Inf7214 – Langage de programmation Perl

Vladimir Makarenkov et Alix Boc

UQAM

Hiver 2010

Documentation recommandée

- Learning Perl, 5th Edition
  Tom Phoenix, Randal L. Schwartz and B.D. Foy
  Publisher: O'Reilly, 2008

- Cours d'introduction au langage Perl en français (DESS TIMH):
  http://www.med.univ-rennes1.fr/~poulique/cours/perl/perl_html/introperl02.html

- Cours du Perl plus avancé en français (François Dagorn et Olivier Salaun):
  http://perso.univ-rennes1.fr/francois.dagorn/perl

- Les diapos du cours seront disponibles sur mon site web:
  http://www.info2.uqam.ca/~makarenv/INF7214.html

Plan

- Introduction au langage Perl
- Structure d’un programme Perl
- Les variables
- Les tableaux
- Les tableaux associatifs
- Les structures de contrôles
- Les boucles
- Les entrées / sorties
- Exercices pour chacun des concepts vus

Introduction au langage PERL

P.E.R.L. signifie Practical Extraction and Report Language
(langage pratique d'extraction et d'édition)

- Créé en 1986 par Larry Wall, pour gérer un système de « News » entre deux réseaux.

- C'est un langage interprété :
  - pas de compilation
  - moins rapide qu'un programme compilé

- **Portable** : existe sous Unix, Linux, Windows, Mac (Amiga, Atari ...)

- **Gratuit** : disponible sur Internet

- **Simple** : quelques commandes permettent de faire ce que fait un programme de 500 lignes en C.

- **Robuste** : pas d'allocation de la mémoire à manipuler, chaînes, piles, pas de limite pour la taille des variables et des tableaux...

Exemples d'un programme Perl

```
1 #!/bin/perl
2
3 print "Bonjour le monde\n";
```

Commentaire obligatoire : indique l'interpréteur Perl

Commande print : afficher à l'écran

```
1   #!/bin/perl
2
3   open(FL,'< monFichier') || die "Problème d'ouverture : $! \n";
4
5   my $i = 0;
6
7   while (my $ligne = <FL>) {
8       $i++;
9    }
10
11  close FL;
12
13  print "Nombre de lignes : $i\n";
```

Ouverture d'un fichier en lecture

Détection d'erreur

Déclaration et initialisation du compteur

Pour chaque ligne lue….

Incrémentation du compteur

Fermeture du fichier

Affichage du contenu du compteur

Exécution d'un programme Perl

Les programmes en Perl ne sont pas compilés, ils sont simplement interprétés par l'interpréteur Perl.

L'usage veut que l'on nomme un programme Perl en utilisant l'extension **.pl**

Exemple :

Soit le fichier : bonjour.pl

```
1 #!/bin/perl
2
3 print "bonjour le monde\n";
```

Il est exécuté en faisant :

>perl bonjour.pl

```
PBG4:~/inf7212_cours09 dcarrey$ perl bonjour.pl
bonjour le monde
PBG4:~/inf7212_cours09 dcarrey$
```

Programme « Hello, world! »

Exemple 1

```
#!/usr/bin/perl
print "Hello, world!\n"; # affiche Hello, world!
```

Exemple 2

```
#!/usr/bin/perl
print # C’est un commentaire
"Hello, world!\n"
; # N’écrivez pas votre code Perl de cette façon
```

Exécution du programme

```
$ perl my_program.pl
```

#Sous Unix et Linux, on ajoute:

```
$ chmod a+x my_program.pl
```

ou

```
$ chmod 755 my_program.pl
```

Les types de données en Perl

1. **Les constantes**

5, -12345, 0.1415, 1.6E16, ‘cerise’, ‘a’, ‘les fruits du palmier sont les dattes’

1. **Les variables scalaires**

Les variables sont précédées du caractère $.

