Inf7214 - Langage de programmation Perl (partie 2)

Vladimir Makarenkov et Alix Boc

UQAM

Hiver 2010

Plan

- Introduction au langage Perl
  - Les boucles
  - Les operateurs conditionnels
  - Le travail avec les fichiers
  - Les variables spéciales
  - Les fonctions système
  - Les fonctions sur les chaînes de caractères
  - Les fonctions sur les tableaux
  - Les fonctions (*subroutines*)
- Exercices

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";
```

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:

```
$string .= "Plus de texte\n";

# 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";
```

Les variables $string et $sum n'ont pas été initialisées 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 ($fruit eq 'banane') {
	  print 'jaune';
	} elsif ($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 <= $#fruits; $i++) {
	  print $fruit[$i];
	}

Boucle « pour tout » :

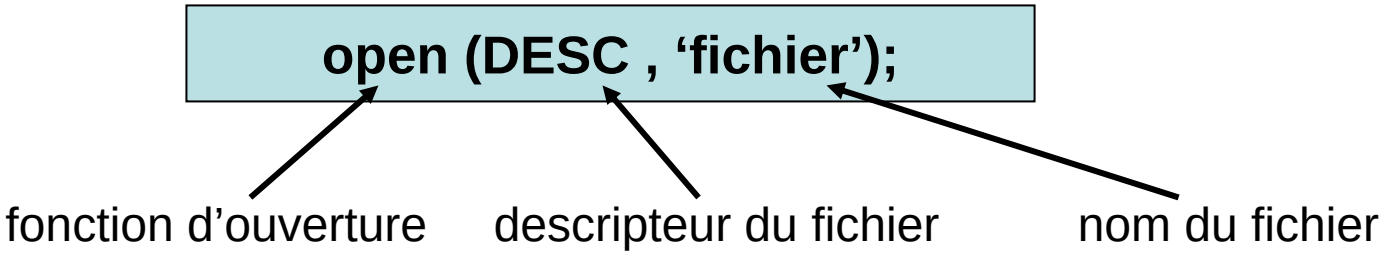
Syntaxe	*Exemple*

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

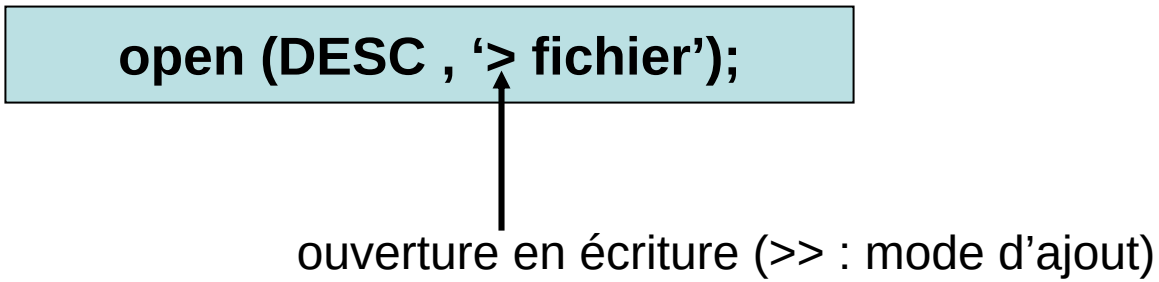
Les fichiers : ouverture

En lecture : la fonction **open** permet d'ouvrir un fichier et prend en paramètre un descripteur (l'objet permettant de manipuler le fichier) et le nom du fichier.

Format :



En écriture : il faut ajouter le symbole '>' devant le nom du fichier. Si le fichier existe, son contenu sera écrasé. En mode d'ajout, on utilise le symbole '>>'. Si le fichier n'existe pas, il sera créé.



Les fichiers : ouverture / fermeture

Gestion d'erreurs : Lorsqu'on ouvre un fichier, il se peut qu'il y ait une erreur. En lecture: le fichier n'existe pas ou ne peut pas être lu. En écriture: le fichier existe, mais on n'a pas le droit d'y écrire.

La fonction **die** permet d'afficher un message et d'arrêter le programme.

