8º SEMINÁRIO LP - 2017/2

PERL

Dhiego Santos Broetto
Israel Pereira de Souza
Lucas Fabio Ferreira Silva

PARTE 01
INTRODUÇÃO

INTRODUÇÃO

- PERL - Practical Extraction and Report Language.
- Criada por Larry Wall, empregado da NSA, em 1987.
- Influenciada por Shell Script, LISP, C/C++, AWK, entre outras.
- Atualmente encontra-se na versão 5.26.1.
- Open Source, licenciado sob Licença Artística e Licença Pública Geral GNU.
- “There’s more than one way to do it”. (Wall, Larry)
- “Perl is a language for getting your job done”. (Wall, Larry)
- Pequeno código faz muito.

SOBRE PERL

- Aplicações:
  - ❑ Desenvolvimento WEB;
  - ❑ Administração de sistemas;
  - ❑ Acesso a banco de dados;
  - ❑ Processamento de textos;
  - ❑ Bioinformática.

SOBRE PERL

- Características:
  - ❑ Sintaxe simples;
  - ❑ Portátil (Multiplataforma);
  - ❑ Multiparadigma: Funcional, Estrutural e Orientada a Objetos;
  - ❑ Excelente para manipulação de textos e arquivos;
  - ❑ Desenvolvimento rápido e eficiente;
  - ❑ Tipagem dinâmica;
  - ❑ Case sensitive;
  - ❑ Gerenciamento de memória feito pela linguagem;
  - ❑ Suporta Unicode.

SINTAXE

- A primeira linha do programa pode começar com o seguinte *shebang*: **#!/usr/bin/perl**, que indica o caminho para o compilador-interpretador.
- Assim como em C, deve-se colocar ponto e virgula **(;)** para finalizar um comando e chaves **({ })** para delimitar um escopo.
- Duas formas de fazer comentário:
  - ❑ Em linha, utilizando a cerquilha (#);
  - ❑ Em blocos, utilizando = no início e **=cut** no final.

O BOM E VELHO “HELLO WORLD”

Código	Output

#!/usr/bin/perl
 	
# Comentário em linha 	
=
Comentário
em
bloco
=cut
 	
if(1){
  print "Olá mundo!\n";
}
	Olá mundo!

ALERTA! USE WARNINGS;

- É só um alerta.
- Será gerado apenas se a opção warnings estiver presente.
- O que é altamente recomendado.
- Pode ser utilizado através de um dos comandos:
  - use warnings;
  - #!/usr/bin/perl –w.

USE WARNINGS;

Código	Output
#!/usr/bin/perl #Sem use warnings; my $x; print $x; print “Use Warnings!!”;	Use Warnings!!
#!/usr/bin/perl #Com use warnings; use warnings; my $x; print $x; print “Eu lhe avisei...”;	Use of uninitialized value $x in print at perl.pl line 6. Eu lhe avisei...

ALGUMAS PALAVRAS-CHAVE

Coluna 1	Coluna 2	Coluna 3	Coluna 4
__DATA__	else	lock	qw
__END__	elsif	lt	qx
__FILE__	eq	m	s
__LINE__	exp	ne	sub
__PACKAGE__	for	no	tr
and	foreach	or	unless
cmp	ge	package	until
continue	gt	q	while
CORE	if	qq	xor
do	le	qr	y

COMPILADA OU INTERPRETADA?

- Perl é compilada ou interpretada?
  - ❑ Compilador que “pensa” ser um interpretador;
  - ❑ Código executável apenas é copiado pra memória, sendo depois utilizado;
  - ❑ Execução eficiente do código compilado combinado com o desenvolvimento rápido de uma linguagem interpretada;
  - ❑ Necessidade de compilar cada vez que é executado;
  - ❑ Conclusão: Perl é compilada "nos bastidores" para uma execução rápida, mas você pode tratá-la como se fosse uma linguagem interpretada.

PARTE 02

AMARRAÇÕES

ESCOPO DE AMARRAÇÃO

- Variáveis por padrão tem escopo global.
- Uma variável global criada dentro de um escopo tem visualização por todo o programa.

Código	Output

$fora = “Fora\n”;
 	
if(1){
  $dentro = $fora = “Dentro\n”;
}
 	
print $fora . $dentro;
	Dentro
	Dentro

MY $GOD

- Para criar uma variável visível somente no escopo em que ela foi definida, basta utilizar o operador my: my $god = 5;

Código	Output

$god = “Dio”;
 	
if(1){
  my $god = “Kami”;
}
 	
print $god;
	Dio

PARTE 03

VALORES E TIPO DE DADOS

TIPOS DE DADOS

- Tipagem fraca.
- Basicamente dois tipos:
  - ❑ Strings: podem ser, ou não, interpoláveis;
  - ❑ Numéricos: Armazenados como inteiros ou ponto flutuante de precisão dupla. Podendo ser especificados como:
    - Integer;
    - Negative Integer;
    - Ponto flutuante;
    - Notação científica;
    - Hexadecimal;
    - Octal.