$i = 0; $c = ‘a’; $mon_fruit_prefere = ‘kiwi’;

$racine_carree_de_2 = 1.41421;

$chaine = ‘100 grammes de $mon_fruit_prefere’; # 100 grammes de $mon_fruit_prefere

$chaine = "100 grammes de $mon_fruit_prefere"; # 100 grammes de kiwi

Attention: Ne mettez pas d’espaces dans les noms des variables.

Un nom peut être aussi long qu’on le veut. Dans les dernières versions de Perl,

les noms de variables peuvent être accentués.

Constantes numériques

Exemples

```
1.25
255.000
255.0

7.25e45         # 7.25 fois 10 puissance 45 (un grand nombre)
-6.5e24         # -6.5 fois 10 puissance 24 (un grand nombre négatif)
-12e-24         # -12 fois 10 puissance -24 (un très petit nombre négatif)
-1.2E-23        # une autre façon de représenter E (lettre majuscule)

61298040283768
61_298_040_283_768

0377            # 377 octal, la même chose que 255 décimal
0xff            # FF hex, la même chose que 255 décimal
0b11111111      # aussi, la même chose que 255 décimal
```

Chaînes de caractères et opérations associées

Exemples

```
'barney‘          # le mot barney
' '               # la chaîne nulle (pas de caractères)
"barney"          # la même chose que 'barney'
"hello world\n"   # hello world, et une nouvelle ligne
'\'\\'            # guillemet simple suivi du backslash
```

Exemples d’opérations associées à des chaîne de caractères

```
"hello" . ' ' . "world"      # la même chose que 'hello world'

"fred" x 3                   # c’est "fredfredfred"
"barney" x (4+1)             # "barney" x 5 ou "barneybarneybarneybarneybarney"
 5 x 4                       # est en réalité "5" x 4, ce qui est "5555"

"Z" . 5 * 7                  # la même chose que "Z" . 35 ou "Z35"
```

Les types de données en Perl : exemple d'execution

Exemple de programme illustrant l'utilisation des variables. Le type de données est défini par l'interpréteur Perl lors de l'affectation (à l'exécution du programme).

Programme:

```
1 #!/bin/perl
2
3 $monNom = 'Alix' ;    # chaine de caracteres
4 $monAge = 31 ;        # entier
5 $maMoyenne = 4.15 ;   # reel
6
7 print "bonjour, je m'appelle $monNom, j'ai $monAge ans, et j'ai eu $maMoyenne de moyenne\n";
```

Donne à l'exécution:

```
PBG4:~/inf7212_cours09 dcarrey$ perl variables.pl
bonjour, je m'appelle Alix, j'ai 31 ans, et j'ai eu 4.15 de moyenne
```

3. Les tableaux

Les tableaux peuvent être utilisés comme des ensembles ou des listes. Il sont précédés du caractère @ :

```
@chiffres = (1,2,3,4,5,6,7,8,9,0);

@fruits = (‘amande’,’fraise’,’cerise’);

@alphabet = (‘a’..’z’);
```

On fait référence à un élément du tableau par son indice :

```
print $chiffre[1];  => ‘2’

print $fruits[0]; => ‘amande’
```

On peut affecter un tableau à un autre tableau :

```
@alphanum = (@alphabet,@chiffre); => (‘a’,’b’,….,’z’,’1’,’2’,…..,’0’);
```

Remarque : on dispose d’une variable spéciale : $#tableau qui indique le dernier indice du tableau (égale à sa taille - 1) : $fruits[$#fruits] => ‘cerise’

On peut référencer une partie d’un tableau :

```
@fruits[0..1] => (‘amande’,’fraise’);
```

Les tableaux (2)

3. **Les tableaux (suite)**

Accès au dernier indice et élément d'un tableau :

```
$rocks[0] = 'bedrock';      # le 1-er élément du tableau ...
$rocks[99] = 'schist';      # le dernier élément du tableau ...

$end = $#rocks;                         # 99, accès au dernier indice du tableau
$number_of_rocks = $end + 1;    # nombre d'éléments du tableau
$rocks[ $#rocks ] = 'hard rock';    # le dernier élément du tableau

$rocks[ -1 ] = 'hard rock';    # accès au dernier élément avec -1
$dead_rock = $rocks[ -3 ];   # on obtient 'bedrock' – le 1-er élément du tableau
$rocks[ -200 ] = 'crystal';    # erreur fatale!
```

