Perl – An Overview

Erwan Lemonnier
DYPL 2009

Today's menu

- A history of Perl
- Code examples
- Modern Perl in practice

A history of Perl

- Created by Larry Wall, released in 1987
  - Python: 1991, Ruby: 1993, Erlang: 1986
  - Java: 1995, Groovy: 2007
- Open-source
- Dynamically typed
- High level scripting language

Philosophy

"There Is More Than One Way To Do It"

Make easy things easy and
hard things possible

Philosophy

- Community driven evolution:
  - CPAN (Perl Archive)
  - Conferences, IRC, online forums
- Organic growth, no strong leadership
  - (chaotic at times)

The evolution of Perl

- A text processing language with great regexps
- Became the language of choice for CGI scripts
- CPAN allowed for fast uncontrolled expansion
- Features progressively introduced in Perl:
  - Object orientation
  - Threads
  - Gazillions of special purpose libraries
  - Meta-object frameworks
  - Functional language features

Perl today

- Perl5
- A powerful but messy language:
  - Messy syntax, no code standard
  - CPAN has modules for everything
  - Each module has its own API and style
  - Good for prototyping new language features
  - Still best for pattern matching

Perl today

- Consequence:
  - Perl is a jungle for the newcomer
  - No longer a trendy language

Perl soon

- Perl needed a rewrite
- Work on Perl6 started in 2000!
- A new language designed by the community
  - Not backward compatible with Perl5
- Close to completion as of 2009:
  - stable specifications
  - virtual machine: parrot
  - compiler: rakudo

Perl6

- A new exciting dynamic language
- Mixes elements from Perl, Python, Ruby, Haskell
- Cool features:
  - Grammars (regexps on steroids)
  - Gradual typing, roles
  - Lazy evaluation
  - Macros

Code Examples

Ex: Fibonacci

```
#!/usr/bin/perl

use Math::BigInt;

$x = Math::BigInt->new(1);
$y = Math::BigInt->new(2);

$stop = 100;

print $x;
print $y;
for ( 3 .. $stop ) {
    ( $x, $y ) = ( $y, $x+$y );
    print $y;
}
```

Ex: Sigils lovers

```
sub multihash_delete {
    my ($hash, $key, $value) = @_;
    my $i;

    return unless ref( $hash->{$key} );
    for ($i = 0; $i < @{ $hash->{$key} }; $i++) {
        if ($hash->{$key}->[$i] eq $value) {
            splice( @{$hash->{$key}}, $i, 1);
            last;
        }
    }

    delete $hash->{$key} unless @{$hash->{$key}};
}
```

Ex: Obfuscated Perl

a curses-based real-time skiing game:

```
undef $/;open(_,$0);/ \dx([\dA-F]*)/while(<_>);@&=split(//,$1);@/=@&;
$".=chr(hex(join("",splice(@&,0,2))))while(@&); eval$”;

($C,$_,@\)=(($a=$/[1]*4)*5+1, q| |x(0x20).q|\||.chr(32)x(0x10).q$*$.
chr(0x20)x(0x10).(pack("CC",124,10)), sub{s/.\|(\s*?)(\S)./\|$1 $2/},
sub{s/\|(\s*?).(\S)/ \|$1$2 /}, sub{$2.$1.$3},sub{$tt=(3*$tt+7)%$C},
sub{$1.$3.$2});
while ($_) {
    select $/, undef, $/, $C/1E3;
    (sysread(STDIN, $k, 1),s/(.)(\*)(.)/(&{$\[(ord($k)-44&2)+2]})/e)
    if (select($a=chr(1),$/,$/,0));

print 0x75736520504F5349583B2024743D6E657720504F5349583A3A5465726D696F73
3B24742D3E67657461747472282830 3D3E2A5F3D5C2423292F32293B2024742D3E
365746C666C61672824742D3E6765746C666C6167267E284543484F7C4543484F4
7C4943414E4F4E29293B2024742D3E7365746363285654494D452C31293B24742D
E736574617474722830 2C544353414E4F57293B24643D224352415348215C6E223B0A;

  ($p?(/.{70}\|$/):(/^\|/))||(&{$\[3]}<$/[0])?($p=!$p):&{$\[$p]}||die("$d");
  (&{$\[3]}<$/[1])&&(s/ \|$/\|/);
  (/\|.*\*.*\|$/)||die("$d");
}
```