TIPOS DE DADOS

- PERL não possui tipos booleanos, condições devem ser compostas por escalares.
- A expressão será considerada falsa apenas se composta por:
  - undef;
  - escalar 0;
  - string “”;
  - string "0“.

PARTE 04

VARIÁVEIS E CONSTANTES

VARIÁVEIS

- Basicamente três tipos:
  - ❑ Escalar;
  - ❑ Array;
  - ❑ Hash.
- Precedidas por um símbolo chamado selo: $ (escalar), @ (array) e % (hash).
- Undef equivale ao null.

ESCALAR

- Variáveis simples
- Representa um valor único, podendo ser um número (inteiro ou ponto flutuante) ou uma string.

Código	Output
$str = “Paulo”; $numInt = 8; $numFloat = 7.5; print “$str tirou $numFloat na P1 e $numInt na P2”;	Paulo tirou 7.5 na P1 e 8 na P2

EXEMPLOS DE ESCALAR

Código	Output
$negative = -300; $scinote= -1.2E-23; $octal = 0377; $hexa = 0xff; print “Negativo: $negative\n”; print “Notação cientifica: $scinote\n”; print “Octal: $octal\n”; print “Hexa: $hexa\n”;	Negativo: -300 Notação cientifica: -1.2e-23 Octal: 255 Hexa: 255

MULTI-LINHAS

- Strings multi-linhas:

Código	Output
$str = ‘Várias linhas na string.’; print $str; print << ‘FIM’; \nAssim também vai. FIM	Várias Linhas na string. Assim também vai.

ARRAY

- Lista de valores, acessíveis a partir de um índice.
- Tamanho do array: @nomedoarray;
- Pode se usar o range operator (..) na criação de arrays.

Código	Output
@array = ('cars', '2', '3.5'); $tam = @array; print "I have $array[1] $array[0]\n"; print "Size: $tam"; @alfa = (a..z); @num = (1..10); print @alfa; print @num;	I have 2 cars Size: 2 abcdefghijklmnopqrstuvwxyz 12345678910

FUNÇÕES DE ARRAY

- Funções importantes de arrays:
  - ❑ Push: Insere elemento no final do array;
  - ❑ Pop: Retira e retorna o último elemento do array;
  - ❑ Shift: Retira e retorna o primeiro elemento;
  - ❑ Unshift: Insere elemento no início da lista;
  - ❑ Sort: Retorna o array ordenado;
  - ❑ Split: Transforma strings em arrays a partir de um delimitador;
  - ❑ Join: Transforma arrays em strings;
  - ❑ Reverse: Retorna o array invertido.

EXEMPLOS DE ARRAY

Código	Output

@a = (2,8,3);
print "1. \@a = @a\n";
push(@a, 9);
print "2. \@a = @a\n";
unshift(@a, 1);
print "3. \@a = @a\n";
pop(@a);
print "4. \@a = @a\n";
shift(@a);
print "5. \@a = @a\n";
@a = sort(@a);
print "6. \@a = @a\n";
	1. @a = 2 8 3
	2. @a = 2 8 3 9
	3. @a = 1 2 8 3 9
	4. @a = 1 2 8 3
	5. @a = 2 8 3
	6. @a = 2 3 8

EXEMPLOS DE ARRAY

Código	Output
$ppl = "Larry,David,Roger,Ken,Michael"; @ppl = split(',', $ppl); print "$ppl[4]\n"; $str = join('-', @ppl); print "$str\n"; @ppl = reverse(@ppl); print "@ppl";	Michael Larry-David-Roger-Ken-Michael Michael Ken Roger David Larry

HASH

- Grupo não ordenado de pares chave-valor, onde as chaves são strings únicas e os valores são escalares.

Código	Output
%age = ('João' => 45, 'Maria' => 30, 'José' => 40); #%age = ('João', 45, 'Maria', 30, 'José', 40); #%age = (-Joao => 45, -Maria => 30, -Jose => 40); print %age; print "$age{'João'}\n"; $age{"Caio"} = 25; print %age;	José40Maria30João45 45 Caio25José40Maria30João45

FUNÇÕES DE HASH

- Funções importantes de hashes:
  - ❑ Keys: Retorna um array com as chaves;
  - ❑ Values: Retorna um array com os valores;
  - ❑ Exists: Verifica se uma chave existe;
  - ❑ Delete: Remove um elemento da hash.

