# Université Gustave Eiffel —— -2020/2021 -

# Perl



Sylvain Lhullier contact@formation-perl.fr https://formation-perl.fr/

La reproduction et/ou diffusion de ce document, même partielle, quel que soit le support, numérique ou non est strictement interdite sans autorisation écrite des avants droit

))(

— Perl —

# ) (

3

#### Pourquoi Perl?

- Langage libre évoluant depuis 30 ans
- Financé par sa communauté
- Ouverture d'esprit communauté Perl
- Portabilité, rétro-compatibilité
- Grande richesse des bibliothèques

Pyhon est un bon langage

C'est un bon choix pour peu que vous soyez à l'aise avec



Développeurs Pytho

Développeurs Perl et Python comparés

• Prédilection pour les données textuelles (à la mode ...) : base de données, réseaux, admin système, formats, textes ...

Glue générale entre presque tout Couteau suisse de l'informatique

Satisfaire les 3 vertus du programmeur selon Lary Wall (créateur de Perl) : la paresse, l'impatience, l'orgueil ⇒ être un fainéant efficace :-)



Développeur Perl en plein travail

) (

— Perl —



# Sylvain Lhullier

Ancien étudiant de l'université (1994-2000) : DEUG  $\rightarrow$  2 ans de thèse Enseignements depuis 1998, formations professionnelles depuis 2001. Depuis 2001: dév, formation, infra, conduite projet, management À ce jour : management d'équipes et direction de projets.

Auteur du «Guide Perl» https://formation-perl.fr/guide-perl.html initialement paru dans Linux-Magazine en 7 articles puis en hors-série.

Membre des Mongueurs de Perl.

Téléchargement des supports : http://ent.u-pem.fr/ Section FOAD https://elearning.u-pem.fr/

© Sylvain Lhullier

Diffusion et reproduction interdites

2

) (

— Perl —



# Programme de l'option

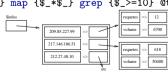
3 séances : bases du langage Perl

• structures contrôle, données, flux, doc ... tableaux dynamiques @hexa=(0..9, 'a'..'f'); tableaux associatifs natifs \$age{Bob}=28;



• programmation fonctionnelle : tri, sélection, traitement print join ', ', sort {\$a <=> \$b} map {\$\_\*\$\_} grep {\$\_>=10} @t;

• références (autovivification) \$infos->{\$ip}{requetes}++;



• expressions régulières : outil puissant d'analyse de texte if( my (m, n) = txt = m/(w+)=(d+)/) { print m n n'; } Perl=référence, aussi présentes dans C, Java, P-langages, Postfix, Apache, etc

• modularité : factorisation de code

© Sylvain Lhullier

Diffusion et reproduction interdites

© Sylvain Lhullier

Diffusion et reproduction interdites



4<sup>ème</sup> séance : bibliothèques Perl et programmation objet

• bibliothèques : web requête/analyse, courriel, SSH, LDAP, XML, SVG, PDF, ODF, YAML, JSON ...

\$p=Net::Ping->new; if(\$p->ping(\$host)){...}
\$ftn=Net::FTP->new(\$host): \$ftn=>get(\$file)

\$ftp=Net::FTP->new(\$host); \$ftp->get(\$file);
\$ua=LWP::UserAgent->new; \$ua->proxy(...);

\$res=\$ua->request(HTTP::Request->new(GET=>\$url)); #HTTP::Response

- POO native: composition, héritage, exception, surcharge opérateur ...
   accesseur polyvalent: \$pers->age(12); \$age=\$pers->age();
- POO Moose : concepts innovants (Perl 6), typage, héritage, trigger
- o délégation d'une méthode à un attribut (ex. structure Perl)
  has stationne=>(isa=>'ArrayRef[Voiture]',handles=>{garer=>'push'});
  \$parking->garer(\$voiture); # Pas de méthode à écrire
- o augmentation de méthode : pré/post-traitement (pex lors héritage/rôle) before garer => sub { # Vérification s'il y a de la place }
- $\circ$ concept de  $\mathbf{r\^{o}le}$  : mi-chemin entre interface et classe abstraite Java

© Sylvain Lhullier

Diffusion et reproduction interdites

5

).(

7





— Perl —



6<sup>ème</sup> séance : web dynamique, bases de données

- Bases de données : requêtes, transactions ...
   Sécurité : éviter les injection SQL
   On parie que vous avez déjà écrit ?
- Tests de non régression
- Benchmarking de code
- $\bullet$  Profilage de code (image)
- Optimisation de code
- Génération de fichier PDF & ODF
- Perl 6 : éléments du langage
- Ouverture à la communauté









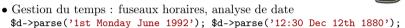


— Perl —



5<sup>ème</sup> séance : outils Perl pour le système

- Bonnes pratiques Toujours coder comme si le gars qui va finir par maintenir votre code est un psychopathe violent qui sait où vous habitez. Rick Osborne
- Appels systèmes : fichiers, processus ...
- Sockets / Threads (exclusion mutuelle aisée)
- Courriels : réception, envoi, attachement, analyse
  \$msg=MIME::Lite->new(To=>'a@dres.se',Subject=>'Ok');
  \$msg->attach(Path=>'image.jpg');
  \$msg->send();



- Gestion des paramètres de la ligne de commande prog -1 --size=14 --coord 6.4 8.1 --rgbcolor 255 0 0 fichiers
- Filtres courts: utiliser Perl en shell (rouge et non Montrouge)
  perl -i 'save/\*.bak' -pe 's/\brouge\b/blanc/' \*.txt

© Sylvain Lhullier

Diffusion et reproduction interdites

6



— Perl —



# Option Perl

Comment travailler cette option?

