
Chapitre 3 : Chaines, tableaux et fonctions

Chaînes de caractères

- Pas de support natif pour l'UTF
 - Caractères sur 1 octets seulement
 - Unicode dans quelques fonctions seulement
- Concaténation : le caractère '.'
 - pas le '+' qui est pour Javascript, en PHP le + **force un transtypage** des arguments en nombres (entiers).
- Le rôle différents des ' et des "
 - **Apostrophe ' : aucune** interprétation des caractères spéciaux sauf pour \'
et \\. Attention, par exemple le \n **n'est pas interprété.**
 - **Guillemet " (double quote) :**
 - **Interprétation** des caractères d'échappement, ex: code en octal ou en décimal.
 - **Interpolation** des variables (avec \$ ou {\$...})
- Format "Here Doc" et "Now Doc"
 - Pour placer du texte brut, avec ou sans interpolation des variables
 - Issu du shell

Chaîne de caractères & interpolation de variable

```
$nom = 'Gwenn';  
print '1) Bonjour $nom \n';  
  
print "2) Bonjour $nom \n";  
  
print '3) Bonjour \\\n';  
  
print "4) Bonjour \\\n";
```

- 1) Bonjour \$nom \n
- 2) Bonjour Gwenn
- 3) Bonjour \\\n
- 4) Bonjour \$nom

Syntaxe générale **interpolation** de variable

```
$nom = 'Gwenn';  
  
print '5) Bonjour {$nom}_Dupont \n';  
  
Print "6) Bonjour {$nom}_Dupont \n";  
  
print "7) Bonjour $nom_Dupont \n";
```

5) Bonjour {\$nom}_Dupont \n

6) Bonjour Gwenn_Dupont

7) Bonjour

Notice: Undefined variable: nom_Dupont

Chaînes de caractères : mélange

```
print '8) Qui es-tu "Gwenn" ? ' ;  
  
print "9) Qui es-tu 'Gwenn' ?\n";  
  
print "10) Qui es-tu \"Gwenn\" ?\n";  
  
print '11) Qui es-tu \'Gwenn\' ? ' ;
```

8) Qui es-tu "Gwenn" ?

9) Qui es-tu 'Gwenn' ?

10) Qui es-tu "Gwenn" ?

11) Qui es-tu 'Gwenn' ?

Chaines de caractères, encodage ASCII explicite

```
print '12) Quel est ce caractère \ ? ' ;  
  
print '13) Quel est ce caractère \\ ? ' ;  
  
print '14) Quel est ce caractère \057 ? ' ;  
  
print "15) Quel est ce caractère \057 ? " ;  
  
print "16) Quel est ce caractère \2F ? " ;  
  
print "17) Quel est ce caractère \x2F ? " ;
```

- 12) Quel est ce caractère \ ?
- 13) Quel est ce caractère \\ ?
- 14) Quel est ce caractère \057 ?
- 15) Quel est ce caractère / ?
- 16) Quel est ce caractère F ?
- 17) Quel est ce caractère / ?

Conclusion : Chaîne de caractères

- Pour les chaînes constantes :
 - Préférer l'apostrophe (EN: simple quote) `'`
- Pour des chaînes avec **interpolation** et/ou **interprétation**
 - Utiliser le guillemet (EN: double quote) `"`
 - Si confusion du nom avec l'environnement ou si usage de la notation **tableau**, ou accès à un attribut, méthode d'un **objet**, utiliser les accolades pour lever l'ambiguïté : `{ $... }`
 - **En cas de doute en pratique**, il vaut mieux simplement faire des concaténation avec l'opérateur `'.'`
- **Ne pas confondre `\n` et `
`**
 - `\n` est interprété dans une chaîne avec guillemet mais **n'a pas d'effet** sur le rendu visuel HTML. Il sert par exemple, à rendre le code HTML plus lisible pour un programmeur humain
 - `
` **n'est pas interprété par PHP**, mais a un effet sur le rendu visuel par le navigateur

Le "Here Doc"

```
<?php
$str = <<<ENDOFDOC
<p> Tout ce texte, sera mis
tel quel dans la variable ! </p>
ENDOFDOC;
```

- Le symbole HereDoc, littéralement, "ici document brut"
<<<
- **Identifiant de fin** : une séquence quelconque de caractères qui n'apparaît **JAMAIS** seule sur une ligne.
- L'identifiant de fin **DOIT** apparaître seul sur au début d'une ligne, sans autre caractères (pas d'espaces).