EXEMPLOS DE HASH

Código	Output

%age = ('João', 45, 'Maria', 30, 'José', 40);
 	
@keys = keys %age;
print "@keys\n";
 	
@values = values %age;
print "@values\n";
 	
if(exists $age{'João'}){
  delete $age{'João'};
}
 	
print %age;
	José João Maria
	40 45 30
	José40Maria30

VARIÁVEIS ESPECIAIS

- Algumas variáveis especiais:
  - $_: Variável padrão;
  - @_: Argumentos para sub-rotinas;
  - @ARGV: Argumentos passados por linha de comando, assim como em C;
  - %ENV: Variáveis de ambientes.
- Outras variáveis especiais podem ser encontradas na documentação da linguagem.

CONSTANTES

- Constantes podem ser declaradas utilizando o pragma constant.

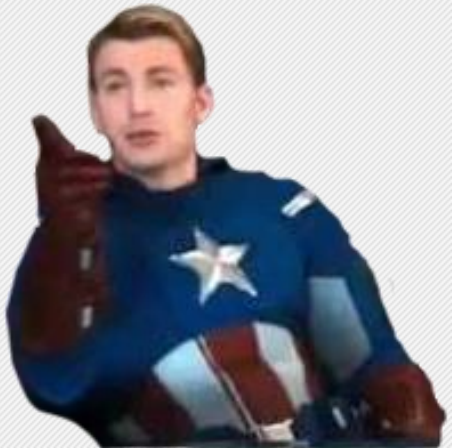
Código	Output
use constant PI => 3.14; $r = 3; $circ = PI * ($r**2); print PI; print "\ncirc = $circ\n";	3.14 circ = 28.26

COLETOR DE LIXO

- Perl possui coletor de lixo.
- Estratégia de contagem de referência.
- Contadores de referência são utilizados para saber quando um bloco deve ser coletado. Quando o contador chega a 0, o bloco é desalocado.

REFERÊNCIAS

- Perl possui manipulação de ponteiros (referências).
- Para referenciar algum tipo é preciso adicionar o caractere \ antecedente à referência.
- Para desreferenciar um tipo, utiliza-se o caractere $ antecedente à referência.
- Escalar, array e hash podem ser referenciados.
- Como array e hash são lineares, referenciar um array é uma saída para se obter um array de arrays.



ESCALAR

- Referências em escalar:

Código	Output

my $texto = “maça”;
my $inteiro = 10;
my $refTexto = \$texto; # ou \“maça”;
my $refInt = \$inteiro; # ou \10;
 	
print “Referência: $refTexto\n”;
print “Escalar: $$refTexto\n”;
	Referência: SCALAR(0x159a7c0)
	Escalar: maça

ARRAY

- Referências em array:

Código	Output

my @vetor = (1,2,3);
my $ref = \@vetor;
${$ref}[1] = 10; # ou $ref->[1] = 10.
 	
print “Referência: $ref\n”;
 	
foreach my $item (@{$ref}){
  print “$item”;
}
	Referência: ARRAY(0x1db67c0)
	1 10 3

HASH

- Referências em hash:

Código	Output
my %hash = (“arroz” => “grao”, “alface” => “salada”); my $ref = \%hash; #${$ref}{‘chave’} ou $ref->{‘chave’} print "${$ref}{'arroz'}\n";	grao

SERIALIZAÇÃO

- Há várias formas de fazer serialização em perl.
- Data::Dumper, Storable e FreezeThaw são os módulos mais populares, porém só são “compreendidas” por perl.
- Produzem um escalar como saída que podem ser armazenados em um arquivo, banco de dados, entre outros.
- Se for necessário comunicação com aplicações escritas em outras linguagens, recomenda-se serialização em JSON.

PARTE 05

EXPRESSÕES E COMANDOS

OPERADORES

- Operadores aritméticos:

Operador	Descrição	Exemplo ($a = 10, %b = 20)
+	Adição	$a + $b = 30.
-	Subtração	$a - $b = -10.
*	Multiplicação	$a * $b = 200.
/	Divisão	$b / $a = 2.
%	Resto da divisão	$b % $a = 0.
**	Exponencial	$a ** $b = 10^{20}.
++	Incremento	$a++ retorna 11.
--	Decremento	$b-- retorna 19.

OPERADORES

- Operadores de atribuição:

Operador	Exemplo
=	$a = 10; #$a recebe 10.
+=	$a += $b; # Equivalente a $a = $a + $b.
-=	$a -= $b; # Equivalente a $a = $a - $b.
*=	$a *= $b; # Equivalente a $a = $a * $b.
/=	$a /= $b; # Equivalente a $a = $a / $b.
%=	$a %= $b; # Equivalente a $a = $a % $b.
**=	$a **= $b; # Equivalente a $a = $a ** $b.

