PERL

“There's more than one way to do it”

Marco Aurélio
ma.brunoro@gmail.com

Marcos Paulo
marcosmachado235@gmail.com

SUMÁRIO

- Introdução
- Tipos de Dados
- Variáveis e Constantes
- Operadores
- Referências
- Condicionais e Loops
- Sub-rotinas
- Modularização e Orientação a Objetos
- Exceções e Concorrência
- Avaliação e Conclusão

INTRODUÇÃO

INTRODUÇÃO

- Criado por Larry Wall em 1987
- Era empregado da NASA
- Linguagem de script de propósito geral para Unix
- Facilitar processamento de relatórios
- Perl 6 é outra linguagem de programação
- "*Practical Extraction and Reporting Language*" na verdade é um *backronym*
- O slogan de Perl é o contrário de uma das frases de *Zen of Python*: *There should be one-- and preferably only one --obvious way to do it*

INTRODUÇÃO

Existem duas “regras” tiradas da documentação oficial do Perl:

- Larry por definição está sempre certo sobre como o Perl deve se comportar. Isto significa que ele tem o poder final do veto na funcionalidade do todo.
- É permitido a Larry mudar de ideia mais tarde sobre qualquer assunto, não importa se ele invocou previamente a regra 1.

INTRODUÇÃO

- Útil em tarefas diárias de administração de sistemas como geração de relatórios (basicamente criada para isso)
- Influenciada por Shell Scripting, C/C++ e awk
- Possui uma interface independente de banco de dados (DBI)
- Linguagem dominante para programação de CGI (common gate interface) e fast-CGI
- Ideal para construção de web robots
- Vasta coleção de programas disponíveis (mais de 25.000)
- Comprehensive Perl Archive Network (CPAN)

“

Perl is the duct tape of the Internet.

-Hassan Schroeder

INTRODUÇÃO

- Open source (escrita em C)
- Linguagem intermediária (como Java, mas nem tanto)
- É possível "compilar" código Perl para bytecode (de Perl) executável e para código C (inclui otimizações)
- Tipagem dinâmica
- Fortemente tipada (?)
- Apresenta outro tipo de módulo: pragma
- Pragmas são módulos que influenciam, em certos aspectos, o tempo de compilação e de execução de código

INTRODUÇÃO

Unix/Linux

✓ Included
(may not be latest)

Mac OS X

✓ Included
(may not be latest)

Windows

Strawberry Perl
ActiveState Perl

TIPOS DE DADOS

Real programmers can write assembly code
in any language.

Larry Wall

TIPOS DE DADOS

- Há basicamente dois tipos de dados em Perl:
  - Números
  - Strings
- Números são inteiros ou floats
- Strings podem ser “interpoláveis” ou não
- O equivalente de Null é *undef*

TIPOS DE DADOS

```php
$a = 10;

$b = -100;

$c = 2.34;

$d = 16.12E14;

$e = 0xffca;

$f = 0577;
```

```php
$g = "Uma string. \$a=$a \n";

$h = 'Outra string. \$a=$a \n';

print $g;
# Uma string. $a=10

print $h;
# Outra string \$a=$a \n
```

TIPOS DE DADOS

```
$a ="Essa string
tem mais de uma linha.
Eu posso escrever
o que eu quiser\n";

$g =<<'TVAR';
Esse texto
todo está
incluído na string
TVAR

$h =<<NOV;
Imagine um
texto enorme aqui \n
NOV
```

```
                                # Unicode
                                $smile = v9786;

                                $martin = v77.97.114.116.105.110;
                                # Martin
```

TIPOS DE DADOS

```php
$a=10;
$b="a";
$c=$a.$b;
print $c,$/; # 10a
```

```php
$a=10;
$b="20a";
$c=$a+$b;
print $c,$/; # 30
```

```php
$a=10;
$b="a";
$c=$a+$b;
print $c,$/; # 10
```

```php
$a=10;
$b="a20";
$c=$a+$b;
print $c,$/; #10
```

VARIÁVEIS E CONSTANTES

I think it's a new feature. Don't tell anyone it was an accident.

Larry Wall

VARIÁVEIS

```
$global = 10;

local $loc = 13;

my $scalar = "string";

@array =
(30, "oi", 3.14);

%hash =
('apple' => 10, 'banana' => 15);

%outrahash =
('apple j',25,'banana v',20);

%maisumahash =
(-apple, 10, -banana, 15);
```

```
print $global; # 10

print $scalar; # string

print @array;
# 30oi3.14

print %hash;
# apple10banana15

print %outrahash;
#apple j25banana v20

print %maisumahash;
#-banana15-apple10
```

VARIÁVEIS

```
@array =
(30, “oi”, 3.14);
```

```
print $array[1];
# oi

print $array[-3];
# 30
```

```
%hash =
('apple' => 10, 'banana' => 15);
```

```
print $hash{'apple'};
# 10
```

```
%outrahash =
('apple j', 25, 'banana v', 20);
```

```
print $outrahash{'apple j'};
# 25
```

```
%maisumahash =
(-apple, 10, -banana, 15);
```

```
print $maisumahash{-banana};
# 15
```

VARIÁVEIS

```
my @array = ("a".."f");

print @array."\n"; # abcdef

print (scalar @array); # 6

print "@array"; # a b c d e f

print $#array; # 5

print "$#array"; # 5
```