Différences entre le "Here Doc" et le "Now Doc"

```
<?php
$nom = "IUT de Grenoble";
$str = <<<ENDOFDOC
<p> Un paragraphe pour
dire bonjour à $nom ! </p>
ENDOFDOC;
```

- Interpolation de la variable \$nom
- Si on ajoute des guillemets à **ENDOFDOC** on obtient un Now Doc qui bloque l'interprétation.
\$str = <<<'ENDOFDOC'

Interpolation des variables dans les chaînes

```
//Syntaxe simple
echo "$A et la valeur de $TAB[0]\n<br/>";

// Syntaxe avec accolades
echo "{$A} et la valeur de {$TAB['truc']}\n<br/>";
```

- Valable pour les chaînes avec guillemets (double quotes) et le HereDoc
- Variables simples interpolées
- Tableaux interpolés **que si l'indice est numérique**
- Pour les autres cas : utiliser les accolades `${..}`.
 - Tableau avec indice chaîne
 - Attribut d'un objet
- NB: dans de doute, utiliser la concaténation

Interpolation vs concaténation

```
//Interpolation
echo "$A et la valeur de ${TAB['truc']}\n<br/>";

// Concaténation
Echo $A.' et la valeur de '.$TAB['truc']. "\n<br/>";
```

- Préférer l'apostrophe quand c'est possible :
 - Les guillemets ont un surcoût dû à l'analyse de la chaîne pour réaliser l'interpolation des variables.

Traitement des chaînes

- Taille d'une chaîne : `strlen()`
- Accès aux caractères : `$prenom[0]`
 - NB: La syntaxe `$prenom{0}` est dépréciée
- Position d'un motif dans une chaîne : `strpos()`
- Remplacer un motif dans une chaîne : `str_replace()`
- Comparaison d'une chaîne à un motif : `ereg()`
`ereg("[A-Za-z]", $chaîne);`
- Changement de casse : `strtoupper()` `strtolower()`
- Echappement et inverse : `addslashes()` `stripslashes()`
`$prenom = addslashes("Eddie");`
`$nom = addslashes ("O'Sullivan");`
`$req = "INSERT INTO selectionneur (nom,`
`prenom)VALUES ('$prenom', '$nom')";`
- ... de l'ordre de 100 fonctions!

Les tableaux (array)

1. Tableau indexé numériquement

```
$tab = array('pomme','poire','pêche');  
$tab[0]= 'pomme'; $tab[1]= 'poire'; $tab[2]= 'pêche';  
$tab[]= 'pomme'; $tab[]= 'poire'; $tab[]= 'pêche';
```

2. **Tableau associatif** : associe une chaîne de caractère à un élément

```
$tab = array (  
    'prenom'      => 'toto',  
    'age'         => 10,  
    'note'        => 0  
);  
echo "{$tab['prenom']} a eu la note {$tab['note']}";
```

3. Tableau multidimensionnel

```
$matrice = array ( array(1,2), array(3,4), array(4,5) );
```

Traitements des tableaux

- Afficher un tableau:

```
print_r($nomTableau); // ou var_dump
```

- Taille d'un tableau:

```
$n=count($nomTableau); // ou sizeof
```

- Convertir une chaîne en tableau (et inversement)

```
$chaîne = "Paul,Dupont,3 rue des prés,38999,Chaille);  
$nomTableau = explode(",", $chaîne);  
(implode)
```

- Parcourir un tableau (foreach, while, for):

```
foreach ($nomTableau as $clef => $valeur) {  
    echo $clef.' : '.$valeur.'<br />';  
}
```

Et beaucoup d'autres choses!

⇒ notion de tableau très riche en PHP

Actions sur les tableaux

```
$T = array(); // Création d'un tableau vide  
$T[] = 'A';  
$T[] = 'B';  
$T[] = 'C';  
var_dump($T);
```

Opérateur d'ajout en fin, création d'index automatiquement

```
array(3) {  
    [0] => string(1) "A"  
    [1] => string(1) "B"  
    [2] => string(1) "C"  
}
```

Tableau : une clé et une valeur

```
$T = array();  
$T[] = 'A';  
$T[] = 'B';  
$T[4] = 'D';  
$T[3] = 'C';  
var_dump($T);
```

- Tableau : une séquence de (clé,valeur)
- Ordre : celui de la **construction** et pas celui des clés !

Tableau : une clé et une valeur

```
$T = array();  
$T[] = 'A';  
$T[] = 'B';  
$T[4] = 'D';  
$T[3] = 'C';  
$T["4"] = 'E'; // un chiffre dans une chaîne est transformé  
var_dump($T);
```

```
array(4) {  
    [0] => string(1) "A"  
    [1] => string(1) "B"  
    [4] => string(1) "E"  
    [3] => string(1) "C"  
}
```

Tableau : clés mélangées

```
$T[ ] = 'A';  
$T[10] = 'B';  
$T['A'] = 0;  
$T['B'] = 10;  
$T[ ] = 'C';  
$T[10.87] = 'Z';  
var_dump($T);
```

A quel indice est 'C' ?