OPERADORES

- Operadores relacionados a strings:

Operador	Descrição	Exemplo
.	Concatenação de strings.	$a = ‘ab’; $b = ‘cd’; $c = $a.$b; #$c = ‘abcd’.
x	Repetição de determinada string.	$a x 3 #abcabcabc.
.=	String recebe a concatenação dela com outra string.	$a .= $b; # Equivalente a $a = $a.$b.
q{ }	Equivalente a aspas simples.	$a = q{abc}; # Equivalente a $a = ‘abc’.
qq{ }	Equivalente a aspas duplas.	$a = qq{abc}; # Equivalente a $a = “abc”.

OPERADORES

- Operadores lógicos:

Operador	Descrição
&	“E” binário.
|	“Ou” binário.
^	“Ou” exclusivo.
~	Complemento de 1.
<<	Shift à esquerda.
>>	Shift à direita.

- Operadores bit-a-bit:

Operador	Descrição
and / &&	“E” lógico.
or / ||	“Ou” lógico.
not / !	Negação lógica.

OPERADORES

- Operadores de comparação:

Numérico	String	Descrição	Exemplo ($a = 10, $b = 20)
==	eq	Igual	$a == $b é falso.
!=	ne	Diferente	$a != $b é verdadeiro.
<	lt	Menor	$a < $b é verdadeiro.
>	gt	Maior	$a > $b é falso.
<=	le	Menor ou igual	$a <= $b é verdadeiro.
>=	ge	Maior ou igual	$a >= $b é falso.
<=>	cmp	Comparação	$a <=> $b retorna -1.

OPERADORES

- Operador range (..):

Código	Output
@a = (1..5); print @a;	12345

- Operador ternário:

Código	Output
$a = 5; $b = 10; $c = ($a>$b) ? "Maior" : "Menor"; print $c;	Menor

CONDICIONAIS – IF, ELSE E ELSIF

- if, else, elsif, unless.
- Caso seja preciso avaliar mais de uma condição o elsif pode ser utilizado.

Código	Output

my $var = 1;
if ($var == 0){
  print '$var = 0';
} elsif ($var == 1){
  print '$var = 1';
} else {
  print '$var = SDS (Só Deus Sabe)';
}
	$var = 1

UNLESS

- O comando unless equivale ao “não if” e pode ser usado com a mesma sintaxe do if.

Código	Output
my $idade = 20; print “Entrada ”; print “não ” unless $idade >= 18; print “permitida.\n”;	Entrada permitida.
my $idade = 17; print “Entrada ”; print “não ” unless $idade >= 18; print “permitida.\n”;	Entrada não permitida.

ITERADORES

- Operadores:
  - while;
  - do-while;
  - for;
  - foreach;
  - until.
- Os operadores while, do-while e for possuem a mesma sintaxe utilizada nas linguagens C, C++ e JAVA.

WHILE

Código	Output

my $contagem = 1;
print “Conte de 1 à 5 enquanto nos escondemos!\n”;
 	
while($contagem <= 5){
  print $contagem . “\n”;
  $contagem++;
}
 	
print “Aí vou eu!\n”;
	Conte de 1 à 5 enquanto nos escondemos!
	1
	2
	3
	4
	5
	Aí vou eu!

DO-WHILE

- Este mesmo código num while não iria funcionar da mesma forma.

Código	Output

$contagem = 1;
print “Conte de 1 à 5 enquanto nos escondemos!\n”;
 	
do{
  print $contagem . “\n”;
  $contagem++;
}while($contagem == 0);
 	
print “Aí vou eu!\n”;
print “Não vale! Você não contou até 5!\n”;
	Conte de 1 à 5 enquanto nos escondemos!
	1
	Aí vou eu!
	Não vale! Você não contou até 5!

FOR

Código	Output

print “Imprimindo os valores do array.\n”;
my $array = (2,4,6,8,10);
 	
for(@array){
  print $_ . “\n”;
}
	Imprimindo os valores do array.
	2
	4
	6
	8
	10

FOREACH

- Seguindo o mesmo exemplo anterior, mas agora com foreach:

Código	Output

print “Imprimindo os valores do array.\n”;
my $array = (2,4,6,8,10);
 	
foreach my $valor (@array){
  print $valor . “\n”;
}
	Imprimindo os valores do array.
	2
	4
	6
	8
	10

UNTIL

- Pode realizar a iteração em um comando específico:

Código	Output
print “Conte apenas de 1 à 5!\n”; my $contador = 0; print “$contador ” until $contador++ > 5; print “Ei! Eu pedi para contar até 5 apenas!!\n”;	Conte apenas de 1 à 5! 1 2 3 4 5 6 Ei! Eu pedi para contar até 5 apenas!!

