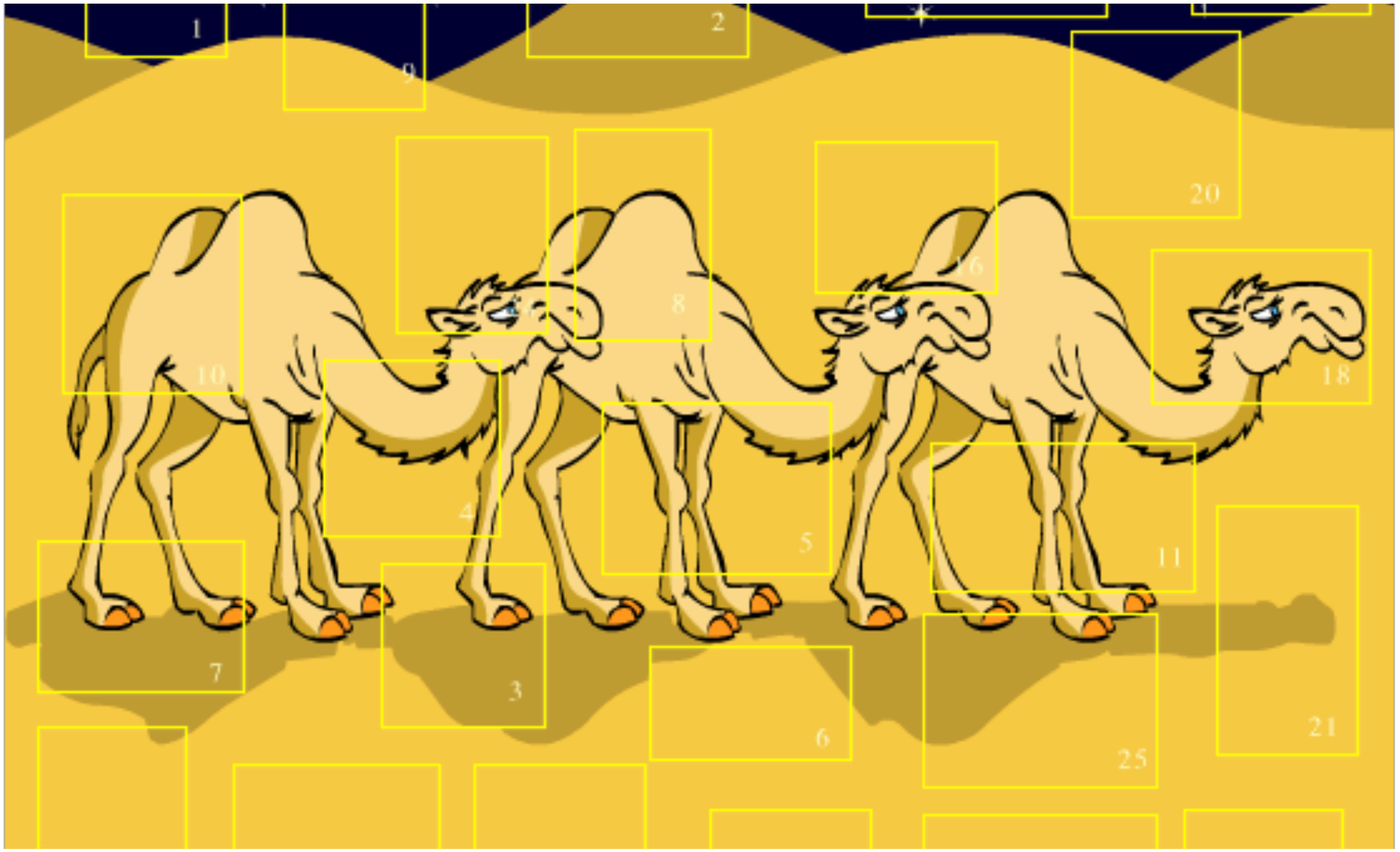


Prerequisites


```
use Perl::PrereqScanner::NotQuiteLite;
use JSON::PP qw(encode_json);
my $scanner = Perl::PrereqScanner::NotQuiteLite->new( suggests => 1 );
my $result = $scanner->scan_file('Rudolph.pm');
say 'requires: '.encode_json( $result->requires->as_string_hash );
say 'suggests: '.encode_json( $result->suggests->as_string_hash );
```

Perl::PrereqScanner::NotSoLite

- **FINDS OPTIONAL DEPENDANCIES TOO**
  - Looks inside string eval code, etc.



Mojolicious on the Command Line

ojo

ojo


```
perl -Mojo -E 'say g("http://goo.gl/EsGk3b")->text'
```

- `-Mojo` is actually shortcut for `use ojo`
- Exports Mojolicious one character functions including…
  - `g(…)` / `h(…)` / `p(…)` / `t(…)` / `u(…)` to do `GET` / `HEAD` / `POST` / `PATCH` / `PUT` requests
  - `r(…)` is a Data::Dumper shortcut
  - `j(…)` turns a data structure to a JSON string, or a JSON string into a data structure
  - `f(…)` creates a Mojo::File instance (like a Path::Tiny path)
  - `c(…)` creates a Mojo::Collection (fancy OO-array)

ojo


```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
            p(
                "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
                {},
                json => {
                    longUrl => $_->attr('href')
                },
            )->json("/id") .
            ": " . $_->text . "\n"
    })->join
```

ojo


```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
            p(
                "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
                {},
                json => {
                    longUrl => $_->attr('href')
                },
            )->json("/id") .
            ": " . $_->text . "\n"
    })->join
```

ojo


```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
            p(
                "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
                {},
                json => {
                    longUrl => $_->attr('href')
                },
            )->json("/id") .
            ": " . $_->text . "\n"
    })->join
```

ojo


```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
            p(
                "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
                {},
                json => {
                    longUrl => $_->attr('href')
                },
            )->json("/id") .
            ": " . $_->text . "\n"
    })->join
```

ojo


```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
            p(
                "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
                {},
                json => {
                    longUrl => $_->attr('href')
                },
            )->json("/id") .
            ": " . $_->text . "\n"
    })->join
```

ojo


```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
        p(
            "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
            {},
            json => {
                longUrl => $_->attr('href')
            },
        )->json("/id") .
        ": " . $_->text . "\n"
})->join
```

ojo


```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
            p(
                "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
                {},
                json => {
                    longUrl => $_->attr('href')
                },
            )->json("/id") .
            ": " . $_->text . "\n"
    })->join
```

ojo

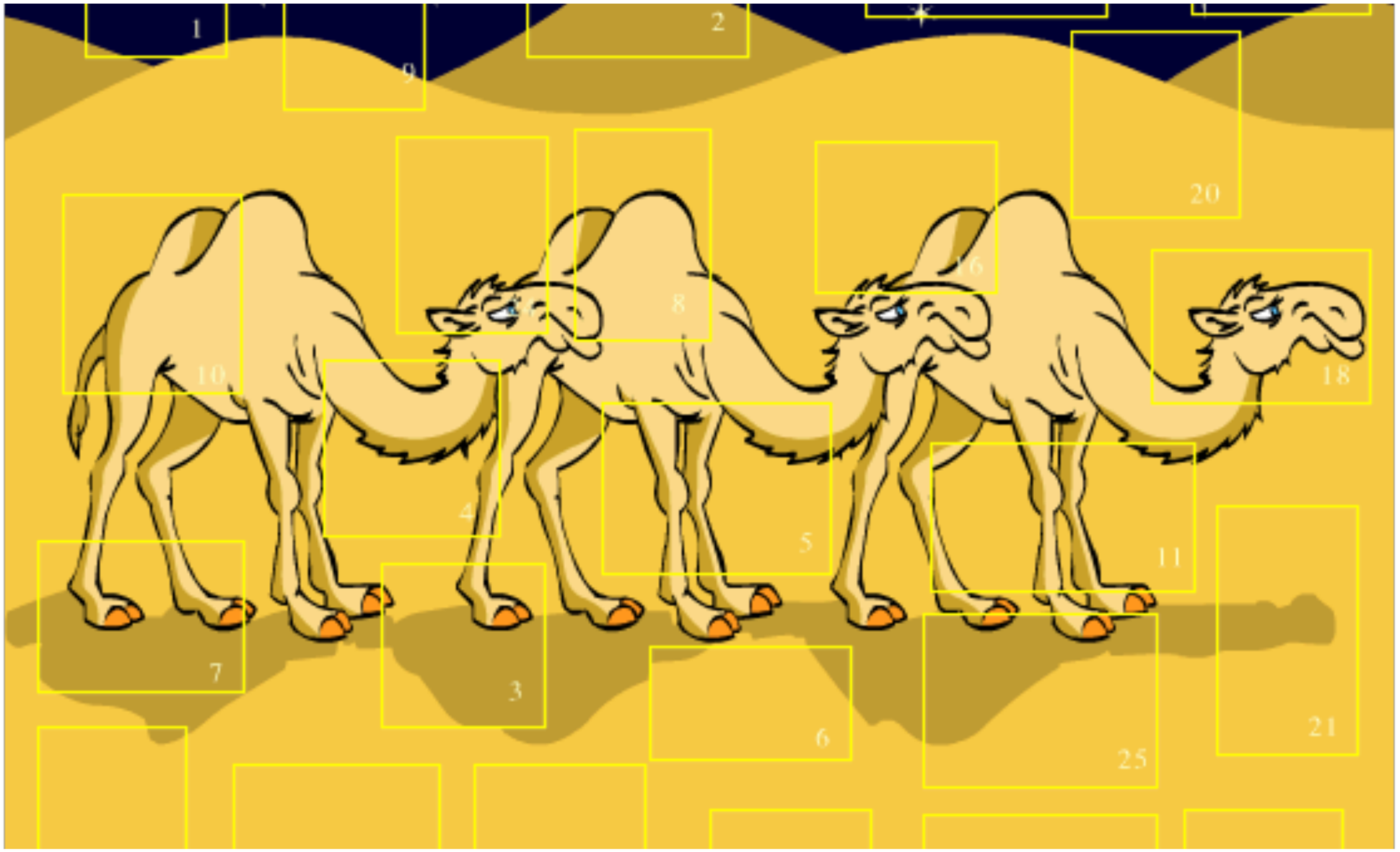

```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
            p(
                "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
                {},
                json => {
                    longUrl => $_->attr('href')
                },
            )->json("/id") .
            ": " . $_->text . "\n"
    })->join
```