- En relisant les cours :
  - o faire un index des notions abordées,
  - o prendre des notes (dégager les points importants),
  - o faire une fiche synthétique (format A5?) par cours.
- $\bullet\,$  En parcourant les notions déjà vues avant de venir en cours.
- En refaisant les TD (sans le cours ?) : sur machine, sur papier.

Évaluation de l'option : examen de 2 heures sur papier (tous documents papier autorisés, ordinateurs/téléphones interdits)

© Sylvain Lhullier

Diffusion et reproduction interdites

----- Perl -----— Cours 1/6 —

# Scalaires, tableaux, fonctions





Sylvain Lhullier contact@formation-perl.fr https://formation-perl.fr/

La reproduction et/ou diffusion de ce document, même partielle, quel que soit le support, numérique ou non, est strictement interdite sans autorisation écrite des ayants droit.

— Perl 1/6: Scalaires, tableaux, fonctions —



11

#### Introduction

Practical Extraction and Report Language

Créé en 1987 par Larry Wall.

Stable: 5.32.0 / 5.30.3 Développement : 5.33

Une mineure par an. Mai/juin 2021: 5.34 / 7?

L'oignon, le symbole de Perl

Inspiration: C, shells, sed, grep, awk, lisp Langage interprété pré-compilé à l'exécution.

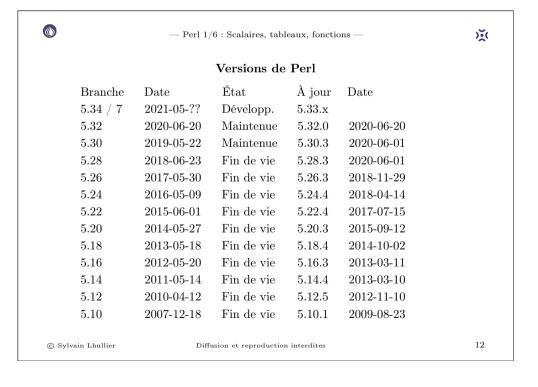
Licence de l'interpréteur : GNU GPL / Artistic License

Perl: le langage perl : l'interpréteur

There is more than one way to do it.

© Sylvain Lhullier Diffusion et reproduction interdites

 Perl 1/6 : Scalaires, tableaux, fonctions — ) • ( Plan des 4 premiers cours Perl Introduction Tables de hachage Prise en main Expressions régulières Scalaires Fichiers, entrée/sorties Structures de contrôle Références Listes et tableaux Modules Fonctions et programme Programmation objet Fonctions sur les listes 10 © Sylvain Lhullier Diffusion et reproduction interdites





# Domaines de prédilection

Ce qui fait la différence entre langages, ce n'est pas ce qu'ils rendent possible, mais ce qu'il rendent facile.

- Prédilection pour les données textuelles (à la mode!) :
  - o Bases de données
  - o Flux et protocoles réseaux : web, courriel, SNMP, LDAP, etc
  - o Administration système : logs, configuration
  - o Manipulation de formats de données : XML, CSV, etc
  - o Algo du texte : génomique, linguistique
- Passerelles applicatives / inter-SI / framework
- Autre: graphisme, système, archivage, chiffrement, GUI, etc.

# Couteau suisse du monde Linux/Unix

Aide à l'interopérabilité : glue générale entre presque tout

© Sylvain Lhullier

Diffusion et reproduction interdites

13

) • (

#### Avantages

- Un vrai langage puissant :
  - o Paradigmes: impératif / fonctionnel / orienté objet,
  - o Récursivité / modularité / exceptions / ramasse-miettes,
  - Tableaux, listes et tables de hachage natifs.
  - o Expressions régulières, support natif d'unicode,
  - o Surcharges d'opérateurs, fermetures (closures),
- Richesse des bibliothèques (efficacité de programmation).
- Ouverture d'esprit de la communauté Perl Perl beaucoup utilisé par adminsys, culture d'outils multiples ⇒ large culture de l'informatique, tolérance
- There is more than one way to do it.

© Sylvain Lhullier

© Sylvain Lhullier

Diffusion et reproduction interdites

14



— Perl 1/6: Scalaires, tableaux, fonctions —



#### Inconvénients

- Langage faiblement typé (scalaires),
- Langage très permissif, liberté de coder
  - o utilisateurs hétérogènes, existant peu recommandable parfois,
  - o exige rigueur et coordination (liberté de choisir ses contraintes),
- Riche donc pas immédiat à maîtriser
- Faible protection attributs et méthodes (nécessité sociale non technique),
- Ramasse-miettes par comptage de références (données cycliques)
- Ce n'est pas à la mode ;-) Ce n'est jamais la bonne solution technique qui est adulée : 8086 vs 68000, Windows vs Linux, The Voice vs chanteurs :-)
- There is more than one way to do it.

