Sub-Rotinas

Algumas vezes, programas podem se tornar tão grandes, que mantê-los, ou seja, corrigir erros, melhorar a performance do código ou criar novas funcionalidades, pode se tornar uma tarefa complicada.

Em alguns casos, certas funcionalidades de um código podem se repetir diversas vezes. Logo, qualquer alteração nesses trechos irá requerer que se altere diversas partes do código. Perl, assim como diversas outras linguagens, fornece meios para modulação de código: as sub-rotinas.

Sub-rotinas, também conhecidas em outras linguagens como métodos, procedimentos ou funções, são trechos de código declarados uma única vez que podem ser chamados diversas vezes durante o programa. Sub-rotinas podem aceitar parâmetros como entrada de dados, realizar processamentos e retornar dados.

Para entendermos melhor uma sub-rotina devemos imaginar como Perl processa as linhas de código, em sequência, uma linha de cada vez. Assim, uma vez que a sub-rotina tenha sido declarada, ela pode ser chamada a qualquer momento e em qualquer ponto do código após a declaração.

Criando sub-rotinas

Sub-rotinas que executam procedimentos simples

Sub-rotinas que recebem parâmetros

Retornando resultados

Variáveis globais

Palavras reservadas

Módulos

Criando sub-rotinas

Sub-rotinas são declaradas através da palavra reservada sub, seguida de um nome. Todos os comandos da sub-rotina devem estar delimitados por chaves. Convém indentar a região interna do bloco para aperfeiçoar a legibilidade, ou seja, aplicar um espaçamento tabular em cada linha de código após a abertura do bloco.

```
sub exemplo_de_sub-rotina {
    # Comandos
}
```

Todo o conteúdo presente no bloco será repetido sempre e unicamente quando a sub-rotina for chamada. Para chamar uma sub-rotina é necessário digitar o nome da mesma seguida ou não de parênteses.

```
# Ambas as formas de chamada funcionam igualmente
exemplo_de_sub-rotina;
exemplo_de_sub-rotina();
```

Em alguns casos, parênteses ajudam a melhorar a visualização do código, entretanto muitos programadores preferem não os utilizar. Em Perl, o uso de parênteses em sub-rotinas é opcional.

A documentação do Perl recomenda o uso do caractere “&” antes do nome de uma sub-rotina, mas isso também é opcional, e grande parte dos desenvolvedores em Perl prefere não adotar tais medidas.

Sub-rotinas que executam procedimentos simples

Sub-rotinas podem ser escritas para executar procedimentos simples como imprimir mensagens em determinadas ocasiões. No exemplo a seguir, faremos um laço de repetição contando de 1 a 20. A seguir verificaremos se o resto da divisão do número por cinco é zero. Se sim, a sub-rotina multiplo_de_cinco será chamada. Se não, a sub-rotina nao_multiplo_de_cinco é quem será chamada. A primeira imprime uma mensagem para o usuário indicando que o número é um múltiplo de cinco, a segunda imprime indicando que não é um múltiplo de cinco.

```
use strict;

# Detectando multiplos de cinco
my $i;

# Subrotinas
sub multiplo_de_cinco{
    print " (SIM), ";
}
sub nao_multiplo_de_cinco{
    print " (NAO), ";
}

print "Detectando numeros entre 1 e 20 multiplos de cin-co:\n";

# Laco principal
for($i = 1; $i <= 20; $i++){

    print "$i";

    # Verifica se a divisao de $i por 5 nao tem resto
    if($i % 5 == 0){
        multiplo_de_cinco();
    }

    else{
        nao_multiplo_de_cinco();
    }

}
# 1 (NAO), 2 (NAO), 3 (NAO), 4 (NAO), 5 (SIM),
# 6 (NAO), 7 (NAO), 8 (NAO), 9 (NAO), 10 (SIM),
# 11 (NAO), 12 (NAO), 13 (NAO), 14 (NAO), 15 (SIM),
# 16 (NAO), 17 (NAO), 18 (NAO), 19 (NAO), 20 (SIM),
```

Observe que o uso de duas sub-rotinas seria desnecessário se pudéssemos enviar o valor atual e testá-lo diretamente na sub-rotina. Aprenderemos a seguir a utilizar parâmetros em sub-rotinas.