ojo


```
use ojo;

print g("http://perlweekly.com/archive/333.html")
    ->dom(".entry-title a")->map( sub {
            p(
                "https://www.googleapis.com/urlshortener/v1/url?key=$ENV{API_KEY}",
                {},
                json => {
                    longUrl => $_->attr('href')
                },
            )->json("/id") .
            ": " . $_->text . "\n"
    })->join
```

ojo


```
https://goo.gl/wS2jSu: Perl Advent Calendar
https://goo.gl/B7HqqD: Perl 6 Advent Calendar
https://goo.gl/xGpyAy: Mojolicious Advent Calendar
https://goo.gl/f9LBKS: Advent Planet 2017
https://goo.gl/8aigk1: London Perl Workshop 2017 - Talks Review
https://goo.gl/j2FzgA: Perl 5 Porters Mailing List Summary: November 21st - December 5th
https://goo.gl/ptZRF3: Why I wrote Keyword::DEVELOPMENT
https://goo.gl/ptZRF3: And here we go!
https://goo.gl/GxETF8: 2017.49 Mischieventing
https://goo.gl/8q2g21: Nominate Perl heroes for the 2017 White Camel Awards
https://goo.gl/VCCoHc: Call for volunteers - TPC::NA 2018 (formerly knows as YAPC::NA)
https://goo.gl/9BRvSA: Booking.com Sponsors the P5CMF
https://goo.gl/XmJ1wT: NICEPERL's lists
https://goo.gl/zPKg2u: Open file to read and write in Perl, oh and lock it too
https://goo.gl/RChQDF: Patreon for Gabor
https://goo.gl/wn3c2Q: Rapid Growth in Perl - developers wanted for an expanding central London company
https://goo.gl/mVZGDQ: Be a trailblazer in a time of growth - Full Stack Architect required for a rapidly expanding London company
https://goo.gl/4UZctX: London's Trendiest Office Space - Lead and Mid-Level Perl Developer roles
```



For Elves, Shorter is Better

MooseX::AttributeShortcuts

Attribute Shorten


```
has lead_reindeer => (
  is      => 'ro',
  isa     => class_type('SantasReindeer'),
  lazy    => 1,
  builder => '_build_lead_reindeer',
);

has second_reindeer => (
  is      => 'ro',
  isa     => class_type('SantasReindeer'),
  lazy    => 1,
  builder => '_build_second_reindeer',
);

has third_reindeer => (
  is      => 'ro',
  isa     => class_type('SantasReindeer'),
  lazy    => 1,
  builder => '_build_third_reindeer',
);

...;  # and on and on for another six reindeer!
```

Attribute Shorten


```
has lead_reindeer => (
  is      => 'ro',
  isa     => class_type('SantasReindeer'),
  lazy    => 1,
  builder => '_build_lead_reindeer',
);

has second_reindeer => (
  is      => 'ro',
  isa     => class_type('SantasReindeer'),
  lazy    => 1,
  builder => '_build_second_reindeer',
);

has third_reindeer => (
  is      => 'ro',
  isa     => class_type('SantasReindeer'),
  lazy    => 1,
  builder => '_build_third_reindeer',
);

...;  # and on and on for another six reindeer!
```

Attribute Shorten


```
use MooseX::AttributeShortcuts;

has [qw(
  lead_reindeer second_reindeer third_reindeer forth_reindeer
  fifth_reindeer sixth_reindeer seveth_reindeer eighth_reindeer
)] => (
  is             => 'lazy',
  isa_instance_of => 'SantasReindeer',
);
```

Attribute Shorten


```
package NorthPole::Types;

use MooseX::Types -declare => [
    qw(
        SantasReindeerWithAGlowingNose
    )
];

subtype(
    SantasReindeerWithAGlowingNose,
    as class_type('SantasReindeer'),
    where {},
    inline_as {
        $_[0]->parent()->_inline_check( $_[1] ) .
            " && $_[1]->glowing_nose";
    }
);
```

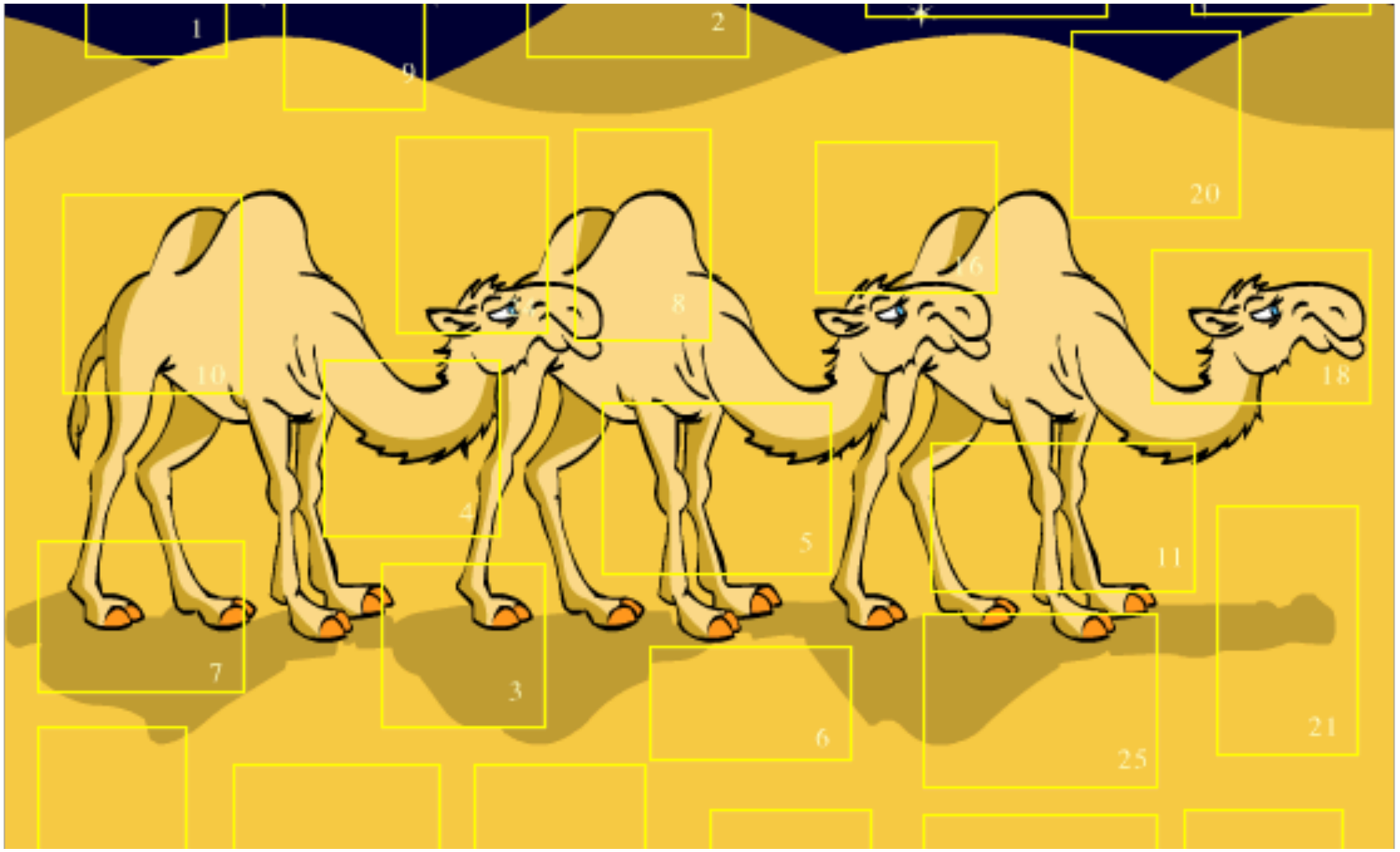
```
use NorthPole::Types qw( SantasReindeerWithAGlowingNose );

has lead_reindeer => (
  is  => 'lazy',
  isa => SantasReindeerWithAGlowingNose,
);
```

Attribute Shorten


```
has lead_reindeer => (
  is  => 'lazy',
  isa => sub {
     die unless blessed && $_->isa('SantasReindeer') && $_->glowing_nose;
  },
);
```

Attribute Shorten


```
has lead_reindeer => (
  is              => 'lazy',
  isa_instance_of => 'SantasReindeer',
  constraint      => sub { $_->glowing_nose },
);
```



Making a constructor argument list, checking it twice.