— Perl 1/6: Scalaires, tableaux, fonctions —



# Sources d'information web sur Perl

• CPAN : distributions et modules

https://www.cpan.org/ https://metacpan.org/

• perldelta : changements dans le langage à chaque version https://metacpan.org/pod/distribution/perl/pod/perldelta.pod

• Pour démarrer : pierres de Rosette

http://hyperpolyglot.org principaux langages 515 langages

http://rosettacode.org

• Guide Perl (S. Lhullier) https://formation-perl.fr/guide-perl.html

• Association Les Monqueurs de Perl

http://mongueurs.net/

© Sylvain Lhullier Diffusion et reproduction interdites

15

Diffusion et reproduction interdites

# Sources d'information : perldoc

perldoc perl - index

perldoc perlop – opérateurs du langage

perldoc perlfunc - fonctions du langage

perldoc perlfaq - FAQ

perldoc perlre - regexp

perldoc perlrun – options de la ligne de commande

etc

perldoc -f fonction perldoc -f sprintf

perldoc -q mots-clefs-FAQ perldoc -q 'macros for vi'

 $\verb|perldoc| module & | perldoc| Net::FTP|$ 

© Sylvain Lhullier Diffusion et reproduction interdites

17

# Bibliographie

Learning Perl / Introduction à Perl Schwartz - O'Reilly

 $\label{eq:programming Perl Programmation en Perl} Programming \ Perl \ / \ Programmation \ en \ Perl \ Wall, Christiansen \& Orwant - O'Reilly$ 



Le «camel book»

Perl best practices / De l'art de programmer en Perl Conway - O'Reilly

Donne à quelqu'un du poisson, il mangera un jour. Donne lui une canne à pêche, il mangera tous les jours. Donne lui un bouquin de référence avec un gros index, il mourra de faim en passant son temps à le consulter.

Rafael Garcia-Suarez

© Sylvain Lhullier

Diffusion et reproduction interdites

18

— Perl1/6 : Scalaires, tableaux, fonctions —



19

#### Bibliographie

Perl cookbook / Perl en action

Christiansen & Torkington - O'Reilly

Advanced Perl programming / Programmation avancée en Perl

Srinivasan - O'Reilly

Object Oriented Perl

Damian Conway - Manning Publications Company

Mastering algorithms with Perl

Orwant, Hietaniemi & MacDonald - O'Reilly

Perl testing: a developer's handbook Langworth & Chromatic - O'Reilly

Perl moderne

Krotkine, Aperghis-Tramoni, Quelin, Bruhat - Pearson

© Sylvain Lhullier Diffusion et reproduction interdites

— Perl 1/6 : Scalaires, tableaux, fonctions —



20

# Quick reference guides

http://www.ch.embnet.org/CoursEMBnet/Pages05/slides/perlref.pdf

https://michaelgoerz.net/refcards/perl\_refcard.pdf

http://www.cheat-sheets.org/saved-copy/perl\_refcard.pdf

http://johnbokma.com/perl/perl-quick-reference-card.pdf

http://www.rexswain.com/perl5.html

Regexp:

http://people.duke.edu/~tkb13/courses/ece560/resources/pregr.pdf

32 pages Perl 5:

http://www.squirrel.nl/pub/perlref-5.004.1.pdf

http://www.netzmafia.de/skripten/perl/perl-qref.pdf

Et le vôtre!

© Sylvain Lhullier Diffusion et reproduction interdites

#### Prise en main

- perl -v affiche le numéro de version
- perl -V affiche les options de compilation de l'exécutable perl, les répertoires des librairies Perl (tableau QINC) ...
- perl -d lance le débogueur
- perl -de 1 mode interactif

© Sylvain Lhullier

Diffusion et reproduction interdites

21

# Lancer un programme Perl

• Code en ligne de commande

```
perl -e 'print("Salut Larry !\n");'
   Salut Larry !
```

• Script en argument

```
fichier salut.pl
    print("Salut Larry !\n");
perl salut.pl
   Salut Larry !
```

© Sylvain Lhullier

Diffusion et reproduction interdites

— Perl 1/6 : Scalaires, tableaux, fonctions —

22

) (



— Perl 1/6 : Scalaires, tableaux, fonctions —



#### Lancer un programme Perl

• Shebang (LA bonne méthode)

```
fichier salut.pl
#!/usr/bin/perl
print("Salut Larry !\n");
```

chmod +x salut.pl ./salut.pl Salut Larry!

• Shebang pour Perl non standard

```
#!/usr/bin/env perl
print("Salut Larry !\n");
```

© Sylvain Lhullier Diffusion et reproduction interdites



23

#### Pragmas

• use strict:

Modifie la phase de compilation. Langage moins permissif: obligation de déclaration des variables, interdiction des références symboliques, vérification de l'existence des fonctions appelées ...

- use warnings; (anciennement flag -w) Modifie la phase d'exécution, affichage d'avertissements : variable non initialisée, conversion numérique partielle ...
- use diagnostics '-l=fr'; Indique à Perl de produire les messages dans leur forme longue
- use 5.010; Prise en compte des fonctionnalités expérimentales depuis 5.10

© Sylvain Lhullier Diffusion et reproduction interdites  $^{24}$ 



# Structures du langage

Point-virgule à la fin de chaque instruction : print(\$v); Espacements libres :

```
if($v==$w){print"0k\n";}if ($v == $w ) {
    print "0k\n";
```

Commentaires : depuis un # jusqu'à la fin de la ligne.

```
# Ceci est un commentaire
print("$s\n"); # Et cela aussi
```

Toujours coder comme si le gars qui va finir par maintenir votre code est un psychopathe violent qui sait où vous habitez. – Rick Osborne

© Sylvain Lhullier

Diffusion et reproduction interdites

25





— Perl 1/6 : Scalaires, tableaux, fonctions —



27

#### Structures de donnée

```
Sigil:
```

```
$ scalaire: $x=42; $y=31.8; $z='bonjour';
```

@ tableau de scalaire : @t = (3, 'vf', -84)

% table de hachage :

%h = ('paul'=>'arg','pierre'=>-84.2,'julie'=>'aussi')

structure	globalité	élément
Scalaire	\$v	
Tableau	@v	<pre>\$v[indice]</pre>
Hash	%v	<pre>\$v{clef}</pre>



Diffusion et reproduction interdites



— Perl 1/6 : Scalaires, tableaux, fonctions —



# Types de données

Simplicité, souplesse et puissance.

