## Perguntas:

1. Como instalar e começar a usar?
2. Quais são os processos de tradução utilizados?
3. Em que paradigmas se encaixa?
4. Os nomes são sensíveis à capitalização?
5. Quais os caracteres aceitos em um nome?
6. Existe alguma restrição de tamanho para nomes?
7. Como é a questão das palavras-chave x palavras reservadas?
8. É possível definir uma variável anônima? Mostre exemplo.
9. A vinculação de tipos (tipagem) é estática ou dinâmica?
10. Quais categorias de variável (Sebesta, Seção 5.4.3) apresenta? Mostre exemplos.
11. Permite ocultamento de nomes (variáveis) em blocos aninhados? Mostre exemplo.
12. Permite definir constantes? Vinculação estática ou dinâmica? Mostre exemplos.
13. Quais os tipos oferecidos? Mostre exemplos de definição de variáveis de cada tipo.
14. Existe o tipo função? São cidadãs de primeira classe? Mostre exemplo.
15. Possui ponteiros ou referências? Permite aritmética de ponteiros?
16. Oferece coletor de lixo? Se sim, qual a técnica utilizada?
17. É possível quebrar seu sistema de tipos (forçar erro de tipo)? Mostre exemplo.
18. Quais os operadores oferecidos? Mostre exemplo de uso de cada operador.
19. Permite sobrecarga de operadores? Mostre exemplo.
20. Quais operadores funcionam com avaliação em curto-circuito?
21. O operador de atribuição funciona como uma expressão?
22. Quais as estruturas de controle (seleção, iteração) oferecidas? Mostre exemplos.
23. Quais sentenças de desvio incondicional oferecidas? Mostre exemplos.
24. Quais os métodos de passagem de parâmetros oferecidos? Mostre exemplos.
25. Permite sobrecarga de subprogramas? Mostre exemplo.
26. Permite subprogramas genéricos? Mostre exemplo.
27. Como é o suporte para definição de Tipos Abstratos de Dados? Mostre exemplo.
28. Permite TADs genéricos/parametrizáveis? Mostre exemplo.
29. Quais as construções de encapsulamento oferecidas? Mostre exemplos.
30. Quais tipos de polimorfismo suporta? Mostre exemplos.
31. Permite herança de tipos? Herança múltipla? Mostre exemplo.
32. Permite sobrescrita de subprogramas? Mostre exemplo.
33. Permite a definição de subprogramas abstratos? Mostre exemplo.
34. Oferece mecanismo de controle de exceções? Mostre exemplo.
35. Possui hierarquia de exceções controlada, como em Java? Qual a raiz?
36. Categoriza as exceções em checadas e não-checadas? Como?
37. Obriga a declaração de exceções lançadas para fora de um subprograma?
38. Como você avalia a LP usando os critérios do Sebesta (Seção 1.3)?
39. Como você avalia a LP usando os critérios do Varejão?

## Respostas:

1. Supondo um ambiente Unix/Linux com um compilador de C, como o gcc, previamente instalado, para instalar a linguagem Perl basta copiar o seguinte comando num terminal: "curl -L http://xrl.us/installperlnix | bash". Após executar o comando, o usuário já pode rodar programas em Perl. Por exemplo, para rodar um programa chamado "exemplo.pl", digita-se o comando "perl exemplo.pl" no terminal, supondo que o usuário esteja com o terminal aberto no diretório onde se encontra o programa. Além de instalar Perl, recomenda-se instalar a ferramenta "cpanm", com o seguinte comando: "cpan App::cpanminus". Esta ferramenta simplifica a instalação de módulos que, por sua vez, facilitam o trabalho do programador pois providenciam várias funcionalidades que não estão disponíveis na instalação original da linguagem Perl.

2. Perl utiliza um método de tradução híbrido. O código em Perl é compilado em bytecode pelo interpretador, e depois é executado.