```
if ( ! open (DESC , '> fichier') ) {
        die "Problème à l'ouverture : $!";
}
```

```
open (DESC , '> fichier') || die "problème à l'ouverture : $!";
```

La variable **$!** Contient le message d'erreur système. On peut aussi afficher son contenu.

La fermeture du fichier s'effectue avec la fonction **close**.
exemple : close(DESC);

Les fichiers : lecture / écriture

La lecture dans un fichier se fait par ligne.

```
while ($ligne = <DESC>) {
    print $ligne ;
}
```

On peut aussi lire tout le fichier et le placer dans un tableau. À chaque indice du tableau, il y aura une ligne du fichier.

```
@ligne = <DESC> ;
```

L'écriture dans un fichier est encore plus simple. Il suffit de spécifier le descripteur du fichier à l'utilisation du **print**.

```
print DESC " allo le monde " ;
```

Les fichiers : lecture / écriture (2)

Quelques exemples équivalents de lecture d'un fichier:

```
while (defined($line = <STDIN>)) {
        print "I saw $line";
}
```

```
while (<STDIN>) {
        print "I saw $_";
}
```

```
while (defined($_ = <STDIN>)) {
        print "I saw $_";
}
```

```
foreach (<STDIN>) {
        print "I saw $_";
}
```

L’opérateur < >

L’opérateur <> (*diamond operator*) est utilisé en Perl pour effectuer la lecture des fichiers dont les noms sont introduits sur la ligne de commande. Si aucun fichier n’est spécifié sur la ligne de commande, l’entrée se fait de l’entrée standard <STDIN>.

La ligne de commande:

$ ./my_program fred barney betty

Deux versions équivalentes du programme qui lisent les lignes des fichiers fred, barney et betty:

```
while (defined($line = <>)) {
        chomp($line);
        print "It was $line that I saw!\n";
}

while (<>) {
        chomp;
        print "It was $_ that I saw!\n";
}
```

Les fichiers : lecture / écriture (3)

Les paramètres de ligne de commande passés au programme en utilisant le tableau ARGV:

```
@ARGV = qw # larry moe curly #;    # force la lecture de ces 3 fichiers
 while (<>) {
            chomp;
            print "On imprime $_ ! \n";
 }
```

La commande print:

```
print <>;              # équivalent à la commande 'cat' de l'UNIX
print sort <>;         # équivalent à la commande 'sort' de l'UNIX

print (2+3);           # Affiche la valeur de 5

print (2+3)*4;         # Oops!

( print(2+3) ) * 4;    # Oops!
```

La sortie formatée avec l'opérateur printf

Quelques exemples d'utilisation de printf:

```
$user = "merlyn";
$days_to_die = 3;
printf "Hello, %s; your password expires in %d days!\n", $user, $days_to_die;
# Imprime: Hello, merlyn; your password expires in 3 days!

printf "%g %g %g\n", 5/2, 51/17, 51 ** 17;      # 2.5 3 1.0683e+29

printf "in %d days!\n", 17.85;                  # imprime: in 17 days!

printf "%6d\n", 42;                             # la sortie est comme suit:    42

printf "%2d\n", 2e3 + 1.95;                     # 2001

printf "%10f\n", 6 * 7 + 2/3;                   # la sortie est comme suit: 42.666667

printf "%10.3f\n", 6 * 7 + 2/3;                 # la sortie est comme suit:      42.667

printf "%10.0f\n", 6 * 7 + 2/3;                 # la sortie est comme suit:          43
```

Les identifiants des fichiers

Les identifiants réservés par Perl:

STDIN, STDOUT, STDERR, DATA, ARGV et ARGVOUT.

Quelques exemples d'ouverture de fichiers:

```
open CONFIG, "dino";        # ouverture du fichier dino
open CONFIG, "<dino";       # ouverture du fichier dino en lecture
open BEDROCK, ">fred";      # ouverture du fichier dino en écriture
open LOG, ">>logfile";      # ouverture du fichier dino en ajout
```

Quelques exemples d'ouverture de fichiers :

```
open CONFIG, "<", "dino";
open BEDROCK, ">", $file_name;
open LOG, ">>", &logfile_name( );
```

Fermeture de fichiers:

```
close BEDROCK;
```

Les fonctions die et warn

La fonction **die** permet d'arrêter le programme et d'afficher un message d'erreur (celui d'utilisateur):

```
if ( ! open LOG, ">>logfile") {
        die "Cannot create logfile !";
 }
```

(ou celui du système d'exploitation):

```
if ( ! open LOG, ">>logfile") {
        die "$!";
 }
```

La fonction **warn** permet aussi d'afficher un message d'erreur, mais n'arrête pas l'exécution du programme:

```
if ( ! open LOG, ">>logfile") {
        warn "Cannot create logfile !";
 }
```

Le changement de la sortie par défaut

Un identifiant de fichier peut être utilisé avec la commande print:

```
print LOG "Captain's log, stardate 3.14159\n";          # la sortie ira au fichier LOG
printf STDERR "%d percent complete.\n", $done/$total * 100;
```

Cette syntaxe est aussi correcte (mais pas recommandée):

```
printf (STDERR "%d percent complete.\n", $done/$total * 100);
printf STDERR ("%d percent complete.\n", $done/$total * 100);
```

Utilisation d’un autre identifiant de sortie par défaut au lieu de STDOUT:

```
select BEDROCK; # la commande select permet de changer l’identifiant par défaut
print "I hope Mr. Slate doesn't find out about this.\n";
print "Wilma!\n";
```

Les fichiers : exemple

```
 1 #!/usr/bin/perl
 2
 3 use strict; use warnings;
 4
 5 open (IN,'input.txt') || die " Erreur lors de l'ouverture du fichier : $!";
 6 open (OUT,'>output.txt') || die "Erreur lors de l'ouverture du fichier : $!";
 7
 8 while (my $ligne = <IN>){
 9     print OUT $ligne;
10 }
11
12 close(IN);
13 close(OUT);
```

Ce programme effectue la copie du fichier *input.txt* vers le fichier *output.txt*.

Les variables spéciales et tableaux spéciaux

Petit récapitulatif des variables spéciales: ce sont les variables sous la forme $c (où c est un caractère non-alphabétique). Les tableaux spéciaux commencent par le caractère @ ou %.

Variable	***Description***
$_	variable par défaut courante (par exemple dans foreach)
$!	dernière erreur (est utilisée pour la détection d'erreurs)
$0	nom du programme
@_	contient les paramètres passés à une fonction
@ARGV	tableau des arguments passés au programme
%ENV	tableau des variables d'environnement
@INC	tous les répertoires Perl contenant des librairies

La ligne de commande

Comme en Java, et dans la plupart des langages de programmation, on peut passer des paramètres au programme sur la ligne de commande. En Perl, ces paramètres sont lus dans la variable spéciale @ARGV.

Exemple : (soit le programme copie.pl)