- Introspection,
- Structures anonymes,
- Autovivification,
- Support natif d'unicode.

# ${\bf Trois\ types:}$

- Scalaire : donnée atomique,
- Tableau (gestion dynamique et automatique de la taille),
- Table de hachage (association clef  $\rightarrow$  valeur) NB: performance.

© Sylvain Lhullier

Diffusion et reproduction interdites

26

— Perl 1/6: Scalaires, tableaux, fonctions —



#### Structures de donnée : scalaires

```
Donnée atomique : chaîne de caractères, nombre \dots
```

```
12 -3.14 "chaîne" 'chaîne' 3e9 034 0xFF
```

Variable: \$x

• déclaration :

```
my x; my x = 10;
```

Portée (visibilité) : le bloc et ses sous-blocs.

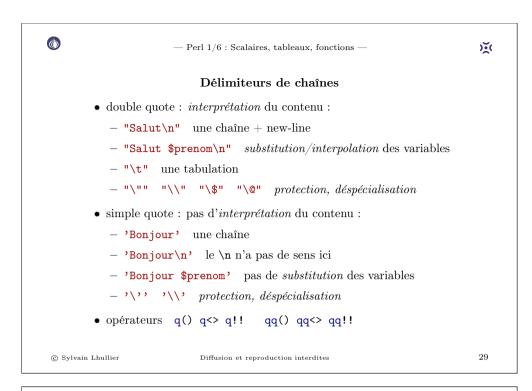
Place libre dans le bloc.

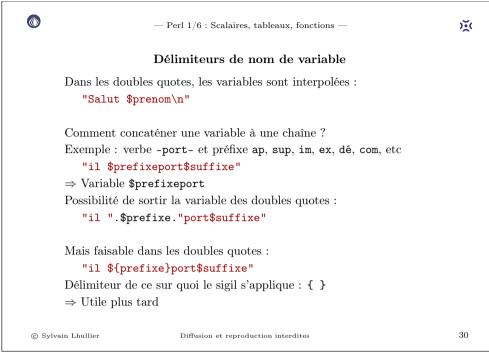
• affectation et utilisation :

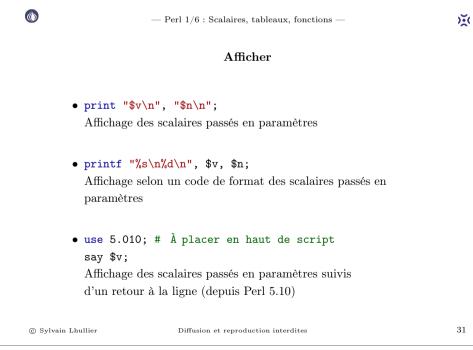
```
$x = $y + 3;
$phrase = "Bonjour $prenom !";
print("$phrase\n");
```

© Sylvain Lhullier

Diffusion et reproduction interdites







```
— Perl 1/6 : Scalaires, tableaux, fonctions —
                                                                                 ) (
                                 La valeur undef
       undef est une valeur particulière scalaire
        • signifie «non défini»
        • valeur par défaut des variables :
                 my x; \Leftrightarrow my = undef;
        • vaut 0 en contexte numérique, " en contexte de chaîne
        • affectation :
                           x = undef; ou undef(x);
                         if( defined($x) ) .....
         • test:
        • Ne pas écrire : if ( $x != undef ) ou autre
        • Opérateur // //=
© Sylvain Lhullier
                                                                                 32
                             Diffusion et reproduction interdites
```





#### — Perl 1/6: Scalaires, tableaux, fonctions —

# )<u>č</u>(

# Opérateurs

- Arithmétiques : + \* / % \*\* puissance
- Chaînes :

```
concaténation $x='bonjour'.$prenom;
x répétition $x='bon'x3 ⇒ 'bonbonbon'
```

- Fonction length
  length(\$x) renvoie le nombre de caractères de la chaîne \$x
  length('bonjour') vaut 7
- Fonctions de manipulation de la «case»:
   lc renvoie la chaîne mise en minuscule
   lcfirst renvoie la chaîne avec la lère lettre en minuscule
   uc renvoie la chaîne mise en majuscule
   ucfirst renvoie la chaîne avec la lère lettre en majuscule
   \$nom = ucfirst(lc(\$nom));

© Sylvain Lhullier

Diffusion et reproduction interdites

33

#### Raccourcis

• Raccourcis

• {In,Dé}crémenteurs

© Sylvain Lhullier

Diffusion et reproduction interdites

34



— Perl1/6 : Scalaires, tableaux, fonctions —



35

#### Fonction substr

substr(\$x, offset, length)

vaut la sous-chaîne de position offset et de longueur length

- substr('bonjour',2,3) vaut 'njo'
- length peut être omis : toute la partie droite est sélectionnée.
- 4ème paramètre : valeur de remplacement

```
my $y = 'salut toi';
substr( $y, 5, 1, 'ation à ');
⇒ $y vaut 'salutation à toi'
⇒ Perl gère lui-même la mémoire!
```

© Sylvain Lhullier

Diffusion et reproduction interdites

— Perl 1/6 : Scalaires, tableaux, fonctions —



# Typage dynamique

Les scalaires ne sont pas typés, mais les valeurs stockées le sont :

- \$x = '1';  $\Rightarrow$  stockage en chaîne (suite de caractères)
- x = 1;  $\Rightarrow$  stockage en entier (4 octets, complément à 2)
- \$x = 1.0;  $\Rightarrow$  stockage en flottant (8 octets, norme IEEE 754)

Influence sur la performance (calculs).

