http://www.absoluta.org/absoluta/cgi/cgi_perl.htm

[CGI]

Tutorial de Perl

---

1 - O que é Perl

Perl significa "Practical Extraction and Report Language" roda em UNIX, MVS, VMS, MS/DOS, Windos, Macintosh, OS/2, Amiga e outros sistemas operacionais. Perl possui funções poderosas para manipulação de textos.A Perl é muito popular para programação de formulários WWW e gateway entre sistemas, banco de dados e usuários, além de ser muito utilizada em tarefas administrativas de sistemas UNIX, onde a linguagem nasceu e cresceu.

Por que usar a Perl

As pessoas estão usando a linguagem Perl porque é rápida, eficiente e de fácil manutenção na programação de uma ampla faixa de tarefas, em particular aquelas que envolvem a manipulação de arquivos de texto. Além disso, existem muitas pessoas usando Perl que estão preparandas para compartilhar seus códigos.

Desenvolvimento Rápido

Muitos projetos de programa são de alto nível em vez de baixo nível. Isso significa que eles tendem a não envolver manipulações em nível de bits, chamadas diretas ao sistema operacional. Em vez disso, eles focalizam a leitura de arquivos, a reformatação de sísas e gravação em saída padrão. A Perl é vigorosa; um pequeno código Perl faz muito. Em termos de linguagem de programação, isso geralmente significa que o código será difícil de ler e penoso de escrever. Mas embora ***Larry Wall***, o autor da Perl, diga que a linguagem é mais funcional do que elegante, a maioria dos programadores rapidamente descobre que o código Perl é muito legível e que não é dificil tornar-se fluente ao escrevê-lo.

Compilada ou Interpretada?

A Perl pertence a qual dessas categorias ?

Bem, a Perl é um pouco especial a esse respeito; ela é um compilador que pensa ser um interpretador. A Perl compila código de programa em código executável antes de executá-lo, portanto existe um estágio de otimização e o código executável trabalha mais rapidamente. Entretanto ela não grava esse código em um arquivo executável separado. Em vez disso, armazena-o na memória e, depois, executa-o.

Isso significa que a Perl combina o ciclo de desenvolvimento rápido de uma linguagem interpretada com a execução eficiente de um código compilado. No entanto, as desvantagens correspondentes também estão lá: a necessidade de compilar o programa cada vez que ele é executado significa uma inicialização mais lenta do que uma linguagem puramente compilada e requer que os desenvolvedores distribuam o código-fonte para os usuários.

Na prática, essa desvantagem não são muito limitadoras. A fase de compilação é extremamente rápida, portanto talvez, você nem note alguma demora entre a ativação do script Perl e o início da execução.

Para finalizar, a Perl é compilada "nos bastidores" para uma execução rápida, mas você pode tratá-la como se fosse uma linguagem interpretada.

Flexibilidade

A perl não foi desenvolvida em um plano abstrato. Ela foi criada para resolver um problema particular e evoluiu de modo a servir para um conjunto ainda mais amplo de problemas do dia-a-dia.

Ela poderia ter sido expandida para lidar com essas tarefas acrecentando-se mais e mais palavras-chave e operadores, o que tornaria a linguagem bem maior. Em vez disso, o núcleo da linguagem Perl iniciou pequeno e ficou mais refinado à medida que o tempo passou. Em alguns aspectos, ela na verdade diminuiu; o número de palavras reservadas da Perl 5 é realmente menos da metada do número de palavras reservadas da Perl 4.

Isso reflete uma preocupação de que a força da Perl resida na sua combinação única de eficiência e flexibilidade. A Perl em si tem crecido lenta e poderosamente, em geral de forma a permitir o acréscimo de melhorias e extensões em vez de receber amarrações.

Extensibilidade

Muito do crescimento da Perl como uma plataforma vem através do uso crescente de bibliotecas (Perl 4) e módulos (Perl 5). Esses são mecanismos que permitem ao desenvolvedores escrever partes auto-suficientes de código Perl que podem ser inseridos em uma aplicaç!ao Perl.

Esses add-ons variam de utilitário regularmente de alto nível, tais como um módulo que acrecenta tags HTML ao texto, até ferramentas de desenvolvimento de baixo nível, sujas e baixas, tais como os profiles e depuradores de códigos.

