Ficha 1

Introdução ao Perl

21 de Fevereiro de 2015

1 Introdução

O Perl é uma linguagem de scripting. As linguagens deste género são normalmente interpretadas e permitem executar um certo tipo de tarefas de uma forma muito eficiente (escrevendo muito pouco código). Estas linguagens são sobretudo vocacionadas para o tratamento de texto.

O Perl é sem dúvida uma das linguagens de scripting mais populares sendo uma das mais antigas. Continua a ser muito usada por causa das bibliotecas que existem.

Segue-se um exemplo de um programa em Perl que não faz mais nada do que imprimir duas mensagens.

```
1  # Isto e um comentario
2  # Um comentario em Perl comeca com um '#'
3  # O '#' nao precisa de estar no inicio da linha
4
5  # Esta biblioteca e importante porque obriga o programador
6  # a ser mais disciplinado
7  # e ajuda-o a encontrar muitos dos erros basicos
8
9  use strict; # Use sempre esta biblioteca
10
11 # Repare que uma linha em perl acaba sempre com um ponto e virgula
12 # O \n que aparece no fim e para indicar ao Perl que se pretende mudar de linha
13 print "ola mundo\neste e o meu primeiro programa em Perl\n";
```

2 Variáveis

O Perl é uma linguagem estranha. Na verdade, os implementadores do Perl deve ter sérios problemas (e tem-nos) e isso nota-se. Uma das coisas que se aponta ao Perl é alguma ilegibilidade. Mas antes de tudo comecemos com o conceito de variável. Uma variável permite armazenar um valor para o usar mais tarde. Segue-se um exemplo.

```
1  use strict;
2
3  my $x = 2; # Estou a declarar uma variavel x com o valor 2
4  print "Este e o valor do x: $x\n";
5  $x = $x + 3;
```

```
6
7 print "Este e o novo valor do x: $x\n";
```

Repare algumas coisas neste exemplo:

1. Uma variável do tipo *escalar*[1] começa sempre por um '$';
2. É conveniente declarar uma variável (na verdade quando estamos no ambiente *strict* é obrigatório) usando o comando `my`;
3. Uma *string* é uma colecção de caracteres e é delimitada por aspas;
4. Dentro de uma *string* é possivel usar o valor de uma variável.

A vantagem do ambiente *strict* é que se não declararmos uma variável obtemos um erro. Isso é extremamente útil já que quando não usamos esse ambiente e usamos uma variável à qual nunca foi atribuído um valor o Perl arbitra um valor por omissão para a variável dependendo do contexto. Esse valor é 0 se estamos no contexto de um número ou string vazia se estamos nesse contexto. Veja agora o exemplo anterior com umas pequenas modificações e tente perceber o que aconteceu.

```
1 #use strict;
2
3 #my $x = 2;
4 print "Este e o valor do x: $x\n";
5 $x = $x + 3;
6
7 print "Este e o novo valor do x: $x\n";
```

3 Leitura

A leitura em Perl é simples mas, como provavelmente já estava à espera, é estranha.

```
1 use strict;
2
3 # Declarar uma variavel e ler uma linha;
4 my $linha = <>;
5
6 print "A linha que li foi : $linha\n";
```

4 Estruturas de controlo

Tal como a maioria das linguagens imperativas o Perl possui várias estruturas de controlo que seguem os moldes clássicos:

if A instrução condicional que permite executar um bloco de código[2] se uma dada condição for verdadeira; existe uma versão mais extensa que usa um `else` tendo neste caso a seguir ao `else` um segundo bloco de código que só é executado se a condição for falsa;

[1] O tipo *escalar* inclui números e letras

[2] Um bloco de código é um conjunto de linhas delimitado por chavetas

unless Funciona exactamente da mesma forma que o `if` mas neste caso o bloco é executado se a condição for falsa, também permite `else`

while É uma estrutura de controle cíclica que executa o bloco enquanto a condição for verdadeira;

until É a estrutura complementar do `while`, neste caso o bloco executa até a condição ser verdadeira;

do while Estrutura de controlo semelhante ao `while` só que a condição é verificada após executar o bloco;

do until Semelhante à anterior para para o `until`;

for Estrutura de controle cíclica que da mesma forma que em `C` ou `Java`;

foreach Estrutura de controle para percorrer uma lista[3]

Segue-se um exemplo que imprime todos os números pares entre 1 e 20 que usa o `for` e o `if`. Utiliza-se também o operador `%` que dá o resto da divisão.

```
use strict;
my $i;

for($i=1; $i <= 20; $i++) {
# Se um numero e par o resto da divisao por 2 da 0
    if($i % 2 == 0) {
        print " $i";
    }
}
```

Quando uma estrutura de controlo está associada a uma única linha pode-se escrever o código de outra maneira (que não necessita de chavetas). O próximo exemplo é muito semelhante ao anterior com a diferença dessa forma mais abreviada de escrever o `for`.

```
use strict;
my $i;

for($i=1; $i <= 20; $i++) {
# Se um numero e par o resto da divisao por 2 da 0
    print " $i" if($i % 2 == 0);
}
```

O próximo exemplo ilustra a utilização do Perl para efectuar uma contagem decrescente.

```
use strict;
my $i;
print "insira o valor inicial da contagem decrescente: ";
$i = <>;
while($i > 0) {
    print "$i\n";
    $i--;
}
```

[3]Falaremos de listas mais tarde

5 Manipulação de texto

Como o Perl é vocacionado para a manipulação de texto é trivial fazer substituições. Na verdade até existem várias formas de o fazer. Existem também formas muito simples de procurar algo num texto.