3. Perl se encaixa nos paradigmas imperativo, orientado a objetos e funcional, portanto, é uma LP multiparadigma.

4. Os nomes em Perl são sensíveis à capitalização.

5. Em Perl os nomes precisam começar com uma letra ou underscore, e podem conter letras, dígitos, underscores ou a sequência especial "::" ou "'". Nesse caso, a parte antes da última sequência de "::" ou "'" é interpretada como um package (namespace). Além disso, variáveis em Perl possuem três tipos: escalar, vetor ou hash. Para cada tipo, há um caracter especial que designa o tipo da variável e que precisa vir antes do primeiro caracter escolhido pelo programador. Esses caracteres são "$" para escalares, "@" para vetores e "%" para hashes.

6. Um nome em Perl não pode conter mais de 251 caracteres.

7. Recomenda-se não usar palavras-chave em Perl como "length", "not" e "my", mas Perl permite que o programador use-as ao seu propósito, caso queira. Como variáveis em Perl são identificadas por caracteres especiais ("$", "@" e "%"), isso torna um pouco menos confuso o uso de palavras-chave. Em Perl há também a opção de sobrecarregar (overload) palavras reservadas para funções ou operadores.

8. Pode-se definir uma estrutura de dados anônima em Perl e acessá-la por meio de referências. Por exemplo:

```
1 my @meals = qw( soup sandwiches pizza );
2 my $anon  = [ @meals ];
3 my $ref   = \@meals;
```

(1) cria-se um vetor

(2) um vetor anônimo que recebeu uma lista (no caso, o vetor "meals" foi transformado numa lista pelos colchetes).
(3) variável que referencia "@meals" (referências em Perl são feitas começando com "\")

No exemplo dado, $ref não tem nenhuma conexão com o vetor "@meals", ou seja, qualquer mudança feita em "@meals" não afetará $ref.

9. A vinculação de tipos em Perl 5 (a versão usada neste trabalho) é dinâmica.

10. Em Perl 5.10+

- Variáveis estáticas (por meio da palavra-chave "state"):

```perl
use 5.010;

sub exemplo_estatica {
    state $var_est = 10;    #esta linha só é executada na primeira chamada do subprograma
    $var_est = $var_est * 2;
    return $var_est;
}

say(exemplo_estatica());    #output: 20
say(exemplo_estatica());    #output: 40
```

- Variáveis dinâmicas da pilha:

```perl
use 5.010;

sub fact{
    my $x = $_[0]; #armazenar argumento passado em variavel
    if($x == 1 || $x == 0){
        return 1;
    }
    else{
        return $x * fact($x - 1);
    }
}

my $a = 5;
say fact($a);
```

- Variáveis dinâmicas do monte implícitas:

```perl
my @array = (0,1,2);
my $str   = "Perl";
```

#vetores e strings em Perl são exemplos de variáveis dinâmicas do monte implícitas

11. Perl permite ocultamento de variáveis em blocos, como mostra o exemplo a seguir:

```perl
use 5.010;

sub subprog0{
    my $x0 = 0;
    sub subprog1{
        my $x1 = 1;
        say($x0);
    }
    subprog1();
    say($x1); #nao imprime nada/erro
}

subprog0();
```

12. Perl permite a definição de constantes, cuja vinculação é estática, por meio do módulo "constant". Por exemplo:

```perl
use constant PI => 3.14159;
```

13. Em Perl existem três tipos: escalares (que podem ser inteiros, floats, strings ou referências e são definidos pelo carácter "$"), vetores (definidos pelo carácter "@") e hashes (definidos pelo carácter "%")
Exemplos:

```perl
my $animal = "Camelo";          #escalar (string)
my $num    = 1.25;              #escalar (float)
my @array  = ("Perl", 1.25, 45) #vetor com tipos distintos de escalares
my %dict   = (                  #hash com par de valores associados
    um   => 1,
    dois => 2
);
```

14. Sim, existe o tipo função, são cidadãs de primeira classe e em Perl este tipo é declarado com a palavra-chave "sub" seguido de chaves "{}".
Exemplo:

```perl
use 5.010;

my $helloWorld = sub {print "Hello World\n"};
sub sayHello{
    my ($f) = @_; #função foi passada como argumento
    $f      ->();
}
sayHello($helloWorld);
```

15. Perl usa referências e essas são declaradas com o uso do caracter "\" precedendo o nome e tipo da variável.

16. Perl não oferece coletor de lixo, mas sim um contador de referências. Quando algo não pode mais ser referenciado, a memória ocupada é retornada ao pool de memória do programa. Perl aloca uma quantidade de memória quando um programa é executado, a qual permanece sob controle do programa durante sua execução.

