Stage Perl
Corrigé des exos

1 Bases : les données

Texte p. 24

1. *Proposer 2 méthodes différentes d'afficher la concaténation de 2 chaînes* `$a1` *et* `$a2`.
   Par exemple:

```
$a = $a1 . $a2;
$a = "$a1$a2";
```

2. *Écrire le programme "Hello world!" en Perl en utilisant une variable pour chaque mot.*

```
#!/usr/bin/perl -w
$a = "Hello";
$b = "world";
print "$a $b!\n";
```

3. *Que se passe-t-il quand on incrémente une variable inexistante ? Quand on l'affiche ? Expérimenter.*

```
#!/usr/bin/perl -w
# use strict; si non commenté, exécution impossible
use warnings;
print $x;
# warnings => Use of uninitialized value $x in print at test.pl line 4.
print "$x\n";
# warnings => Use of uninitialized value $x in concatenation (.) or string...
$y++;
print "$y\n"; # undef + 1 = 1
print ++$z."\n" # 1
# warnings => Name "main::z" used only once: possible typo at test.pl line 10.
```

4. *Qu'affichent les instructions suivantes ?*
   On obtient *2+1*. À différencier de : `$b = 2 + $a`.

5. *Les expressions suivantes sont-elles vraies ou fausses ?*
   C'est l'opérateur qui fixe les types, et éventuellemnt implique des conversions. `(1<"un")` équivaut à `(1<0)`, donc faux. Par contre, dans l'ordre lexicographique imposé par `lt`, le caractère *1* est avant le *u*, donc vrai.

6. *Que donne* `print((2+3)x4);` *?*
   D'abord les parenthèses intérieures : *5*, puis répété 4 fois (`x4`), donc après une conversion implicite d'un nombre en texte : *5555*.

Tableaux p. 28

1. *Quelle est la différence entre* `print @a` *et* `print "@a"` *?*
   Par défaut, Perl affiche un tableau dans une chaîne en séparant les éléments par un espace. En dehors d'une chaîne, il n'y a pas de séparateur.

```
#!/usr/bin/perl -w
@a=(1,2,3,4,5,6);
print "@a\n"; # 1 2 3 4 5 6
print @a, "\n"; # 123456
```

   La raison est que "print" affiche chacun de ses arguments. Dans `print @a`, chaque élément de `@a` est un argument de `print`.

2. *Qu'affiche le programme suivant ?*
   La ligne `$b[1]=@a;` convertit un tableau en scalaire, donc renvoie la taille du tableau. Puisque `$b[1]` vaut donc 5, le programme affiche *3 5 1*.

3. *Écrire un programme qui affiche le dernier et le premier de ses arguments.*
   Il y a bien sûr plusieurs manières de faire. En voici une :

```
print "Premier : $ARGV[0]\nDernier : $ARGV[@ARGV-1]\n";
```

   Ou plus simplement :

```
print "Premier : $ARGV[0]\nDernier : $ARGV[-1]\n";
```

4. *Échanger deux variables scalaires sans utiliser de variable intermédiaire.*
   Il faut utiliser une liste.

```
($b,$a) = ($a,$b);
```

5. *Écrire un programme qui affiche l'argument dont le rang est donné par le dernier argument.*
   Sans variable intermédiaire :

```
#!/usr/bin/perl -w
print $ARGV[$ARGV[-1]-1] . "\n";
```

6. *Saisir des valeurs ligne par ligne jusqu'à apparition d'une ligne vide. Demander alors le numéro de la ligne à afficher parmi celles saisies.*
   La ligne contient tout ce qui a été saisi, y compris le retour à la ligne.

```
#!/usr/bin/perl -w
print "saisie : ";
while ($in=<STDIN>) {
        if ($in eq "\n") {
                print "Quelle ligne ? ";
                $in=<STDIN>;
                print $tab[$in];
                exit;
        } else {
                @tab = (@tab, $in);
                print "saisie : ";
        }
}
```

   L'écriture peut être simplifiée en utilisant `push` qui sera évoqué un peu plus tard.