A possibilidade de usar extensões como essas é um avanço marcante no desenvolvimento de uma linguagem regularmente polida e tem ajudado a alimentar o crescimento no uso da Perl. Isso facilita aos desenvolvedores Perl compartilhar seus códigos com outras pessoas; a chegada dos objetos na Perl 5 torna possível as metodologias de projeto estruturado para aplicações Perl. A linguagem atingiu a maior idade sem perder sua flexibilidade ou sua força natural.

Segurança

A segurança é uma quetão importante ao escrever programas para sistemas administrativos e na Internet em geral. Usando a Perl para criar scripts no servidor web, você pode facilmente se resguardar contra aqueles usuários que tentam inserir, sorrateiramente, comandos no servidor para execuções em seu próprio benefício. Há também um excelente módulo na Perl 5 chamado *pgperl*, que permite ao seu servidor usar a técnicas públicas de codificação para salvaguardar dados importantes dos curiosos.

Ubiqüidade

Muitas pessoas na web já usam Perl! Caminhar a favor da corrente nem sempre é o me;hor método, mas Perl tem crescido com a web. Há bastante experiência acumulada se você precisar de conselhos. Os desenvolvedores Perl estão bem concientes das questões da web à medida que passam para a Perl. E muitos módulos da Perl foram construídos visando especialmente à web.

Por que você deseja Usar Perl

Existem muitas razões para usar a Perl. Ela é concisa, eficiente, flexível e versátil. A Perl é particularmente adequada para o ttrabalho de desenvolvimento na web onde a saída de texto é uma preocupação importante. Se as razões previamente descritas não são suficientes, considere isso: a Perl é totalmente grátis.

2 - Um programa básico

Vejamos um programa básico para iniciar nossos estudos.

```
#!/usr/local/bin/perl
#
# Nosso primeiro programa
#

print 'Ola mundo.';     # display da mensagem
```

Agora vamos discutir cada uma destas partes.

A Primeira linha

Todo programa Perl inicia com esta linha

```
#!/usr/local/bin/perl
```

Esta linha pode variar de sistema para sistema. Esta linha informa onde está o compilador para rodar o programa Perl.

Se você não sabe onde esta o Perl no seu sistema, tente um dos comando abaixo (UNIX):

```
% which perl
ou
% where perl
ou
% find / -name perl -print
ou
? Pergunte para o administrado da sua máquina... :))
```

Comentários

Comentários podem ser inseridos no programa com o símbolo #, e tudo que estiver depois do # até o final da linha é ignorado (*com exceção da primeira linha*) a unica forma de realizar comentários por vária linha é usando um # por linha.

Toda instrução Perl deve ser encerrada com um ponto e vírgula. Veja a última linha do programa acima.

Impresão simples

A função print imprime qualquer informação. No programa acima ela imprime a string Ola mundo. e foi encerrda com um ponto e vírgula.

Talvez o programa acima produza um resultado inesperado (não rode) o próximo passo e preparar o programa para rodar.

3 - Executando um programa

Usando o editor de texto da sua preferência digite o programa acima e salve. Emacs é um bom editor para esta finalidade por que ele tem um modo Perl que formata as linhas quando você tecla tab (use 'M-xperl-mode'). Mas use o editor no qual você se sinta mais confortável.

Após ter digitado e salvo o programa tenha certeza de que ele é executável usando o seguinte comando no prompt do UNIX:

```
% chmod 0755 <programa>
```

Agora para executar o programa basta digitar uma das linhas abaixo:

```
% perl <programa>
% ./<programa>
% <programa>
```

Se alguma coisa de errado acntecer talvez você receba uma mensagem de erro ou talvez não receba nada. Você pode rodar o programa no modo de notificação:

```
% perl -w <programa>
```

Isto vai mostrar alertas e outras mensagens antes da execução do programa. Para rodar o programa no modo debug use o seguinte comando:

```
% perl -d <programa>
```

Quando o arquivo é executado a Perl primeiramente compila e em seguida executa a versão compilada.Após uma pequena pausa para compilação o programa roda silenciosamente.

4 - Variaveis escalares

O tipo mais basico de variável em Perl e a variável escalar. Variáveis escalar podem conter úmeros, textos e letras e os números com as letras são perfeitamente intercambiáveis. Poe exemplo:

```
$prioridade = 9;
```

Seta a variável escalar $prioridade para 9, mas você pode colocar uma string na mesma variável.

```
$prioridade = 'alta';
```

Perl pode receber números e strings, veja:

```
$prioridade = '9';
$padrao     = '0009';
```

e pode realizar operações aritiméticas entre outras.