Imagine que quer ler uma linha do teclado e escrever encontrei se algures no meio dessa linha existir a sequência de letras "scripting". Eis um peequeno programa que faz isso.

```
use strict;

# Ler uma linha;
my $linha = <>;

print "encontrei\n" if $linha =~ /scripting/;
```

Eis outra forma de o fazer.

```
use strict;

# Ler uma linha;
$_ = <>;

print "encontrei\n" if /scripting/;
```

Repare que na segunda versão você não explicitou uma variável quando utilizou a procura. O que acontece é que a linha foi armazenada numa variável implícita chamada `$_`.

A função de procura é chamada `m//`. Repare que nos exemplos acima não utilizámos o `m`. Normalmente só se vê a utilização do `m` quando se quer utilizar outros separadores.

```
# Ler uma linha;
$_ = <>;

print "encontrei\n" if m#scripting#;
```

Repare que neste caso utilizámos o m porque quisemos usar o `#` como o separador. Os separadores podem ser ou o mesmo caracter ou, no caso de parentesis (curvos, rectos ou chavetas) usando ambos como se vê no exemplo seguinte.

```
# Ler uma linha;
$_ = <>;

print "encontrei\n" if m{scripting}; # poderia ser m(scripting) ou m[scripting]
```

Se quisermos substituir letras podemos usar o construtor `y///`. Este construtor recebe entre as barras dois conjuntos de letras sendo o primeiro conjunto de letras o que se pretende substituir e o segundo conjunto o que se utiliza para a substituição. Eis um exemplo em que se substituem as vogais pela maiúscula correspondente.

```
use strict;

# Ler uma linha;
$_ = <>;
```

```
y/aeiou/AEIOU/;
print;
```

Notas sobre este exemplo:

- Mais uma vez a variável `$_` foi usada implícitamente;
- A variável foi usada implicitamente na leitura, substituição e escrita;
- Quando se invoca a função `print` sem parâmetros ele vai buscar o que quer imprimir à mesma variável.

Se quisermos substituir mais do que um caracter temos que usar o comando `s///`. O seu funcionamento simplificado é muito semelhante ao comando `y///`. O próximo exemplo lê uma linha e substitui `spln` por `scripting para linguagem natural`.

```
use strict;

# Ler uma linha;
$_ = <>;
s/spln/scripting para linguagem natural/; # Repare que aqui tambem se podem usar outros
    separadores, e.g. s(spln){scripting para linguagem natural}
print;
```

6 Comprimento de sequências e subsequências

Existem duas funções úteis para a manipulação de sequências: o comprimento (função `length`) e a extracção de subsequências (função `subseq`). Seguem-se alguns exemplos tirados do manual.

```
  my $s = "The black cat climbed the green tree";
  print length $s; # Imprime 36
  print "\n";
  my $color = substr $s, 4, 5;     # black
  my $middle = substr $s, 4, -11;  # black cat climbed the
  my $end   = substr $s, 14;       # climbed the green tree
  my $tail  = substr $s, -4;       # tree
  my $z     = substr $s, -4, 2;    # tr
  print "\'$color\' \'$middle\' \'$end\' \'$tail\' \'$z\'\n";
```

7 Tratamento de um conjunto de linhas

Imagine que quer tratar um ficheiro inteiro e não só uma linha. Isso é muito fácil em Perl. Veja se percebe o programa seguinte que simplesmente lê todas as linhas que são introduzidas e as imprime. O ciclo `while` funciona porque o operador `<>` que lê uma linha devolve um valor que é interpretado como falso quando já não existem mais linhas para ler. Este programa também conta quantas linhas foram lidas.

```
use strict;

my $linhas = 0;

# Ler uma linha de cada vez
while(<>) { # Repare que a leitura usa a variavel por omissao $_
    print;
    $linhas = $linhas + 1;
}
print "Num. de linhas: $linhas\n";
```

8 Exercícios

1. Escreva um programa que leia um texto e numere as linhas, isto é, que imprima cada linha precedida do nº de linha correspondente;
2. Escreva um programa que leia um texto e o escreva, substituindo as vogais por asteriscos;
3. Escreva um programa que leia um texto e o escreva em maiúsculas;
4. Escreva um programa que leia uma frase e a escreva colocando a primeira letra em maiúsculas;
5. Escreva um programa que leia um texto e que conte o número de linhas que contém a palavra "linguagem";
6. Escreva um programa que leia um texto e conte o número de ocorrências de cada vogal;
7. Escreva um programa que leia um texto e imprima cada frase ao contrário;
8. Escreva um programa que leia um texto em que cada linha é um nome de uma pessoa e que escreva o primeiro nome, as iniciais de cada nome intermédio e o último nome (por exemplo, se o nome fosse Mário Alberto Nobre Lopes Soares deveria escrever Mário A. N. L. Soares);
9. Escreva um programa que leia um texto e conte o nº de ocorrências de *scripting*, *linguagem*, *processamento*, *processamento de linguagem* e *linguagem natural*;
10. Escreva um programa que leia um texto e conte o nº de ocorrências de palavras capitalizadas.