7. *Comment extraire un sous-tableau ? Expérimenter. Comment insérer un élément dans un tableau ?*
   Un sous-tableau est un tableau, donc on commence par `@`. Il est alors naturel d'écrire :

```
@tab[1,3,5]
```

   Et même :

```
@a = (10 .. 19);
@b = (5, 8, 9);
print @a[@b]; # 151819
```

   Pour l'insertion, il faut obtenir le tableau de 0 à n, puis de n+1 jusqu'à la fin. Par exemple, pour insérer "insert" au rang 5 d'un tableau :

```
@a = ( @a[0..4], "insert", @a[5..(@a-1)] );
```

Hashes p. 33

1. *Écrire un programme qui convertisse un chiffre donné en toutes lettres en argument en valeur numérique.*

```
#!/usr/bin/perl -w
my %chiffres = ( "un"    => 1,
                 "deux"  => 2,
                 "trois" => 3,
               );
my $entree = $ARGV[0];
print $chiffres{$entree}."\n";
```

2. *Écrire un programme qui saisisse les noms et les notes d'élèves.*

```
#!/usr/bin/perl -w
my %devoir; # le hash a remplir
print "Eleve : ";
while (my $eleve=<STDIN>) {
        my $note = <STDIN>;
        $devoir{$eleve} = $note;
}
```

3. *Qu'affiche le programme suivant ?*
Le programme affiche les clefs de %ENV et leurs valeurs. La liste des clefs est triée. Comme %ENV contient les variables d'environnement, on obtient leur liste complète et ordonnée.

4. *Saisir un mot par ligne, et dire si ce mot a déjà été saisi.*

```
#!/usr/bin/perl -w
use strict;
my %saisi = ();
while (my $in = <STDIN>) {
        if (exists $saisi{$in}) {
                print "Ce mot a déjà été saisi $saisi{$in} fois.\n";
        }
        $saisi{$in}++;
}
```

2 Bases : les structures de contrôle

Structures principales p. 40

1. *Écrire un programme qui affiche ses arguments, un par ligne.*

```
#!/usr/bin/perl -w
use strict; use warnings;
foreach my $arg (@ARGV) {
        print "$arg\n";
}
```

2. *Programmer le jeu de devinette d'un entier fixé entre 1 et 99.*
La documentation de Perl nous propose une fonction `rand` qui rendra le jeu plus intéressant.

```
#!/usr/bin/perl -w
use strict; use warnings;
my $magique = int(rand(100));
while (my $in = <STDIN>) {
        if ($in > $magique) {
                print "Trop grand\n";
        } elsif ($in < $magique) {
                print "Trop petit\n";
        } else {
                print "Youpi !";
        }
}
```

3. *Afficher la somme des arguments du programme en écrivant l'opération.*

```perl
#!/usr/bin/perl -w
# my $somme =0;  # facultatif !
foreach $arg (@ARGV) {
        print "$arg + ";
        $somme += $arg;
}
print "\b = $somme\n";
```

4. *Écrire un programme qui affiche ses arguments accompagnés de leur nombre de caractères.*
   Avec l'aide-mémoire ou man perlfunc, on trouve la fonction `length`.

```perl
#!/usr/bin/perl -w
foreach my $a (@ARGV) {
        print "$a fait ".length($a)." caractère(s) de long.\n";
}
```

5. *Afficher les cubes des entiers de 1 à 20, d'abord à la façon du C, puis en utilisant une liste.*

```perl
for($i=1; $i<=20; $i++) {
        printf "%d\n",  $i*$i*$i;
}
foreach $i (1..20) {
        print $i**3 . "\n";
}
```

3 Bases : les fonctions p. 48

1. *Pourquoi ne peut-on pas recevoir comme arguments deux tableaux distincts ?* Parce que les arguments d'une fonction sont placés dans le tableau `@_` et qu'un tableau ne peut contenir de tableau.

```perl
#!/usr/bin/perl -w
sub marchepas {
        my (@a,@b) = @_;  # ERREUR !
```

2. *Écrire une fonction qui affiche "Vrai" si son argument est vrai, faux sinon.*

```perl
#!/usr/bin/perl -w
sub vrai {
        $arg = shift @_;
        # $arg = shift; # idem
        # $arg = $_[0]; # idem
        if ($arg) {
                print "Vrai\n";
        } else {
                print "Faux\n" unless $_[0];
        }
}
```

On peut aussi accèder directement à l'argument.

```perl
#!/usr/bin/perl -w
sub vrai {
        if ($_[0]) {
                print "Vrai\n";
        } else {
                print "Faux\n" unless $_[0];
        }
}
```