Geralmente o nome da variavel consiste de números, letras e undescore, mas elas não pode iniciar com um número e com a variável especial $_. Perl diferencia cada baixa para casa alta, então $a é diferente de $A

operadores

A Perl usa todos os operadores aritméticos do C:

```
$a = 1 + 2;    # Soma 1 em 2 e armazena o resultado em $a
$a = 3 - 4;    # Subtrai 4 de 3 e armazena o resultado em $a
$a = 5 * 6;    # Multiplica 5 por 6
$a = 7 / 8;    # Divide 7 por 8 e obtem 0,875
$a = 9 ** 10;    # Eleva nove a décima potência
$a = 5 % 2;    # Armazena em $a o resto da divisão de 5 por 2
++$a;        # Inclementa $a e depois retorna
$a++;        # Retorna $a e depois inclementa
--$a;        # Declementa $a e depois retorna
$a--;        # Retorna $a e depois declementa
```

Para strings, estas são algumas das maneira:

```
$a = $b . $c;     # Concatena $b com $c
$a = $b x $c;    # $b é repetida $c vezes
```

Para designar valores temos as seguintes formas:

```
$a = $b;     # Coloca em $a o conteúdo de $b
$a += $b;  # Soma o valor de $b ao valor de $a
$a -= $b;   # Subtrai o valor de $b do valor de $a
$a .= $b;   # Concatena $b a $a
```

Note que quando a Perl designa um valor com $a = $b ela faz uma cópia de $b e depois coloca em $a. Se depois você alterar o conteúdo de $b isto não altera $a.

Estes são apenas alguns dos operadores possíveis.

Interpolação

O código abaixo imprime banana e goiaba, usando concatenação:

```
$a = 'banana';
$b = 'goiaba';
print $a . ' e ' .  $b
```

Isso seria melhor se incluíse-mos somente uma string, mas a linha

```
print '$a  e  $b';
```

imprime as literais $a e $b e isso não ajuda muito. Mas, podemos usar aspas duplas invés de aspas simples:

```
print "$a  e  $b";
```

As aspas duplas força a interpolação de qualquer código, incluindo a interpretação das variáveis. Esta é uma forma de tratamento muito original e simpática. Outros códigos que podem ser interpolados incluem caracteres especiais como nova linha (\n) e TAB (\t).

Exercício

Este exercíco é para re-escrever o programa "Ola mundo!":
a) A string deve estar armazena em uma variável;
b) Esta variável deve ser impressa com um caracter de nova linha.
Use aspas duplas e não use concatenação. Tenha certeza de que o programa pode rodar antes de tentar processa-lo.

Arrays

Um dos tipos mais interesantes de variáveis é a variável array, ela é uma lista de variáveis escalares (isto é, números e *strings*). Variáveis array tem o formato semelhante ao da variável escalar exceto que elas são pré-fixadas com o símbolo **@**. O exemplo abaixo mostra uma lsiat de três elementos na variável array @frutas e dois elementos na variável @musica.

```
@frutas = ("banana", "goiaba", "mamao");
@musica = ("MPB", "samba");
```

Você pode acessar o conteúdo de um array usando um índice que começa com zero e colchetes são usados para especificar o índice. O exemplo abaixo retona mamao, Note que trocamos @ por $, por que mamao é escalar.

```
$frutas[2]
```

Tratamento de array

Em toda a Perl a mesma expresão em um contexto diferente pode produzir um resultado diferente. O exemplo abaixo mostra que a primeira instrução é equivalente a segunda.

```
@maismusica = ("baiao", @musica, "pagode");
@maismusica = ("baiao", "MPB", "samba", "pagode");
```

Para adicionar um novo elemento na array use a função push:

```
push(@frutas, "abacaxi");
```

agora abacaxi foi inserido no final da array frutas. Para colocar dois ou mais itens em um array use uma das seguintes formas:

```
push(@frutas, "abacaxi", "acerola");
push(@frutas, ("abacaxi", "acerola");
push(@frutas, @maisfrutas);
```

A função push além de acresenta os novos elementos na array, retorna o novo número de elementos contido na array.