Les tableaux (3)

3. **Les tableaux (suite)**

Quelques exemples :

```
$fred[0] = "yabba";
$fred[1] = "dabba";
$fred[2] = "doo";
$number = 2.71828;
print $fred[$number - 1]; # la même chose que $fred[1]
$blank = $fred[ 142_857 ]; # un élément non-utilisé du tableau contient undef
$blanc = $mel; # la variable $mel non-utilisée donne aussi undef
$rocks[0] = 'bedrock'; # un élément du tableau ...
$rocks[1] = 'slate';# un autre élément du tableau ...
$rocks[2] = 'lava'; # et un autre ...
$rocks[99] = 'schist'; # maintenant il y a 96 éléments undef
```

Les tableaux (4)

3. **Les tableaux (suite)**

Exemples des tableaux :

```
("fred", 4.5)          # deux valeurs "fred" et 4.5
(1..5)                 # la même chose que (1, 2, 3, 4, 5)
(1.7..5.7)             # la même chose, mais les deux nombres seront tronqués
(5..1)                 # tableau vide : " .. " on compte seulement en avant
(0, 2..6, 10, 12)      # la même chose que (0, 2, 3, 4, 5, 6, 10, 12)
($m..$n)               # l'intervalle déterminé par les valeurs courantes de $m et $n
(0..$#rocks)           # les indices du tableau rock du diapo précédent
($m, 17)               # deux valeurs: la valeur courante de $m et 17
($m+$o, $p+$q)         # les sommes de 2 variables
("fred", "barney", "betty", "wilma", "dino") # tableau des chaînes de caractères
qw( fred barney betty wilma dino ) # la même chose que la ligne précédente
```

Les tableaux (5)

3. **Les tableaux (suite)**

Assignations des valeurs aux tableaux :

```
($fred, $barney, $dino) = ("flintstone", "rubble", undef);
@rocks = qw/ bedrock slate lava /;
@tiny = ( ); # le tableau vide
@giant = 1..1e5; # un tableau avec 100,000 éléments
```

Les fonctions pop, push, shift et unshift :

```
@array = 5..9;
$fred = pop(@array); # $fred sera égal à 9, @array est comme suit: (5, 6, 7, 8)
push(@array, 0); # @array @array est maintenant comme suit: (5, 6, 7, 8, 0)
push @array, 8; # @array @array est maintenant comme suit: (5, 6, 7, 8, 0, 8)
@array = qw# dino fred barney #;
$m = shift(@array); # $m sera égale à "dino", @array est comme suit: ("fred", "barney")
unshift(@array, "claude"); # @array est comme suit: ("claude", "fred", "barney")
```

Opérations sur les tableaux

Structure de contrôle foreach :

```
foreach $rock (qw/ bedrock slate lava /) {
print "One rock is $rock.\n"; # Affiche les noms de 3 éléments du tableau @rock
 }
```

Variable par défaut $_ :

```
foreach (1..10) {              # utilise  $_  par défaut
 print "I can count to $_!\n";
 }

$_ = "ABCD abcd\n";
print;                         # affiche  $_  par défaut
```

Opérations sur les tableaux (2)

Opérateur reverse :

```
@fred = 6..10;
@barney = reverse(@fred); # donne 10, 9, 8, 7, 6
@wilma = reverse 6..10; # donne la même chose, mais sans un autre tableau
@fred = reverse @fred; # ajoute le résultat dans le même tableau
reverse @fred; # Erreur ! – ne change pas @fred
```