VARIÁVEIS

```
my @array = ("ab".."er");

print "@array";
```

```
# ab ac ad ae af ag ah ai aj ak al am
an ao ap aq ar as at au av aw ax ay az
ba bb bc bd be bf bg bh bi bj bk bl bm
bn bo bp bq br bs bt bu bv bw bx by bz
ca cb cc cd ce cf cg ch ci cj ck cl cm
cn co cp cq cr cs ct cu cv cw cx cy cz
da db dc dd de df dg dh di dj dk dl dm
dn do dp dq dr ds dt du dv dw dx dy dz
ea eb ec ed ee ef eg eh ei ej ek el em
en eo ep eq er
```

VARIÁVEIS

```perl
my @array = ("a".."f");
my @arrayb = @array[2,3,5];
my @arrayc = @array[1..3];

print "@arrayb"; # c d f

print "@arrayc"; # b c d
```

```perl
my $texto = "eu não vou à padaria!";
my @array = split(" ",$texto,3);

print @array; # eunãovou à padaria!
```

```perl
my @array = ("a".."f");
my @a = splice(@array,1,2,4,5);

print "@array"; # a 4 5 d e f

print "@a"; # b c
```

```perl
my @array = (1..4);
my $algo = join("-",@array);

print $algo; # 1-2-3-4
```

VARIÁVEIS

```perl
my @array = ("a".."f");

print "@array";                    # a b c d e f
print "\n";

my $a = push(@array,1);            # 7, a b c d e f 1
print $a.", @array"."\n";

my $b = pop(@array);               # 1, a b c d e f
print $b.", @array"."\n";

my $c = shift(@array);             # a, b c d e f
print $c.", @array"."\n";

my $d = unshift(@array,2);         # 6, 2 b c d e f
print $d.", @array"."\n";
```

VARIÁVEIS

```perl
my @array = ("ham","plate","egg","bun","cheese");
@array = sort(@array);

# sort("sub-rotina",@array)
# sort($sub-rotina,@array)
# sort({ sub-rotina },@array)
```

```perl
print "@array"; # bun cheese egg ham plate
```

```perl
my @array1 = ("a".."d");
my @array2 = (1..4);

my @array = (@array1,@array2);
print "@array"; # a b c d 1 2 3 4
```

“

Perl programming is an *empirical* science!

–Larry Wall

VARIÁVEIS

```
%hash =
('apple' => 10, 'banana' => 15);
%outrahash =
('apple j', 25, 'banana v', 20);
%maisumahash =
(-apple, 10, -banana, 15);

print %hash;                          # apple10banana15
print (scalar %hash);                 # 2/8
print "%hash";                        # %hash

print %outrahash;                     # apple j25banana v20
print (scalar %outrahash);            # 2/8
print "%outrahash";                   # %outrahash

print %maisumahash;                   # -banana15-apple10
print (scalar %maisumahash);          # 2/8
print "%maisumahash";                 # %maisumahash
```

VARIÁVEIS

```perl
my %data = (-JohnPaul => 45, -Lisa => 30, -Kumar => 40);
my @array = @data{-Lisa, -JohnPaul};

print "@array"; # 30 45

@array = keys %data;

print "@array"; # -Lisa -Kumar -JohnPaul

@array = values %data;

print "@array"; # 30 40 45

if( exists( $data{-Marvin} ) ) { # I don't know age of Marvin
    print "Marvin is $data{-Marvin} years old";
}
else {
    print "I don't know age of Marvin";
}

delete $data{-JohnPaul};

if( exists( $data{-JohnPaul} ) ) { # JohnPaul no longer exists
    print "JohnPaul is in %data";
}
else {
    print "JohnPaul no longer exists";
}
```

VARIÁVEIS

```perl
@array =
(30, "oi", 3.14);

%hash =
('apple' => 10, 'banana' => 15);

$array[5] = "perdido";

$hash{'orange'} = 10;

$novoarray[1] = 2;

$novoarray2[3];
```

```perl
print @array;
# após dois warnings
# 30oi3.14perdido

print (keys %hash);
# orangeapplebanana

print $#novoarray;
# sem uso de strict
# 1

print $#novoarray2;
#sem uso de strict
# -1
```

VARIÁVEIS

```perl
my $scalar = 5;
my $scalarb = $scalar;

$scalarb = 0;

print $scalar; # 5

my @array = (1,2,3,4);
my @arrayb = @array;

$arrayb[1] = 0;

print $array[1]; # 2

my %hash = (2,3,4,5);
my %hashb = %hash;

$hashb{2} = 0;

print $hash{2}; # 3
```

VARIÁVEIS

```
sub count {
    state $counter = print "world",$/;
    $counter++;
    return $counter;
}

print "hello",$/;
print count(),$/;
print count(),$/;
print count(),$/;

# hello
# world
# 2
# 3
# 4
```

```
{
    my $counter = 0;
    sub count {
        $counter++;
        return $counter;
    }
}

print count(); # 1
print count(); # 2
print count(); # 3
```

VARIÁVEIS ESPECIAIS

```
print "arquivo: ".__FILE__;
print "linha: ".__LINE__;
print "pacote: ".__PACKAGE__;
```

VARIÁVEIS ESPECIAIS

```
$[ = 1;
```

```
my @array = (2..5);
```

```
print $array[2];
```

```
# 3
```

VARIÁVEIS ESPECIAIS

```
$!                  $-                  $?
$"                  $`                  $DEBUGGING
$^S                 $.                  $@                  %ENV
$LIST_SEPARATOR     $a                  $[                  $^F
$#                  $PERL_VERSION       $\                  $^H
$^T                 $/                  $]                  %OVERLOAD
$$                  $ACCUMULATOR        $ERRNO              $^I
$%                  $0                  $^                  $^L
$&                  $ARG                $EUID               @+
$^UTF8LOCALE        $PID                $UID                $^M
$'                  $:                  $^A                 @-
$^V                 $ARGV               $WARNING            $^N
$(                  $;                  $^C                 @_
$^W                 $b                  $|                  $^O
$)                  $<                  $~                  @ARGV
$*                  $BASETIME           $^D                 $^OPEN
$+                  $PROCESS_ID         %!                  $^P
$^X                 $=                  $^E
$,                  $>                  %^H
$_                  $COMPILING          $^ENCODING
```

http://learn.perl.org/docs/keywords.html

“

I'm reminded of the day my daughter came in, looked over my shoulder at some Perl 4 code, and said, ‘What is that, swearing?’