Para remover o último item de uma lista e retorna-lo, use a função pop. Usando pop na nossa lista original, ela retorna mamao e @frutas fica com dois elementos:

```
$umafruta = pop(@frutas);    # Agora $umafruta = "mamao"
```

então vimos que é possivel transforma uma variável array em uma variável escalar. A linha

```
$f  =  @frutas;
```

armazena em $f o número de elementos em @frutas, mas a linha

```
$f  =  "@frutas";
```

retorna quais são os elemento com um espaço em branco entre eles. Este espaço ode ser trocado por qualquer caracter, para isto basta trocar o valor da variável especial **$".** Esta é apenas uma das várias variáveis especiais da Perl

Arrays podem ser usadas para atribuir valores a variáveis escalares, veja:

```
($a, $b)  =  ($c, $d)    # É semelhante a $a  =  $c;  $b  =  $d
($a, $b)  =  @frutas;    # Colocar os dois primeiros itens da variável @frutas nas variávesi $a e $b.
($a, @qualquerfruta)  =  @frutas;    # coloca o primeiro elemento da variável @frutas em $a, os demais são
                                     # são armazenados na variável @qualquerfruta
(@qualquerfruta, $a)  =  @frutas;    # @qualquerfruta é igual a @frutas e $a é indefinida
```

Para você reculperar o valor do índice do último elemento de @frutas use a seguinte instrução:

```
$#frutas
```

Imprimindo arrays

Dependendo do contexto o resultado pode ser diferente, veja:

```
print @frutas;       # Sem aspas
print "@frutas";     # Entre aspas
print @frutas."";    # Num contexto escalar
```

Exercício

Teste os três exemplos acima para ver o resultado.

6 - Arquivos

Vejamos um programa Perl básico que realiza algo semelhante ao comando *cat* do UNIX sobre um arquivo.

```
#!/usr/bin/perl
#
# Programa para abris o aquivo passwd, ler o conteúdo, imprimir e fechar o arquivo.
#
$arquivo  =  '/etc/passwd';        # Nome do arquivo
open(INFO, $arquivo);              # Abre o arquivo
@linhas  =  <INFO>;                # Le o conteúdo e armazena em um array
close(INFO);                       # Fecha o arquivo
print @linhas;                     # Imprime o array
```

A função open abre o arquivo para entrada (i.e. para leitura). O primeiro parâmetro (INFO) é o apelido do arquivo

(*filehandle*) que permite a Perl fazer referências ao arquivo no futuro. O segundo parâmetro ($arquivo) é uma espressão que informa o nome do arquivo. Se o nome do arquivo estiver entre aspas simples ele realizará uma expansão de shell. A expressão '~/notas/lista' não será interpetrada corretamente. Se você que forçar uma expansão de shell use os sinais de maior que e menor que: <~/notas/lista>. A função *close* informa para a Perl fechar o arquivo.

Temos mais alguns pontos para tratar com relação ao apelido do arquivo (*filehanding*). Primeiro, a função *open* pode especificar se um arquivo deve ser aberto para entrada, saída ou adicionar. Para realizar estes pedidos anteceda o nome do arquivo com > para saída e >> para adicionar:

```
open(INFO, $arquivo);          # Abre para entrada
open(INFO, ">$arquivo");      # Abre para saída
open(INFO, ">>$arquivo");    # Abre para adicionar
opne(INFO, "<$arquivo");      # Também abre para entrada
```

Segundo, se você quer gravar alguma coisa no arquivo ele já deve estar aberto para saída ai você pode usar a função print com um parâmetro extra. Para gravar uma *string* em m arquivo com o apelido (*filehandle*) INFO use:

```
print INFO "Esta linha sera gravada no arquivo\n";
```

Terceiro, você pode usar o formato abaixo para abrir respectivamente a entrada padrão (geralmente o teclado) e a saída padrão (geralmente o video):

```
open(INFO, '-');       # Abre a entrada padrão
open(INFO, '>');       # Abre a saída padrão
```

No nosso programa de exemplo (acima) lemos a informação apartir de um arquivo. O arquirvo é INFO, e para ler os dados apartir dele a Perl usa o seguinte formato:

```
@linhas  =  <INFO>;
```

Le o arquivo referênciado pelo apelido (*filehandle*) INFO e armazena as linhas no array @linhas. Note que a expressão <INFO> le todos os registro do arquivo numa única vez.Isso por que a leituta está dentro de um contexto de variável array. Se @linhas for trocada por uma variável escalar $linhas só poderemos ler uma linha por vez. Nos dois casos cada linha e armazenada completamente e encerrada com um caracter de nova linha **\n**.

