Rappels de Perl

Février 2016

Bérénice Batut
berenice.batut@udamail.fr

PDF

Représentation des données

Variable

```
$chaine = "une chaine\n";
```

Données simples

```php
$nombre = 1;
$nombre = $nombre + 10;
$chaine = "nombre : ";
$chaine = $chaine.$nombre;
chop($chaine);
```

Quel est le contenu de `$chaine` ?

- ☐ `nombre : 11`
- ☐ `nombre : 1`
- ☐ `ombre : 11`
- ☐ `nombre :`

Nombre entier ou réel

- Opérateurs : `+ - * / ** %`
- Comparaison : `< > <= >= == !=`

Chaîne de caractères

- Définition avec des quotes simples ou doubles
- Opérateurs : `.`
- Comparaison : `lt gt le ge eq ne`
- Manipulation :
  - `length($chaine)`
  - `chop($chaine)`
  - `chomp($chaine)`
  - `reverse($chaine)`
  - `substr($chaine, 1, 3)`
  - `index($chaine, " ", 1)`
  - `rindex($chaine, " ")`

Données simples

```php
$nombre = 1;
$nombre = $nombre + 10;
$chaine = "nombre : ";
$chaine = $chaine.$nombre;
chop($chaine);
```

Quel est le contenu de `$chaine` ?

- ☐ `nombre : 11`
- ☐ `nombre : 1`
- ☐ `ombre : 11`
- ☐ `nombre :`

Tableaux

Tableaux

```
$nombre= 12;
$chaine = ($nombre+1)." ".($nombre+2);
@tab = ($nombre, ($chaine,$chaine));
```

Quel est le contenu de `$tab[1]`?

- ☐ Un tableau contenant 2 éléments `$chaine`
- ☐ `$chaine`
- ☐ `$nombre`
- ☐ Rien

Tableaux

- Définition : `@tab`
- Initiation :
  - `@tab = (3, "chaine");`
  - `@t = @tab;`
  - `@tab = ((1,2),(3,4));`

Tableaux

- Parcours :
  - Début à 0
  - `print $tab[1];`
  - `$tab[2] = 1;`
- Manipulation :
  - `exists($tab[10])`
  - `defined($tab[10])`
  - `unshift(@tab,5,6)`
  - `$v = shift(@tab);`

Tableaux

- Manipulation :
  - `push(@tab, 20);`
  - `$u = pop(@tab);`
  - `reverse(@tab);`
  - `@t = qw(Découpage d'une liste);`
  - `join(" ", @tab);`
  - `sort(@tab)`
  - `@s = grep( /^aa/, @t );`
  - `@s = map( { -$_ } @t );`

Tableaux

```
$nombre= 12;
$chaine = ($nombre+1)." ".($nombre+2);
@tab = ($nombre, ($chaine,$chaine));
```

Quel est le contenu de `$tab[1]`?

- ☐ Un tableau contenant 2 éléments `$chaine`
- ☐ `$chaine`
- ☐ `$nombre`
- ☐ Rien

Tableaux associatifs

Tableaux associatifs

```
%hash = ("id1" => 1, "id2" => "chaine");
$id = "id";
$hash{$id} = "je ne sais pas";
@tab = each(%hash);
@tab = each(%hash);
```

Quel contient `$tab[1]`?

- ☐ Rien
- ☐ `id2`
- ☐ `1`
- ☐ `chaine`

Tableaux associatifs

- Définition : %hash
- Initiation :
  - `@hash = ("id1" => 1, "id2" => "chaine");`
- Parcours :
  - `print $hash{"id2"};`
  - `$hash{"id1"} = 3;`
  - `$hash{"id3"} = (1, 4);`

Tableaux associatifs

- Manipulation :
  - `keys(%hash)`
  - `values(%hash)`
  - `each(%hash)`
  - `delete(%hash, "id1");`
  - `exists(%hash{"id3"});`
  - `reverse(%hash);`
  - Autovivification

Tableaux associatifs

```
%hash = ("id1" => 1, "id2" => "chaine");
$id = "id";
$hash{$id} = "je ne sais pas";
@tab = each(%hash);
@tab = each(%hash);
```

Quel contient `$tab[1]`?

- ☐ Rien
- ☐ `id2`
- ☐ `1`
- ☐ `chaine`

Structures de contrôle

Instructions de test

Instructions de test

```
$c1 = "chaine";
$t = $c1 eq "chaine2";
if ( $t ){
    print "1";
}elsif ( 1 < 10 & 2 > 3 ){
    print "2";
}elsif ( 2 | "0" ){
    print "3";
}else{
    print "4";
}
```

Que va-t-il s'afficher?