Opérateur sort :

```
@rocks = qw/ bedrock slate rubble granite /;
@sorted = sort(@rocks); # donne bedrock, granite, rubble, slate
@back = reverse sort @rocks; # les valeurs varieront de slate à bedrock
@rocks = sort @rocks; # ajoute le résultat ordonné dans le même tableau
@numbers = sort 97..102; # donne 100, 101, 102, 97, 98, 99
```

Les tableaux : un exemple d'exécution

L'exemple ci-dessous illustre l'utilisation d'un tableau. Dans ce tableau on insère les différents éléments d'un jeu de carte.

```
 1 #!/bin/perl
 2
 3 @tableau=('01'..'10','valet','dame','roi');
 4
 5 print "Nombre d'elements : " . ($#tableau +1) . "\n";
 6
 7 for ($i=0;$i<($#tableau+1);$i++){
 8     print "$tableau[$i] ";
 9 }
10 print "\n";
```

Donne à l'exécution:

```
PBG4:~/inf7212_cours09 dcarrey$ perl variables.pl
Nombre d'elements : 13
01 02 03 04 05 06 07 08 09 10 valet dame roi
```

4. Les tableaux associatifs (indicés)

Ils sont toujours précédés du caractère % :

```
%prix = (‘amande’=>30, ‘fraise’=>9, ‘cerise’=>25);
                    ou
%prix = (amande=>30, fraise=>9, cerise=>25);
```

On référence ensuite un élément du tableau par :

```
$prix{‘cerise’}      # =25
     ou
$prix{cerise}        # =25
```

On peut ajouter des éléments à un tableau associatif :

```
%chiffre = ();
$chiffre{un} = 1;
$chiffre{deux} = 2;
```

```
print $chiffre{un};   # affiche 1
$var = ‘deux’;
print $chiffre{$var}; # affiche 2
```

Les tableaux associatifs (2)

Remarques :

On peut avoir un tableau de tableaux :

```
%saison = (
        ‘abricot’ => [‘été’],
        ‘fraise’   => [‘printemps’,’été’],
        ‘pomme’=> [‘automne’,’hiver’]);
```

Et accéder à un champs en faisant :

```
print $saison{‘fraise’}[0] ;  # affiche printemps
```

Le nombre d’éléments du tableau associé à fraise est : $#{ ${‘fraise’} } + 1

Les tableaux associatifs (3)

Dans quels cas on utilise les tableaux associatifs (key => value):

1. Association (Prénom, Nom de famille).
   Exemple: (Pierre, Bouchard).

2. Association (Nom de l'hôte, Adresse IP).
   Exemple: (arabica.info.uqam.ca, 123.45.67.89).

3. Association opposée (Adresse IP, Nom de l'hôte).
   Exemple: (123.45.67.89, arabica.info.uqam.ca).

4. Association (Mot, Nombre d'occurrences de ce mot).
   Exemple: (« fois », 5).

5. Association (Nom d'usager, Nombre de disques utilisés par cet usager).
   Exemple: (User_Makarenkov, 5).

6. Association (Numéro de permis de conduire, Nom de la personne).
   Exemple: (ABCD123456, Tremblay).

Utilisation des tableaux associatifs

Exemples:

```
$family_name{"fred"} = "astaire";  # affecte une nouvelle valeur à un élément
$bedrock = $family_name{"fred"}; # donne "astaire"; ancienne valeur est perdue


$family_name{"wilma"} = "flintstone"; # ajoute une nouvelle clef et un nouveau contenu
$family_name{"betty"} .= $family_name{"barney"}; # créé un élément si nécessaire


$granite = $family_name{"larry"}; # Si « larry » n'existe pas, donne undef


%new_hash = %old_hash; # une façon de copier des tableaux associatifs


%some_hash = ("foo", 35, "bar", 12.4, 2.5, "hello", "wilma", 1.72e30, "betty", "bye\n");
 # une autre façon de définir un tableau associatif
```