© Sylvain Lhullier

Diffusion et reproduction interdites



#### Notion de contexte

Valeur d'expression : dépend de la position dans le code (le contexte)

Types de contexte :

- scalaire (numérique, chaîne, booléen) : monovalué
- de liste (lire la suite) : multivalué

Par exemple, où trouve-t-on les contextes?

- booléen ? Dans les tests : if( \$x ) { ... }
- de chaîne ? Dans les concaténations : \$x . \$y
- numérique ? Dans les calculs : x + v

Règles de conversion chaîne→ numérique :

- '23.5'  $\rightarrow 23.5$
- '23.5re'  $\rightarrow$  23.5  $\Rightarrow$  message d'avertissement si use warnings;
- 're23.5'  $\to 0$ ⇒ message d'avertissement si use warnings;

© Sylvain Lhullier

© Sylvain Lhullier

Diffusion et reproduction interdites

37



— Perl 1/6: Scalaires, tableaux, fonctions —



39

#### Structures de contrôle : conditions

Diffusion et reproduction interdites

```
• if ( condition ) { code1 } ( else { code2 } )
  NB: { } obligatoires
         if( x == 0 ) {
            print "\$x nul\n";
         } else {
            print "\$x non nul\n";
  Incorrect:
          if( condition1 ) { code1 }
          else if( condition2 ) { code2 }
  Correct:
          if( condition1 ) { code1 }
          else { if( condition2) { code2} } }
```

Constantes

```
use constant DEBUG => 0:
     use constant {
           PROTO => 'http',
           PORT => 80,
     };
     print PROTO."\n" if DEBUG;
     use constant PI
                         => 4*atan2(1,1);
     # Calcule la valeur une bonne fois pour toute
     use constant JOURS => qw(lundi mardi mercredi
                    jeudi vendredi samedi dimanche);
     print 'Premier jour de la semaine : ', (JOURS)[0], ".\n";
                                                                 38
© Sylvain Lhullier
```

— Perl 1/6: Scalaires, tableaux, fonctions —

— Perl 1/6 : Scalaires, tableaux, fonctions —



) • (

#### Structures de contrôle : conditions

Diffusion et reproduction interdites

```
• elsif pour ne pas surcharger en accolades :
  if( condition1 ) { code1 }
  elsif( condition2 ) { code2 }
  elsif( condition3 ) { code3 }
  else { code4 }
         if( x == 0 ) {
            print "\$x nul\n";
         } elsif( $x < 0 ) {</pre>
            print "\$x négatif\n";
         } else {
            print "\$x positif\n";
```

© Sylvain Lhullier

Diffusion et reproduction interdites

```
— Perl 1/6 : Scalaires, tableaux, fonctions —
```

#### Structures de contrôle : conditions

- instruction if condition; print "\$s\n" if defined \$s; Pas de parenthèse obligatoire, ni d'accolade. Pas de else possible.
- unless( condition )  $\Leftrightarrow$  if(!condition)

© Sylvain Lhullier

Diffusion et reproduction interdites

41

) (

43

)•(



Opérateurs de tests

Les opérateur de tests imposent un contexte à leur opérandes.

contexte imposé	numérique	chaîne
égalité	==	eq
différence	!=	ne
infériorité	<	lt
supériorité	>	gt
inf ou égal	<=	le
sup ou égal	>=	ge
comparaison	<=>	cmp

```
if('0' == '00') if(x != undef)
if('0' eq'00') \Rightarrow if(defined x)
```

© Sylvain Lhullier

Diffusion et reproduction interdites



— Perl 1/6: Scalaires, tableaux, fonctions —



# Booléens et opérateurs logiques

Booléens : scalaires

- Valeurs fausses: '', undef, 0, '0'
- Valeurs vraies: 1, 'blabla', -4.2, '00', 0.0, etc
- if( \$x ) { ... }

Opérateur logiques, 2 familles :

- Forte priorité ! && ||
- Faible priorité not and or

```
if( (a \mid \mid b) && (c \mid \mid d) ) if( a \mid \mid b and c \mid \mid d )
```

perldoc perlop

© Sylvain Lhullier

Diffusion et reproduction interdites

42

— Perl 1/6 : Scalaires, tableaux, fonctions —



#### Structures de contrôle : boucles

• for (initialisation; condition; incrément) { code }

© Sylvain Lhullier

Diffusion et reproduction interdites

```
Structures de contrôle : boucles

• while(condition) { code }
instruction while condition;
do { code } while(condition);

my $e = 0;
while($e <= 10) {
print "$e\n";
$e++;
}

• until(condition) \( \Delta \) while(!condition)

© Sylvain Lhullier

Diffusion et reproduction interdites
```

```
— Perl 1/6: Scalaires, tableaux, fonctions —
                                                                       ).(
                        Premier exemple simple
      #!/usr/bin/perl
      use strict;
      use warnings;
                                                9 impair
      my $somme = 0;
                                                -----
      for( my $i=9; $i>0; $i-- ) {
                                                7 impair
         if( $i % 2 != 0 ) {
                                                ----
             print("$i impair\n");
                                                5 impair
                                                3 impair
         print( '-'x$i . "\n")
             if $i % 3 == 0:
                                                1 impair
         $somme += $i;
                                                45
      print("$somme\n");
© Sylvain Lhullier
                                                                       47
                         Diffusion et reproduction interdites
```

```
— Perl 1/6: Scalaires, tableaux, fonctions —
                                                                                ) • (
                       Structures de contrôle : boucles
       Sauf pour do while:
        • last : sortie de boucle
        • next : arrêt de l'exécution du bloc
           (retour à l'évaluation de la condition)
        • redo : recommence l'exécution du bloc courant
           (sans passer par l'évaluation de la condition)
              for( my $e=0 ; $e<=10; $e++ ) {</pre>
                 next if $e % 2 == 0; # ne pas traiter les pairs
                 print "$e\n":
                 last if $e == $valeur; # mettre fin à la boucle
              }
                                                                                46
 © Sylvain Lhullier
                            Diffusion et reproduction interdites
```