Sub-rotinas que recebem parâmetros

Sub-rotinas podem receber dados, enviados através de argumentos, como parâmetros. Para isso serão necessários dois passos: (i) envio da variável como argumento na chamada da sub-rotina; e (ii) leitura do parâmetro dentro da sub-rotina.

```
sub exemplo_de_sub-rotina{
    $primeiro_argumento = $_[0];
    $segundo_argumento = $_[1];
    $terceiro_argumento = $_[2];
    # ...
}

# Chamada
exemplo_de_sub-rotina(1948.90, “Hello”, “Terceiro argumen-to”);
```

Os parâmetros enviados podem ser de qualquer tipo, numérico ou strings. A sub-rotina recebe os argumentos na variável “$_”. É possível acessá-los através do índice na ordem de envio.

Veja o exemplo anterior alterado para receber argumentos:

```perl
use strict;

# Detectando multiplos de cinco
my $i;

# Subrotinas
sub testa_multiplo_de_cinco{

    my $argumento = $_[0];

    if($argumento % 5 == 0){
        print "$argumento (SIM), ";
    }

    else{
        print "$argumento (NAO), ";
    }
}

print "Detectando numeros entre 1 e 20 multiplos de cin-co:\n";

# Laco principal
for($i = 1; $i <= 20; $i++){
    testa_multiplo_de_cinco($i);
}

# Detectando numeros entre 1 e 20 multiplos de cinco:
# 1 (NAO), 2 (NAO), 3 (NAO), 4 (NAO), 5 (SIM),
# 6 (NAO), 7 (NAO), 8 (NAO), 9 (NAO), 10 (SIM),
# 11 (NAO), 12 (NAO), 13 (NAO), 14 (NAO), 15 (SIM),
# 16 (NAO), 17 (NAO), 18 (NAO), 19 (NAO), 20 (SIM),
```

Lembre-se que argumentos são enviados na chamada da sub-rotina. E essa recebe argumentos como parâmetros.

Retornando resultados

Além de receber argumentos e realizar tarefas, sub-rotinas podem retornar informações através da palavra reservada return.

No exemplo a seguir, a sub-rotina detectará se um número é múltiplo de cinco e retornará o quadrado desse valor. Nesse exemplo vamos utilizar duas sub-rotinas diferentes. A primeira fará o teste se o numeral recebido é realmente um múltiplo de cinco. Se for, a própria sub-rotina chamará uma outra sub-rotina para calcular o valor do numeral ao quadrado. Preste atenção nos dados retornados por ambas as rotinas.

```perl
use strict;

my $i;
my $i2;

# Subrotinas
sub testa_multiplo_de_cinco{
    my $n = $_[0];
    if($n % 5 == 0){
        my $n2 = calcula_quadrado($n);
        return $n2;
    }
    else{
        return -1;
    }
}

sub calcula_quadrado{
    my $n = $_[0];
    return $n**2;
}

print "Detectando o quadrado de numeros entre 1 e 20 multi-plos de cinco:\n";

# Laco principal
for($i = 1; $i <= 20; $i++){

    $i2 = testa_multiplo_de_cinco($i);

    if($i2 != -1){
        print "O quadrado de $i eh $i2\n";
    }

}

# O quadrado de 5 eh 25
# O quadrado de 10 eh 100
# O quadrado de 15 eh 225
# O quadrado de 20 eh 400
```

Ao executar esse script, Perl incialmente lê as sub-rotinas, mas sem executá-las. Em seguida, imprime uma mensagem na tela para o usuário e chama a sub-rotina testa_multiplo_de_cinco 20 vezes, cada vez enviando um valor diferente através da variável $i. A sub-rotina testa_multiplo_de_cinco, por sua vez, testa se o valor é múltiplo de cinco, e se for ela chama a sub-rotina calcula_quadrado, que retorna o quadrado do valor enviado. Se o valor testado não for múltiplo de cinco, a sub-rotina retornará -1. A escolha de -1 foi especifica para o exemplo, pois sabemos que nenhum número elevado ao quadrado poderá obter como resultado um valor negativo. Assim, o valor retornado por testa_multiplo_de_cinco será recebido pela variável $i2, que a testará para que apenas resultados diferentes de -1 sejam exibidos.