17. Não, Perl verifica por erros de tipo e os preveni em tempo de execução. Como Perl trata inteiros, floats e strings como escalares, há conversão automática entre estes diferentes tipos, como o exemplo a seguir ilustra:

```perl
$a=10;
$b="a";
$c=$a.$b;
print $c;        #output: 10a
```

Em Perl, isto não é uma falha do sistema de tipos, mas sim uma característica da LP.

18.

**Aritméticos:**

+ (adição) Exemplo: $a + $b
- (subtração) Exemplo: $a - $b
* (multiplicação) Exemplo: $a * $b
/ (divisão) Exemplo: $a / $b
% (módulo) Exemplo: $a % $b
** (expoente) Exemplo: $a ** $b

**Igualdade:**

== (igual a) Exemplo: $a == $b
!= (diferente de) Exemplo: $a != $b
<=> (retorna -1 se $a menor que $b, 0 se $a for igual a $b ou 1 se $a for maior que $b) Exemplo: $a <=> $b

> (maior que) Exemplo: $a > $b
< (menor que) Exemplo: $a < $b
>= (maior ou igual a) Exemplo: $a >= $b
<= (menor ou igual a) Exemplo: $a <= $b

**Equidade:**

lt (Retorna true se $a for uma string menor que $b) Exemplo: $a lt $b
gt (Retorna true se $a for uma string maior que $b) Exemplo: $a gt $b
le (Retorna true se $a for uma string menor ou igual a $b) Exemplo: $a le $b
ge (Retorna true se $a for uma string maior ou igual a $b) Exemplo: $a ge $b
eq (Retorna true se $a for uma string igual a $b) Exemplo: $a eq $b
ne (Retorna true se $a for uma string diferente de $b) Exemplo: $a ne $b
cmp (retorna -1 se a string $a for menor que a string $b, 0 se for igual, ou 1 se for maior) Exemplo: $a cmp $b

**Atribuição:**

= (atribuir valor) Exemplo: $c = $a + $b
+= (somar e atribuir ao valor) Exemplo: $b += $a
-= (subtrair e atribuir ao valor) Exemplo: $b -= $a
*= (multiplicar e atribuir ao valor) Exemplo: $b *= $a
/= (dividir e atribuir ao valor) Exemplo: $b /= $a
%= (módulo e atribuir ao valor) Exemplo: $b %= $a
**= (potenciação e atribuir ao valor) Exemplo: $b **= $a

**Bitwise:**

& (AND binário) Exemplo: $a & $b
| (OR binário) Exemplo: $a | $b
^ (XOR binário) Exemplo: $a ^ $b
~ (Complemento de 2) Exemplo: ~$a
<< (Shift binário de $a a esquerda pelo número de bits especificado por $b) Exemplo: $a << $b
>> (Shift binário de $a a direita pelo número de bits especificado por $b) Exemplo: $a >> $b

**Lógicos:**

and (AND lógico) Exemplo: $a and $b
&& (AND lógico tipo C; se o operando da esquerda for falso, o operando da direita nem é avaliado) Exemplo: $a && $b
or (OR lógico) Exemplo: $a or $b
|| (OR lógico tipo C; se o operando da esquerda for verdadeiro, o operando da direita nem é avaliado) Exemplo: $a || $b
not (NOT lógico; inverte estado lógico de uma operação) Exemplo: not($a and $b)

**Aspas:**

q{} (coloca aspas simples numa string) Exemplo: q{abcd} = 'abcd'
qq{} (coloca aspas duplas numa string) Exemplo: qq{abcd} = "abcd"

qx{} (coloca aspas invertidas numa string)  Exemplo: qq{abcd} = `abcd`

**Outros:**

. (concatena duas strings)  Exemplo: "abc"."def" = "abcdef"

x (retorna uma string contendo operando da direita repetido pelo número de vezes especificado pelo operando da direita)  Exemplo: '-' x 3 = '---'