MooseX::StrictConstructor

Strict Constructor


```
# create the test sleigh
my $sleigh = SantasSleigh->new(
    lead_raindeer => Robbie->new,
);

say ref $sleigh->lead_reindeer;
```

Strict Constructor

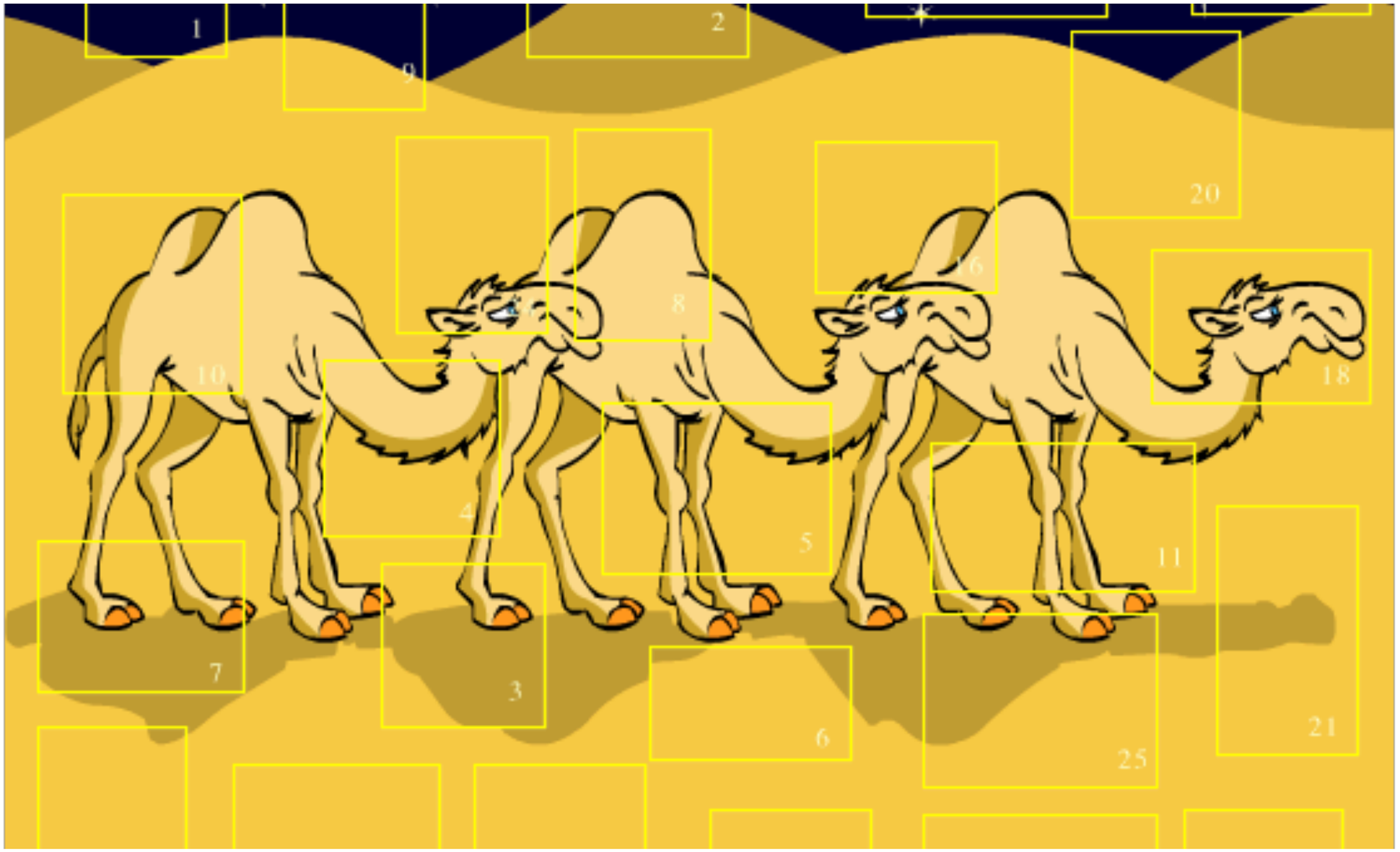

```
# create the test sleigh
my $sleigh = SantasSleigh->new(
    lead_raindeer => Robbie->new,
);

say ref $sleigh->lead_reindeer;
```

Strict Constructor


```
# create the      sleigh
my $sleigh = SantasSleigh->new(
    lead_raindeer => Robbie->new,
);

say ref $sleigh->lead_reindeer;
```

Strict Constructor


```
package SantasSleigh;
use Moose;
use MooseX::StrictConstructor;
```



Project Multipli-sleigh-ion

MooseX::ClassAttribute

Class Attributes

- An attribute shared between all instances of a class!

```
class_has carrots_on_sleigh => {
    is      => 'rw',
    isa     => 'Int',
    default => 100,
};
```

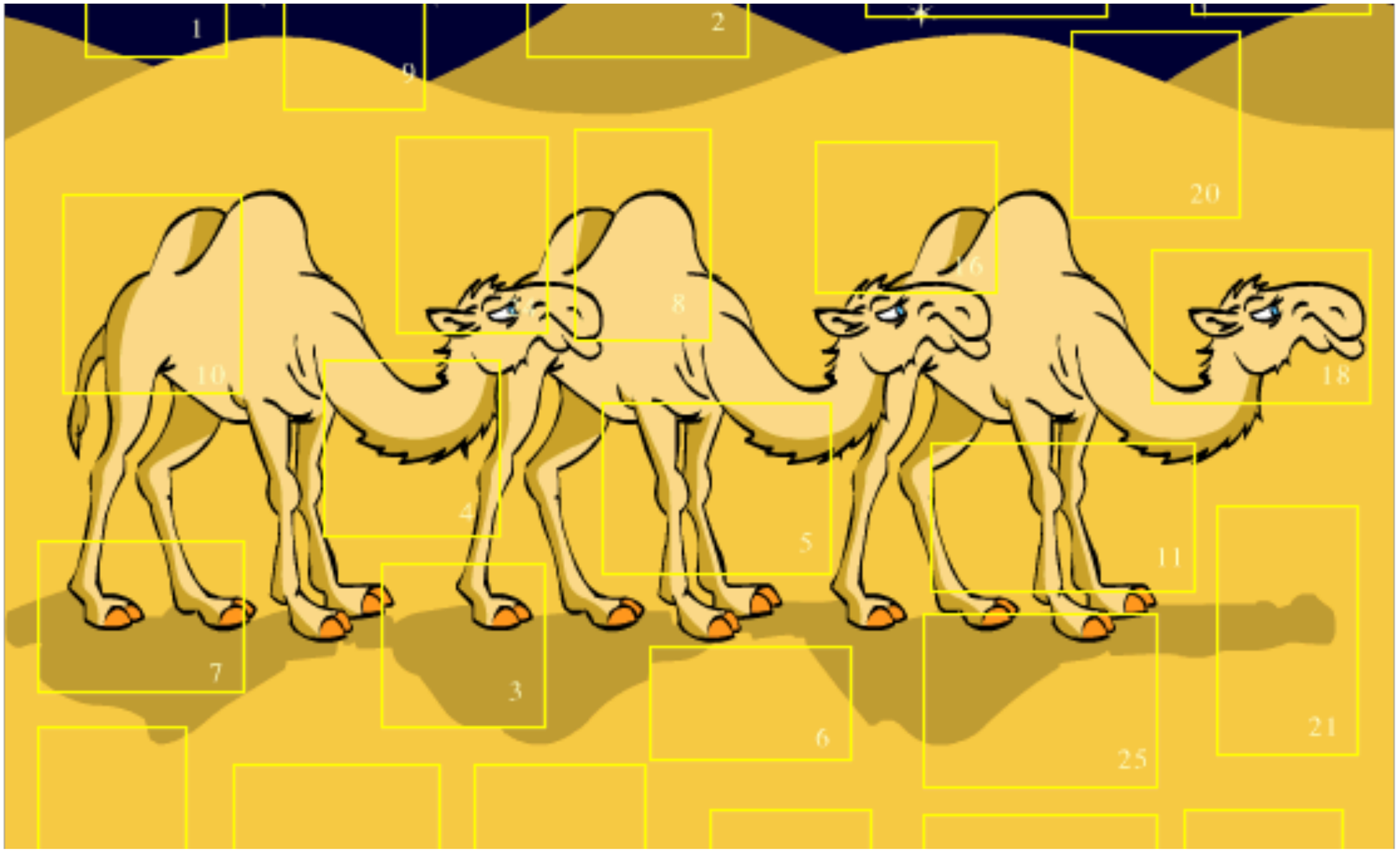
```
sub eat_food ($self) {
    $self->carrots_on_sleigh( $self->carrots_on_sleigh - 1 );
    ...
};
```

Class Attributes

- An attribute shared between all instances of a class!

```
class_has carrots_on_sleigh => {
    is      => 'rw',
    isa     => 'Int',
    default => 100,
    traits  => ['Counter'],
    handles => {
       _carrot_eaten => 'dec',
    },
};
```

```
sub eat_food ($self) {
    $self->_carrot_eaten;
    ...
};
```



Maybe...not

MooseX::LazyRequire,
MooseX::UndefTolerant::Attribute

Maybe...not

- Maybe is a code smell

```
has gps_unit => (
    is  => 'ro',
    isa => Maybe[class_type('GPSUnit')],
);

say $basic_sleigh->gps_unit->name
    if defined $basic_sleigh->gps_unit;
```