```
 1 #!/usr/bin/perl
 2
 3 use strict; use warnings;
 4
 5 open (IN,$ARGV[0]) || die " Erreur lors de l'ouverture du fichier $ARGV[0] : $!";
 6 open (OUT,"> $ARGV[1]") || die "Erreur lors de l'ouverture du fichier $ARGV[1] : $!";
 7
 8 while (my $ligne = <IN>){
 9     print OUT $ligne;
10 }
11
12 close(IN);
13 close(OUT);
```

Appel : > perl copie.pl input.txt output.txt

Les fonctions système

Nom	*Description*	*Exemple*
print	Affichage d'une chaîne	print 'bonjour le monde';
die	Arrêt du programme en affichant un message	if ($fruit -ne 'fraise') { die ' dommage !'; }
exit	Arrêt du programme.	if ($fruit ne 'fraise') { exit; }
system	Exécute un programme du système	system 'mkdir inf7212';
sleep n	Le programme « dort » pendant n secondes.	sleep 100;

Les fonctions sur les tableaux associatifs

Nom	*Description*	*Exemple*

each	Donne les couples clé/valeur d'un tableau indicé.	while (($fruit,$valeur) = each(%fruit)) {
		  print "kilo de $fruit : $valeur";
		}

values	Toutes les valeurs (contenus) d'un tableau indicé.	print 'les prix:', join (', ', values(%prix));
keys	Toutes les clés d'un tableau indicé.	print 'les fruits:', join (', ', keys(%prix));

exists	Indique si un élément a été défini.	if (exists $prix{'kiwi'}) {
		  print $prix{'kiwi'};
		} else {
		  print 'Je ne connais pas le prix du kiwi !';
		}

delete	Supprime un élément.	delete $prix{'cerise'};

Les fonctions mathématiques

Nom	*Description*	*Exemple*
sin	sinus	$res = sin (0) ;
cos	cosinus	$res = cos ($val) ;
log	logarithme	$res = log (20) ;
int	valeur entière	$res = int (10.2) ; => 10
sqrt	racine carrée	$res = sqrt (20) ;
rand(n)	entiers pseudo-aléatoire	$res = rand(100) ;
abs	valeur absolue	$res = abs (-10) ; => 10

Les fonctions sur les chaînes de caractères

Nom	*Description*	*Exemple*
chop	Enlève le dernier caractère de la chaîne.	$ch = ‘cerises’; chop($ch); => cerise
chomp	Enlève le dernier caractère de la chaîne si c’est un saut de ligne.	
length	Retourne la longueur de la chaîne.	$res = length (‘cerise’); => 6
uc	Retourne la chaîne en majuscule.	$res = uc (‘cerise’); => CERISE
lc	Retourne la chaîne en minuscule.	$res = lc (‘POIRE’); => poire
lcfirst	Retourne la chaîne avec le premier caractère en minuscule.	$res = ucfirst (‘La cerise’); => la cerise
ucfirst	Retourne la chaîne avec le premier caractère en majuscule.	$res = ucfirst (‘la cerise’); => La cerise
split	Sépare la chaîne en plusieurs éléments selon un séparateur.	@res = split(‘/’, ‘amande/fraise/cerise’); => (amande,fraise,cerise)
substr	Retourne une sous-chaîne; l’indice de départ et le nombre de caractères sont spécifiés.	$res = substr(‘pomme’,0,3); => ‘pom’
index	Retourne la position d’une sous-chaîne.	$res = index(‘le temps des cerises,’des’); => 9

Les fonctions sur les tableaux

Nom	*Description*	*Exemple*
grep	Recherche une expression dans un tableau.	if (grep(/$f/,@fruits)) { print ‘fruit connu’;}
join	Regroupe les éléments d’un tableau dans une chaîne.	print join(‘,’ , @fruits); => Affiche : ‘amande,fraise,cerise’
pop	Retourne le dernier élément du tableau.	print pop(@fruits); => affiche 'cerise', @fruits devient ('amande','fraise')
push	Ajoute un élément en fin de tableau.	push(@fruits, 'abricot');=> @fruits devient ('amande','fraise','abricot')
shift	Retourne le premier élément du tableau.	print shift(@fruits)=> Affiche 'amande', @fruits devient ('fraise','abricot')
unshift	Ajoute un élément en début de tableau.	unshift (’pomme', @fruits);=> @fruits devient (’pomme', 'fraise','abricot')
sort	Trie le tableau en ordre "ASCII-bétique".	@fruits = sort (@fruits);=> @fruits devient ('abricot', 'fraise’,’pomme’)
reverse	Inverse le tableau	@fruits = reverse (@fruits);=> @fruits devient ('pomme’, 'fraise', 'abricot')
Splice (tableau, début, nb)	Enlève *nb* éléments du tableau à partir de l’indice début.	@derniers = splice (@fruits, 0, 1); => @derniers devient(’fraise', 'abricot') @fruits devient (’pomme')

Un exemple de programme

Exercice : que fait-il ce programme ?

