# Programação Comparativa: Perl/PHP. Perl a Linguagem Livre

Ole Peter Smith, IME, UFG, ole@mat.ufg.br

Semana de Integração
Faculdade Estácio de Sá - Goiânia-GO
Curso de Redes de Computadores
12/05/2011

TI**A**MTOWTDI:
There is **Always** More Than One Way To Do It!
*Larry Wall*

# Me & Programming

- 1980: Draw Bicycle Whell (Amiga/Basic)
- 1990: Turbo Pascal 5.0
  Conjugating Spanish Verbs...
- 1992: PhD in Optimization
  AutoCAD/LISP, C/C++, Shells
- 1996: Split, join (regex) in C/C++...
  $\rightsquigarrow$ Failure
- 1996: Systems Administrator, MAT, DTU
- 1996, Perl: System - Web - Tk
- 2002: Brasil
- 2004: SAdE, Sistema Adm. Escolar » No DBs!
- 2009: IME, UFG
- PHP: WOOIDs, SiPE, SiDS, SiVent

# What does the Free in 'Free Software' Mean?

- No problem may be solved
  Using the same parameters that lead to it
  *Einstein*
- I have no Rules is a Rule...
- Axioms - Paradigmas - Dogmas
- Question, Always!
- Free Thinking:
  My Rights Starts Where Yours End
- Isonomy
- Tolerances
- Free Knowledge vs. Science

# Using Perl we are Free to do:

- Linux user/group adm
- Quota adm
- Backup adm
- Tk Window App
- CGI programming
- Lowlevel networking
- Perl & Expect: Controle command-line programs
- ...
- Exercise 1:
  Rename the php executable on your Linux & Restart
  What Happened?
- Exercise 2:
  Rename perl executable on your Linux & Restart
  What Happened?

# Perl: use strict

Perl

```
use strict;

my $var=0;
print $ver;
```

Compile time error!

PHP

```
my $var=0;
print $ver;
```

'No' Problem

Liberty is to be strict, if and when we want...

`perl -w`: Stricter!
Perl also used for critical operations
Taint

# Variables

| Type | Perl | PHP |
|---|---|---|
| Scalar | $ | $ |
| List | @ | $ |
| Hash | % | $ |

```
my $scalar="Oi";
my @list=();
$list[3]=$scalar;
my %list=();
$list{ "key" }=$scalar;
my $list=\@list; #Ref to list
print $list->[3];
my $list=\%list; #Ref to hash
print $list->{ "key" };
```

# Lists - Perl

```
my @list1=("Ole","Fernando");
my @list2=qw/Ole Peter Smith/;
my @list=grep { /e$/ } @list2;
push(@list,"da","Silva",@list2);
my @sobjects=grep { $_{ "Name" }=~/^O/ } @objects;
map { $_=$_{ "Name" } } @objects;
my $path=join("/",$path1,"tmp",@list2);
foreach my $item (@list) #No counter
{
    $item...
}
        for (my $n=0;$n<@list;$n++) { $list[$n]... }
```

# Lists - PHP

```
$list1=array("Ole","Fernando");
#qw: Não tem!
$list=preg_grep('/e$/',$list1);
array_push($list,"da");
array_push($list,"Silva");
#map: Não tem!
$path=join("/",array($path1,"tmp",$list2)); #WRONG!
foreach ($list as $item) { $item... }
foreach ($list as $id => $item) { $item... }
for ($n=0;$n<count($list);$n++) { $list[$n]... }
```

# Associated Arrays - Perl

- ```
  my %list1=
  (
     "Name" => "Ole",
     "Email" => "ole.ufg@gmail.com",
  );
  my %list2=
  (
     "Name","Ole",
     "Email","ole@mat.ufg.br",
  );
  my %list=(%list1,%list2); #Email is 'ole@mat...'
  print $list{ "Name" };
  my @list=%list;
  ```
- List: Even number of elements


  ```

# Associated Arrays - PHP

- Maintain counter
- ```
  foreach ($list as $key => $value)
  {
      ...
  }
  ```


  ```

# Functions

- Perl:

  One Argument:

```
sub MyFunction
{
    my $arg1=$_;
}
```

  List of Arguments:

```
sub MyFunction
{
    my @args=@_;
}
```

- PHP:

```
function MyFunction($arg1,$arg2=FALSE,...) { ... }
```

# Return two Values: Perl

```
sub MyFunc
{
    ...
    return ($res1,$res2);
}

sub MyOtherFunc
{
    ...
    my ($res1,$res2)=MyFunc();
}
```

# Return two Values: PHP

```
sub MyFunc
{
   ...
   return array($res1,$res2);
}

sub MyOtherFunc
{
   ...
   $list=MyFunc();

   $res1=$list[0];
   $res2=$list[1];
}
```

# Named Arguments, Perl

```
sub MyFunc
{
   my %args=@_;
   if ($args{ "BoldFirst" }) ...
   unless ($args{ "TableHead" }) ...
}
sub MyCallingFunc
{
   ...
   MyFunc
   (
      "BoldFirst" => 1,
      "TableHead" => 0,
   );
}
```

# Named Arguments, PHP?

```
function MyFunc($args)
{
   if ($args[ "BoldFirst" ]) ...
}
sub MyCallingFunc
{
   MyFunc(
      array(
         "BoldFirst" => 1,
         "TableHead" => 0,
      )
   );
}
```

Rather clumbsy - to say the least

# Objects in Perl

- @INC: List of include paths
- @ISA: List of classes/namespaces
- bless

```
sub new
{
    my ($pkg,%args)=@_;
    %args=(%Defaults,%args);
    my $this={};
    bless($this,$pkg);
    $this->Init(%args);
    return $this;
}
```

- Objects may divided in several files, same package.

# Objects in Perl

- Basic classes, ex:
  HTML in Html.pm
  LaTeXin LaTeX.pm
  Sql in Sql.pm
  Mail in Mail.pm
- ```
  use Object;
  use MyHtml;
  use MyLatex;
  unshift(@ISA,qw/Object/);
  unshift(@ISA,qw/MyHtml/);
  unshift(@ISA,qw/MyLatex/);
  ```
- Include necessary ones...
- Why Linear Inheritance?


  ```

# Conclusion

- Free to write as I think
- unless vs. if (!)
- TI**A**MTOWTDI
- Discutir programação - não linguagens
- Real Programmers can write assembly code in any language... Larry Wall