Maybe...not

- Maybe is a code smell

```
has gps_unit => (
    is        => 'ro',
    isa       => class_type('GPSUnit'),
    predicate => 'has_gps_unit',
);

say $basic_sleigh->gps_unit->name
    if $basic_sleigh->has_gps_unit;
```

Maybe…not

- Maybe is a code smell

```
has gps_unit => (
    is        => 'ro',
    isa       => class_type('GPSUnit'),
    predicate => 'has_gps_unit',
);

say $basic_sleigh->gps_unit->name
    if $basic_sleigh->has_gps_unit;
```

```
my $gps = $basic_sleigh->gps_unit;    // undef
```

Maybe...not

- Maybe is a code smell

```
use MooseX::LazyRequire;

has gps_unit => (
    is           => 'ro',
    isa          => class_type('GPSUnit'),
    predicate    => 'has_gps_unit',
    lazy_require => 1,
);

say $basic_sleigh->gps_unit->name
    if $basic_sleigh->has_gps_unit;
```

```
my $gps = $basic_sleigh->gps_unit;    // BOOOOOOOOM!
```

Maybe…not

- Maybe is a code smell

```
sub sleigh ($gps = undef) {
    return Sleigh->new(
        name => 'Mk III',
        gps  => $gps,
    );
}
```

```
my $super_sleigh = sleigh($tomtom);   // okay
my $basic_sleigh = sleigh();          // BOOM!
```

`

- Maybe is a code smell

```
sub sleigh ($gps = undef) {
    return Sleigh->new(
        name => 'Mk III',
        ($gps ? ( gps_unit => $gps ) : ()),
    );
}
```

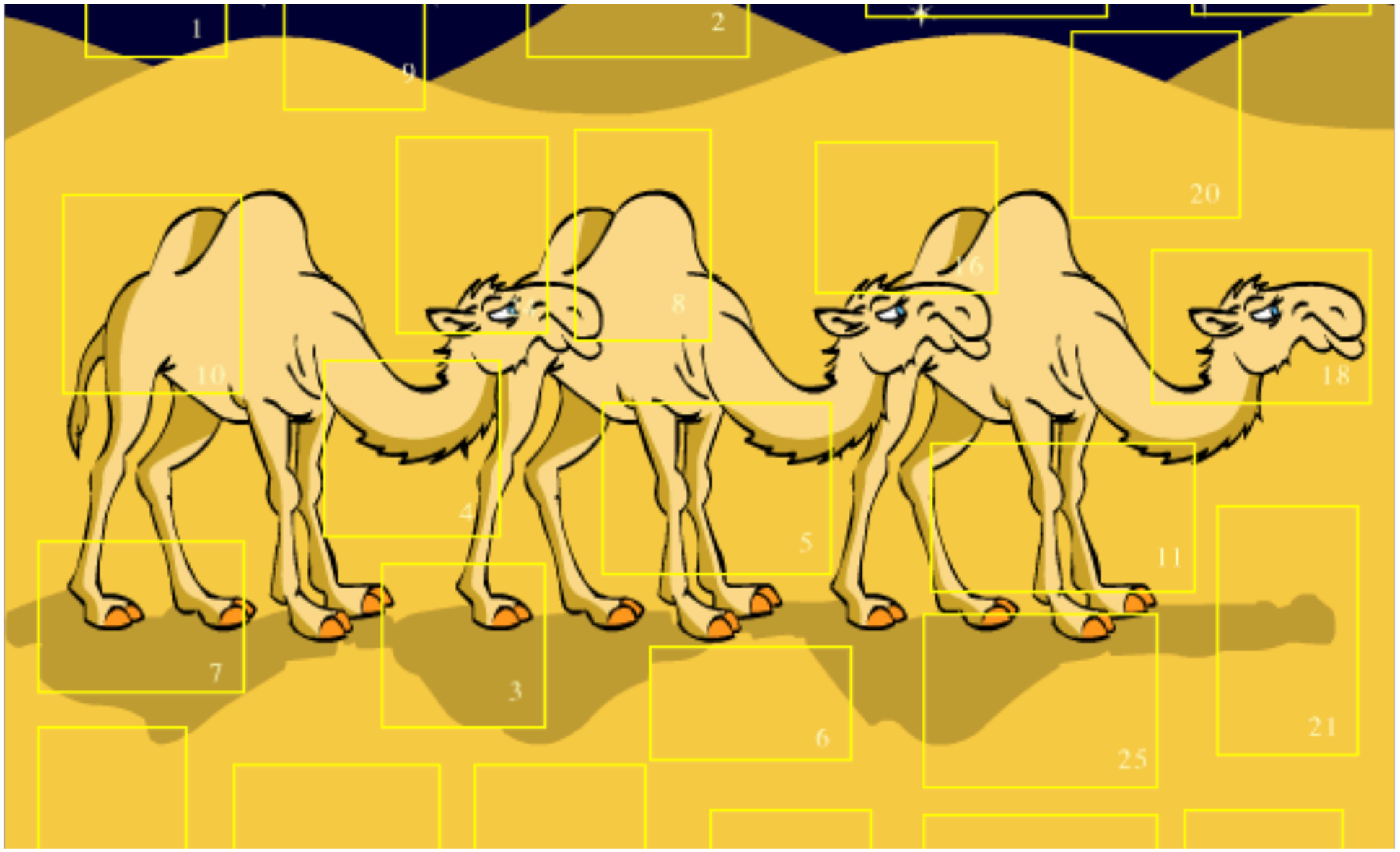
```
my $super_sleigh = sleigh($tomtom);   // okay
my $basic_sleigh = sleigh();          // okay
```

Maybe...not

- Maybe is a code smell

```
use MooseX::LazyRequire;
use MooseX::UndefTolerant::Attribute;

has gps_unit => (
    traits        => [ UndefTolerant ],
    is            => 'ro',
    predicate     => 1,
    lazy_required => 1,
);
```

```
my $super_sleigh = sleigh($tomtom);   // okay
my $basic_sleigh = sleigh();          // okay
```



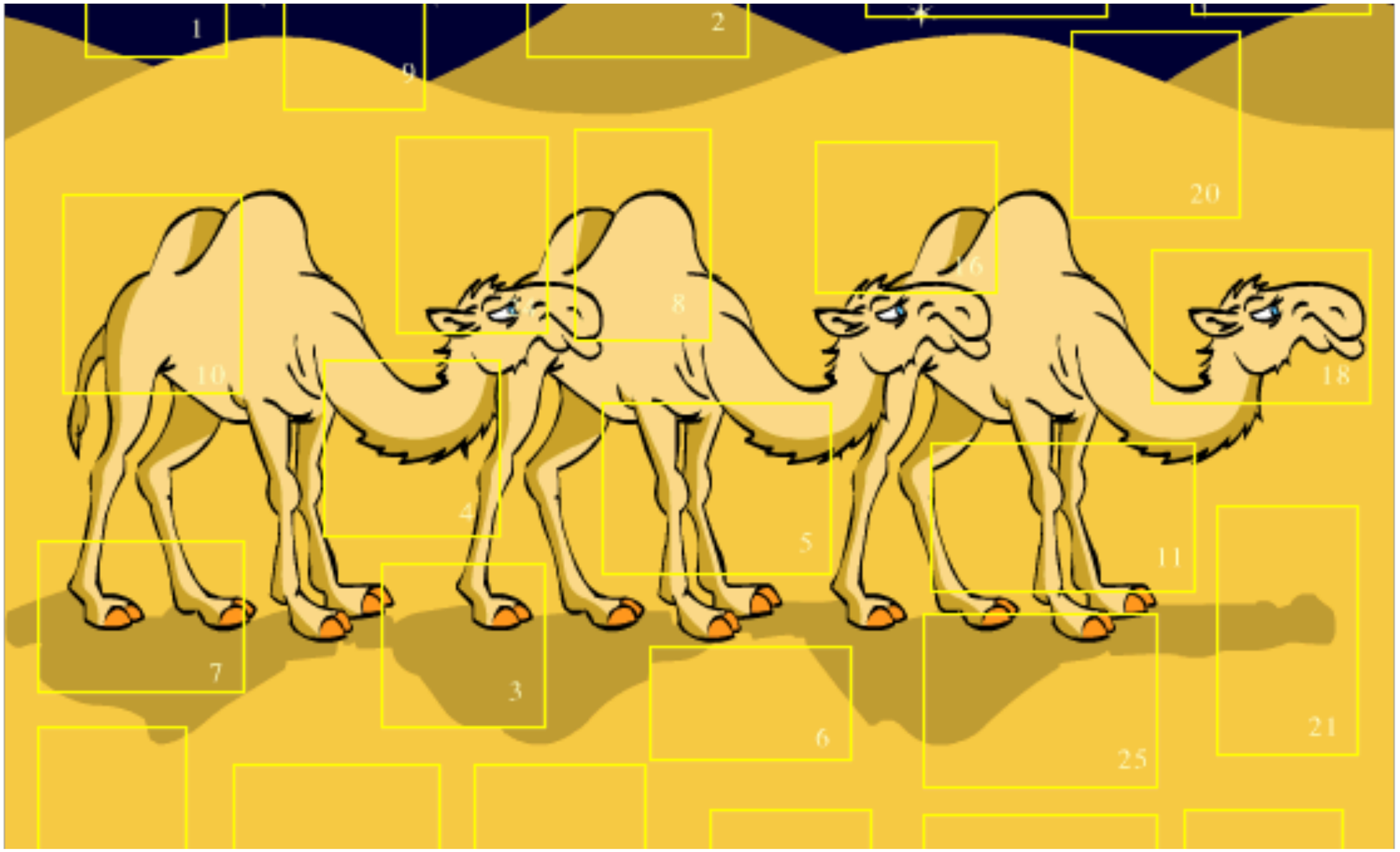
Reindeer

Reindeer

Reindeer


```
use Reindeer;
```

- Moose, but with “more antlers”
- Opinionated version of Moose, i.e. Moose with most of the add-ons we just discussed loaded for you, plus much more