–Larry Wall

CONSTANTES

```
use constant PI    => 4 * atan2(1, 1);
use constant DEBUG => 0;
use constant {
    SEC   => 0,
    MIN   => 1,
    HOUR  => 2,
    MDAY  => 3,
    MON   => 4,
    YEAR  => 5,
    WDAY  => 6,
    YDAY  => 7,
    ISDST => 8,
};
use constant WEEKDAYS => qw(
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
);
print "Today is ", (WEEKDAYS)[ (localtime)[WDAY] ], ".\n";
```

CONSTANTES

```perl
use Readonly;

Readonly my $SPEED_OF_LIGHT => 299_792_458; # m/s
Readonly my %DATA => (
Mercury => [0.4,     0.055    ],
Venus   => [0.7,     0.815    ],
Earth   => [1,       1        ],
Mars    => [1.5,     0.107    ],
Ceres   => [2.77,    0.00015  ],
Jupiter => [5.2,   318        ],
Saturn  => [9.5,    95        ],
Uranus  => [19.6,   14        ],
Neptune => [30,     17        ],
Pluto   => [39,   0.00218     ],
Charon  => [39,   0.000254    ],
);
Readonly my @PLANETS => sort keys %DATA;
```

http://perlmaven.com/constants-and-read-only-variables-in-perl

OPERADORES

Although the Perl Slogan is There's More Than One Way to Do It, I hesitate to make 10 ways to do something.

Larry Wall

OPERADORES

```
+
-
*
/
%
**

==
!=
<=>
<
>
<=
>=
```

```
lt
gt
le
ge
eq
ne
cmp

=
+=
-=
*=
/=
%=
**=
```

```
&
|
^
~
<<
>>

and
&&
or
||
not
!
```

```
.
x
..
++
--
->
```

OPERADORES

```
my $inp = <STDIN>;
```

```
my $inp = <>;
```

```
while(<>)
{
    print $_;
}
```

```
while(<>)
{
    print;
}
```

OPERADORES

```
q{ }        q% %        q. .

qq{ }       qq$ $       qq. .

qx{ }       qx@ @       qx. .

q( )        q d d       q; ;

qq( )       qq l l      qq; ;

qx( )       qx p p      qx; ;

q& &        q 9 9       q# #

qq& &       qq 0 0      qq# #

qx& &       qx 7 7      qx# #
```

“

In general, if you think something isn’t in Perl, try it out, because it usually is.

–Larry Wall

OPERADORES

q{}	Literal	não
qq{}	Literal	sim
qx{}	Comando	sim*
qw{}	Lista de palavras	não
m{}	Padrão encontrado	sim*
qr{}	Padrão	sim*
s{}{}	Substituição	sim*
tr{}{}	Transliteração	não
y{}{}	Transliteração	não
<<EOF	here-doc	sim*

```
my $s = $d =~ tr/e/E/;
```

OPERADORES

```
left        terms and list operators (leftward)
left        ->
nonassoc    ++ --
right       **
right       ! ~ \ and unary + and -
left        =~ !~
left        * / % x
left        + - .
left        << >>
nonassoc    named unary operators
nonassoc    < > <= >= lt gt le ge
nonassoc    == != <=> eq ne cmp ~~
left        &
left        | ^
left        &&
left        || //
nonassoc    ..  ...
right       ?:
right       = += -= *= etc. goto last next redo dump
left        , =>
nonassoc    list operators (rightward)
right       not
left        and
left        or xor
```

OPERADORES

```
1.      # These evaluate exit before doing the print:
2.      print($foo, exit);  # Obviously not what you want.
3.      print $foo, exit;   # Nor is this.
4.
5.      # These do the print before evaluating exit:
6.      (print $foo), exit; # This is what you want.
7.      print($foo), exit;  # Or this.
8.      print ($foo), exit; # Or even this.
```

http://perldoc.perl.org/perlop.html

REFERÊNCIAS

Let's say the docs present a simplified view of reality...

Larry Wall

REFERÊNCIAS

- É assim que estruturas de dados mais complexas são construídas
- O operador *backslash* cria uma nova referência para a variável
- Cada referência guarda o tipo de dado que referencia
- Mesmo que a tabela de símbolos seja apagada, essa referência permanece
- Evita a cópia de estruturas completas

REFERÊNCIAS

```
$scalarref = \$foo;
$arrayref  = \@ARGV;
$hashref   = \%ENV;
$coderef   = \&handler;
$globref   = \*foo;

$scalar = ${$scalarref};
$scalar2 = $$scalarref;

# [@{$arrayref}]
@array = @{$arrayref};
@array2 = @$arrayref;

%hash = %{hashref};
%hash2 = %$hashref;

&$coderef();
&{$coderef}();

${* ${ *globref{SCALAR} }{SCALAR} };
```

```
$sc = "essa string";
print ${ *sc{SCALAR} };
# essa string
```

```
$sc = "essa string";
*globref{SCALAR} = \$sc;
print ${ *globref{SCALAR} };
# essa string
```

```
$sc = "essa string";
*globref = \$sc;
print ${ *globref{SCALAR} };
# essa string
```