```
1 #!/usr/bin/perl
2
3 use strict; use warnings;
4
5 my @tab = (1..49);
6 my $i=0;
7
8 print splice(@tab, int (rand($#tab+1)),1) . "\n" while ($i++ < 7 );
```

Un exemple bioinformatique

```
#!/usr/bin/perl -w
# Recherche une sous-séquence de nucléotides dans une séquence d'ADN

my $target = "ACACCA";
my $search_string =
'CCACACCACACCCACACACCCACACACCACACCACACACACACACACA'.
'CATCCTAACACTACCCTAACACAGCCCTAATCTAACCCTCTCAACTTTTT'.
'ACCCTCCATTACCCTGCCTCCACTCGTTACCCTGTCCCATTCAACGAAC';
my @matches;

# Recherche une sous-séquence correspondante parmi les lettres 1-6 de la
# variable $search_string, puis parmi les lettres 2-7, et ainsi de suite. Enregistre
# l'indice de départ de chaque sous-séquence trouvée.

foreach my $i (0..length ($search_string)) {
        if ( $target eq substr( $search_string, $i, length ($target) ) ) {
                push (@matches, $i);
        }
}

print "Ma sous-séquence a été trouvée dans les positions: @matches.\n";
print "C'est la fin !\n";
```

Exercices (1)

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 etre *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 triés dans l'ordre croissant.

Coder ces programmes à la démo !!

Exercices (2)

1. Écrivez un programme qui demande à l'utilisateur son prénom et affiche à la sortie son nom de famille. Utilisez les noms des gens que vous connaissez.

2. Écrivez un programme qui lit des séries de mots (un mot par ligne) jusqu'à la fin de la saisie et puis imprime un résumé indiquant combien de fois chaque mot a été entré. Si l'entrée était: fred, barney, fred, dino, wilma, fred (tous les mots entrés sur les lignes séparées), la sortie doit être : fred – 3 fois, barney – 1 fois, dino – 1 fois, wilma – 1 fois. Pour complexifier la tache, affichez les mots en question dans l'ordre alphabétique.

3. Écrivez un programme qui fonction comme la commande *cat* de UNIX mais imprime le contenu d'un fichier dans l'ordre inverse (certains systèmes d'exploitation ont une commande de ce genre appelée *tac*). Si vous lancez votre programme *tac.pl* de cette façon (sur la ligne de commande)
   tac fred barney betty,
   la sortie doit être comme suit:
   le fichier *betty* imprimé dans l'ordre inverse (en commençant par la dernière ligne), puis le fichier *barney,* et puis le fichier *fred*, les deux derniers fichiers dans l'ordre inverse également.

Coder ces programmes à la démo !!

Exercice (tri bulle)

Écrire une fonction permettant d'effectuer le tri bulle d'entiers.

Écrire un programme permettant de lire sur la ligne de commande les valeurs à trier, puis d'effectuer le tri.

Pseudo-code de l'algorithme du tri bulle :

```
PRODECURE Tri_bulle (Tableau a[1:n])
        VARIABLE permut : Booleen;
        REPETER
                permut = FAUX
                POUR i VARIANT DE 1 à N-1 FAIRE
                        SI a[i] > a[i+1] ALORS
                                echanger a[i] et a[i+1]
                                permut = VRAI
                        FIN SI
                FIN POUR
        TANT QUE permut = VRAI
FIN PROCEDURE
```