3. *Que fait la fonction suivante ?*
   Elle reçoit en argument un tableau (ou une liste de variables) et renvoie un tableau contenant ces valeurs incrémentées de 1. Les arguments ne sont pas modifiés.

```perl
@a = (1..4);
$b = -5;
print fonction(@a,$b); # affiche "2345-4"
# @a = (1..4);
# $b = -5;
print fonction(5); # affiche "6"
```

4. *Que fait la fonction suivante ?*
   Elle reçoit en argument un tableau (ou une liste de variables) et les multiplie par 2.

```perl
@a = (1..4);
$b = -5;
fonction(@a,$b);
# @a = (2,4,6,8);
# $b = -10;
fonction(5); # ERREUR !
```

4 Bases : commandes essentielles

Commandes essentielles p. 55

1. *Lire des nombres passés en argument au programme et en afficher la liste triée du plus grand au plus petit sous la forme "25 > 17 > 12 > 9 > 5 > 2".*
   La commande `sort` trie selon l'ordre lexicographique. Mais un coup d'œil à la page de documentation grâce à perldoc -f sort nous montre l'option *ad hoc* de cette commande.

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my @sorted = sort {$b<=>$a} @ARGV;
my $ready = join(" > ",@sorted);
print "$ready\n";
```

   Il est possible de faire plus court et moins propre :

```perl
#!/usr/bin/perl -w
print join(" > ",sort {$b<=>$a} @ARGV)."\n";
```

2. *Calculer l'intersection de deux tableaux.*
   Voici une astuce courante en Perl : on stocke dans un hash les éléments de `@a` pour pouvoir tester directement leur existence (sans parcourir tout le tableau).

```perl
#!/usr/bin/perl -w
use strict;
my @a = (1..20);
my @b = (-1,2,5,7,22,18);
my @intersection = ();
my %in_a; # les éléments vus dans @a
foreach my $e (@a) {
        $in_a{$e}++;
}
my @intersection; # le résultat
foreach my $e (@b) {
        push @intersection, $e if $in_a{$e};
}
print "(", join(",",@intersection), ")\n";
```

3. *Calculer le produit scalaire de 2 vecteurs.*
   Parfois, on est obligé de recourir à une boucle `for` classique. Mais la partie intéressante du programme est dans la lecture des données.

```perl
#!/usr/bin/perl -w
print "Premier vecteur : ";
$str_u = <STDIN>;
chomp($str_u);
@u = split(/,/,$str_u);
print "Second vecteur : ";
$str_v = <STDIN>;
chomp($str_v);
@v = split(/,/,$str_v);
my $ps=0;
for(my $i=0; $i<@u; $i++) {
        $ps += $u[$i] * $v[$i];
}
print "Produit scalaire : $ps\n";
```

4. *Lister les clés d'un* hash *associées à une valeur donnée en argument.*

```perl
#!/usr/bin/perl -w
%h = qw( un 1 deux 2 UN 1 oui 1 non 0);
foreach $key (keys %h) {
        print "'$key' " if ($h{$key}==$ARGV[0]);
}
print "\n";
```

5 Bases : Entrées/Sorties

Entrées/Sorties p. 61

1. *Lire* `/etc/passwd` *et produire un hash* `%root`

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my %root = ();
open my $passwd, "<", "/etc/passwd"
        or die "Erreur : $!";
while (my $ligne = <$passwd>) {
        chomp $ligne;
        my @champs = split /:/, $ligne;
        if ($champs[0] eq "root") {
                %root = (
                        "home"  => $champs[-2],
                        "shell" => $champs[-1],
                        "uid"           => $champs[3],
                        "gid"           => $champs[2],
                );
                last; # pour sortir de la boucle
        }
}
```

2. *Écrire un filtre qui compte le nombre de lignes et le nombre de caractères.*

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my $nblignes = 0;
my $nbcar = 0;
while (my $in = <>) {
        $nblignes++;
        $nbcar += length($in);
}
print "Lignes : $nblignes\nCaracteres : $nbcar\n";
```

3. *Écrire une procédure de sauvegarde de hash et une procédure de lecture de hash dans un fichier.*
On suppose que le hash ne contient pas de retour à la ligne. Ce "\n" va nous servir de séparateur.