case study:
a Circle object

Circle object: just a 'blessed' structure

```
package Circle;

sub new {
    my ($class,$x,$y,$r)
    my $self = {
        x => $x,
        y => $y,
        r => $r,
    };
    return bless($self,'Circle');
}

sub area {
    my $self = shift;
    return $self->{r}^2*3.1415;
}
```

Circle: using accessors to hide implementation

```
package Circle;
use accessors ('x','y','r');

sub new {
    my ($class,$x,$y,$r)
    my $self = bless({},'Circle');
    $self->x($x);
    $self->y($y);
    $self->r($r);
    return $self;
}

sub area {
    my $self = shift;
    return $self->r^2*3.1415;
}
```

Circle: validate arguments with contracts

```
package Circle;
use accessors ('x','y','r');
use Sub::Contract;

contract('new')->in(undef,\&is_float,\&is_float,\&is_float)
               ->enable;

sub new {
    my ($class,$x,$y,$r)
    my $self = bless({},'Circle');
    $self->x($x);
    $self->y($y);
    $self->r($r);
    return $self;
}

contract('area')->in(undef)->out(\&is_float)
               ->enable;

sub area {
    return $_[0]->r^2*3.1415;
}
```

Circle: use a different object syntax

```
package Circle;
use Moose;     # a meta-object framework

has 'x' => (is => 'rw', isa => 'Int');
has 'y' => (is => 'rw', isa => 'Int');
has 'r' => (is => 'rw', isa => 'Int');

sub area {
    return $_[0]->r*3.1415;
}

# creates a default constructor:
# my $c = new Circle(x => 1, y => 1, r => 3);
```

Circle

- We could go on a long time:
  - Add exception handling
  - Use other meta-object frameworks
- And still, it's just a circle.

Perl in practice

Perl's strenghts

- regexps
- Complex data structures
- Powerful introspection mechanisms
- Dynamic compilation and injection of new code (methods, classes...)
- Source filters
- Hooks to the bytecode interpretor
- Anonymous subs (closures)

CPAN

- Hard things, once solved, are made easy via CPAN modules
- CPAN = Comprehensive Perl Archive Network
  - Anyone can upload modules to CPAN
  - Code has to have unit-tests, documentation and examples
  - CPAN testers: automated test reports
  - CPAN quality: automated quality control

Hot on CPAN

- Meta Object frameworks
- Object Relational database Mappers: Jifty::DBI, Class::DBIx
- Frameworks for fast web development: Jifty, Catalyst, Rose
- Perl6 features in Perl5

Ex: Python decorators

```
use Python::Decorator;

# the 2 lines above 'sub incr' are Python-style decorators.
# they add memoizing and debugging behaviors to incr()

@memoize          # decorator without arguments
@debug("incr")    # decorator with arguments
sub incr {
    return $_[0]+1;
}
```

Ex: Meta Object Framework

```
package Point;
use Moose; # automatically turns on strict and warnings

has 'x' => (is => 'rw', isa => 'Int');
has 'y' => (is => 'rw', isa => 'Int');

sub clear {
    my $self = shift;
    $self->x(0);
    $self->y(0);
}

package Point3D;
use Moose;

extends 'Point';

has 'z' => (is => 'rw', isa => 'Int');

after 'clear' => sub {
    my $self = shift;
    $self->z(0);
};
```

Ex: Meta Object Protocol

- A meta object protocol is an API to an object system.
  - It abstracts the components of an object system (classes, object, methods, object attributes, etc.)
  - Meta object frameworks are implemented on top of them
- Class::MOP
- MOP

Your way in the jungle

- Read books
- Browse CPAN and read the sources
- Go to conferences/workshops
- Get in touch with the community

Questions?