I/O

Código	Output
print “Digite seu nome: ”; $a = <STDIN>; #$a = <>; chomp($a); #remove o \n; print “Nome: $a”;	Digite seu nome: Daniel Nome: Daniel

EXPRESSÕES REGULARES

- Perl possui ferramentas de expressões regulares nativamente.
- Usadas para checar o formato de uma string, formatá-la, substituir dados, capturar dados, entre outros.
- Geralmente designada entre barras (/), e seu reconhecimento pode ser feito através do operador lógico “=~” ou “!~”.
- Três operadores:
  - m//: Correspondência;
  - s///: Substituição;
  - tr///: Transliteração.
- Cada um tem modificares (parâmetro).

EXPRESSÕES REGULARES

- Correspondência:

Código	Output

$str = “Larry Wall and Perl”;
 	
if($str =~ m/wall/i){
  print “Antes: $`\n”;
  print “Encontrado: $&\n”;
  print “Depois: $’\n”;
}else{
  print “Não encontrado.”;
}
	Antes: Larry
	Encontrado: Wall
	Depois: and Perl

Modificador	Descrição
i	Case insensitive.
m	Parar em caso de \n.
o	Avalia a expressão somente uma vez.
s	Não parar em \n.
x	Permite espaço na expressão.
g	Todas as correspondências.

EXPRESSÕES REGULARES

- Substituição:

Código	Output
$str = 'My cats are not friendly with other cats'; print "$str\n\n"; $str =~ s/cats/dogs/g; print $str;	My cats are not friendly with other cats My dogs are not friendly with other dogs

Modificador	Descrição
i/m/o/s/x/g	Mesmo de correspondência.
e	Execução de comandos.

EXPRESSÕES REGULARES

- Transliteração:

Código	Output
$str = 'abcABBC123'; $str =~ tr/a-z//d; print "$str\n"; $str =~ tr/A-Z//cd; print "$str\n"; $str =~ tr/A-Z//s; print $str;	ABBC123 ABBC ABC

Modificador	Descrição
c	Complemento.
d	Exclui caracteres encontrados mas não substituídos.
s	Elimina repetições.

DESVIOS INCONDICIONAIS

- Escape
  - ❑ last: Sai do ciclo mais interno;
  - ❑ next: Salta para a próxima iteração do ciclo mais interno;
  - ❑ redo: Salta para o ínicio do ciclo mais interno, sem reavaliar a condição;
  - ❑ return EXPRESSÃO: sai da subrotina retornando o valor da EXPRESSÃO.

- Desvio irrestrito
  - ❑ goto: Avança para a próxima instrução determinada pela marca.

LAST

Código	Output

my @vetor = (1,2,3,4,5);
 	
for my $i (@vetor) {
 print "$i ";
 	
  if($i == 2) {
  last;
  }
}
	1 2

NEXT

Código	Output

my @vetor = (1,2,3,4,5);
 	
for my $i (@vetor) {
  if($i == 2) {
  next;
  }
 	
 print "$i ";
}
	1 3 4 5

REDO

- Este desvio pode acabar em loop infinito, caso não tenha os devidos cuidados.

Código	Output

my @vetor = (1,2,3,4,5);
 	
FLAG: for my $i (@vetor) {
 print "$i ";
 	
 if($i == 2) {
 $i = "teste"; # Cuidado! Possibilidade de loop infinito.
 redo FLAG;
 }
}
	1 2 teste 3 4 5

GOTO

Código	Output
my $i = 10; goto LABEL; $i = 20; LABEL: print "$i";	10

PARTE 06

MODULARIZAÇÃO

SUB-ROTINAS

- As sub-rotinas (funções) precisam da palavra-chave *sub* para serem declaradas.

Código	Output

sub classe{
  print “classe!\n”;
}
 	
print “Olá ”;
classe;
	Olá classe!

PASSAGEM DE ARGUMENTOS

- Sub-rotinas permitem uma quantidade variável de parâmetros.

Código	Output

sub imprimeParametros{
  for (@_){
  print “$_ ”;
  }
}
 	
my @array = (“primeiro”, “segundo”, 75);
 	
imprimeParametros(10, “casa”, @array, $array[2]);
	10 casa primeiro segundo 75 75

PASSAGEM É POR REFERÊNCIA E POSICIONAL

- A passagem dos parâmetros é feita por referências e ao mesmo tempo é posicional.
- É possível modificar o conteúdo dos parâmetros permanentemente dentro das sub-rotinas.