Exercício

Modificar o programa de exemplo (acima) de modo que cala linha impressa seja precedida do simbolo #. Você tem que incluir uma linha e modificar outra linha (esta é a dica ;)). Use a variável **$" .** Coisas estranhas podem acontecer, você pode encontra ajuda usando a opção **-w** mencioda no tópico executando um programa.

7 - Controle de estrutura

Muitas possibilidades interesantes surgem quando introduzimos controles de estrutura e loops. A Perl suporta vários tipos diferentes de controle de estrutura que são melhores que os controles do C, mas são muito similar as estruturas do Pascal. Agora vamos discutir um pouco sobre estas estruturas:

7.1 foreach

Para ler todas a linhas de um array ou outro tipo de estrutura de lista a Perl usa a estrutura foreach.

```
@frutas = ("banana", "goiaba", "mamao");
foreach $maisfruta (@frutas)
  {
   print "$maisfruta\n";
   print "Delicia...delicia ...\n";
  }
```

As ações que devem ser repetidas várias vezes estão entre chaves. Na primeira vez da execução do bloco a variável $maisfruta recebe o valor do primeiro item de @frutas. Na próxima vez ela recebe o valor do segundo item de @frutas e este loop repete-se até o fim de @frutas, ou seja, até ler todos os itens de @frutas. Se @frutas estiver vazia quando iniciar o loop o bloco de ações nunca será executado.

7.2 Testes
A próxima estrutura é utilizada para testar se uma condição é verdadeira ou falsa. Na Perl qualquer valor diferente de zero e qualquer string que não esteja vazia é considerada verdadeira. O número zero, o zero dentro de uma string e strings vazias são considerados falso. Vejamos alguns exemplos com números e strings:

```
$a = = $b    # testa se $a é numericamente igual a $b, atenção NÃO use o operador =
$a != $b     # testa se $a é numericamente diferente de $b
$a eq $b     # testa se a string $a é igual a $b
$a ne $b     # testa se a string $a é diferente de $b
```

Podemos usar o "e" lógico, "ou" lógico e o "não" lógico:

```
($a && $b)    # $a e $b são verdadeiros ?
($a || $b)    # $a ou $b são verdadeiro ?
!($a)         # $a é falso ?
```

7.3 for
A estrutura for da Perl é similar a estrutura for do C.

```
for (inicializacao; teste; inclemento)
  {
   primeira ação
   segunda ação
   etc
  }
```

Primeiro para todo tratamento a inicialização é executada. Depois enquanto o teste for verdadeiro o bloco de ações é executado. Após a execução do bloco o inclemento é executado. Vejamos um exemplo que imprime os número de 0 a 9:

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

7.4 while e until
Vejamos um programa que le algumas entradas do teclado e não continua enquanto a senha não for correta:

```
#!/usr/local/bin/perl

print "Senha?";                  # pergunta pela senha de entrada
$a = <STDIN>;                   # recebe a entrada
chop $a;                        # remove a nova linha (\n) do final da entrada
while ($a ne "alexandra")       # enquanto a entrada estiver errada
   {
    print "Senha invalida. Tente novamente. Senha?";    # pergunta novamente
    $a = <STDIN>;       # recebe a entrada novamente
    chop $a;            # retira a nova linha (\n) novamente
    }
```

O bloco de ações é executa enquanto a entrada não for verdadeira. A estrutura while é muito simples, mas esta é a oportunidade para notarmos algumas coisas. Primeiro, podemos receber uma entrada do teclado quando abrimos primeiramente o arquivo. Segundo, quando a senha é digitada $a recebe o valor incluindo uma nova linha (\n) no final. A função chop remove o último caractere de uma string, neste caso a nova linha (\n).

Para testar o oposto, nos usamos a estrutura until. Ela executa o bloco de ações até que a condição seja verdadeira, e não enquanto é verdadeira.

