Les sous-programmes et leurs spécificités en Perl

par François Lieuze (autres articles)

Date de publication : 13/07/2006

Dernière mise à jour : 13/07/2006

I - Introduction

Un sous-programme est un ensemble d'instructions regroupées de manière à être appelées à l'aide d'une seule instruction. Les sous-programmes sont souvent nommés. Ils peuvent avoir des paramètres qui sont des valeurs d'entrées qu'on leur fournit en les appelant. Ils peuvent également retourner une valeur à la fin de leur appel. On peut introduire des distinctions entre les sous-programmes : ceux qui retournent une valeur et ceux qui n'en retournent pas. Les premiers sont appelés des fonctions, les seconds des procédures.

Certains langages font la distinction entre procédure et fonction, ce n'est pas le cas de Perl. Aussi, pour la suite de ce tutoriel, j'utiliserais les termes sous-programmes et fonctions indifféremment.

C'est en réalité une petite erreur d'appeler un sous-programme une fonction. En effet, à l'origine le terme fonction désignait les fonctions internes de Perl, comme print. Mais le terme de fonction est tellement utilisé pour désigner un sous-programme et surtout est tellement plus court (n'oubliez pas que la paresse est une des vertus du programmeur selon Larry Wall, créateur du langage), je me permettrais de l'utiliser.

Dans la suite de ce tutoriel, je vais vous apprendre comment définir et appeler une fonction en Perl.

II - Déclaration

II-1 - Sans paramètres

En Perl, les fonctions sont introduites par le mot clé sub (vous souvenez-vous de ma petite erreur ?). Ainsi, pour définir une fonction qui affiche le fameux Hello World, il suffit de faire ceci :

```
sub fonc
{
 print "Hello World";
}
```

II-2 - Avec des paramètres

Maintenant, définissons une fonction qui met bout à bout deux chaînes de caractère qu'elle reçoit en paramètre et affiche le résultat.

Pour ça, Perl diffère largement de la plupart des langages classiques : il stocke les paramètres que l'on lui donne dans un tableau spécial nommé @_ . Ainsi, pour accéder à ces paramètres, il faut accéder au tableau @_. Définissons la fonction :

```
sub concat
{
 my ($chaine1, $chaine2) = @_;
 print $chaine1.$chaine2;
}
```

Une autre façon très utilisée pour accéder aux paramètres d'une fonction et d'utiliser la fonction shift, qui supprime et renvoie le premier élément du tableau qu'elle reçoit en paramètre. Si on ne lui donne aucun paramètre, la fonction shift s'applique au tableau @_. Ainsi, il est fréquent de voir :

```
my $foo = shift, #$foo contiendra le premier élément de @_
```

N'oubliez pas qu'en Perl, les tableaux sont interpolés. Donc si vous voulez passer un tableau comme paramètre à une fonction, il y a deux solutions : les références et les prototypes (que nous verront plus tard)

II-3 - Avec une valeur de retour

Le mot clé return permet de retourner une valeur au programme appelant. Par exemple, si nous voulons que la fonction précédente renvoie les deux chaînes mise bout à bout plutôt que de les afficher, il faut faire ainsi :

```
sub concat
{
 my ($chaine1, $chaine2) = @_;
 return $chaine1.$chaine2;
}
```

Il est tout à fait possible d'utiliser une liste, un tableau ou un hash comme valeur de retour.

II-3 - Appel

Maintenant, voyons comment appeler une fonction. Pour ça, Perl s'inspire grandement du C : il suffit d'indiquer son nom et de le faire de suivre de parenthèses, à l'intérieur desquels on indique éventuellement les paramètres séparés d'une virgule. Des exemples seront plus parlant :

```
foo (); #appelle la fonction foo
foo ("Salut"); #appelle la fonction foo et lui passe la chaîne de caractère Salut
foo ($var1, $var2); #appelle la fonction foo et lui passe les variables $var1 et $var2;
my $res = bar($var1, $var2); #appelle la fonction bar, lui passe les variables $var1 et $var2 et
 stocke sa valeur de retour dans $res
```