Les fonction pour traiter des tableaux associatifs

Fonctions keys et values:

```
my %hash = ("a" => 1, "b" => 2, "c" => 3);

my @k = keys %hash;      # donne le tableau de clefs
my @v = values %hash;    # donne le tableau de contenus (valeurs)
my $count = keys %hash; # donne 3, voulant dire 3 paires clef/contenu
```

Fonction each:

```
while ( ($key, $value) = each %hash ) {
print "$key => $value\n";
}

foreach $key (sort keys %hash) {
        $value = $hash{$key};
        print "$key => $value\n";
        # On pourrait éviter la variable supplémentaire $value:
        # print "$key => $hash{$key}\n";
 }
```

Utilisation typique des tableaux associatifs

Exemple:

```
$books{"fred"} = 3;
$books{"wilma"} = 1;
if ($books{$someone}) {
        print "$someone a au moins un livre emprunté.\n";
}
foreach $person (sort keys %books) { # chaque personne, dans l'ordre
        if ($books{$person}) {

                print "$person a $books{$person} livre(s) emprunté(s).\n"; # fred a 3 livres
        }
}
```

Fonctions exists et delete:

```
if (exists $books{"dino"}) {
        print "Hey, il y a une carte de bibliothèque pour dino!\n";
}
my $person = "betty";
delete $books{$person}; # Enlève la carte de bibliothèque de la $person
```

Les tableaux associatifs : exemple

Exemples de programmes montrant la manipulation des tableaux associatifs.

```perl
#!/bin/perl

%prix = ('abricot' => 10 , 'fraise' => 15 , 'pomme' => 5);

print "Prix de la pomme : $prix{'pomme'} ";
```

```perl
#!/bin/perl

%saison = ('abricot' => ['ete'],
           'fraise' => ['printemps','ete'],
           'pomme'=> ['automne','hiver']);

# affichage des saisons pour les fraises

print "Nombre de saison : " . ( $#{$saison{'fraise'}} +1 ) ."\n";
print $saison{'fraise'}[0]." \n";
print $saison{'fraise'}[1]." \n";
```

Les expressions (les nombres)

Voici un aperçu des opérateurs binaires et unaires sur les nombres.

Opérateur	*Nom*	*Exemple*
+	addition	2 + 3
-	soustraction	5 - 6
*	multiplication	2 * 5
/	division	6 / 4
%	modulo	7 % 2
**	exponentiation	2 ** 3

Opérateur	*Nom*	*Exemple*
++	Pré-incrémentation	$res = ++ $val
++	Post-incrémentation	$res = $val ++
--	Pré-décrémentation	$res = -- $val
--	Post-décrémentation	$res = $val --

Les expressions (les nombres)

Combinaison de l'opérateur d'affectation avec les autres :

Opérateur	*Exemple*	*Instruction équivalente*
=	$res = 3;	$res = 3;
+=	$res += 2;	$res = $res + 2;
-=	$res -= 4;	$res = $res - 4;
*=	$res *= 2;	$res = $res * 2;
/=	$res /= 1;	$res = $res / 1;
%=	$res %= 5;	$res = $res % 5;
**=	$res **= 2;	$res = $res ** 2;

Opérateurs de comparaison :

Opérateur	*Nom*
>	supérieur
>=	Supérieur ou égal
<	inférieur
<=	Inférieur ou égal
==	égal
!=	différent

Priorité des opérateurs et comparaison des variables

Exemples de calcul:

```
4 ** 3 ** 2          # 4 ** (3 ** 2), ou 4 ** 9 (association droite)
72 / 12 / 3          # (72 / 12) / 3, ou 6/3, ou 2 (association gauche)
36 / 6 * 3           # (36/6)*3, ou 18
```

Exemples de comparaison:

```
35 != 30 + 5         # faux
35 = = 35.0          # vrai
'35' eq '35.0'       # faux (comparaison des séquences)
'fred' lt 'barney'   # faux
'fred' lt 'free'     # vrai
'fred' eq "fred"     # vrai
'fred' eq 'Fred'     # faux
' ' gt ''            # vrai
```

Les expressions (les chaînes de caractères)

Voici un aperçu des différents opérateurs permettant de combiner des chaînes de caractères.