```
#!/usr/bin/perl -w
sub savehash {
        my %hash = @_;
        open FILE, ">", "hash.save" or die "Erreur : $!";
        foreach $clef (keys %hash) {
                print FILE "$clef\n$hash{$clef}\n";
        }
        close FILE;
}
sub loadhash {
        my %hash;
        open FILE, "<", "hash.save" or die "Erreur : $!";
        while ($clef=<FILE>) {
           chomp $clef;
           $valeur = <FILE>;
           chomp $valeur;
           $hash{$clef} = $valeur;
        }
        close FILE;
        return %hash;
}
```

4. *Écrire un programme qui affiche la fréquence des mots dans un fichier.*

```
#!/usr/bin/perl -w
use strict; use warnings;
my %frequence = ();
while (my $ligne = <>) {
        my @mots = split / /, $ligne;
        foreach my $mot (@mots) {
        if (exists $frequence{$mot}) {
            $frequence{$mot}++;
        } else {
            $frequence{$mot} = 1;
        }
        }
}
# Affichage
foreach my $mot (keys %frequence) {
        print "$mot -> $frequence{$mot} fois.\n";
}
# Affichage plus complexe
my @sorted_words = sort
    {$frequence{$b} <=> $frequence{$a}}
    keys(%frequence);
foreach my $mot (@sorted_words) {
        print "$mot -> $frequence{$mot} fois.\n";
}
```

6 Expressions régulières

Matching p. 69,72

1. *Que donnent les regexps suivantes ?*
vrai, vrai, faux
faux, faux, vrai, vrai

faux, vrai, vrai, faux
vrai, faux

2. *Écrire un programme qui reçoit en argument une regexp de matching et teste cette regexp sur chaque ligne lue.*

```
#!/usr/bin/perl -w
my $regexp = shift; # la regexp est sortie de @ARGV
while ($ligne=<>) {
        if ($ligne =~ m/$regexp/) {
                print "         -> la ligne correspond !";
        }
}
```

A utiliser avec perl matching.pl '\w+\s?\?' pour appliquer `m/\w+\s?\?/`.

3. *Proposer et tester des regexp qui reconnaissent…*

```
/a+s?/
/^\n$/
/^\s*\n$/
/[A-Z].+[.?!]/
```

4. *Écrire une fonction qui teste si une chaîne est un entier naturel, un réel, ou n'est pas un nombre.*

```
#!/usr/bin/perl -w
use strict; use warnings;
sub typedenombre {
        my $n = shift;
        if ($n =~ /^-?\d+$/) {
                print "$ or $n =~ /^?\d*\.\d+$/) {
```

5. *Tester si une chaîne est valide comme numéro d'immatriculation de véhicule.*

```
$_ = "numero d'immatriculation";
if (m/^\d{2,4}[A-Z]{2,3}\d\d/
    or m/^\d{2,4}[A-Z]{2,3}2[AB]/) {
        print "Le numero semble correct.\n";
}
```

Substitution p. 75

1. *Proposer et tester une substitution qui…*

```
s/^.//    # ou s/.//
s/barb/tart/
```

Attention, `s/b/t/` changerait *barbare* en *tarbare*.

2. *Écrire une fonction qui supprime les blancs de début et de fin de ligne pour le texte passé en argument.*
Deux modes de fonctionnement de la fonction sont possibles. D'abord, à la façon de `chomp`.

```
#!/usr/bin/perl -w
sub nettoie {
        $_[0] =~ s/^\s+//;
        $_[0] =~ s/\s+$//;
}
```

Ensuite, en renvoyant le résultat sans modifier l'argument.

```perl
#!/usr/bin/perl -w
use strict; use warnings;
sub nettoie {
        my $arg = shift;
        $arg =~ s/^\s+//;
        $arg =~ s/\s+$//;
        return $arg;
}
```

3. *Remplacer "oui" par "non" quelle que soit la casse du premier.*

```perl
s/[Oo][Uu][Ii]/non/
```

4. *Écrire un filtre qui reçoit en argument une regexp de substitution et l'applique à chaque ligne.*

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my $regexp = shift; # la regexp est sortie de @ARGV
while (<>) {
        if (eval $regexp) {
                print "         -> la ligne devenue : $_";
        }
}
```

5. *Écrire une variante de l'exercice 4 qui reçoive une chaîne (et non un motif) à remplacer et sa substitution.* Pour que la chaîne ne soit pas interprêtée comme un motif, c'est-à-dire pour que "." reste un point et non la classe "n'importe quel caractère", il faudrait protéger tous les caractères spéciaux avec un \. Au lieu de cela, Perl propose un opérateur `\Q` (pour Quote) qui se charge de cette tâche jusqu'au `\E` qui suit (ou la fin de l'expression).