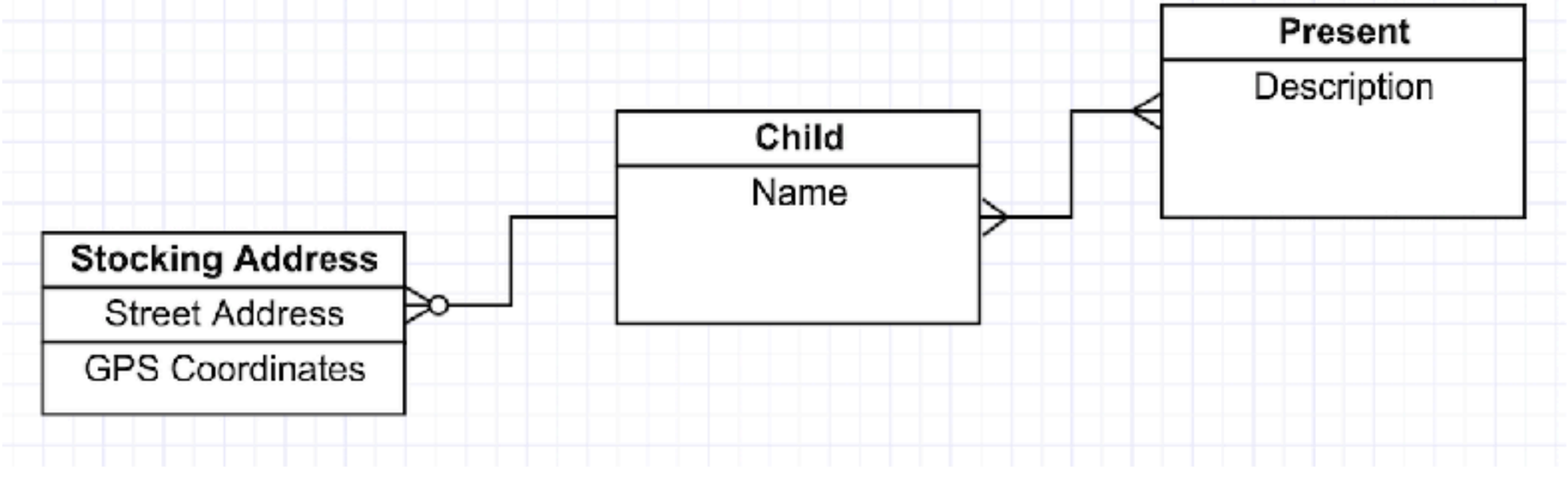


Quickly Building DBIx::Class Schemas

dbicdump, Fixtures, Reply

Quick Schemas

- Turning a database into a DBIx::Schema



Quick Schemas

- Turning a database into a DBIx::Schema

```
bash$ dbicdump -o dump_directory=./lib \
   Prototype::Schema 'dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'
```

Quick Schemas

- Turning a database into a DBIx::Schema

```
bash$ dbicdump -o dump_directory=./lib \
   Prototype::Schema 'dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'
```

```
Dumping manual schema for Prototype::Schema to directory ./lib ...
   Schema dump completed.
```

Quick Schemas

- Turning a database into a DBIx::Schema

```
bash$ dbicdump -o dump_directory=./lib \
   Prototype::Schema 'dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'
```

```
 Dumping manual schema for Prototype::Schema to directory ./lib ...
    Schema dump completed.
```

```
bash$ find .
.
./lib
./lib/Prototype
./lib/Prototype/Schema
./lib/Prototype/Schema/Result
./lib/Prototype/Schema/Result/Child.pm
./lib/Prototype/Schema/Result/ChildPresent.pm
./lib/Prototype/Schema/Result/Present.pm
./lib/Prototype/Schema/Result/StockingAddress.pm
./lib/Prototype/Schema.pm
```

Quick Schemas

- Populate the database (don't sweat the details)

```
fixtures/StockingAddress.json:
    [
        {
            "stocking_address_id": "db1e1ce1-bc05-4931-b79f-4356ea6270ff",
            "street_address": "671 Lincoln Ave. Winnetka, Illinois",
            "lat": -87.7358245,
            "lon": 42.109756
        }
    ]

    fixtures/Present.json:
    [
        {
            "present_id": "a9c15d33-b157-4513-9638-926d7793fdb1",
            "description": "BB Gun"
        },
        {c
            "present_id": "d321221e-6f1e-41ec-82b2-bda7d4783569",
            "description": "Micro Machines"
        },
        {
            "present_id": "108bebb9-871b-45a6-b4e0-36fc720b2165",
            "description": "Blowtorch"
        }
    ]

    fixtures/Child.json:
    [
        {
            "child_id": "f69eaf6c-bf77-4b29-9eca-78cda6fd2db7",
            "name": "Kevin McCallister",
            "stocking_address_id": "db1e1ce1-bc05-4931-b79f-4356ea6270ff",
            "child_presents": [
                { "present_id": "a9c15d33-b157-4513-9638-926d7793fdb1" },
                { "present_id": "d321221e-6f1e-41ec-82b2-bda7d4783569" },
                { "present_id": "108bebb9-871b-45a6-b4e0-36fc720b2165" }
            ]
        }
    ]
```

```
#!/usr/bin/perl

use v5.22;
use warnings;
use lib qw(lib);

use JSON::PP qw( decode_json );
use Path::Tiny qw( path );
use Prototype::Schema;

# connect to the database
my $schema = Prototype::Schema->connect(
    'dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432',
);

$schema->txn_do(sub {
    # run with constraints disabled until the end of a transaction so we
    # don't have to worry about the order in which we're inserting
    # fixtures into the database
    $schema->storage->dbh->do('SET CONSTRAINTS ALL DEFERRED');

    # for each json file
    for my $file ( grep { /[.]json$/ } path('fixtures')->children ) {
        # decode the json file
        my $ds = decode_json( path( $file )->slurp );

        # turn fixtures/Foo.json to Foo
        my $name = path( $file )->basename('.json');

        # get the result set
        my $rs = $schema->resultset($name);

        # and insert each json object into the database
        for my $row (@{ $ds }) {
            $rs->create( $row );
        }
    }
});
```