Opérateur	*Nom*	*Exemple*	*Valeur de l'expression*
.	concaténation	"Je " . "suis"	"Je suis"
x	duplication	"a" x 4	"aaaa"
gt	Plus grand que	"abc" gt "def"	faux
ge	Plus grand ou égal	"abc" ge "def"	faux
lt	Plus petit que	"abc" lt "def"	vrai
le	Plus petit ou égal	"abc" le "def"	vrai
ne	Non-égalité	"Suis" ne "suis"	vrai
eq	Égalité	"Suis" eq "suis"	faux

Opérateur	*Exemple*	*Instruction équivalente*
=	$res = "Salut";	-
.=	$res .= " toi ";	"Salut toi "
x=	$res x= 3;	"Salut toi Salut toi Salut toi "

Les expressions logiques

Les valeurs de vérité des expressions s'évaluent selon la table suivante :

Valeur de vérité	***Expressions ayant cette valeur de vérité***
faux	Toutes les valeurs numériques nulles : 0 0.0 0.0e3 … Les 2 chaînes de caractères suivantes : "" "0"
vrai	Toutes les autres expressions, comme par exemple 4 "allo" "0.0" …

Les expressions logiques (2)

Voici la liste des opérateurs logiques :

Opérateur	***Nom***	***Exemples***	***Valeurs des expressions***
&& (and)	*Et* logique	"abc" eq "abc" && 5 > 2	Vrai
|| (or)	*Ou* logique	"abc" ne "abc" || 5 < 2	Vrai
! (not)	*Non* logique	! ("abc" ne "abc")	Vrai

La saisie au clavier

Perl offre des moyens relativement simples à utiliser pour communiquer avec la console. Nous avons déjà vu comment écrire avec la console.

Écriture :

```
print "mon message s’affiche dans la console";
```

La lecture s’effectue en appelant les symboles suivants : <>

Lecture :

```
$monNom = <>;
```

Pour être plus précis, on lit dans le STDIN (l’entrée standard) et on écrit dans le STDOUT (la sortie standard).

Écriture : print STDOUT "mon message s’affiche à la console";

Lecture : $monNom = <STDIN> ;

La saisie au clavier (2)

$madonna = <STDIN>; # on lit la ligne courante sur l'entrée standard

@madonna = <STDIN>; # on lit toutes les lignes restantes sur l'entrée standard

La lecture peut se faire depuis un fichier, par exemple.

La commande Ctrl-Z indique la fin du texte saisi au clavier sous DOS et Windows.

Chaque ligne est retournée comme un élément distinct du tableau.

Exemple:

```
@lines = <STDIN>;          # lire toutes les lignes sur STDIN
chomp(@lines);             # éliminer les sauts de lignes
```

ou tout simplement:

```
chomp(@lines = <STDIN>); # Lire les lignes sans les sauts de lignes
```

Exercices (1)

1. Écrire un programme qui calcule la circonférence d'un cercle avec le rayon de 12.5. La circonférence est 2π fois le rayon de la circonférence (la valeur de π est approximativement 3.141592654).

2. Modifiez le programme précédent pour permettre à l'utilisateur du programme de saisir la valeur du rayon.

3. Modifiez le programme précédent en sorte que si l'utilisateur saisit le nombre inférieur à 0, la valeur reportée sera 0 (plutôt que négative).

4. Écrire un programme qui lit les valeurs de 2 nombres et affiche à l'écran leur produit.

5. Écrire un programme qui lit une chaîne de caractères et un nombre et affiche, sur les lignes séparées, la chaine en question le nombre de fois indiqué par le nombre saisi. Si l'utilisateur saisit "Fred" and "3", la sortie sera donc:
   Fred
   Fred
   Fred

Coder ces programmes à la démo !!