Outra técnica é colocar o estrutura while ou until no final do bloco de ações. Isso requer a presença do operador **do** para marcar o início do bloco de ações e o teste (while ou until) fica no final do bloco. Se você esquecer o **do** sinto muito mas não vai funcionar. Vejamos o exemplo:

```
#!/usr/local/bin/perl

do
  {
   print "Senha?";           # pergunta pela senha de entrada
   $a = <STDIN>;        # recebe a entrada
   chop $a;                # remove a nova linha (\n) do final da entrada
   }
while ($a ne "alexandra")    # volta para o do enquanto a entrada estiver errada
```

7.5 Exercício
Modifique o programa do exercício anterior de forma que cada linha lida seja impressa precedida de um número seqüencial. Você vai ter algo como o seguinte:

```
1  root:jgutgtyi878jhg:0:0:Super usuario:/:/bin/csh
2  frank:*:10:12:Frank Ned:/home/frank:/bin/csh
3  ricardo:*:10:15:Ricardo Toledo:/home/ricardo:/bin/csh
```

Uma dica é usar a seguinte estrututa:

```
while ($linha = <INFO>)
  {
   seu código... ;))
   }
```

Quando você tiver conseguido fazer isso, você pode mudar o programa de forma que o número das linhas sejam impresso no seguinte formato 001, 002, 003, ..., 010, 011, 012, etc. Para fazer isso você deve simplesmente mudar uma linha inserindo quatro caracteres.... boa sorte.

8 - Condicionais

A Perl permite operadores condicionais if / then / else.

```
if ($a)
  {
   print "A string não está vazia...\n";
   }
else
  {
   print "A está vazia ...\n";
  }
```

Lembre-se uma string vazia é considerada falsa.

É possivel colocar várias alternativas com operadores condicionais:

```
if (!$a)
 {
  print "A string está vazia ...\n";
 }
elsif (length($a) = = 1)
 {
  print "A string tem um caractere...\n";
 }
elsif (length($a) = = 2)
 {
  print "A string tem dois caracteres...\n";
 }
else
 {
  print "A string tem vários caracteres...\n";
 }
```

Note que elsif NÃO tem o "e".

8.1 - Exercícios
Localize um arquivo grande que contenha texto e brancos. No exercício anterior você imprimiu o arquivo de senhas com as linhas numeradas. Altere o programa de forma que ele trabalhe com arquivos de texto. Agora altere o programa novamente de forma que ele não imprima nem conte as linhas em branco. Lembre-se que quando uma linha é lida, automaticamente é incluido um caractere de nova linha ao final da linha lida.

9 -String

Uma das propriedades mais utilizadas da Perl (se não a mais utilizada) são as poderosas facilidades para manipulação de string. Esta propriedade é conhecida como expressões regulares (RE) que é copiada de muitas das ferramentas do UNIX.

9.1 Expressões regulares (RE)

Uma expressão regular fica dentro de barras (/RE/), e comparação é feita com o operador =~ . A expressão abaixo é verdadeira se a *string* "**ola**" aparecer na variável **$sentenca**.

```
$sentenca =~ /ola/
```

As expressões regulares (RE) são *case sensitive*, ou seja diferenciam maiusculas de minusculas, a expressão acima seria falsa se:

```
$sentenca = "Ola meus amigos"
```

por que o "O" de ola está em maiusculo.

O operador != é utilizado para a negação da RE. Veja o exemplo abaixo

```
$sentenca != /ola/
```

é verdadeiro por que a *string* **ola** não aparece na sentença.

9.2 A variável especial $_

Nos podemos usar a condicional da seguinte forma

```
if ($sentenca =~ /casa/)
 {
  print "Nos estamos falando da linda casa...\n";
 }
```

a mensagem será impressa se a *string* **casa** estiver na variável **$sentenca**, veja:

```
$sentenca = "A linda casa de campo";
$sentenca = "O casarao da avenida das palmeiras";
$sentenca = "O casaco de vento";
```

logo, no nosso exemplo a mensagem seria impressa em qualquer um dos casos.

Mas frequentemente nós comparamos a sentença com a variável especial $_, que é escalar. Se nós fizermos isso, poderemos desconsiderar os operadores de igualdade (=~) e diferença (!=) e escrever a sentença como segue abaixo:

```
if (/casa/)
 {
  print "Nos estamos falando da linda casa...\n";
 }
```

A variável $_ é o padrão (*default*) para muitas operações da Perl e consequente você vai encontrar este tipo de comparação com muita freqüência.