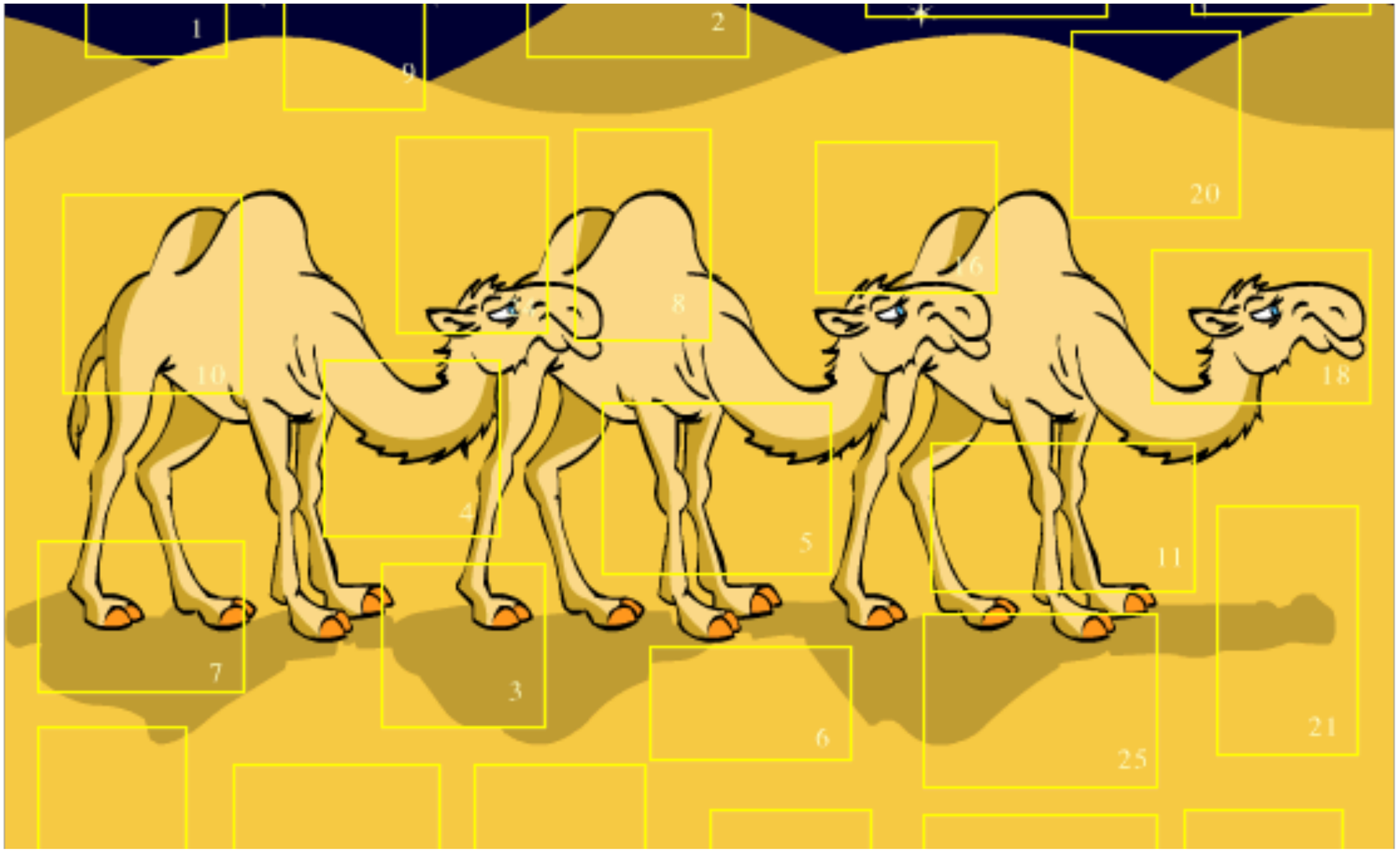
Quick Schemas

- We can use “Reply” the perl REPL with the DB

```
bash$ reply -Ilib -MPrototype::Schema
```


```
bash$ reply -Ilib -MPrototype::Schema
0> my $schema = Prototype::Schema->connect('dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'); 1
$res[0] = 1

1> my $child_rs = $schema->resultset('Child'); 1
$res[1] = 1

2> my $kevin = $child_rs->find("f69eaf6c-bf77-4b29-9eca-78cda6fd2db7"); 1
$res[2] = 1

3> $kevin->name;
$res[3] = 'Kevin McCallister'

4> my $p = $kevin->presents; 1
$res[4] = 1

5> $p->next->description;
$res[5] = 'BB Gun'

6> $p->next->description;
$res[6] = 'Micro Machines'

7> $p->next->description;
$res[7] = 'Blowtorch'

8> $kevin->name('Kevin McCallister (aka Macaulay Culkin)');
$res[8] = 'Kevin McCallister (aka Macaulay Culkin)'

9> $kevin->update; 1
$res[9] = 1

10> my $children = $child_rs->search_rs( name => 'Kevin McCallister (aka Macaulay Culkin)' ); 1
$res[10] = 1

11> $children->first->child_id
$res[11] = 'f69eaf6c-bf77-4b29-9eca-78cda6fd2db7'

12> $children->first->stocking_address->street_address
$res[12] = '671 Lincoln Ave. Winnetka, Illinois'

14>
```


```
bash$ reply -Ilib -MPrototype::Schema
0> my $schema = Prototype::Schema->connect('dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'); 1
$res[0] = 1

1> my $child_rs = $schema->resultset('Child'); 1
$res[1] = 1

2> my $kevin = $child_rs->find("f69eaf6c-bf77-4b29-9eca-78cda6fd2db7"); 1
$res[2] = 1

3> $kevin->name;
$res[3] = 'Kevin McCallister'

4> my $p = $kevin->presents; 1
$res[4] = 1

5> $p->next->description;
$res[5] = 'BB Gun'

6> $p->next->description;
$res[6] = 'Micro Machines'

7> $p->next->description;
$res[7] = 'Blowtorch'

8> $kevin->name('Kevin McCallister (aka Macaulay Culkin)');
$res[8] = 'Kevin McCallister (aka Macaulay Culkin)'

9> $kevin->update; 1
$res[9] = 1

10> my $children = $child_rs->search_rs( name => 'Kevin McCallister (aka Macaulay Culkin)' ); 1
$res[10] = 1

11> $children->first->child_id
$res[11] = 'f69eaf6c-bf77-4b29-9eca-78cda6fd2db7'

12> $children->first->stocking_address->street_address
$res[12] = '671 Lincoln Ave. Winnetka, Illinois'

14>
```



Custom Relationships with DBIx::Class

might_have, DBIC_TRACE

Relationships


```
CREATE TABLE reason (
    child_id UUID NOT NULL,
    tag TEXT NOT NULL,
    notes TEXT
);

ALTER TABLE ONLY reason
    ADD CONSTRAINT reason_pkey
    PRIMARY KEY (child_id, tag);

ALTER TABLE ONLY reason
    ADD CONSTRAINT child_idfkey
    FOREIGN KEY (child_id)
    REFERENCES  child(child_id)
    DEFERRABLE;
```

Relationships


```
CREATE TABLE reason (
    child_id UUID NOT NULL,
    tag TEXT NOT NULL,
    notes TEXT
);

ALTER TABLE ONLY reason
    ADD CONSTRAINT reason_pkey
    PRIMARY KEY (child_id, tag);

ALTER TABLE ONLY reason
    ADD CONSTRAINT child_idfkey
    FOREIGN KEY (child_id)
    REFERENCES  child(child_id)
    DEFERRABLE;
```

Relationships


```
CREATE TABLE reason (
    child_id UUID NOT NULL,
    tag TEXT NOT NULL,
    notes TEXT
);

ALTER TABLE ONLY reason
    ADD CONSTRAINT reason_pkey
    PRIMARY KEY (child_id, tag);

ALTER TABLE ONLY reason
    ADD CONSTRAINT child_idfkey
    FOREIGN KEY (child_id)
    REFERENCES  child(child_id)
    DEFERRABLE;
```

Relationships


```
bash$ dbicdump -o dump_directory=./lib \
   Prototype::Schema 'dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'
```

```
=head2 reasons

Type: has_many

Related object: L<Prototype::Schema::Result::Reason>

=cut

__PACKAGE__->has_many(
  "reasons",
  "Prototype::Schema::Result::Reason",
  { "foreign.child_id" => "self.child_id" },
  { cascade_copy => 0, cascade_delete => 0 },
);
```

Relationships

Solving Problems with Dogs

```
sub delivery_problem_with_dog {
    my $self = shift;
    my $rs = $self->search_related('Reason', { tag => 'dog' });
    return $rs->first;  # will be undef if there's no matching row
}
```

Relationships

Solving Problems with Dogs

```
sub delivery_problem_with_dog {
    my $self = shift;
    my $rs = $self->search_related('Reason', { tag => 'dog' });
    return $rs->first;  # will be undef if there's no matching row
}
```

```
if ($child->delivery_problem_with_dog )) {
    break_out_the_bacon();
}
```

Relationships


```
bash$ DBIC_TRACE=1 reply -Ilib -MPrototype::Schema
0> my $schema = Prototype::Schema-
>connect('dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'); 1;
$res[0] = 1

1> my $child = $schema->resultset('Child')->find("f69eaf6c-
bf77-4b29-9eca-78cda6fd2db7"); 1
SELECT me.child_id, me.stocking_address_id, me.name FROM child me
WHERE ( me.child_id = ? ):
'f69eaf6c-bf77-4b29-9eca-78cda6fd2db7'
$res[1] = 1

2> $child->delivery_problem_with_dog && "yes"
SELECT me.child_id, me.tag, me.notes FROM reason me
WHERE ( ( me.child_id = ? AND tag = ? ) ):
'f69eaf6c-bf77-4b29-9eca-78cda6fd2db7', 'dog'
$res[2] = ‘yes'
```

Relationships

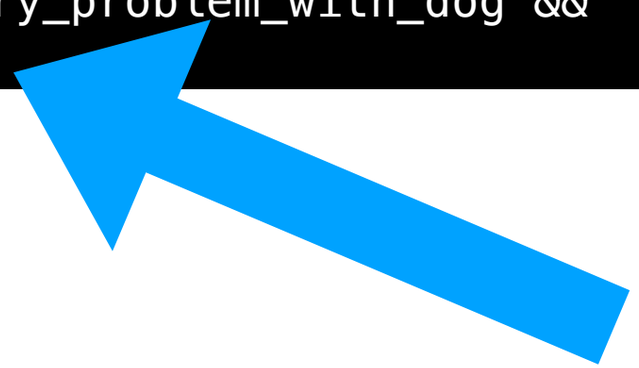

```
__PACKAGE__->might_have(
  "delivery_problem_with_dog",
  "Prototype::Schema::Result::Reason",
  sub {
    my $args = shift;

    return {
      "$args->{foreign_alias}.child_id"
        => { -ident => "$args->{self_alias}.child_id" },
      "$args->{foreign_alias}.tag" => { '=', 'dog' },
    }
  },
);
```