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my $chaine = shift; # la chaine est sorti de @ARGV
my $subst = shift;
while (<>) {
        if (eval "s/\Q$chaine\E/$subst/g") {
                print "         -> la ligne devenue : $_";
        }
}
```

Autres opérateurs p. 80

1. *Proposer des regexp qui...*

```perl
s/(\w+)(\s+)(\w+)/$3$2$1/
s/(\S)([;:!?])/$1 $2/g
s/^(\d{4})\.(\d\d)\.(\d\d)$/$3 $mois{$2} $1/  # avec %mois=qw(01 janvier 02 ...
```

2. *Programmer un filtre qui compte le nombre de signes de ponctuation.*
L'opérateur `tr///` renvoie le nombre de substitutions effectuées.

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my $ponct = 0;
while (<>) {
        $ponct += tr/?.,;:!/?.,;:!/;
}
print "Nombre de signes : $ponct\n";
```

3. *Écrire un filtre comptabilisant le nombre d'occurrences d'un motif passé en argument.*

```perl
#!/usr/bin/perl -w
use strict; use warnings;
die "arg ?\n" unless @ARGV;
my $motif = shift; # on enlève le motif de @ARGV
my $occ = 0;
while (<>) {
        while (m/$motif/g) {
                $occ++;
        }
}
print "Nombre d'occurrences : $occ\n";
```

4. *Convertir toutes les balises HTML d'un fichier en majuscules.*
   Il faut faire appel à un opérateur très particulier : `\U` met en majuscule le texte qui suit, jusqu'à ce qu'il rencontre un `\E`.

```perl
#!/usr/bin/perl -w
while (<>) {
        s/<(\w+)/<\U$1/g;
}
```

   En fait, l'utilisation serait plutôt en ligne de commande:

```
perl -i -p -e 's/<(\w+)/<\U$1/g' nomsdefichiers
```

   Une autre possibilité serait d'utiliser le modificateur `s///e`. Se reporter à la documentation.

7 Références et structures de données p. 86

1. *Comparer `\@a` et `[@a]`*
   `\@a` crée une référence vers `@a` alors que `[@a]` crée une référence vers un nouveau tableau qui est une copie de `@a`.

```perl
@a = (1,2,3);
$r1 = [@a];
shift @$r1;   # @a indentique, $r1=[2,3]
$r2 = \@a;
shift $@r2;   # @a=(2,3),     @$r2=[2,3]
```

2. *Écrire une fonction qui lit deux tableaux et renvoie les éléments du premier que ne sont pas dans le second.*
   Les arguments seront des références aux tableaux puisque passer directements 2 tableaux en arguments est impossible.

```perl
#!/usr/bin/perl -w
use strict; use warnings;
sub remove {
        my ($ra, $rb) = @_; # on copie les références
        # on construit un hash d'appartenance au second tableau
        my %in_b;
        foreach my $ele_b (@$rb) {
                $in_b{$ele_b}++;
        }
        my @result;
        foreach my $ele_a (@$ra) {
                push(@result, $ele_a) unless (exists $in_b{$ele_a});
        }
        return @result;
}

@a = (1..20);
@b = (5,1,6,5,-1,2.1,15);
print 'A - B = ', join(", ", remove(\@a,\@b)), "\n";
```

3. *Écrire un programme qui saisisse ligne par ligne un tableau à deux dimensions, puis demande quelle ligne afficher.*
   Dans l'idéal, il faudrait vérifier la taille des lignes, afficher des messages, etc.

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my @tableau = ();
while(my $ligne = <STDIN>) {
        last if ($ligne eq "\n");
        chomp $ligne;
        my @vecteur = split(/,/,$ligne);
        push @tableau, [@vecteur];
}
print "Afficher la ligne : ";
my $l = <STDIN>;
chomp $l;
print join(", ", @{$tableau[$l]}) ."\n";
# Attention, les indices commencent avec 0, pas 1
print "Ligne 1, colonne 3 : ", $tableau[1][3], "\n";
```

4. *Reprendre l'exemple précédent et contruire une variable qui stocke les lignes et les colonnes du tableau.* Une liste de listes ne suffira pas. Il faut avoir deux groupes de listes, un pour les colonnes, et un pour les lignes. Ici, `$tab{$ligne}` est une référence vers un tableau ligne à ligne (donc `@{$tab{$ligne}}` est identique au tableau 2D de l'exercice précédent).