Código	Output

sub altera{
  $_[0] = 5;
}
my @array = (1, 2, “Perl”);
altera(@array);
print “@array”;
	5 2 Perl

RETURN

- O retorno das funções se assemelham à C, C++ e JAVA.
- Porém não é necessário dizer qual será o retorno da função.

Código	Output

sub soma{
  my($a, $b) = @_;
  return($a, $b, $a + $b);
}
 	
$val = soma(5,10);
print (“Valor da soma: $val\n");
 	
@val = soma(5,10);
print (“Valores e a soma: @val\n");
	Valor da soma: 15
	Valores e a soma: 5 10 15

ORIENTAÇÃO A OBJETOS

- Perl permite programação orientada a objetos.
- Classes são criadas através de pacotes (.pm).
- Objetos são estruturas de dados associadas a uma classe, utilizando a função *bless*.
- A função *bless* retorna uma referência para o pacote, que se torna um objeto.
- O construtor pode receber qualquer nome, porém convencionou-se chamá-lo de new.
- isa() é um método da classe UNIVERSAL e todas as classes são subclasses de UNIVERSAL, logo todo objeto possui o método isa().

ORIENTAÇÃO A OBJETOS

```perl
#Pessoa.pm
package Pessoa;

sub new{
    my $class = shift;
    my $self = {
        nome => shift,
        idade => shift
    };
    bless($self, $class);
    return $self;
}

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

```perl
sub getIdade{
    my $self = shift;
    return $self->{idade};
}

sub setNome{
    my ($self, $nome)= @_;
    $self->{nome} = $nome;
    return $self;
}

sub setIdade{
    my ($self, $idade)= @_;
    $self->{idade} = $idade;
    return $self;
}

1;
```

ORIENTAÇÃO A OBJETOS

- Além de herança simples também é possível fazer herança múltipla.
- @ISA: Array contendo as classes herdadas.
- Comando SUPER para utilizar métodos da super classe.

```perl
#Aluno.pm
package Aluno;

use Pessoa; # Herdando de Pessoa

@ISA = qw(Pessoa); #Aluno é uma pessoa

sub new{
    my $class = shift;
    my $self = {
        nome => shift,
        idade => shift,
        cr => shift
    };
    bless($self, $class);
    return $self;
}

sub getCR{
    my $self = shift;
    return $self->{cr};
}

sub getIdade{
    my $self = shift;
    $self->SUPER::getIdade();
}

1;
```

ORIENTAÇÃO A OBJETOS

```perl
use Pessoa;
use Aluno;

my $p1 = Pessoa->new('Joao', 21);
my $a1 = Aluno->new('Jose', 23, 7.5);

$a = $p1->getNome();

print "$a\n\n"; # Joao

$a1->setNome('Junior');
$cr = $a1->getCR();
$nomeA1 = $a1->getNome();

print "Aluno: $nomeA1\nCR: $cr\n";
# Aluno: Junior
# CR: 7.5
```

ARQUIVOS

- Função open para abrir o arquivo e close para fechar.
- A função open recebe um escalar para navegar no arquivo, modo como o arquivo será aberto e o caminho do arquivo.
- Modos para lidar com arquivos mais utilizados:
  - <: Leitura;
  - >: Escrita com sobrescrita;
  - +<: Leitura e escrita;
  - >>: Escrita no final do arquivo (append).

ARQUIVOS

- Escrevendo em um arquivo:

Código

```
my $arq = "teste.txt";
my $a = 10;

open(my $fh, '>', $arq) or die "Algo deu errado";
print $fh "Escrevendo alguma coisa\n";
print $fh "Escalar $a";
close $fh;
```

ARQUIVOS

- Lendo um arquivo:

Código	Output

my $arq = "teste.txt";
 	
open($fh, '<', $arq) or die "Algo deu errado";
while(<$fh>){
print "$_";
}
close $fh;
	Escrevendo alguma coisa
	Escalar 10

PARTE 07

POLIMORFISMO

AD-HOC

- Coerção:

Código	Output
$num = 3; $str = “5”; print $num + $str; # Converte $str para inteiro. print “\n” . $num . $str; # Converte $num para string.	8 35

AD-HOC

- Sobrecarga:

Código

```
# Sobrescrita de operadores utilizando ‘overload’

package Teste

use overload
      “+” => "newAdd",
      “-” => /newSub;
```

UNIVERSAL

- Paramétrico:
  - A linguagem faz isso automaticamente, já que não é necessário dizer quais serão os tipos de parâmetros e o tipo de retorno de uma função.

Código	Output

sub soma{
  return $_[0] + $_[1];
}
 	
my $str = “3”;
my $num = 2;
print soma($num, $num) . “\n”;
print soma($str, $num) . “\n”;
print soma($str, $str) . “\n”;
	4
	5
	6

UNIVERSAL

- Inclusão:

Código

```
package Aluno

use Pessoa
@ISA = qw(Pessoa);

sub new {
...
}
```

PARTE 08
EXCEÇÕES

EXCEÇÕES

- Perl não possui nativamente mecanismos de tratamento de exceções, sendo necessário um controle por parte do programador, assim como em C.
- warn(): Indica um erro e imprime uma mensagem na saída de erro padrão, STDERR.
- die(): Imprime mensagem em STDERR, porém encerra o programa.