— Perl 1/6 : Scalaires, tableaux, fonctions —

# Type de données : liste

- liste vide : ()
- listes: (2,5,1,0,-4) ou (2,'âge',"bonjour \$prenom")
   ⇒ liste ≡ liste de scalaires
- 256 éléments : (0..255)
- 37 éléments : (1..10, 'âge', 'a'..'z')
- (\$debut..\$fin) pas toujours possible  $\Rightarrow$  liste vide
- (1,2,(1..10), 'âge',('b',6),4) ⇒ une liste à 16 éléments «Aplatissement» ou «linéarisation» des listes
- (2,3) x4  $\Rightarrow$  (2,3,2,3,2,3,2,3)

© Sylvain Lhullier Diffusion et reproduction interdites

48

) (

#### Structure de données : tableau

Tableau = mono-dimensionnel

• déclaration d'un tableau : my @t;

$$mv @t = ();$$

NB: Pas de taille à déclarer (taille dynamique)

• déclaration et initialisation d'un tableau par une liste :

@t				
0	-3	1	-2	2 -1
	23		ab	-54.4

$$my \ Ot = (4..12); # 9 valeurs$$

• accéder à un élément : \$t[i] Attention : \$

\$t[0]: premier élément, puis \$t[1] \$t[2] etc

t[-1]: dernier élément, puis t[-2] at [-3] et c

© Sylvain Lhullier

Diffusion et reproduction interdites

49





51

#### Tableau

— Perl 1/6: Scalaires, tableaux, fonctions —

⇒ les éléments intermédiaires n'existent pas • \$t[100]=6;

0	-101	1	-100	2	-99		100	-1
23	3	;	ab	-	-54.4		6	

Attention: tableau à trous, à éviter!

• On peut tester l'existence d'un élément :

```
if( exists( $t[1] ) ) { ... }
```

• \$#t : indice du dernier élément du tableau @t (affectable!)

$$#t = 12;$$

⇒ le tableau réduit de taille (perte des valeurs)

© Sylvain Lhullier

Diffusion et reproduction interdites

#### Tableau : éléments

• Éléments non définis : undef (\$t[3] et suivants, \$t[-4] et précédents)

• Ajouter un élément au tableau :

```
my @t = (23, ab', -54.4);
```

\$t[3] = 'def'; (impossible avec les indices négatifs)

- $\Rightarrow$  Allongement du tableau
- ⇒ Perl gère lui-même la mémoire

© Sylvain Lhullier

Diffusion et reproduction interdites

50

— Perl 1/6 : Scalaires, tableaux, fonctions —



#### Tableau: substitution et contexte

• Substitution entre doubles-quotes :

```
⇒ représentation des éléments séparés par un espace.
```

```
my @t = (-12.5, 'he', 0xFF);
my s = 0t; \Rightarrow -12.5 \text{ he } 255
                                               print "@t\n";
```

• En contexte scalaire (chaîne, numérique, booléen) :

```
⇒ renvoie sa taille.
```

```
my v = 0t; \Rightarrow 3
                                         @t."\n"
   my $v = scalar( @t );
                                         @t+10
                                         if( @t ) { ... }
                                         if( @t > 0 ) { ... }
print(@t."\n");
                     print("@t\n");
```

© Sylvain Lhullier

Diffusion et reproduction interdites

```
— Perl 1/6: Scalaires, tableaux, fonctions —
```



# ) • (

#### Liste et tableau : affectations et contexte

```
• Affectation et liste :
```

$$(\$x,\$y) = (1,2);$$
  
 $(\$x,\$y) = (1);$ 

$$(\$x,\$y) = (1,2,3);$$

$$(x x y) = Fonction()$$

$$(x,y) = Fonction();$$

(
$$x,$$
  $y$ ) = 1;  $\Rightarrow$  Comme le cas ( $x,$   $y$ ) = (1);

$$(x,y) = (y,x);$$

Perl ne se prend pas les pieds dans le tapis : très utile plus tard

• Affectation et tableau :

• Affectations différentes : \$x=@t: (\$x) = 0t:

© Sylvain Lhullier

Diffusion et reproduction interdites

53

# Liste/tableau: affectation et déclaration multiples

• Différence d'affectation depuis liste et tableau :

$$x = Qt;$$
  $\Rightarrow$  taille du tableau

$$x = (y, z); \Rightarrow \text{recopie de } y$$

\$x = Fonction(); ⇒ recopie du 1er élément renvoyé

• Déclaration multiple :

my 
$$x, y; \Rightarrow \text{erreur}$$

$$my (\$x,\$y) = (1,2);$$

• Copie de tableau à tableau : @t2 = @t;

© Sylvain Lhullier

Diffusion et reproduction interdites

54

) (



— Perl 1/6: Scalaires, tableaux, fonctions —



#### Liste: «aplatissement»

• Right-value :