.. (range; retorna uma lista com valores incrementados um a um desde o operando da esquerda até o operando da direita)  Exemplo: (2..5) = (2,3,4,5)

++ (incrementar valor em um)  Exemplo: $a++

-- (decrementar valor em um)  Exemplo: $a--

-> (acessar variável ou método de objeto) Exemplo: $objeto->$a

19. Perl permite sobrecarga de operadores, como no seguinte exemplo:

```perl
use 5.010;

package Number;

use overload
    "-" => \&minus;

sub minus{
    my ($val1, $val2) = @_; #armazena os argumentos passados à função.
    my $sum = $val1 + $val2;
    return $sum;
}

package main;
$x = Number->new(11);
$y = 6 - $x;
```

20. Os operadores “ &&, and, ||, or “ funcionam como avaliação de curto-circuito.

21. Em Perl, uma expressão é uma sequência de literais, variáveis e funções conectadas por um ou mais de um **operador** que se avaliam em um escalar de valor único ou vetor. No caso, o operador de atribuição “=” se encaixa nesta descrição. O seguinte exemplo ilustra isso:

```perl
use 5.010;

if(my $i = 1 == 1){
    print "True";
}else{
    print "False";
}
#output: "True"
```

22. A linguagem fornece as estruturas básicas de controles, como: if e unless, que são estruturas de controle de seleção/condicionais.E temos também while, for, unti, foreach, do-while, que são estruturas de controle interativas. Abaixo temos alguns exemplos:

```
if ($idade < 18){
  print "Menor de idade";
} else {
  print "Maior de idade";
}

if ($temperatura < 20){
  print "Esta frio";
} elsif($temperatura < 24) {
  print "Está agradavel";
} else {
  print "Está quente"
}

for ($i = 0; $i < 10; $i++){
  print $i."\n";
}


my listaNomes = ("Iury", "João", "Victor");
foreach my $nome (@listaNomes){
  print $nome."\n"
}

my $i = 10;
while( $i >= 0){
    print $i."\n";
    $i--;
}
```

23. Sentenças de desvios incondicionais em Perl são:

- **return [EXPRESSÃO]** - sai da subrotina devolvendo o valor da EXPRESSÃO
- **goto [RÓTULO]** - desvia para a sentença indicada
- **last [RÓTULO]** - desvia para a primeira sentença após o laço (famoso break)
- **next [RÓTULO]** - inicia uma nova iteração do laço
- **redo [RÓTULO]** - reinicia a iteração corrente do laço

24. Em perl a passagem de parâmetros não é necessário que seja definida definida. Os parâmetros passados na chamada da sub-rotina ficam armazenados em um array '@_'. Exemplo:

```
 3 sub maior{
 4     if($_[0]>$_[1]){
 5         $_[0];
 6     }else{
 7         $_[1];
 8     }
 9 }
10
11 $num1 = 5;
12 $num2 = 7;
13
14 $maior = maior($num1,$num2);
15
16 print $maior; # Resultado: 7
```

25. Sobrecarga de subprogramas:

Sobrecarga de Operadores: Em perl temos a possibilidade de programar em OO. E portanto conseguimos realizar sobrecarga operadores de uma classes utilizando 'overload'. Exemplo:

```
 3 package Classe;
 4
 5 use overload
 6     "-" => "meuMenos",
 7     "+" => \meuMais;
```

26. Não possui suporte.

27. Não possui suporte.

28. Não possui suporte.

29.

Encapsulamento em perl é feito da seguinte maneira:

```
package Automovel;

sub new{
  my $class => shift;
  my $self => {
    dono => shift;
    placa => shift;
  }

  bless $class, $self;
}

sub getDono{
  my $self = shift;
  $self->dono = shift if @_;
  return $self{dono};
}

sub getPlaca{
  my $self = shift;
  $self->placa= shift if @_;
  return $self{placa};
}
```

Onde temos acesso as variáveis ‘dono’ e ‘placa’ dentro dos métodos ‘getDono’ e ‘getPlaca’.