```perl
#!/usr/bin/perl -w

# la structure de données
my %tab = { ligne => [],
            col   => [] } ;

# on saisit la liste des lignes
while(my $ligne = <STDIN>) {
        last if ($ligne eq "\n");
        chomp $ligne;
        my @vecteur = split(/,/,$ligne);
        push @{$tab{ligne}}, [@vecteur];
}

# on construit la liste des colonnes
my $max_l = scalar @{$tab{ligne}};
my $max_c = scalar @{$tab{ligne}[0]};
for(my $i=0; $i<$max_l; $i++) {
        for(my $j=0; $j<$max_c; $j++) {
                $tab{col}[$j][$i] = $tab{ligne}[$i][$j];
        }
}
```

5. *Stocker dans un hash anonyme les informations de /etc/passwd et y accéder à la demande.*

```perl
#! /usr/bin/perl -w

use strict;
use warnings;

open my $passwd, "< /etc/passwd"
        or die "Erreur : $!";

my %users;
while (my $ligne = <$passwd>) {
        chomp $ligne;
        my @champs = split /:/, $ligne;
        my %info = (
              login => $champs[0],
              perm  => $champs[1],
              uid   => $champs[2],
              gid   => $champs[3],
              descr => $champs[4],
              home  => $champs[5],
              shell => $champs[6],
    );
        $users{$champs[0]} = { %info };
}

print "Utilisateur : ";
my $u = <STDIN>;
chomp $u;
print "login/perm/uid/gid/descr/home/shell ? ";
my $c = <STDIN>;
chomp $c;
print "Pour $u : $c = '". $users{$u}{$c}."', \n";
```

8 Modules

Modules p. 92

1. *Utiliser* `XML::Simple` *pour sauvegarder et charger un hash dans un fichier.*
   En fait, ces fonctions ne sont pas limitées à des hashes, une structure de type

```perl
%biblio = (
  livres => [
              { titre => "La lumière qui s'éteint",
                auteur => "R. Kipling" },
              { titre => "Les sept piliers de la sagesse",
                auteur => "T. E. Lawrence" },
            ],
  disques => [ ],
};
```

   sera tout aussi bien sauvegardée/chargée.

```perl
#!/usr/bin/perl -w
use strict; use warnings;
use XML::Simple;

# variable globale pour normaliser les accès XML
my $xs = new XML::Simple( KeyAttr => [] );

my %h = ( 'un' => 1, "deux" => 2 );
print "Nombre de clefs avant : ".keys(%h)."\n";
savehash(%h);
%h = ( "trois" => 3 );
print "Nombre de clefs pendant : ".keys(%h)."\n";
%h = loadhash();
print "Nombre de clefs après : ".keys(%h)."\n";

# sauvegarde un hash donné en argument
sub savehash {
        my %hash = @_;
        my $xml = $xs->XMLout(\%hash, OutputFile => "hash.save");
}

# chargement
sub loadhash {
        my $refhash = $xs->XMLin("hash.save");
        return %$refhash;
}
```

On peut aussi proposer une solution qui passe des références et non des hashes.

```perl
savehash(\%h);
loadhash(\%h);

# sauvegarde un hash donné en argument
sub savehash {
        my $rhash = shift;
        my $xml = $xs->XMLout($rhash, OutputFile => "hash.save");
}

# chargement
sub loadhash {
        my $rhash = shift;
        my $refhash = $xs->XMLin("hash.save");
        $rhash = $refhash; # $rhash référence le même hash que $refhash
                           # sans copie de données
}
```

2. *Écrire un programme qui affiche les titres du Monde en ligne.*
   Un travail délicat serait de traiter le HTML avec l'un des nombreux modules Perl appropriés, comme par exemple HTML::TreeBuilder. Ici, des regexps suffisent.

```perl
#!/usr/bin/perl -w
use LWP::Simple;
use strict; use warnings;
$_ = get("http://www.lemonde.fr");
my @titres = ();
while (m/<div class=tit\d>(.+?)<\/div>/ig) {
        my $t = $1;
        $t =~ s/<[^>]+>//g; # on efface les balises
        push @titres, "$t\n";
}
print @titres;
```

3. *On souhaite que les éléments d'un hash respectent l'ordre dans lequel ils ont été créés.*
   On interroge la FAQ de Perl : `perldoc -q order hash`. Il n'y a plus qu'à utiliser cpan pour installer le module Perl et tester le code proposé par la FAQ.

9 Interactions

Système de fichiers p. 97

1. *Afficher tous les fichiers de `/bin/`, triés selon leur taille.*
   On peut trouver la taille avec stat, mais il est plus simple d'utiliser -s :

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my @fichiers = sort { -s $a <=> -s $b } glob("/bin/*");
print join("\n", @fichiers), "\n";
```