 $\Rightarrow$  Qt2 vaut la liste (10,1,2,'age',20)

```
print(@t,"\n"); # Aplatissement
print(@t."\n"); # Contexte scalaire
```

print("@t\n" ); # Interpolation

• Left-value:

$$(\$x, @t) = @s;$$

\$x vaut le premier élément de @s

Ot «absorbe» tous les autres éléments

$$(\$x, 0t, \$y, 0t2) = 0s;$$

© Sylvain Lhullier Diffusion et reproduction interdites 55

— Perl 1/6: Scalaires, tableaux, fonctions — Manipulation de tableaux et listes

• Ajout et suppression à gauche dans un tableau :

$$- \; \text{unshift(@t,5,6);} \qquad \Rightarrow \text{@t=(5,6,1,2,3,4)}$$

$$-$$
 \$v = shift(@t);  $\Rightarrow$  \$v=1 @t=(2,3,4)

• Ajout et suppression à droite dans un tableau :

$$- \text{ push(Qt,5,6)}; \Rightarrow \text{Qt=(1,2,3,4,5,6)}$$

$$- $v = pop(@t);$$
  $\Rightarrow $v=4 @t=(1,2,3)$ 

• Inversion de liste:  $@s = reverse(@t); \Rightarrow @s=(4,3,2,1)$ 

Comment remplacer ces fonctions par des instructions du langage?

© Sylvain Lhullier

Diffusion et reproduction interdites

# Tableau: splice et @ARGV

Remplacer des éléments : splice(tableau, début, nbr, remplacement) Si pas de remplacement, suppression :

```
my @t = ('a'..'z');
@s = splice( @t, 3, 2 ); # Supprime et renvoie ('d','e')
```

Remplacer les éléments supprimés :

```
@s = splice( @t, 3, 2, ('A'..'Z') );
```

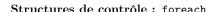
**QARGV**: tableau des arguments de la ligne de commande (nom du programme : \$0)

© Sylvain Lhullier

Diffusion et reproduction interdites

57

) • (



• foreach variable ( liste ) { code } parcours non destructif

La variable est un alias pour chaque valeur successive de la liste.

NB1: Ne pas modifier la liste lors parcours

NB2: Modifier le variable modifie la liste

```
• foreach $v (1,2,3,4) { ... }
foreach $v (@t) { ... }
foreach $v (@t,5,@t2,$x,'azerty') { ... }
foreach $v (1..10) { ... }
foreach $v (reverse @t) { ... }
```

- foreach my \$v ( liste ) { code }
  foreach (@t) { fonction( \$\_ ); } \$\_ variable par défaut
- next valeur suivante (fin de l'exécution du bloc)
  last sortie de boucle

© Sylvain Lhullier

Diffusion et reproduction interdites

58



— Perl 1/6 : Scalaires, tableaux, fonctions —



#### Création de listes avec l'opérateur qu

```
• Il peut être pénible d'écrire :

@t = ( 'Ceci', 'est', 'un', 'peu', 'ennuyeux,', 'non', '?');
```

```
• L'opérateur qw découpe une chaîne et renvoie une liste
Séparateur : les caractères espaces ou \n\t\r\f (\s+)
@t = qw(Cela est bien plus facile, non ?);
```

- @t = qw/ attention 'aux erreurs' stupides /; ⇒ 4 éléments
- foreach \$v (qw(liste de mots)) { ... }

59

— Perl 1/6: Scalaires, tableaux, fonctions —



#### Concept de fonction

Regroupement d'instruction qui peut être utilisé plusieurs fois.

Prend des valeurs en entrée (appelées paramètres) et renvoie des valeurs de retour.

Comporte des instructions et des appels à d'autres fonctions.

Deux phases : une déclaration puis des utilisations

Une même fonction peut être appelée avec des paramètres différents :

```
my $val1 = calcul( 2, 'aa' );
my $val2 = calcul( 10, 'bb' );
```

print push length sont des fonctions.

if for foreach qw ne sont pas des fonctions mais des instructions.

© Sylvain Lhullier Diffusion et reproduction interdites

60

© Sylvain Lhullier

Diffusion et reproduction interdites

```
— Perl 1/6 : Scalaires, tableaux, fonctions —
```



#### Écrire une fonction

```
    sub calcul {
        my ($nom,$age,$nbr) = @_; #arguments scalaires
        ...
        return $r;
    }
    my $val1 = calcul('Jean',42,7.5);
    my $val2 = calcul('Paul',18,-0.4);
    Si 1 seul argument : my ($x) = @_;
        → () obligatoires
    my ($x,@t) = @_;
        → @t reçoit les arguments restant
```

© Sylvain Lhullier

Diffusion et reproduction interdites

61

— Perl 1/6 : Scalaires, tableaux, fonctions —



63

# Appel de fonction

```
Utiliser calcul('Jean',42,7.5); ou calcul 'Jean',42,7.5; ?
```

Plus léger sans parenthèses, pour les fonctions de bases :

 ${\tt print\ pop\ map\ etc}$ 

Cas à problème :

```
print ($i*$j)."\n";
```

 $\Rightarrow$  la priorité n'est pas celle qu'on croit

print( (\$i\*\$j)."\n");

© Sylvain Lhullier Diffusion et reproduction interdites



— Perl 1/6: Scalaires, tableaux, fonctions —



#### Écrire une fonction

```
Appel:
    calcul('Jean',42,7.5); $v = calcul('Jean',42);
    $v = calcul('Jean',42,7.5); $v = calcul('Jean',42,7.5,'Paul');
    calcul 'Jean',42,7.5; (à éviter)
Renvoyer une liste:
    return ($y,$z);
    return @t;
    Appel:
    ($j,$k) = calcul('Jean',42,7.5);
    @s = calcul('Jean',42,7.5);
```