30. Perl suporta polimorfismo de coerção, sobrecarga e inclusão. Exemplos:
Coerção: Acontece quando um tipo primitivo ou um objeto é ‘convertido’ em outro tipo de objeto ou tipo primitivo

```perl
#Exemplo 1:
my $num1 = "2";
my $num2 = 3;
my $num3 = $num1 + $num2;    #Converte a string "2"
                             #implicitamente para inteiro
                             #e faz a soma
print $num3; #Resposta: 5


#Exemplo 2:
my $tam = @lista; #tam recebe o tamanho da lista


#Exemplo 3:
my $num = 2;
my @lista = $num; # Faz a lista com 1 elemento
```

Sobrecarga: Temos também a sobrecarga de operadores que é realizada através do método overload. Os argumentos de overload são do tipo “chave” “valor”. Os valores podem ser sub-rotinas, referências a sub-rotinas ou sub-rotinas anônimas - ou seja, qualquer coisa. Os valores especificados como strings são interpretados como nomes de métodos..

```perl
package Classe;

use overload
    "-" => "meuMenos",
    "+" => \meuMais;
```

Inclusão: exemplos são herança entre pacotes, quando usados como classes.

31. Permite herança simples e múltipla, quando utilizado um pacote ().
Em Perl, a herança (simples ou múltipla) é implementada através do vetor `@ISA`.

```
package Automovel;

sub new{
  my $class => shift;
  my $self => {
    dono => shift;
    placa => shift;
  }

  bless $class, $self;
```

```
}

sub getDono{
  my $self = shift;
  $self->dono = shift if @_;
  return $self{dono};
}

sub getPlaca{
  my $self = shift;
  $self->placa= shift if @_;
  return $self{placa};
}

package Moto;
use Automovel;

@ISA=(Moto);

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



```

32. Perl permite a sobrescrita.
```
package Pessoa;

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


sub getNome{
  my ($self) = @_;
```

```
  return $self->{ _nome };
}

sub getIdade{
  my ($self) = @_;
  return $self->{ _idade };
}

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

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

sub saySobremim{
  my ($self) = @_;

  return "Meu nome é ".($self->{ _nome })." e tenho".($self->{ _idade })."\n";
}


#=======================================================

package Programador;
use Pessoa;

@ISA=(Empregado);

sub new{
  my $class=shift;
  my $self=Pessoa->new();

  return (bless($class, $self));
}

# Sobrescrevendo o método saySobremim
sub saySobremim{
  my ($self) = @_;
```

```
  return "Meu nome é ".($self->{ _nome })." e tenho".($self->{ _idade })." E minha função é programar\n";
}
```

33. Não possui suporte.

34. A partir da versão 5.12 foi criado o modulo Error.pm que introduziu os blocos try/catch a linguagem.
Perl não oferece um mecanismo para tratamento de exceções, porém inclui facilidades que permitem a implementação de funcionalidades similares, como exemplo o tratamento de erros. Para tratamento de erros, temos a função ‘die()’ que é utilizada para terminar um programa e apresentar uma mensagem para que o usuário possa ler. Outra boa utilização de função para o tratamento de erros é a função ‘warn()’ que imprime diretamente na saída de erros padrão, informando a linha onde ocorreu um determinado erro.

35. Não possui suporte.

36. Não possui suporte.

37. Não possui suporte.

38. De acordo com os critérios de avaliação de Sebesta, perl é uma linguagem que:
- Não possui legibilidade
- Não possui confiabilidade
- Possui facilidade de escrita/redigibilidade
- Alto custo

39. De acordo com os critérios de avaliação de Varejão, perl é uma linguagem que:
- Não possui legibilidade
- Possui redigibilidade
- Não possui confiabilidade
- Possui eficiência
- Não possui facilidade no aprendizado
- possui ortogonalidade
- Possui reusabilidade
- possui modificabilidade
- Possui portabilidade