REFERÊNCIAS

```perl
my $arrayref = [1, 2, ['a', 'b', 'c']];

print $arrayref->[2][1]; # b
```

```perl
my $hashref = {
    'Adam'  => 'Eve',
    'Clyde' => 'Bonnie',
};
```

```perl
my @array = (1,2,3,[‘a','b','c']);

print $array[3][1]; # b
```

```perl
my @list = (\$a, \@b, \%c);

@list = \($a, @b, %c);
```

REFERÊNCIAS

```perl
# retorna referência para hash
sub hashem {        +{ @_ } }
sub hashem { return { @_ } }

# bloco de código
sub showem {         { @_ } }   # evitar pois é ambíguo
sub showem {        {; @_ } }
sub showem { { return @_ } }
```

REFERÊNCIAS

```
$coderef = sub { print "Boink!\n" };
```

```
sub newprint {
    my $x = shift;
    return sub { my $y = shift; print "$x, $y!\n"; };
}

$h = newprint("Howdy");
$g = newprint(“Greetings");

# Time passes...
&$h("world"); # Howdy, world!
&$g("earthlings"); # Greetings, earthlings!
```

```
my $objref = new Doggie( Tail => 'short', Ears => 'long' );
```

REFERÊNCIAS

```perl
my $foo;
$foo = \$foo;
```

```perl
ref($referencia)
# SCALAR
# ARRAY
# HASH
# CODE
# GLOB
# REF
```

CONDICIONAIS E LOOPS

The computer should be doing the hard work. That's what it's paid to do, after all.

Larry Wall

CONDICIONAIS

```
if(expr1){
    # Code
}
elsif(expr2){
    # Code
}
elsif(expr3){
    # Code
}
else{
    # Code
}
```

```
unless(expr1){
    # Code
}
elsif(expr2){
    # Code
}
elsif(expr3){
    # Code
}
else{
    # Code
}
```

```
$v = Exp1 ? Exp2 : Exp3;
```

```
print $y,$/ if $y < 1;
```

CONDICIONAIS

```
my $v = 6;

switch($v)
{
    case 6
    {
        print 1,$/;
        next;
    }
    case { 6 => 4, "b" => "g" }
    {
        print 2,$/;
        next;
    }
    case [1,2,3,6]
    {
        print 3,$/;
    }
    else
    {
        print 4,$/;
    }
}
```

CONDICIONAIS

- São considerados false:
  - número 0
  - string ‘0’
  - string “”
  - lista vazia ()
  - *undef*

LOOPS

```perl
my $a = 0;
while($a++<5)
{
    print $a," ";
}
# 1 2 3 4 5
until(--$a<1)
{
    print $a," ";
}
# 5 4 3 2 1
```

```perl
for($a = 0; $a < 5; $a++)
{
    print $a," ";
} # 0 1 2 3 4
print $a,$/; # 5
```

```perl
my $a = 0;
do{
    print $a," ";
    $a++;
}while( $a < 5 );
# 0 1 2 3 4
```

```perl
foreach $a (1..5)
{
    print $a," ";
} # 1 2 3 4 5
# print $a,$/;
```

LOOPS

```
while(condition){
    statement(s);
}continue{
    statement(s);
}

foreach $var (list){
    statement(s);
}continue{
    statement(s);
}
```

```
OUTER:while( condition ){
    statement(s);
INNER:while ( condition ){
        if( condition ){
            statement(s)
            next OUTER;
        }
        statement(s);
    }
    statement(s);
}
```

LOOPS

```
OUTER:while( condition ){
    statement(s);
INNER:while ( condition ){
    if( condition ){
        statement(s)
        last OUTER;
    }
    statement(s);
}
    statement(s);
}
```

```
$a = 0;
while($a < 10){
    if( $a == 5 ){
        $a++;
        redo;
    }
    print "$a ";
}continue{
    $a++;
}
# 0 1 2 3 4 6 7 8 9
```

“

Just put in another goto, and then it'll be readable.

–Larry Wall

LOOPS

```
goto LABEL;

goto EXPR;

goto &NAME;
```

```
LOOP:do
{
    statement(s);
    if( condition ){
        goto LOOP;
    }
}while( condition );
```

```
LOOP:do
{
    statement(s);
    if( condition ){
        goto "LO"."OP";
    }
}while( condition );
```

```
sub ei
{
    print "hey\n";
}

sub ho
{
    print "ho\n";
    goto &ei;
    print "eu aqui\n";
}

ho();
# ho
# hey
```

SUB-ROTINAS

And I don't like doing silly things (except on purpose).

Larry Wall

SUB-ROTINAS

- Permite declaração de sub-rotinas
- Parâmetros são acessados pela variável @_
- Passagem de parâmetros por referência
- Se não há *return* e a última linha for:
  - uma expressão, retorna o valor da expressão
  - um loop, o valor de retorno é não especificado
  - nada (sub-rotina vazia), retorna uma lista vazia
- Variáveis não declaradas privadas são globais

SUB-ROTINAS

```
sub max {
    my $max = shift(@_);
    foreach $foo (@_) {
        $max = $foo if $max < $foo;
    }
    return $max;
}
$bestday = max($mon,$tue,$wed,$thu,$fri);
```

```
sub foo ($first, $, $third) {
    return "first=$first, third=$third";
}

sub foo ($left, $right = 0) {
    return $left + $right;
}
```

SUB-ROTINAS

```perl
my $coderef = \&rutina;
my $y = 3;

&{$coderef}($y);
print $y,$/; # 4

sub rutina
{
    #    my $x = shift;
    $_[0] = 4;
}
```

SUB-ROTINAS

```perl
my $file = File->new( $path, $data );
my $method = 'save';
$file->$method();
# $file->save()
```

```perl
my $class = 'File';
my $file = $class->new( $path, $data );
```

MODULARIZAÇÃO E ORIENTAÇÃO A OBJETOS

Just don't create a file called -rf.