Bem, quando eu estava iniciando meus estudos da Perl, tive dificuldades para entender o funcionamento da variável $_ , então vou montar um outro exemplo para os que tiverem dificuldades como eu tive.

$_ é como uma copia da última operação de atribuição, então se:

```
$teste = "ola";
```

e se fizermos:

```
print $_;
```

o resultado sera:

```
ola
```

Espero que tenha entendido, caso não esteja claro envie email para podemos montar um exemplo mais esclarecedor.

9.3 Mais expressões regulares (RE)
Podemos utilizar alguns caracteres especiais com expressões regulares (RE) e isso lhe dá muito poder e fica com uma aparecia meio compricada. :). Seria uma boa idéia se você tiver calma no estudo das REs; sua criatividade pode algumas vezes tornar-se estado da arte

Vejamos algumas RE e seu significado:

```
.    # qualquer caractere simples exceto nova linha ( \n )
^    # o início da linha ou string
$    # o fim da linha ou string
*    # nehum ou vários do último caractere
+    # um ou vários do último caractere
?    # nenhum ou um do último caractere
```

agora vamos ver alguns exemplos de sua aplicação. Lembre-se eles devem estra entre barras (/ RE /) para serem usados.

```
t.e    # t seguido de qualquer caractere seguido de e
       #  isso é verdade para tre ou tle
       # mas é falso para te ou tale


^f    # f no início da linha ou string
^ftp    # ftp no início da linha ou string
e$    # e no final da linha ou string
tle$    # tle no final da linha ou string
und*    # isso é verdade para un ou und ou undd ou unddd, etc
.*    # qualquer string exceto nova linha ( \n ), por que o  .  marca qualquer coisa nemos nova linha
       # e o * significa nehum ou vários do último caractere
^$    # uma linha vazia
```

Existem muitas opções. Colchetes([ ]) são utilizados para marcar qualquer conjunto de caracteres que estejam dentro deles. Dentro dos colchetes um - significa entre e um ^ no início significa não.

```
[qjk]       # q ou j ou k
[^qjk]      # não q ou não j ou não k
[a-z]       # qualquer coisa entre a e z inclusive
[^a-z]      # não para letras em minusculo
[a-zA-Z] # qualquer letra
[a-z]+      # qualquer seqüência de letras minusculas.
```

Neste ponto você já tem condições de pular para o exercício e faze-lo, o resto são referências.

Uma barra vertical | representa um ou e parentese (...) são usados para agrupamento de strings:

```
frank|ned    # frank ou ned
(tr|br)incar  # trincar ou brincar
(da)+         # da ou dada ou dadada ou ...
```

Mais caracteres especiais:

```
\n      # nova linha
\t      # um tab
\w     # qualquer palavra alfanumérica - semelhante a [a-zA-Z0-9_]
\W    # qualquer palavra sem letras ou números - semelhante a [^a-zA-Z0-9_]
\d    # qualquer dígito - semelhate a [0-9]
\D    # NÃO dígitos - semelhante a [^0-9]
\s    # qualquer caractere de espaço: espaço, tab, nona linha, etc
\S    # qualquer caractere diferente de espaço
\b    # qualquer palavra dentro do limite, ou seja fora de [ ]
\B    # qualquer palafra fora do limite
```

Alguns caracteres são um caso peculiar quando trata-se expressões regulares, se você precisar usa-los devem ser precedidod de uma barra invertida:

```
\|       # barra vertical
\[       # abertura de colchete
\)       # fechamento de parentese
\*       # asterisco
\ ^      # símbolo de circunflexo
\@       # símbolo de arroba
\/       # barra
\\       # barra invertida
```

9.4 Alguns exemplos de RE

9.5 Exercícios

Referência :

- http://agora.leeds.ac.uk/Perl/start.html
- Learning Perl (ISBN 1-56592-042-2)
- Programming Perl (ISBN 1-56592-149-6)
- Perl 5 Desktop Referenc (ISBN 1-56592-187-9)
- Teach Yourself CGI programming with Perl 5 in a week (ISBN 1-57521-196-3)
- Perl 5 quick reference (ISBN 85-352-0144-0)

Verdade @bsoluta - 30/04/98 - frank@cnpq.br