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4

Instructions de test

```
if (expression booléenne) {
    instructions;
}elsif (expression booléenne) {
    instructions;
}else{
    instructions;
}
```

Booléens

- Valeurs fausses
  - `0` : Entier valant zéro
  - `"0"` ou `'0'`
  - `""` ou `''`
  - `undef`
- Valeurs vraies
  - Toutes les autres valeurs

Opérateurs de tests

Type	Nombres	Chaînes
égalité	==	eq
différence	!=	ne
infériorité	<	lt
supériorité	>	gt
inf ou égal	<=	le
sup ou égal	>=	ge
comparaison	<=>	cmp

Opérateurs booléens

- &
- |
- !

Instructions de test

```
$c1 = "chaine";
$t = $c1 eq "chaine2";
if ( $t ){
    print "1";
}elsif ( 1 < 10 & 2 > 3 ){
    print "2";
}elsif ( 2 | "0" ){
    print "3";
}else{
    print "4";
}
```

Que va-t-il s'afficher?

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4

Boucles

Boucles

```
@liste = ("2","22", "10", "100");
foreach $_ (@liste)
{
    if ( $_ lt "20"){
        $_ = $_ + 1;
    }
}
```

Que contient `@liste`?

- ☐ `@liste = (2, 22, 11, 100)`
- ☐ `@liste = (3, 22, 11, 101)`
- ☐ `@liste = (2, 22, 10, 100)`
- ☐ `@liste = (2, 23, 10, 100)`

Boucles

```
while (condition){
    instructions;
}
```

```
for(initialisation; condition; incrément){
    instructions;
}
```

```
foreach variable (liste) {
    instructions;
}
```

Que choisir? Quand?

Boucles

- `next`
- `last`
- `redo`

Boucles

```
@liste = ("2","22", "10", "100");
foreach $_ (@liste)
{
    if ( $_ lt "20"){
        $_ = $_ + 1;
    }
}
```

Que contient `@liste`?

- ☐ `@liste = (2, 22, 11, 100)`
- ☐ `@liste = (3, 22, 11, 101)`
- ☐ `@liste = (2, 22, 10, 100)`
- ☐ `@liste = (2, 23, 10, 100)`

Manipulation de fichiers

Manipulation de fichiers

Quelle commande permet d'ajouter à la fin d'un fichier après avoir vérifier que le fichier est non vide?

- ☐
```
if -s file {open(FILE, ">file");
write("text"); close(FILE) }
```
- ☐
```
if -z file {open(FILE, ">>file");
write("text"); close(FILE) }
```
- ☐
```
if -w file {open(FILE, ">>file");
write("text"); close(FILE) }
```
- ☐
```
if -s file {open(FILE, ">>file");
write("text"); close(FILE) }
```

Ouverture de fichiers

```
open(FILE, "nom du fichier") or die("open: $!");
```

Caractère(s)	Mode d'ouverture
<	lecture
>	écriture (écrasement)
>>	écriture (ajout)
+>	lecture et écriture (écrasement)
+<	lecture et écriture (ajout)

Fermeture

```
close(FILE);
```

Lecture / écriture de fichiers

- Lecture

```
while($ligne = < FILE >){
    instructions;
}
```

- Écriture

```
print FILE ”chaine à écrire\n”;
```

Opérateurs sur les noms de fichier

`if operateur nom_fichier`

- `-e` : chemin valable
- `-f` : fichier normal
- `-d` : répertoire
- `-l` : lien symbolique
- `-r` : droit de lecture sur le fichier
- `-w` : droit d'écriture sur le fichier
- `-x` : droit d'exécution du fichier
- `-o` : appartenance à l'utilisateur qui exécute le programme
- `-z` : fichier vide
- `-s` : fichier non vide

Manipulation de fichiers

Quelle commande permet d'ajouter à la fin d'un fichier après avoir vérifier que le fichier est non vide?

- ☐
```
if -s file {open(FILE, ">file");
write("text"); close(FILE) }
```
- ☐
```
if -z file {open(FILE, ">>file");
write("text"); close(FILE) }
```
- ☐
```
if -w file {open(FILE, ">>file");
write("text"); close(FILE) }
```
- ☐
```
if -s file {open(FILE, ">>file");
write("text"); close(FILE) }
```

Expression régulières

Expression régulières

```
if( $v =~ m/^"([a-z]{4,})",/ ) {
    print "$1\n";
}
```

Pour quelle chaine de caractères quelque chose sera affiché?