Larry Wall

MODULARIZAÇÃO

- Pacotes (classes) e módulos
- Pacotes definem o namespace do escopo
- Um namespace é representado por uma tabela de símbolos
- Após a declaração de um pacote, variáreis definidas pertencerão à este pacote
- Módulos são arquivos de extensão .pm que possuem sub-rotinas em um namespace (pacote) próprio

MODULARIZAÇÃO

```
# MyModule.pm
package MyModule;

use strict;
use Exporter;
use vars qw($VERSION @ISA @EXPORT @EXPORT_OK %EXPORT_TAGS);

$VERSION     = 1.00;
@ISA         = qw(Exporter);
@EXPORT      = ();
@EXPORT_OK   = qw(func1 func2);
%EXPORT_TAGS = ( DEFAULT => [qw(&func1)],
Both    => [qw(&func1 &func2)]);

sub func1  { return reverse @_  }
sub func2  { return map{ uc }@_ }

1;
```

MODULARIZAÇÃO

- Também existem blocos BEGIN { ... } e END { ... }
- O bloco BEGIN é executado antes de qualquer expressão do script
- O bloco END é executado logo antes do interpretador finalizar

```
package Foo;
print "Begin and Block Demo\n";

BEGIN {
    print "This is BEGIN Block\n"
}

END {
    print "This is END Block\n"
}
1;
```

ORIENTAÇÃO A OBJETOS

- Uma classe é um pacote que contém métodos necessários para criação e manipulação dos objetos
- Em Perl, objetos são simplesmente estruturas de dados que foram, explicitamente, associadas a uma classe
- Essa associação é feita com uso de uma função: *bless*
- O construtor pode receber qualquer nome
- Qualquer sub-rotina que use *bless* em uma estrutura de dados e em uma classe, é um construtor válido
- Atributos do objeto geralmente são armazenados no próprio objeto (objeto é uma hash anônima, atributo pode ser chave)

ORIENTAÇÃO A OBJETOS

```
package File;
sub new {
    my $class = shift;
    return bless {}, $class;
}

package File;
sub load {
    my $class = shift;
    return bless {}, $class;
}
```

```
sub new {
    my $class = shift;
    my $self = {};
    bless $self, $class;
    $self->_initialize();
    return $self;
}
```

ORIENTAÇÃO A OBJETOS

```
package File;
sub new {
    my $class = shift;
    my ( $path, $data ) = @_;
    my $self = bless {
        path => $path,
        data => $data,
    }, $class;
    return $self;
}
```

```
sub path {
    my $self = shift;
    return $self->{path};
}
```

```
sub path {
    my $self = shift;
    if (@_) {
        $self->{path} = shift;
    }
    return $self->{path};
}
```

ORIENTAÇÃO A OBJETO

```perl
sub save {
    my $self = shift;
    open my $fh, '>', $self->path() or die $!;
    print {$fh} $self->data()       or die $!;
    close $fh                       or die $!;
}
```

```perl
my $pod = File->new( 'perlobj.pod', $data );
$pod->save();
```

```perl
package File::MP3;
use parent 'File';  # sets @File::MP3::ISA = ('File');
my $mp3 = File::MP3->new( 'Andvari.mp3', $data );
$mp3->save();
```