   Pour ne pas faire un appel système à chaque comparaison lors du tri, il faut stocker les tailles :

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my @fichiers;
foreach my $fichier (glob("/bin/*")) {
    my %h = (
        nom => $fichier,
        taille => -s $fichier,
    );
    push @fichiers, { %h };
}
sort { $a{taille} <=> $b{taille} } @fichiers;
foreach my $fichier (@fichiers) {
    print $fichier{nom}, "\n";
}
```

   Ce qui peut s'abréger ainsi :

```perl
#!/usr/bin/perl -w
use strict; use warnings;
my @fichiers = map { $_[0] } sort { $a[1] <=> $b[1] }
                            map { [$_, -s $_] } glob("/bin/*");
print join("\n", @fichiers), "\n";
```

   Cette construction s'appelle *Schwartzian Transform.*

2. *Écrire un programme qui change les droits d'accès pour toutes les entrées dans un répertoire donné.*
   Il faut commencer par chercher la commande avec man perlfunc, puis la syntaxe de chmod avec perldoc -f chmod.

```perl
#!/usr/bin/perl -w
chmod $ARGV[0], glob("*");
```

3. *Reprendre le programme précédent pour permettre de donner des droits différents aux fichiers et aux répertoires.*

```perl
#!/usr/bin/perl -w
use strict; use warnings;
foreach my $f (glob "*") {
        if (-f $f) {
                chmod $ARGV[0], $f;
        } elsif (-d $f) {
                chmod $ARGV[1], $f;
        } else {
                print "$f n'est ni un fichier ni un répertoire !";
        }
}
```

4. *Ajouter une option récursive au programme précédent.*
   Le module File::Find va nous être utile. Sa documentation nous indique sa syntaxe et que la variable `$File::Find::name` contient le nom complet de chaque fichier examiné.

```perl
#!/usr/bin/perl -w
use strict; use warnings;
use File::Find;

find(\&wanted, ".");

sub wanted {
        my $f = File::Find::name;
        if (-f $f) {
                chmod $ARGV[0], $f;
        } elsif (-d $f) {
                chmod $ARGV[1], $f;
        } else {
                print "$f n'est ni un fichier ni un répertoire !";
        }
}
```

Bases de données p. 100

1. *Utiliser DBI et SQLite pour remplir une base de données avec /etc/passwd.*
   Avec SQLite, pas besoin de login/mot de passe, contrairement à un fonctionnement client/serveur comme MySQL. Par contre, le nom de la base est aussi par défaut le nom du fichier où elle sera placée. On ajoute ici deux options recommandées par la doc de DBI (cf perldoc DBI).

```perl
#!/usr/bin/perl -w
use strict; use warnings;

use DBI;
my $dbh = DBI->connect("DBI:SQLite:base.sqlite","","",
                  { RaiseError => 1, AutoCommit => 1 });

# Creation de la table
$dbh->do("CREATE TABLE passwd
                (login, x, uid, gid, description, home, shell)");

# Remplissage
my $insert = $dbh->prepare("INSERT INTO passwd VALUES (?,?,?,?,?,?,?)");
open $passwd, "<", "/etc/passwd"
        or die "Erreur : $!";
while (my $ligne = <$passwd>) {
        chomp $ligne;
    my @fields = split /:/, $ligne;
        $insert->execute(@fields);
}
close $passwd;

# Interrogation
my $getbylogin = $dbh->prepare("SELECT * FROM passwd WHERE login=?");
print "Login : ";
while (my $login = <STDIN>) {
        last if $login eq "\n";
        chomp $login;
        $getbylogin->execute($login);
        my $rdata = $getbylogin->fetchrow_hashref;
        foreach my $clef (sort keys %$rdata) {
                print "  $clef => $rdata->{$clef}\n";
        }
        print "Login : ";
}
```

On voit ici les deux modes de requête SQL : préparation+exécution ou action directe. Et pas besoin de protéger les arguments, quand la requête est préparée, c'est automatique ! D'autres fonctions peuvent simplifier la tâche, comme par exemple `$dbh->selectall_arrayref()`.