RÉFÉRENCE RAPIDE PERL

Variables, regex, E/S fichier, références, modules essentiels

Bases

Hello World

```
#!/usr/bin/perl
use strict;
use warnings;
print "Hello, World!\n";
say "Hello, World!";  # with use feature 'say';
```

Exécuter Perl

```
perl script.pl           # run a file
perl -e 'print "hi\n"'   # run inline
perl -ne 'print' file    # process file line by line
```

Commentaires et documentation

```
# single-line comment
=pod
Multi-line POD documentation
=cut
```

Variables

Sigils

`$scalar`	Valeur unique (chaîne, nombre, référence)
`@array`	Liste ordonnée de scalaires
`%hash`	Paires clé-valeur
`$array[0]`	Accéder à un élément de tableau (contexte scalaire)
`$hash{key}`	Accéder à une valeur de hash (contexte scalaire)

Variables scalaires

```
my $name = "Perl";         # string
my $version = 5.40;        # number
my $count = 42;            # integer
my $undef;                 # undefined (undef)
my $combined = "$name v$version";  # interpolation
```

Contexte

```
my @arr = (1, 2, 3);
my $count = @arr;          # scalar context: 3
my @copy = @arr;           # list context: (1, 2, 3)
my $len = scalar @arr;     # force scalar context
```

Variables spéciales

`$_`	Variable par défaut (topic)
`@_`	Arguments de sous-routine
`$!`	Message d'erreur système
`$@`	Erreur provenant de eval
`$0`	Nom du programme
`@ARGV`	Arguments de la ligne de commande
`%ENV`	Variables d'environnement

Opérateurs

Opérateurs de comparaison

`==, !=, <, >, <=, >=`	Comparaison numérique
`eq, ne, lt, gt, le, ge`	Comparaison de chaînes
`<=>`	Opérateur spaceship numérique (retourne -1, 0, 1)
`cmp`	Opérateur spaceship de chaînes
`=~`	Correspondance / liaison regex
`!~`	Correspondance regex niée

Opérateurs de chaînes

```
my $full = "Hello" . " " . "World";  # concatenation
my $line = "-" x 40;                 # repetition
my $len = length($full);             # 11
```

Opérateurs logiques

`&& / and`	ET logique (faible priorité : `and`)
`|| / or`	OU logique (faible priorité : `or`)
`//`	Defined-or (retourne gauche si défini)
`! / not`	NON logique
`? :`	Conditionnel ternaire

Flux de contrôle

Conditionnelles

```
if ($x > 0) { print "positive\n"; }
elsif ($x == 0) { print "zero\n"; }
else { print "negative\n"; }
print "yes\n" if $condition;     # postfix if
print "no\n" unless $condition;  # postfix unless
```

Boucles

```
for my $i (0..9) { print "$i\n"; }
foreach my $item (@array) { print "$item\n"; }
while ($line = <STDIN>) { chomp $line; }
until ($done) { last if check(); }
```

Contrôle de boucle

`next`	Passer à l'itération suivante (comme `continue`)
`last`	Quitter la boucle (comme `break`)
`redo`	Recommencer l'itération courante
`next LABEL`	Passer à l'itération suivante de la boucle étiquetée
`last LABEL`	Quitter la boucle étiquetée

Given / When

```
use feature 'switch';
given ($status) {
    when ("ok")    { say "success"; }
    when ("error") { say "failed"; }
    default        { say "unknown"; }
}
```

Sous-routines

Sous-routine de base

```
sub greet {
    my ($name) = @_;
    return "Hello, $name!";
}
my $msg = greet("Alice");
```

Paramètres par défaut et nommés

```
sub connect {
    my (%opts) = @_;
    my $host = $opts{host} // "localhost";
    my $port = $opts{port} // 5432;
    return "$host:$port";
}
connect(host => "db.example.com", port => 3306);
```

Références de sous-routines

```
my $double = sub { return $_[0] * 2; };
print $double->(5);   # 10
my @sorted = sort { $a <=> $b } @nums;
```

Prototypes et signatures

```
use feature 'signatures';
sub add($a, $b) { return $a + $b; }
sub greet($name, $greeting = "Hello") {
    return "$greeting, $name!";
}
```

Regex

Correspondance

```
if ($str =~ /pattern/) { print "matched\n"; }
if ($str =~ /(\d+)/) { print "number: $1\n"; }
my @matches = ($str =~ /(\w+)/g);  # all matches
```

Substitution

```
$str =~ s/old/new/;      # first occurrence
$str =~ s/old/new/g;     # global (all occurrences)
$str =~ s/^\s+|\s+$//g;  # trim whitespace
(my $clean = $str) =~ s/\W//g;  # non-destructive copy
```