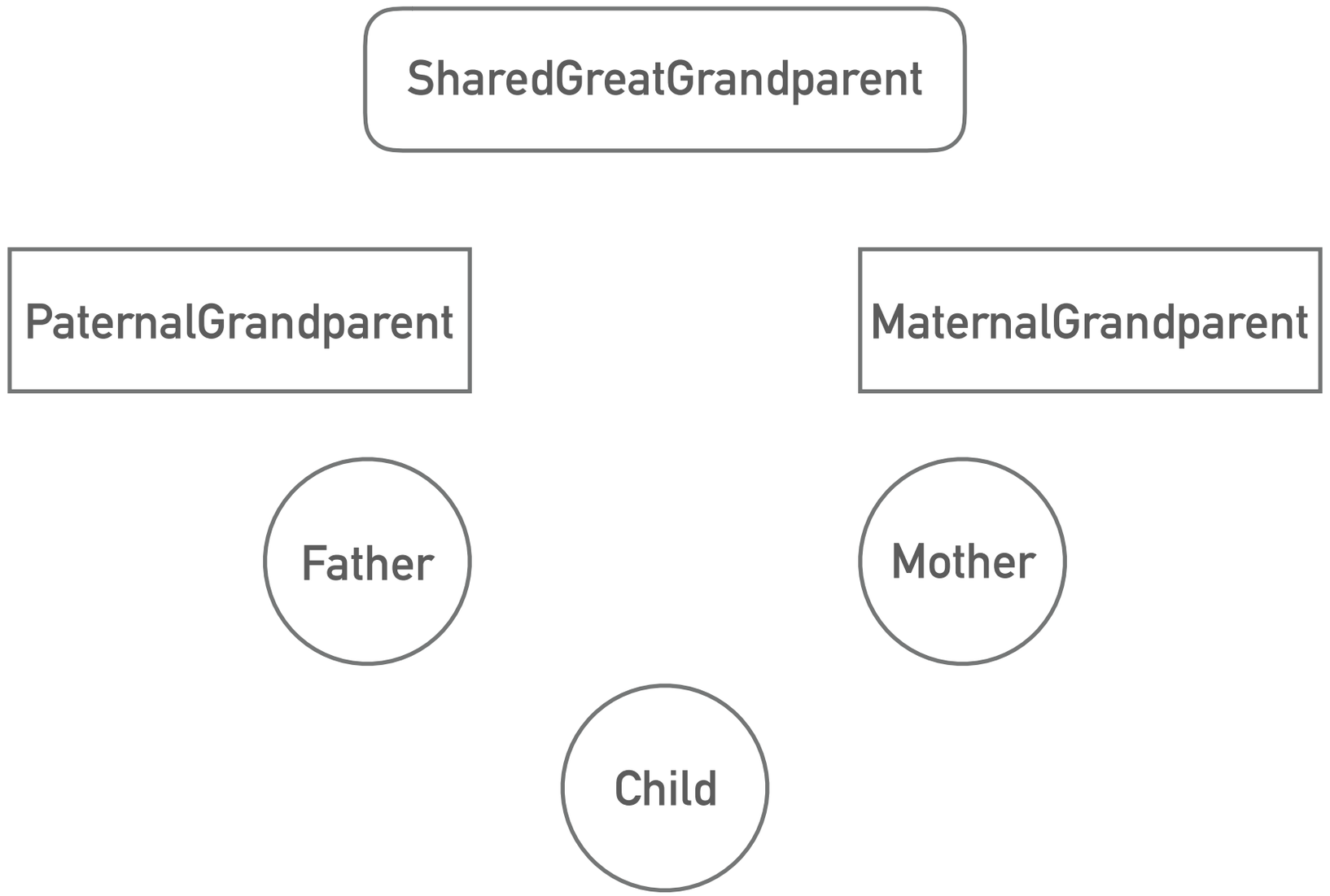
ORIENTAÇÃO A OBJETO

Herança múltipla pode indicar problemas de design, mas Perl sempre fornecerá corda suficiente para você se enforcar, se pedir por isso.

http://perldoc.perl.org/perlobj.html

ORIENTAÇÃO A OBJETO

```
package Child;
use parent 'Father', 'Mother';
```



ORIENTAÇÃO A OBJETO

```perl
sub save {
    my $self = shift;
    say 'Prepare to rock';
    $self->SUPER::save();
}
```

```perl
SUPER::save($thing);   # FAIL: looks for save() sub in package SUPER
SUPER->save($thing);   # FAIL: looks for save() method in class SUPER
$thing->SUPER::save(); # Okay: looks for save() method in parent classes
```

SUPER

```
package A;
sub new {
    return bless {}, shift;
}
sub speak {
    my $self = shift;
    say 'A';
}
package B;
use parent -norequire, 'A';
sub speak {
    my $self = shift;
    $self->SUPER::speak();
    say 'B';
}
package C;
use parent -norequire, 'B';
sub speak {
    my $self = shift;
    $self->SUPER::speak();
    say 'C';
}
my $c = C->new();
$c->speak();
```

```
# A
# B
# C
```

EXCEÇÕES E CONCORRÊNCIA

I don't know if it's what you want, but it's what you get.

Larry Wall

EXCEÇÕES

- É necessário importar a biblioteca *Try::Tiny* para usar bloco try-catch
- Maneira mais simples: uso de *eval()* ou *eval {}*
- *eval* executa o código ou expressão separadamente, como se fosse um pequeno programa
- Quando a forma *eval {BLOCK}* é usada, o código do bloco é avaliado em tempo de compilação
- *eval* retorna o resultado da última expressão ou o valor indicado por *return*
- Melhor uso: com indicadores de erro

EXCEÇÕES

- *warn*: indica um erro e imprime uma mensagem em STDERR

```
chdir('/etc') or warn "Can't change directory";
```

- *die*: mesmo que *warn*, mas também chama *exit*

```
chdir('/etc') or die "Can't change directory";
```

EXCEÇÕES

- *warn* e *die* podem ser insuficientes para o usuário final
- É necessário uma maneira de melhor indicar problemas ao usar outros módulos
- Solução: usar módulo *Carp*, que contém as funções *carp*, *cluck*, *croak* e *confess*

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp;

sub function {
    warn "Error in module!";
}
1;
```

```
use T;
function();

# Error in module! at T.pm line 9.
```

EXCEÇÕES

- *carp*: equivalente a *warn*

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp;

sub function {
    carp "Error in module!";
}
1;
```

```
use T;
function();

# Error in module! at test.pl line 4
```

EXCEÇÕES

- *cluck*: como o *carp*, mas imprime o *stack trace* de todos os módulos até a chamada da função

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp qw(cluck);

sub function {
    cluck "Error in module!";
}
1;
```

```
use T;
function();

# Error in module! at T.pm line 9
#    T::function() called at test.pl line 4
```

EXCEÇÕES

- *croak*: como o *die*, mas reporta a quem chamou a função, ou seja, um nível acima

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp;

sub function {
    croak "Error in module!";
}
1;
```

```
use T;
function();

# Error in module! at test.pl line 4
```

EXCEÇÕES

- *confess*: imprime o *stack trace* do script de origem à função no módulo e chama *exit*

```
package T;

require Exporter;
@ISA = qw/Exporter/;
@EXPORT = qw/function/;
use Carp;

sub function {
    confess "Error in module!";
}
1;
```

```
use T;
function();

# Error in module! at T.pm line 9
#    T::function() called at test.pl line 4
```

EXCEÇÕES

```
eval {
    die "Oops!";
    1;
} or do {
    my $e = $@;
    print("Something went wrong: $e\n");
};
```

```
unless(chdir("/etc")){
    die "Error: Can't change directory - $!";
}
```

```
die "Error: Can't change directory!: $!" unless(chdir("/etc"));
```

```
print(exists($hash{value}) ? 'There' : 'Missing',"\n");
```

“

I don't know if it's what you want, but it's what you get.