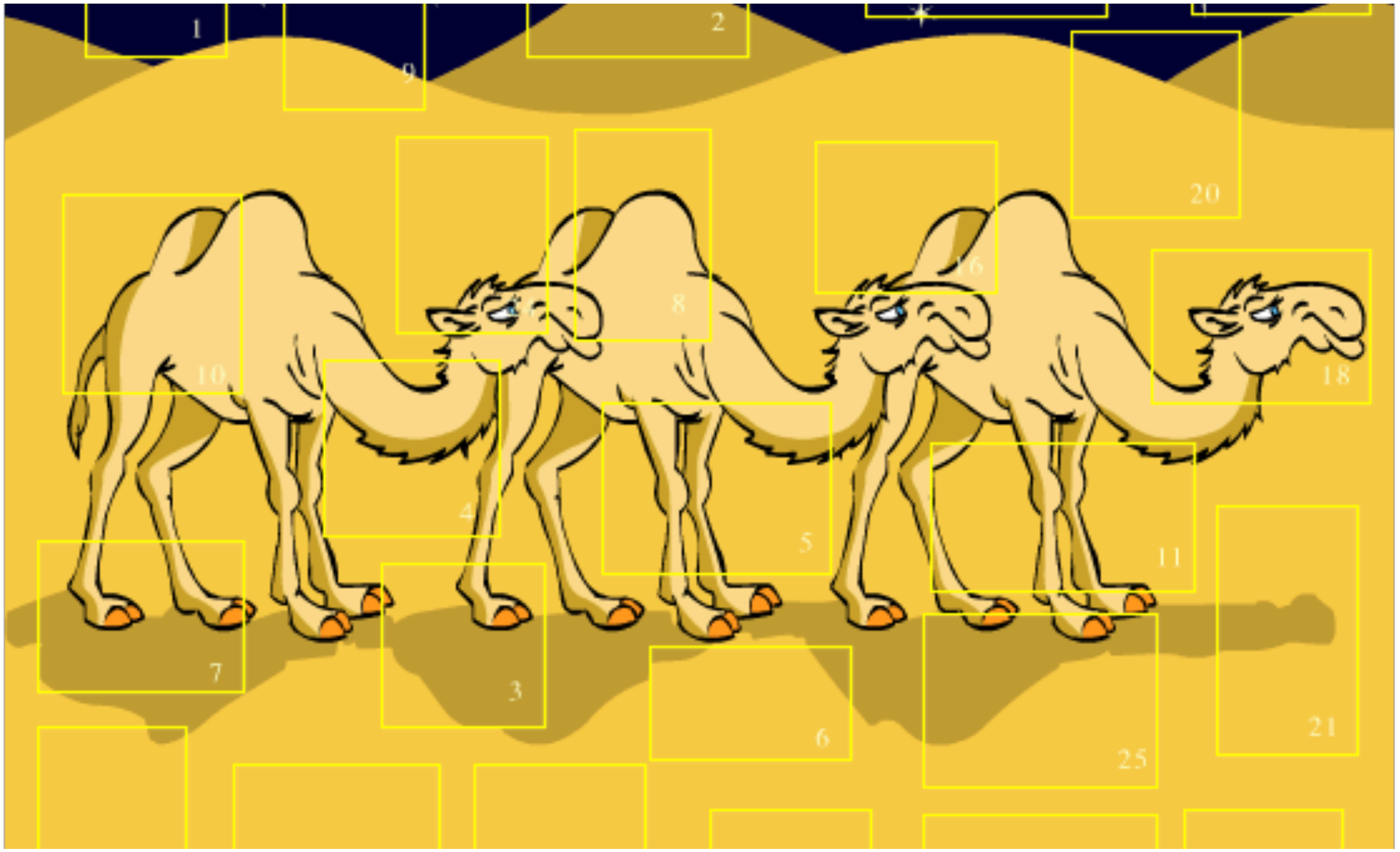
Relationships


```
bash$ DBIC_TRACE=1 reply -Ilib -MPrototype::Schema
0> my $schema = Prototype::Schema-
>connect('dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'); 1;
$res[0] = 1

1> my $child = $schema->resultset(‘Child')->find("f69eaf6c-
bf77-4b29-9eca-78cda6fd2db7",
  { prefetch => "delivery_problem_with_dog" }); 1
SELECT me.child_id, me.stocking_address_id, me.name,
      delivery_problem_with_dog.child_id, delivery_problem_with_dog.tag,
      delivery_problem_with_dog.notes FROM child me
LEFT JOIN reason delivery_problem_with_dog ON (
 delivery_problem_with_dog.child_id = me.child_id
 AND delivery_problem_with_dog.tag = ?
) WHERE ( me.child_id = ? ):
    'dog', 'f69eaf6c-bf77-4b29-9eca-78cda6fd2db7'
$res[1] = 1

2> $child->delivery_problem_with_dog && "yes"
$res[2] = 'yes'
```

Relationships


```
bash$ DBIC_TRACE=1 reply -Ilib -MPrototype::Schema
0> my $schema = Prototype::Schema-
>connect('dbi:Pg:dbname=prototype;host=127.0.0.1;port=5432'); 1;
$res[0] = 1

1> my $child = $schema->resultset('Child')->find("f69eaf6c-
bf77-4b29-9eca-78cda6fd2db7",
  { prefetch => "delivery_problem_with_dog" }); 1
SELECT me.child_id, me.stocking_address_id, me.name,
      delivery_problem_with_dog.child_id, delivery_problem_with_dog.tag,
      delivery_problem_with_dog.notes FROM child me
LEFT JOIN reason delivery_problem_with_dog ON (
 delivery_problem_with_dog.child_id = me.child_id
 AND delivery_problem_with_dog.tag = ?
) WHERE ( me.child_id = ? ):
    'dog', 'f69eaf6c-bf77-4b29-9eca-78cda6fd2db7'
$res[1] = 1

2> $child->delivery_problem_with_dog && "yes"
$res[2] = 'yes'
```





Speedy Validation

Param::ValidationCompiler

Validation

- Params::ValidationCompiler
- Faster than most other pure Perl parameter checkers
- Allows us to re-use Moose types, Type::Tiny types, etc

Validation


```
use Params::ValidationCompiler qw( validation_for );
use MooseX::Types::Common::String qw( NonEmptySimpleStr );
use MooseX::Types::Common::Numeric qw( PositiveInt );

my $validator = validation_for(
    params => {
        present_name => { type => NonEmptySimpleStr },
        qty          => { type => PositiveInt, default => 1 },
    },
);

sub load_sled {
    my $self = shift;
    my %p    = $validator->(@_);

    my $present_name = $p{present_name};
    my $qty          = $p{qty};
```

Validation


```
use Params::ValidationCompiler qw( validation_for );
use MooseX::Types::Common::String qw( NonEmptySimpleStr );
use MooseX::Types::Common::Numeric qw( PositiveInt );

my $validator = validation_for(
    params => {
        present_name => { type => NonEmptySimpleStr },
        qty          => { type => PositiveInt, default => 1 },
    },
);

sub load_sled {
    my $self = shift;
    my %p    = $validator->(@_);

    my $present_name = $p{present_name};
    my $qty          = $p{qty};
```

Validation

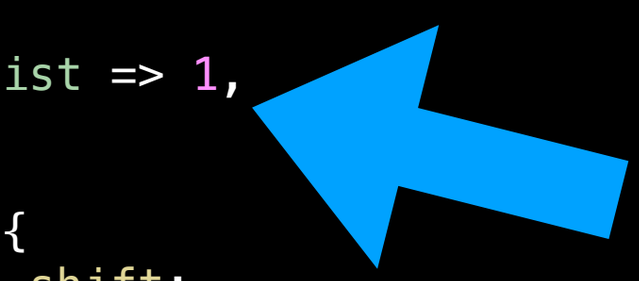

```
use Params::ValidationCompiler qw( validation_for );
use MooseX::Types::Common::String qw( NonEmptySimpleStr );
use MooseX::Types::Common::Numeric qw( PositiveInt );

my $validator = validation_for(
    params => {
        present_name => { type => NonEmptySimpleStr },
        qty          => { type => PositiveInt, default => 1 },
    },
);

sub load_sled {
    my $self = shift;
    my %p    = $validator->(@_);

    my $present_name = $p{present_name};
    my $qty          = $p{qty};
```