Modificateurs

`/i`	Insensible à la casse
`/g`	Global (toutes les correspondances)
`/m`	Multi-ligne (`^` et `$` correspondent aux limites de ligne)
`/s`	Ligne unique (`.` correspond aux sauts de ligne)
`/x`	Étendu (autoriser les espaces et commentaires)

Motifs courants

`\d, \D`	Chiffre / non-chiffre
`\w, \W`	Caractère de mot / non-mot
`\s, \S`	Espace blanc / non-espace
`\b`	Limite de mot
`(?: ... )`	Groupe non-capturant
`(?<name> ... )`	Capture nommée (accès via `$+{name}`)

E/S fichier

Ouvrir et lire

```
open(my $fh, '<', 'data.txt') or die "Cannot open: $!";
while (my $line = <$fh>) {
    chomp $line;
    print "$line\n";
}
close($fh);
```

Écrire et ajouter

```
open(my $fh, '>', 'out.txt') or die "Cannot open: $!";
print $fh "Hello\n";
close($fh);
open(my $fh, '>>', 'log.txt') or die "Cannot open: $!";
print $fh "entry\n";
close($fh);
```

Lire le fichier entier

```
use File::Slurp;
my $content = read_file('data.txt');
my @lines = read_file('data.txt', chomp => 1);
```

Tests de fichier

`-e $path`	Le fichier existe
`-f $path`	Est un fichier normal
`-d $path`	Est un répertoire
`-r / -w / -x`	Lisible / inscriptible / exécutable
`-s $path`	Taille du fichier en octets (0 si vide)
`-z $path`	Le fichier est de taille nulle

Tableaux et hashs

Tableaux

```
my @arr = (1, 2, 3, 4, 5);
push @arr, 6;              # append
my $last = pop @arr;       # remove last
my $first = shift @arr;    # remove first
unshift @arr, 0;           # prepend
my @slice = @arr[1..3];    # slice
```

Fonctions de tableau

`scalar @arr`	Nombre d'éléments
`push / pop`	Ajouter/supprimer à la fin
`shift / unshift`	Supprimer/ajouter au début
`splice(@a, 2, 1)`	Supprimer 1 élément à l'index 2
`sort @arr`	Trier alphabétiquement
`reverse @arr`	Inverser l'ordre
`grep { /pat/ } @arr`	Filtrer par motif
`map { $_ * 2 } @arr`	Transformer chaque élément
`join(',', @arr)`	Joindre en chaîne

Hashs

```
my %user = (name => "Alice", age => 30);
$user{email} = "a\@b.com";    # add pair
delete $user{age};            # remove pair
my @keys = keys %user;
my @vals = values %user;
```

Itération de hash

```
while (my ($k, $v) = each %hash) {
    print "$k => $v\n";
}
for my $key (sort keys %hash) {
    print "$key: $hash{$key}\n";
}
```

Références

Créer des références

```
my $scalar_ref = \$name;
my $array_ref  = \@arr;
my $hash_ref   = \%hash;
my $anon_arr   = [1, 2, 3];       # anonymous array ref
my $anon_hash  = {a => 1, b => 2}; # anonymous hash ref
```

Déréférencement

```
print $$scalar_ref;          # dereference scalar
print $array_ref->[0];       # arrow notation
print $hash_ref->{key};      # arrow notation
my @copy = @$array_ref;      # dereference to array
my %copy = %$hash_ref;       # dereference to hash
```

Structures de données complexes

```
my @users = (
    { name => "Alice", age => 30 },
    { name => "Bob",   age => 25 },
);
print $users[0]->{name};   # "Alice"
```

Fonction ref()

`ref($r) eq 'SCALAR'`	Référence à un scalaire
`ref($r) eq 'ARRAY'`	Référence à un tableau
`ref($r) eq 'HASH'`	Référence à un hash
`ref($r) eq 'CODE'`	Référence à une sous-routine

Modules

Utiliser des modules

```
use strict;
use warnings;
use List::Util qw(sum max min);
use File::Basename;
use Cwd qw(abs_path);
```

Créer un module

```
# MyModule.pm
package MyModule;
use Exporter 'import';
our @EXPORT_OK = qw(helper);
sub helper { return "help"; }
1;  # module must return true
```

Modules noyau courants

`List::Util`	sum, max, min, reduce, any, all
`File::Basename`	basename, dirname, fileparse
`File::Path`	make_path, remove_tree
`Getopt::Long`	Analyse des options de ligne de commande
`JSON`	encode_json, decode_json
`LWP::Simple`	get($url) — client HTTP simple
`Data::Dumper`	Débogage de structures de données
`Carp`	croak, confess — meilleurs messages d'erreur

CPAN

```
cpan install Module::Name        # install from CPAN
cpanm Module::Name               # cpanminus (faster)
perldoc Module::Name             # read module docs
```