```
$a = $b/$c;
if($c == 0){
    warn('Denominador inválido');
}
```

```
open (my $fh, '>', "texto.txt") or die "Algum erro";
```

EXCEÇÕES

- Funções alternativas para tratamento de exceções:
  - ❑ carp(): Equivalente ao warn();
  - ❑ cluck(): Imprime mensagem + stack trace;
  - ❑ croak(): Semelhante ao die() porém reporta quem chamou a função;
  - ❑ confess(): Semelhante ao croak() porém com back trace.

PARTE 9

CONCORRÊNCIA

FORK

- A linguagem possui suporte ao fork.
- O ID do processo atual fica armazenada na variável $$.
- A função defined retorna verdadeiro se $pid não for undef.

Código	Output

if(($pid = fork) != 0){
  wait();
  print "$$: Nãããããão!\n“;
}else{
  print "$$: Luke, eu sou seu pai!\n“;
}
die "fork falhou: " unless defined $pid;
	8564: Luke, eu sou seu pai!
	8563: Nãããããão!

USE THREADS;

- É possível criar e manipular threads em PERL.
- É necessário incluir o módulo threads ao cabeçalho do arquivo, na forma:
  - use threads.
- Principais funções:
  - create(): Cria uma nova thread;
  - join(): Espera a thread finalizar, libera a memória relacionada e retorna o conteúdo de retorno da thread, caso ela tenha feito;
  - detach(): Se não há necessidade do retorno da thread, nem mesmo da finalização da thread é possível utilizar a sub-rotina que liberará a memória da thread, quando esta finalizar.

USO DE THREADS

Código	Output

use threads;
 	
sub funcao{
  foreach my $item (@_){
  print "$item ";
  }
  return “\nThread finalizada!”;
}
 	
my $thr = threads->create(\&funcao, 42, "Hello", 3.14);
print $thr->join;
	42 Hello 3.14
	Thread finalizada!

THREAD::SEMAPHORE

- Para controlar condições de corrida é possível fazer o uso de semáforos.
- Utilizar semáforos é necessário incluir ao cabeçalho do arquivo:
  - use threads;
  - use Thread::Semaphore.
- Existem alguns métodos importantes para controlar o semáforo:
  - up(): Incrementa a variável de controle;
  - down(): Decrementa a variável de controle;
  - down_nb(): Decrementa o semáforo apenas se obtiver sucesso imediato;
  - down_force(): Decrementa o semáforo mesmo se o contador estiver abaixo de 0;
  - down_timed(TIMEOUT): Decrementa o semáforo num tempo determinado em segundos.

EXEMPLO DE SEMAPHORE

Parte 1

```
use threads;
use Thread::Semaphore;
my $sem = Thread::Semaphore->new();
my $posRub = 1;
my $posSch = 1;
sub func{
    my $corredor = shift;
    my $i = 0;
    while ($i < 10){
        if($corredor == 1){
            $sem->down;
            print "Rub: ";
            for (my $rub = 0; $rub < $posRub; $rub++){
                print "-";
            }
            print ">\n";
            $posRub++;
            sleep(1);
            $sem->up;
```

Parte 2

