Inf7212 – Langage de programmation Perl

Vladimir Makarenkov et Alix Boc

UQAM

Automne 2012

Syntaxe générale

Chaque instruction doit être terminée par un point-virgule. Un passage à la ligne ne signifie pas une fin d'instruction.

```
my $a = 'a'
print $a;
```

il manque ';'

Ce programme est erroné !

```
my $a = 'Une très longue chaîne de caractères qui ne peut s'écrire sur
 une seule ligne';
```

OK !

Les commentaires commencent par un #.
Tout le reste de la ligne est considéré comme un commentaire.

```
#voici un commentaire
my $a = 3;  # et en voici un autre
```

Un bloc est un ensemble de commandes entourées par des accolades, chaque commande étant suivie d'un point-virgule. Les variables déclarées à l'intérieur d'un bloc sont invisibles ailleurs.

Soyons rigoureux (2)

Il est fortement recommandé d'utiliser les modules **strict** et **warnings**. Le fait de les inclure au début du programme vous oblige à déclarer toutes les variables que vous utilisez (à l'aide de **my**).

```
$mavariable = 5;
print $Mavariable;
```

Le m est en majuscule dans la seconde ligne, et n'affichera donc pas 5 !

```
use strict;
use warnings;

my $mavariable=5 ;  # l'usage de my devient obligatoire
print $Mavariable;
```

Vous aurez le message : « Global symbol "$Mavariable" requires explicit package name »

Soyons rigoureux (2)

L'ajout des modules **strict** et **warnings** oblige la définitions des variables avant de les utiliser.

```
1 #!/bin/perl
2 use strict;
3 use warnings;
4
5 my $monNom = 'Alix' ;    # chaine de caracteres
6 my $monAge = 31 ;        # entier
7 my $maMoyenne = 4.15 ;   # reel
8
9 print "bonjour, je m'appelle $monNom, j'ai $monAge ans\n";
```

La saisie au clavier (3)

La fonction **chomp**($variable) permet de supprimer le saut de ligne à la fin d'une chaîne de caractères.

```
 1 #!/bin/perl
 2 use strict;
 3 use warnings;
 4
 5 my $monNom;
 6 my $monAge;
 7
 8 print "Ton nom : ";
 9 $monNom = <>;
10 chomp($monNom);
11 print "Ton age : ";
12 $monAge = <>;
13 chomp($monAge);
14
15 print "bonjour, je m'appelle $monNom, j'ai $monAge ans\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

La table ASCII

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

:

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 valeur undef

La valeur **undef** est associée à toute variable qui n'a pas été initialisée. Dans les expressions arithmétiques cette **undef** prend la valeur de 0.

Exemple:

```
# La somme des nombre impaires
$n = 1;
while ($n < 10) {
        $sum += $n;
        $n += 2;        # On passe à un nombre impaire suivant
}
print "Le total est: $sum.\n";
```

La variable $sum n'a pas été initialisée au départ.

Les expressions conditionnelles

Syntaxe	*Exemple*

if (expression) {
  bloc;
}
	if ($prix {'datte'} > 20) {
	  print 'Les dattes sont un peu chères';
	}

if (expression) {
  bloc 1;
} else {
  bloc 2;
}
	if ($fruit eq 'fraise') {
	  print 'Parfait' ;
	} else {
	  print 'On veut la fraise';
	}

if (expression) {
  bloc 1;
} elseif {
  bloc 2;
} elseif {
  bloc 3;
}
	if (($fruit eq 'cerise') or ($fruit eq 'fraise')) {
	  print 'rouge' ;
	} elseif ($fruit eq 'banane') {
	  print 'jaune';
	} elseif ($fruit eq 'kiwi') {
	  print 'vert'
	} else {
	  print 'je ne sais pas';
	}

Les expressions conditionnelles (2)

Remarque:

Il existe une autre notation : commande **if** (expression)

Ex : **print** ‘les dattes sont un peu chères’ **if** ($prix{‘datte’} > 20);

On peut utiliser la condition inversée:

```
unless (expression) {
        block;
}
```

Si la valeur de l’expression est FAUX alors le bloc sera exécuté!

Les boucles

Tant que :

Syntaxe	*Exemple*

while (expression) {
  bloc;
}
	my $mon_argent = 100;
	while ($mon_argent > $prix{‘cerise’}) {
	  $mon_argent -= $prix{‘cerise’};
	  print ‘Et un kilo de cerise !’;
	}

Différentes conditions d’arrêt :

Syntaxe	*Exemple*

do {
  bloc;
} while (expression);
	# Recherche du premier fruit <= 10 $
	my $i = 0; my $f;
	do {
	  $f = $fruits[$i];
	  $i++;
	} while ($prix($f) > 10);
	print "Je prends : $f ";

do {
  bloc;
} until (expression)
	my $i = 10;
	do {
	  print $i;
	  $i--;
	} until ($i == 0);

Les boucles

Boucle « pour » :

Syntaxe	*Exemple*

for (init ; condition ; cmd) {
  bloc;
}
	for ($i=0 ; $i <= $N; $i++) {
	  print $fruit[$i];
	}

Boucle « pour tout » :

Syntaxe	*Exemple*

foreach element (tableau) {
  bloc;
}
	foreach $f (@fruits) {
	  print $f;
	}

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)

Les tableaux (suite)

Accès au dernier indice et au dernier 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)

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)

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)

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
```

Exercices (2)

1. Écrivez un programme qui lit une liste des chaînes de caractères saisie au clavier (sur des lignes séparées) et l'imprime dans l'ordre inverse. Tapez Ctrl-D sous Unix ou Ctrl-Z sous Windows pour arrêter la saisie.

2. Écrivez un programme qui lit une liste des chaînes de caractères saisie au clavier (sur des lignes séparées) et l'imprime ensuite dans l'ordre «ASCIIbetique». Par exemple, si les chaînes lues sont: *fred, barney, wilma, betty*, la sortie doit être: *barney betty fred wilma*. Toutes les chaines doivent être situées sur la même ligne à la sortie. Quelle est la solution alternative ?

3. Écrivez un programme qui lit la liste des nombres (sur des lignes séparées) et puis affiche la liste des carrés de ces nombres triée dans l'ordre croissant.

Coder ces programmes à la démo !!