É importante destacar que ambas sub-rotinas utilizaram variáveis com mesmo nome ($n). Assim, podemos concluir que variáveis presentes em sub-rotinas são locais, ou seja, uma variável criada em uma sub-rotina não interfere na variável criado por outra. O conteúdo de uma variável criada em uma sub-rotina pode ser acessado fora dela se for retornado através da palavra-chave return, e tal variável será recebida em outra sub-rotina através da variável “$_”.

Variáveis globais

Entretanto existem variáveis válidas em quaisquer partes do código, inclusive dentro de sub-rotinas. São as chamadas variáveis globais (disponíveis a partir da versão 5.6.0 do Perl). Elas podem ser declaradas por meio da palavra reservada our.

```perl
use strict;

sub teste_variaveis{

    print our $i;
    $i = "Podemos modificar variaveis globais em qual-quer parte do codigo.\n";

}

our $i = "Isso eh uma variavel global.\n";

teste_variaveis();

print $i;
```

Observe que foi possível imprimir e alterar o conteúdo da variável $i dentro de uma sub-rotina sem precisar recebê-la como parâmetro, e ainda imprimir modificações sem a necessidade de retorná-la.

Palavras reservadas

Como já dito anteriormente, Perl tem uma lista de palavras reservadas que são utilizadas como operadores ou funções nativas. Tais palavras não podem ser utilizadas como nomes de sub-rotinas. Segue abaixo uma pequena lista com palavras reservadas.

alarm	chomp	chop	close
defined	delete	die	do
each	eof	eval	exists
exit	gmtime	join	keys
last	length	localtime	my
next	open	our	pack
package	pop	print	printf
push	read	redo	ref
return	scalar	shift	sleep
sort	splice	split	splintf
sub	substr	system	time
undef	unpack	unshift	wantarray
warn	while	write	x

Módulos

Falamos anteriormente que módulos agregam novas funcionalidades não nativas ao Perl. Agora que já apresentamos as sub-rotinas, podemos ter uma visão melhor sobre o que é um módulo.

Em Perl, um módulo pode ser visto como um pacote (package) dentro de um arquivo. Módulos armazenam diversas sub-rotinas e variáveis. Módulos foram criados para permitir o reaproveitamento de código. Assim, módulos personalizados devem ser criados quando temos sub-rotinas que podem ser aproveitadas por muitos scripts.

Como exemplo, vamos criar um módulo chamado Calculadora. Módulos devem ser salvos na extensão ".pm", logo nosso arquivo deverá ser chamado de "Calculadora.pm".

O arquivo de um módulo requer algumas informações iniciais, como: a declaração de seu nome através da palavra reservada package, as sub-rotinas que queremos reutilizar, e no final do arquivo, a linha “1;”, para que o módulo retorne true (verdadeiro) quando for utilizado.

```
package Calculadora;

use strict;
use warnings;
use Exporter qw(import);

our @EXPORT_OK = qw(soma divisao multiplicacao subtracao);

sub soma {
  my $x = $_[0];
  my $y = $_[1];
  return $x + $y;
}

sub divisao {
  my $x = $_[0];
  my $y = $_[1];
  return $x / $y;
}

sub multiplicacao {
  my $x = $_[0];
  my $y = $_[1];
  return $x * $y;
}

sub subtracao {
  my $x = $_[0];
  my $y = $_[1];
  return $x - $y;
}

1;
```

Utilizar módulos melhora a legibilidade de seus códigos. Um módulo pode ser carregado de três maneiras distintas: usando as palavras reservadas do, use ou require. Por exemplo, para carregar um módulo basta apenas utilizar as linhas “use nome_do_modulo;”.

Ao chamar cada sub-rotina é necessário indicar o nome do módulo seguido dos caracteres “::” e do nome da sub-rotina. Veja abaixo como seria o uso das sub-rotinas do módulo Calculadora.

```
use strict;
use Calculadora;

# SOMA
print "2 + 2 = ";
print Calculadora::soma(2,2);

# SUBTRACAO
print "\n10 - 7 = ";
print Calculadora::subtracao(10,7);

# DIVISAO
print "\n10 / 5 = ";
print Calculadora::divisao(10,5);

# MULTIPLICACAO
print "\n7 * 8 = ";
print Calculadora::multiplicacao(7,8);

print "\n";
```

O desenvolvimento de módulos auxilia no reaproveitamento de código, entretanto sua construção pode ser um pouco complexa. Para mais informações, consulte a documentação oficial do Perl.