- ☐ "cette" chaine
- ☐ "celle",ci
- ☐ et "celle",la
- ☐ "ou",elle

Composants

- Motif
- Chaine de caractères à évaluer
- Correspondances

Fonctionnalités

- Recherche de correspondances

```
m/motif/
```

- Substitution

```
s/motif/chaine_de_remplacement/
```

Liaison d'une variable à une expression

=~

- Recherche de correspondances

```
$s =~ m/motif/
```

- Substitution

```
$s =~ s/motif/chaine/
```

Composants de motif : Caractères

`m/a/`

- Caractères à déspecifier avec `\`
  - `\ | ( ) [ ] { } ^ $ * + ? . /`
- Caractères spéciaux

Motif	Caractère
`\n`	saut de ligne
`\r`	retour chariot
`\t`	tabulation
`\f`	saut de page
`\e`	échappement

Composants de motif : Ensembles (1)

- N'importe quel caractère : `.`
- Ensemble : `[atc]`
- Intervalle : `[a-z]`
- Complémentaire : `[^ar]`

Composants de motif : Quantificateurs

	Motif présent	**Exemple**	**Mots matchés**
`*`	0 fois ou plus	`m/a*/`	mot vide, a, aa ...
`+`	1 fois ou plus	`m/a+/`	a, aa, aaa ...
`?`	0 ou 1 fois	`m/a?/`	mot vide ou a
`{n}`	n fois	`m/a{4}/`	aaaa
`{n,}`	au moins n fois	`m/a{2,}/`	aa, aaa, aaaa ...
`{,n}`	au plus n fois	`m/a{,2}/`	mot vide, a ou aa
`{n,m}`	entre m et n fois	`m/a{2,5}/`	aa, aaa ou aaaaa

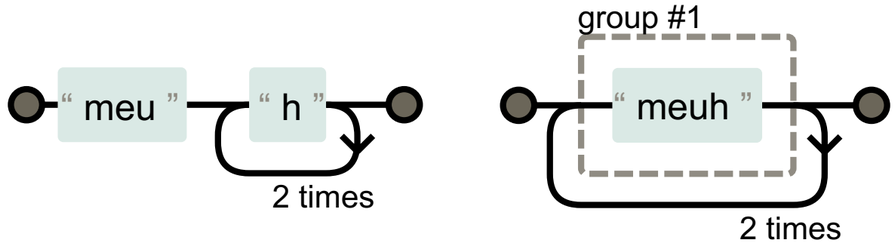
Composants de motif : Ensembles (2)

- `\d` = `[0-9]`
- `\D` = `[^0-9]`
- `\w` = `[0-9a-zA-Z_]`
- `\W` = `[^0-9a-zA-Z_]`
- `\s` = `[ \n\t\r\f]`
- `\S` = `[^ \n\t\r\f]`

Composants de motif

- Regroupement de plusieurs motifs : ()

`m/meuh{3}/` VS `m/(meuh){3}/`



- Alternatives : |
- Assertions = position dans l'expression
  - Début : ^
  - Fin : $

Expression régulières

```
if( $v =~ m/^"([a-z]{4,})",/ ) {
    print "$1\n";
}
```

Pour quelle chaine de caractères quelque chose sera affiché?

- ☐ "cette" chaine
- ☐ "celle",ci
- ☐ et "celle",la
- ☐ "ou",elle

Fonctions

Fonctions

```perl
my $a = 3;
sub fonction{
    my ($a) = @_;
    return ($a+2,$a+1);
}
my ($b,$a) = fonction($a+2);
```

Quelle est la valeur de *a* en fin d'exécution?

- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7

Fonction

Ensemble d'instructions regroupées de manière à être utilisées plusieurs fois sans avoir à dupliquer du code

Déclaration d'une fonction

```
sub nomFonction {
   my ($x,$y,$t) = @_;
   ... instructions ...
   return $z;
}
```

Appel à une fonction

```php
$v = nomFonction(10, 20, 30);
```

Visibilité des variables

- Variable locale
- Variable globale
- Variable déclarée avec `my` visible jusqu'à la fin du plus petit bloc qui l'englobe

Renvoi de plusieurs valeurs

Utilisation d'une liste

Fonctions

```
my $a = 3;
sub fonction{
    my ($a) = @_;
    return ($a+2,$a+1);
}
my ($b,$a) = fonction($a+2);
```

Quelle est la valeur de *a* en fin d'exécution?

- ☐ 3
- ☐ 4
- ☐ 5
- ☐ 6
- ☐ 7

Références

- Guide Perl
- Visualisation d'expressions régulières
- Pour aller plus loin sur les expressions régulières

Perl orientée object