–Larry Wall

CONCORRÊNCIA

Por que é considerada tão ruim?

WARNING

The "interpreter-based threads" provided by Perl are not the fast, lightweight system for multitasking that one might expect or hope for. Threads are implemented in a way that make them easy to misuse. Few people know how to use them correctly or will be able to provide help.

The use of interpreter-based threads in perl is officially discouraged.

CONCORRÊNCIA

- Cada *thread* inicia um interpretador Perl para si, ou seja, não temos compartilhamento de dados, por *default*
- Isso significa que para cada *thread* criada, toda e qualquer estrutura de dados será copiada! (!!!!!)
- Variáveis definidas fora da *thread* também serão copiadas
- Variáveis compartilhadas são como variáveis “comuns”, que gastam mais recursos para manter a sincronia e o ar de especial (memória e CPU)
- Resumo: não usem *threads* em Perl (a não ser que seja de algum módulo específico no CPAN)

AVALIAÇÃO E CONCLUSÃO

Just don't compare it with a real language,
or you'll be unhappy…

Larry Wall

AVALIAÇÃO

Critérios Gerais	C++	Java	Perl
Aplicabilidade	Sim	Parcial	Parcial
Confiabilidade	Não	Sim	Sim
Aprendizado	Difícil	Médio	Variável
Eficiência	Boa	Média	Média
Portabilidade	Baixa	Boa	Boa
Método de Projeto	Estruturado e OO	OO	Script, Estruturado e OO
Evolutibilidade	Ruim	Boa	Péssima
Reusabilidade	Boa	Boa	Ótima
Integração	Boa	Média	Boa

AVALIAÇÃO

Critérios Específicos	C++	Java	Perl
Escopo	Estático	Estático	Estático e Dinâmico
Tipos Primitivos e Compostos	Sim	Sim	Sim
Gerenciamento de Memória	Programador	Sistema	Sistema
Persistência	Bibliotecas	JDBC, Bibliotecas e Serialização	Módulos e Serialização
Passagem de Parâmetros	Cópia e Referência	Cópia e Referência	Referência

CONCLUSÃO

- Perl é uma ótima linguagem de programação
- Possui ferramentas como expressões regulares nativamente
- Possui suporte para execução de códigos externos
- Aprendizado inicial é fácil, pois não faltam opções para o programador (como diz o *slogan*)
- A linguagem torna-se mais difícil de aprender com o passar do tempo, pois apresenta tópicos densos e de difícil compreensão
- Exemplo: tabelas de símbolo e globs
- Mas Perl tem algo a mais para quem a usa, pois...

“

It's all magic.

–Larry Wall

REFERÊNCIAS BIBLIOGRÁFICAS

http://www.tutorialspoint.com/perl/index.htm

http://qntm.org/files/perl/perl.html

http://etutorials.org/Programming/perl+bioinformatics/Part+I+Object-Oriented+Programming+in+Perl/Chapter+1.+Modular+Programming+with+Perl/

http://stackoverflow.com/questions/11019979/how-does-perl-polymorphism-work

http://stackoverflow.com/questions/10342875/how-to-properly-use-the-try-catch-in-perl-that-error-pm-provides

http://stackoverflow.com/questions/3395117/are-perl-strings-immutable

http://www.ehow.com/info_12295724_programming-language-called-duct-tape-internet.html

http://archive.oreilly.com/pub/a/oreilly/perl/news/importance_0498.html

http://www.teaser.fr/~amajorel/psgfcp/

http://perldoc.perl.org/threads.html

http://perldoc.perl.org/perlop.html

http://perldoc.perl.org/perlpragma.html

http://perldoc.perl.org/perlref.html

http://perldoc.perl.org/perlreftut.html

http://perldoc.perl.org/Carp.html

http://perldoc.perl.org/functions/eval.html

http://perldoc.perl.org/perlxs.html

http://perldoc.perl.org/perlretut.html

http://learn.perl.org/docs/keywords.html

http://perldoc.perl.org/constant.html

http://www.perlmonks.org/?node_id=290607

http://www.perlmonks.org/?node_id=288022

http://www.perlmonks.org/?node_id=211441

http://www.perlmonks.org/?node_id=102347

http://www.perlmonks.org/?node_id=15838

http://www.perlmonks.org/?node_id=78523

http://perlmaven.com/constants-and-read-only-variables-in-perl

http://www.sarand.com/td/ref_perl_reserve.html

https://en.wikiquote.org/wiki/Larry_Wall

https://pt.wikipedia.org/wiki/Larry_Wall

CÓDIGO CONFUSO

```
#!/usr/bin/perl

not exp log srand xor s qq qx xor
s x x length uc ord and print chr
ord for qw q join use sub tied qx
xor eval xor print qq q q xor int
eval lc q m cos and print chr ord
for qw y abs ne open tied hex exp
ref y m xor scalar srand print qq
q q xor int eval lc qq y sqrt cos
and print chr ord for qw x printf
each return local x y or print qq
s s and eval q s undef or oct xor
time xor ref print chr int ord lc
foreach qw y hex alarm chdir kill
exec return y s gt sin sort split
```

ESCOPO DINÂMICO

```
sub dinamica
{
    return $x;
}

sub principal
{
    local $x = 6;
    dinamica();
}

print principal(),$/;
# 6
```