```
        }else{
            $sem->down;
            print "Sch: ";
            for (my $sch = 0; $sch < $posSch; $sch++){
                print "-";
            }
            print ">\n";
            $posSch++;
            sleep(1);
            $sem->up;
        }
        $i++;
    }
}
my $rub = threads->new(\&func, 1);
my $sch = threads->new(\&func, 2);
$rub->join;
$sch->join;
```

OUTPUT

Output	Sch venceu!
Rub: -> Sch: -> Rub: --> Sch: --> Sch: ---> Rub: ---> Sch: ----> Sch: -----> Sch: ------> Rub: ----> Sch: -------> Sch: --------> Rub: -----> Sch: ---------->	Rub: ------> Sch: ----------> #Venceu! Rub: -------> Rub: --------> Rub: ---------> Rub: ---------->

PARTE EXTRA
PERL 6

PERL 6

- PERL 6 está sendo desenvolvida por uma equipe dedicada e por voluntários.
  - ❑ *"You can help too. The only requirement is that you know how to be nice to all kinds of people (and butterflies)."*

- Disponível em dezembro de 2015 com a versão do compilador RAKUDO PERL 6.

Perl 6

PERL 5 VS PERL 6

- Existem algumas diferenças de sintaxe entre PERL 5 e PERL 6.
- Em PERL 6 é obrigatório o uso do espaço após do uso de uma palavra-chave da linguagem.

Perl 5	Perl 6
my($alfa, $beta); if($alfa < 0) { ... }	my ($alpha, $beta); if ($alfa < 0) { ... } if $alfa < 0 { ... }

INDEXAÇÃO

- A indexação de arrays e hashes não utilizam mais o selo $.

Perl 5	Perl 6
my %calorias = (‘maça’ => 10); print $calorias{"maça"}; my @vetor = (0, 1, 2); print $vetor[2];	my %calorias = (‘maça’ => 10) print %calorias{"maça"}; # % ao invés de $. my @vetor = (0, 1, 2); print @vetor[2]; # @ ao invés de $.

REFERENCIAÇÃO

Perl 5	Perl 6

my $aref = \@aaa;
 	
my $href = \%hhh;
 	
my $sref = \&foo;
	my $aref = item(@aaa);
	
	my $href = item(%hhh);
	
	my $sref = &foo;

DESREFERENCIAÇÃO

- Em PERL 6 não é mais utilizado a seta ‘->’, mas apenas o ponto.

Perl 5	Perl 6
print $arrayref->[7]; print $hashref->{'chave'}; print $subref->($foo, $bar); print ${$scalar_ref};	print $arrayref.[7]; print $hashref.{'fire bad’}; print $subref.($foo, $bar); print $($scalar_ref); # parenteses no lugar das chaves.

PARTE 10

AVALIAÇÃO DA LINGUAGEM

AVALIAÇÃO DA LINGUAGEM

Critérios Gerais	C	JAVA	PERL 5
Aplicabilidade	Sim	Parcial	Parcial
Confiabilidade	Não	Sim	Não
Aprendizado	Não	Não	Sim
Eficiência	Sim	Parcial	Parcial
Portabilidade	Não	Sim	Sim
Método de Projeto	Estruturado	OO	Estruturado, OO e Funcional
Evolutibilidade	Não	Sim	Parcial
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Sim

AVALIAÇÃO DA LINGUAGEM

Critérios Específicos	C	JAVA	PERL 5
Escopo	Sim	Sim	Parcial
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Linguagem	Linguagem
Persistência dos dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Biblioteca de sub-rotinas, serialização.
Passagem de parâmetros	Lista variável e por valor	Lista variável, por valor e por cópia de referência.	Lista variável por referência.

AVALIAÇÃO DA LINGUAGEM

Critérios Específicos	C	JAVA	PERL 5
Encapsulamento e proteção	Parcial	Sim	Não
Sistema de tipos	Não	Sim	Parcial
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Sim	Não
Concorrência	Não (biblioteca de funções)	Sim	Sim

REFERÊNCIAS BIBLIOGRÁFICAS

- https://perldoc.perl.org
- https://perlmaven.com
- https://br.perlmaven.com/
- https://br.perlmaven.com/uso-de-valor-nao-inicializado
- https://qntm.org/files/perl/perl.html
- https://www.perl.com/
- http://www.troubleshooters.com/codecorn/littperl/perlsub.htm
- https://sites.google.com/site/mbusigin/articles-papers/a-journeyman-s-guide-to-concurrency-in-perl
- https://www.perl.com/pub/2003/07/22/overloading.html

REFERÊNCIAS BIBLIOGRÁFICAS

- https://www.tutorialspoint.com/perl/
- https://github.com/Perl/perl5
- http://www.w3ii.com/en-US/perl/default.html
- https://giovannireisnunes.wordpress.com/2015/06/26/um-basico-de-orientacao-a-objetos-em-perl/
- http://www.perlmonks.org/?node_id=814784
- http://www.perlmonks.org/?node_id=51097
- http://montegasppa.blogspot.com.br/2006/09/orientao-objetos-em-perl.html
- https://br.perlmaven.com/escrevendo-em-arquivos-com-perl
- https://www.devmedia.com.br/perl-para-que/12787

REFERÊNCIAS BIBLIOGRÁFICAS

- https://metacpan.org/pod/UNIVERSAL
- https://perldoc.perl.org/perlfaq7.html
- https://stackoverflow.com/questions/5376559/is-perl-a-compiled-or-an-interpreted-programming-language
- https://perlmaven.com/hashbang