© Sylvain Lhullier

— Perl 1/6 : Scalaires, tableaux, fonctions —

Diffusion et reproduction interdites



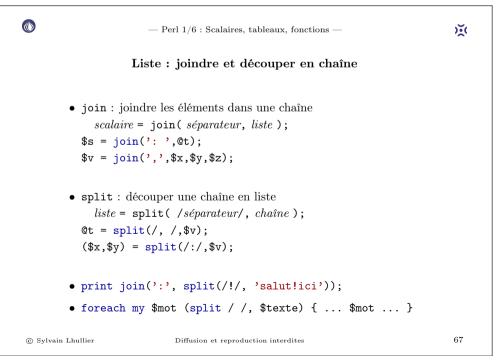
62

# Tableaux et appel de fonction

© Sylvain Lhullier

Diffusion et reproduction interdites

```
) • (
                       — Perl 1/6: Scalaires, tableaux, fonctions —
                   Prototype de fonction (expérimental)
      use v5.20:
      use feature qw(signatures);
      no warnings qw(experimental::signatures);
      sub f1( $x, @t ) { # également @_
         print "$x\n";
         print "@t\n":
      f1( 'texte', 1..10 );
      Ne règle pas le problème de l'applatissement :
       sub f2( @t, @t2 ) { ... }
       Slurpy parameter not last at fichier.pl line 12.
                                                                               65
© Sylvain Lhullier
                           Diffusion et reproduction interdites
```

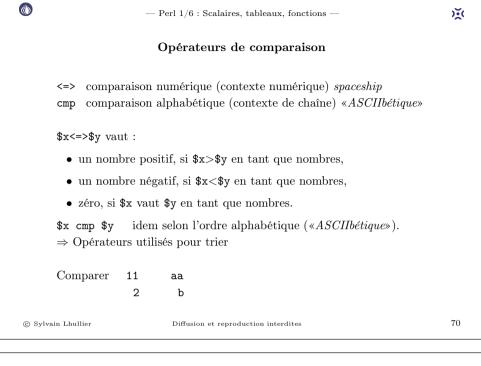


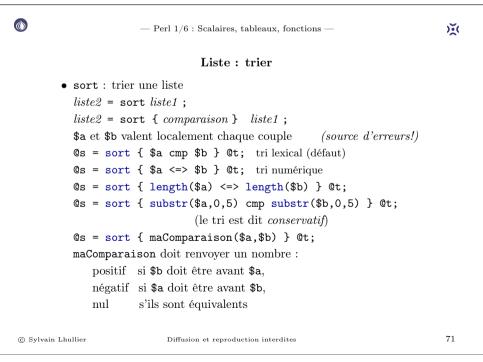
```
    Perl 1/6 : Scalaires, tableaux, fonctions —

                                                                               ) • (
                         Structure d'un programme
      #!/usr/bin/perl
                                           shebana
                                           rèales strictes
       use strict:
                                           activation des avertissements
       use warnings;
      my $t = 'Bonjour Larry';
                                           variable globale
      print "$t\n";
                                           avec ou sans ()
       sub SommeProduit {
          mv ($x,$v) = 0_{:}
                                           paramètres de la fonction
          mv $m = x*v;
                                          variable locale
         printf("%5d\n", $m);
                                           printf comme en C
                                           valeurs de retour de la fonction
         return ($x+$y,$m);
                                          NB: pas de conflit entre $t et @t
      my @t = SommeProduit(3,5);
      print "$t[0] $t[1]\n";
                                           ni avec %t ou sub t
      my @t2 = SommeProduit(2,7);
                                                                               66
© Sylvain Lhullier
                            Diffusion et reproduction interdites
```

— Perl 1/6: Scalaires, tableaux, fonctions — ) ( Liste: effectuer une sélection • grep : sélectionner les éléments liste2 = grep { test } liste1; liste2 = grep /reqexp/, liste1;  $\rightarrow$  sélection des éléments pour lesquels l'expression est vraie.  $_{-}$  vaut localement chaque élément de  $\it liste1$  $0s = grep \{ \$_ > 4 \} 0t;$ @s = grep { length(\$\_) > 10 } @t; Os = grep { \$\_ % 2 != 0 } Ot;  $@s = grep { $_>0 and $_!=4 } @t;$ @s = grep { monTest(\$\_) } @t; @s = grep /^aa/, @t; © Sylvain Lhullier 68 Diffusion et reproduction interdites

```
— Perl 1/6: Scalaires, tableaux, fonctions —
                                                                               ) • (
                       Liste: appliquer un traitement
        • map : appliquer une fonction à tous les éléments
          liste2 = map { expression } liste1;
          → création de la liste où chaque élément vaudra
               expression de chaque élément de liste1
          $_ vaut localement chaque élément de liste1
           0s = map \{ \$_* * 4 \} 0t;
           @s = map { $_ * $_ } @t;
           @s = map { uc($_) } @t;
           @s = map { '"'.$_.'"'} @t;
           @s = map { monCalcul($_) } @t;
          Modifier $_ modifie liste1: map { $_ *= 4 } @t;
                                                                               69
© Sylvain Lhullier
                            Diffusion et reproduction interdites
```





```
— Perl 1/6 : Scalaires, tableaux, fonctions —
                                                                               ) (
                            Fonctions sur les listes
       Ces fonctions sont auto-applicables:
        • @t = grep { $_ >= 10 } @t;
        • @t = map { $_ * $_ } @t;
        • @t = sort { $a <=> $b } @t:
       Chaînage possible entre fonctions:
       foreach my $e ( sort { $a <=> $b }
                             map { $_ * $_ }
                                grep { $_ >= 10 }
                                   @t. )
       { ... }
© Sylvain Lhullier
                                                                               72
                            Diffusion et reproduction interdites
```