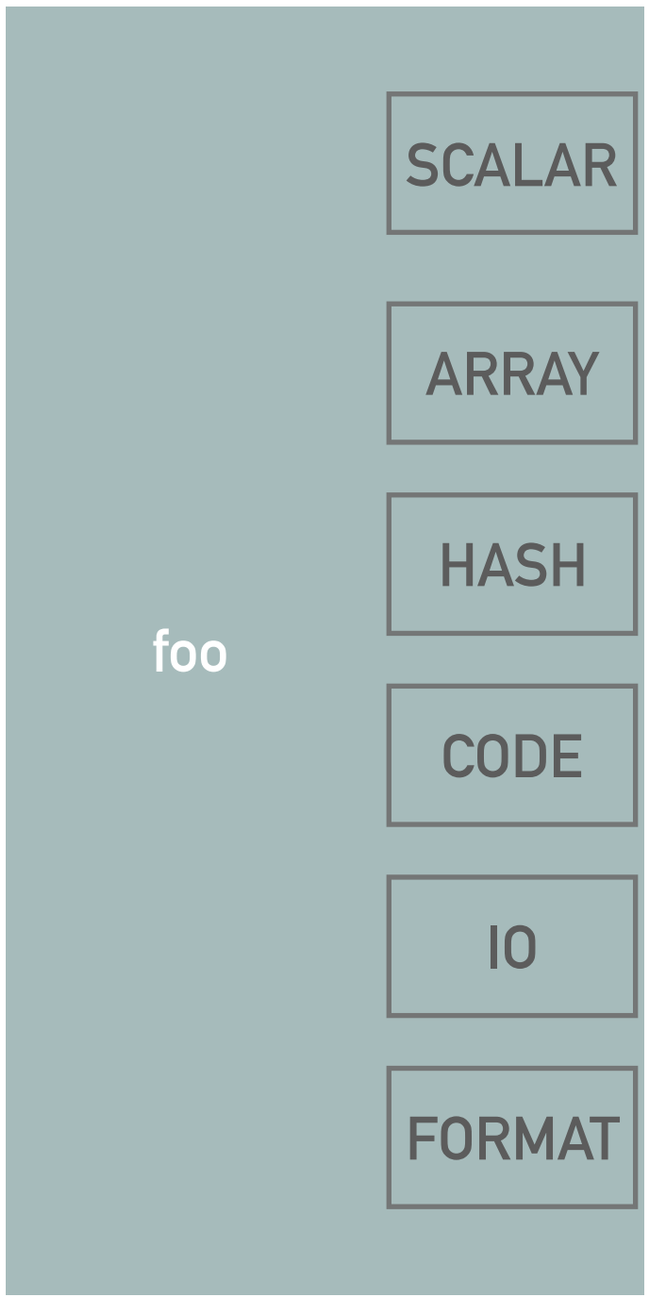
CONVERSÃO DE REFERÊNCIAS

```perl
my @array = (1,2,3,4);
my $rf = \@array;

my %hash = %{rf};

print %hash,$/;
# Not a HASH reference at ./teste.pl line 8.
```

GLOB



EXECUTANDO CÓDIGOS EXTERNOS

```perl
my $a = system('pwd');
# /Users/mabrunoro/Public/default/lptrab/test

print $a,$/;
# 0

# exec não retorna ao script
```

```perl
$a = qx/ls/;
print $a;
# Classe.pl
# T.pm
# array.pl
# hash.pl
# hello_world.pl
# scalar.pl
# teste.pl

# open usa um pipe do comando para o script (ou o contrário)
```

ARQUIVOS

```
use IO::File;
$fh = IO::File->new();
if ($fh->open("< file")) {
    print <$fh>;
    $fh->close;
}
$fh = IO::File->new("> file");
if (defined $fh) {
    print $fh "bar\n";
    $fh->close;
}
```

```
$fh = IO::File->new("file", "r");
if (defined $fh) {
    print <$fh>;
    undef $fh;       # automatically closes the file
}
$fh = IO::File->new("file", O_WRONLY|O_APPEND);
if (defined $fh) {
    print $fh "corge\n";
    $pos = $fh->getpos;
    $fh->setpos($pos);
    undef $fh;       # automatically closes the file
}
```

EXPRESSÕES REGULARES

```perl
$greeting = "World";
print "It matches\n" if "Hello World" =~ /$greeting/;

$foo = 'house';
'cathouse' =~ /cat$foo/;   # matches
'housecat' =~ /${foo}cat/; # matches

$x = 'bcr';
/[$x]at/;   # matches 'bat, 'cat', or 'rat'

/[^0-9]/;  # matches a non-numeric character
```

CHAMANDO FUNÇÕES PELO NOME

```
my $f = 'foo';

sub foo
{
    print "oaksiaaibf\n";
}

$f->();
&$f();
```

TRANSLITERAÇÃO

```perl
my $text = 'abc bad acdf';
print $text,$/; # abc bad acdf

$text =~ tr/a/z/;
print $text,$/; # zbc bzd zcdf
```

- Permite sobrecarga de operadores com uso do pragma *overload*
- ~“\x{3B1}” é “\x{FFFF_FC4E}”
- Serialização YAML e JSON
- .. retorna true no mínimo uma vez, quando o valor da esquerda é true, ... não
- É possível ler arquivos .csv (podemos usar *split*)
- Sintaxe de Perl: http://perldoc.perl.org/perlsyn.html
- Licenças: http://perldoc.perl.org/index-licence.html
- Se o módulo *utf8* é usado, Perl trata o código como Unicode, senão trata como ASCII
- Não há aritmética de referências
- Strings são mutáveis (associadas a um buffer) e podem sofrer transliteração