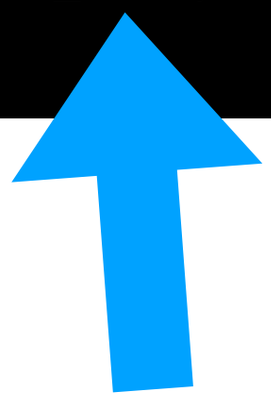
Validation

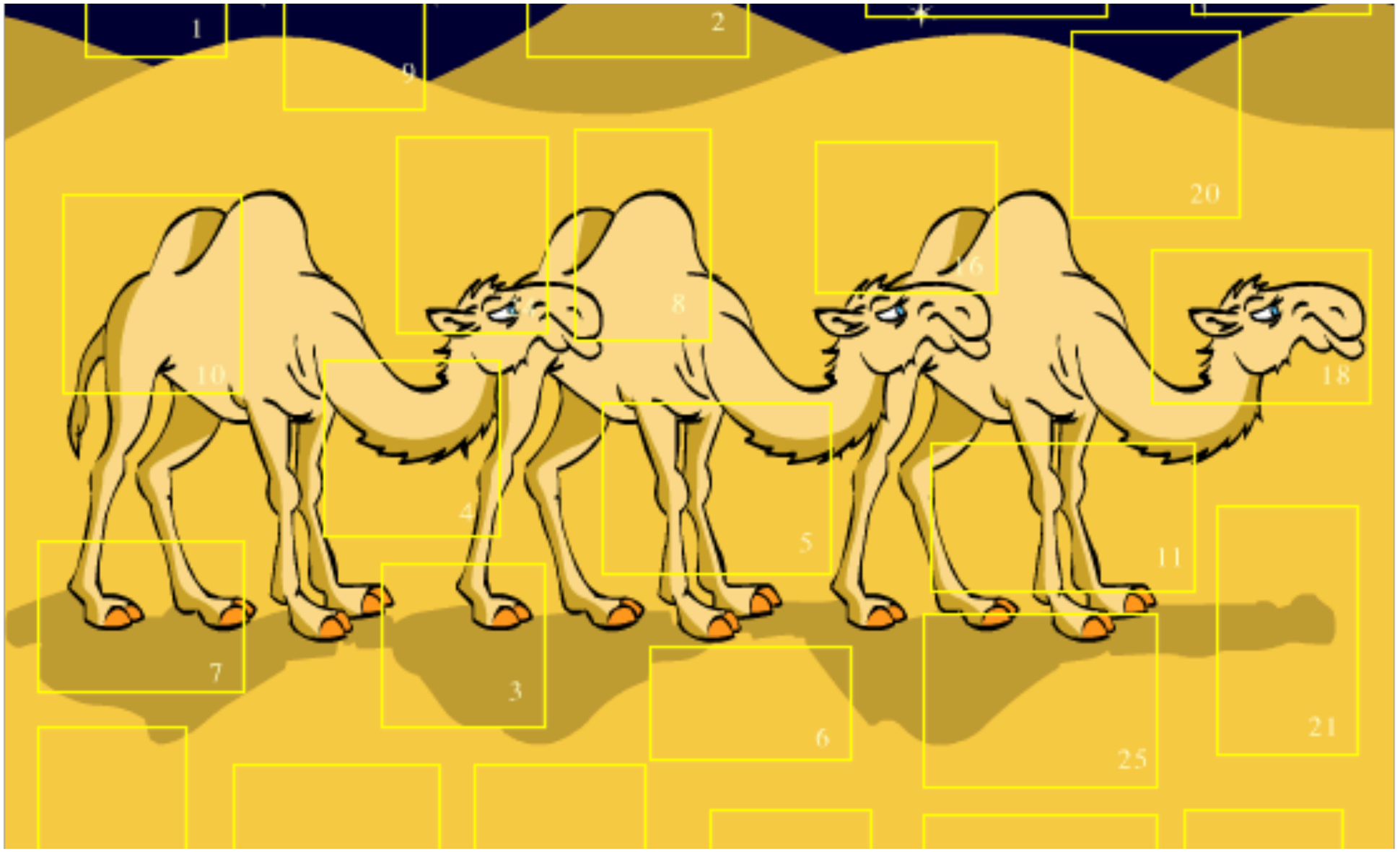

```
use Params::ValidationCompiler qw( validation_for );
use MooseX::Types::Common::String qw( NonEmptySimpleStr );
use MooseX::Types::Common::Numeric qw( PositiveInt );

my $validator = validation_for(
    params => {
        present_name => { type => NonEmptySimpleStr },
        qty          => { type => PositiveInt, default => 1 },
    },
);

sub load_
    my $se    = shift;
    my %p     = $validator->(@_);

    my $present_name = $p{present_name};
    my $qty          = $p{qty};
```

Validation


```
use Params::ValidationCompiler qw( validation_for );
use MooseX::Types::Common::String qw( NonEmptySimpleStr );
use MooseX::Types::Common::Numeric qw( PositiveInt );

my $validator = validation_for(
    params => [
        present_name => { type => NonEmptySimpleStr },
        qty          => { type => PositiveInt, default => 1 },
    ],
    named_to_list => 1,
);

sub load_sled {
    my $self = shift;
    my ($present_name, $qty) = $validator->(@_);
```



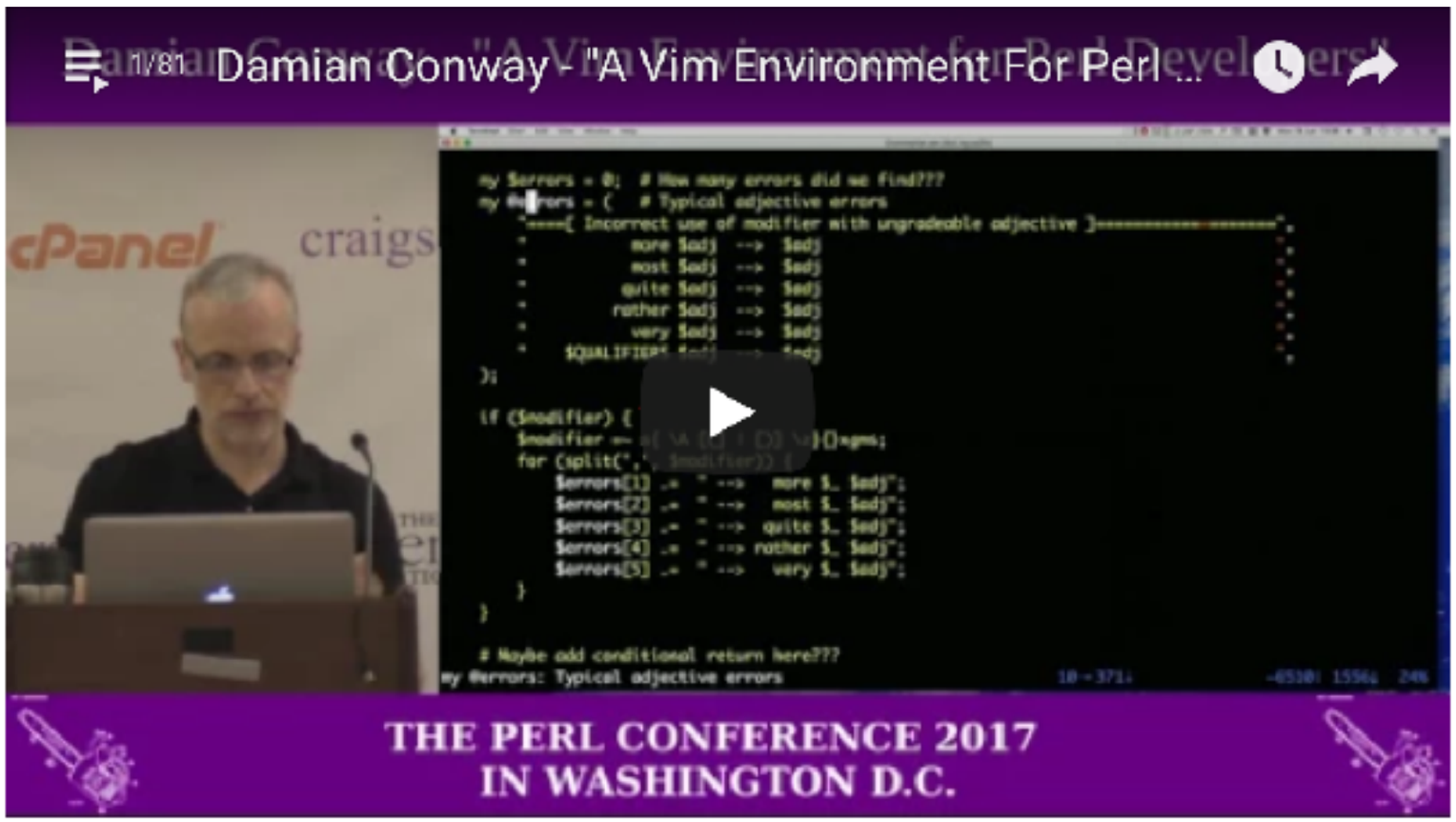




Watching The Perl Conference

TPC Videos, ojo, WWW::YouTube::Download, xargs

YouTube Scrape



- All the videos from TPC 2017 are on YouTube
- Wouldn’t it be nice to download them all?

YouTube Scrape


```
perl -Mojo -E \
     'g(shift)->dom(".pl-video-title-link")->map(sub {say $_->attr("href")})' \
     'https://www.youtube.com/playlist?list=PLA9_Hq3zhoFxdSVDA4v9Af3iutQxLI14m'
```

YouTube Scrape


```
perl -Mojo -E \
     'g(shift)->dom(".pl-video-title-link")->map(sub {say $_->attr("href")})' \
     'https://www.youtube.com/playlist?list=PLA9_Hq3zhoFxdSVDA4v9Af3iutQxLI14m'
```

- WWW::YouTube::Download provides youtube-download

```
youtube-download /watch?v=DVKfBV2xnggperl
```

YouTube Scrape


```
perl -Mojo -E \
     'g(shift)->dom(".pl-video-title-link")->map(sub {say $_->attr("href")})' \
     'https://www.youtube.com/playlist?list=PLA9_Hq3zhoFxdSVDA4v9Af3iutQxLI14m'
     | xargs -n1 youtube-download
```

YouTube Scrape


```
perl -Mojo -E \
     'g(shift)->dom(".pl-video-title-link")->map(sub {say $_->attr("href")})' \
     'https://www.youtube.com/playlist?list=PLA9_Hq3zhoFxdSVDA4v9Af3iutQxLI14m'
     | xargs -n1 youtube-download
```

```
--> Working on oka4wcsrg0c
  Downloading `oka4wcsrg0c.mp4`
  126211392/126211392 (100.00%)
  Download successful!
  --> Working on DVKfBV2xngg
  Downloading `DVKfBV2xngg.mp4`
  42843944/42843944 (100.00%)
  Download successful!
  --> Working on vQ5qWey_SO4
  Downloading `vQ5qWey_SO4.mp4`
  36677196/36677196 (100.00%)
  Download successful!
```

The End

http://perladvent.org/2017

- **Next year: http://cfp.perladvent.org/**