Array(5) {

[0] => string(1) "A"

[10] => string(1) **"Z"** // Un flottant est transformé en entier

["A"] => int(0)

["B"] => int(10)

[11] => string(1) "C"

}

Parcours d'un tableau

```
$T[ ] = 'A';  
$T[10] = 'B';  
$T['A'] = 0;  
$T['B'] = 10;  
foreach ($T as $K => $V) {  
    var_dump($K);  
    var_dump($V);  
}
```

int(0)

string(1) "A"

int(10)

string(1) "B"

string(1) "A"

int(0)

string(1) "B"

int(10)

Parcours d'un tableau

```
$EVAL[ 'Alice' ] = 11.5;
$EVAL[ 'Mohamed' ] = 17;
$EVAL[ 'Julien' ] = 9;
$TOTAL = 0.0;
$NB_NOTE = 0;
foreach($EVAL as $NOTE) {
    $TOTAL += $NOTE;
    $NB_NOTE += 1;
}
printf("Moyenne = %f\n", $TOTAL/$NB_NOTE);
```

- Pas nécessaire de connaître la clé pour faire un parcours

Moyenne = 12.500000

Fonctions sur les tableaux

```
$EVAL['Alice'] = 11.5;  
$EVAL['Mohamed'] = 17;  
$EVAL['Julien'] = 9;  
printf("Moyenne = %f\n", array_sum($EVAL)/count($EVAL));
```

- Beaucoup de fonctions prédéfinies

count() : compte le nombre d'éléments

array_sum() : somme de tous les éléments

array_shift() : retourne la première valeur et réduit de 1 le tableau

array_pop() : retourne la dernière valeur et réduit de 1 le tableau

ksort() : trie le tableau selon ses clés

... environ 80 fonctions pour les tableaux

Tri des valeurs avec perte des clés

```
$VOISINE[ 'Nom' ] = 'Varde';  
$VOISINE[ 'Prenom' ] = 'Melba';  
$VOISINE[ 'Adresse' ] = '9 rue Baratin';  
  
sort( $VOISINE );  
var_dump( $VOISINE )
```

```
array(3) {  
  [0] => string(13) "9 rue Baratin"  
  [1] => string(5) "Melba"  
  [2] => string(5) "Varde"  
}
```

Tri des valeurs avec conservation des clés

```
$VOISINE[0] = 'Varde';  
$VOISINE[1] = 'Melba';  
$VOISINE[2] = '9 rue Baratin';  
  
asort($VOISINE);  
var_dump($VOISINE)
```

```
array(3) {  
    [2] => string(13) "9 rue Baratin"  
    [1] => string(5) "Melba"  
    [0] => string(5) "Varde"  
}
```

Tri des clés (avec conservation des valeurs)

```
$VOISINE[ 'Nom' ] = 'Varde';  
$VOISINE[ 'Prenom' ] = 'Melba';  
$VOISINE[ 'Adresse' ] = '9 rue Baratin';  
  
ksort( $VOISINE );  
var_dump( $VOISINE )
```

```
array(3) {  
    ["Adresse"] => string(13) "9 rue Baratin"  
    ["Nom"] => string(5) "Varde"  
    ["Prenom"] => string(5) "Melba"  
}
```


Blocs et portée des variables

```
$A = 10;  
{  
    print("$A\n");  
    $A = 20;  
    print("$A\n");  
}  
print("$A\n");
```

- Les blocs sont délimités par { }
- Attention : les blocs (sauf ceux des fonctions), **ne permettent pas de déclarer des variables locales**, contrairement aux autres langages (ADA, C, C++)

Affichage : **10 20 20**

La variable A est modifiée

Les fonctions (PHP 7)

```
function nom_fonction (type param_1, ..., type param_n): type {  
    instruction_1;  
    ...  
    instruction_m;  
    [return $resultat;]  
}
```

- Association d'un nom de fonction, de noms de paramètres et d'un bloc.
- Type de paramètres et type de retour de valeurs, optionnels mais conseillés (PHP 7 uniquement)
- Une fonction retourne toujours quelque chose : null par défaut.
- PHP met à disposition un très grand nombre de fonctions (dites natives)
<http://www.php.net>

Type de paramètres des fonctions (PHP 7.3)

```
function inc(?int i,int delta) : ?int {  
    if (i === null) { return null; }  
    else { return i + delta;}  
}  
  
// Retourne la première clé ou se trouve val, null sinon  
Function find(array a, string val): ?string {  
    ...  
}
```

- Tous les types de bases : **int, float, bool, string, array, callable**
- Toutes les classes
- Type "**nullable**" : rends le bon type ou la valeur nulle ('?' avant le type).
Utile par exemple pour une recherche qui n'a rien trouvé (ex: find)
- Identifier les "procédures" avec le type de retour "**void**"
- Valeur par défaut aux paramètres (idem C++)

Passage de paramètres : **par valeur** (i.e. copie)

```
$A = 10;
function afficher(int $A) { // CONTEXTE DE DECLARATION
    print("$A\n");
    $A = 20;
    print("$A\n");
}
afficher($A);                // CONTEXTE D'UTILISATION
print("$A\n");
```

- Inspiré du langage C : par défaut, passage par **copie**.
- Les paramètres sont vus comme **des variables locales** initialisées lors de l'appel (contexte d'utilisation)
- Affiche : **10 20 10**
- Attention : tableaux copiés sur la pile si **modifié** dans la fonction, sinon l'optimiseur de l'interprétation ne fait pas de copie (*copy on write*)

Passage de paramètres : par référence

```
$A = 10;
function afficher(int &$A) { // CONTEXTE DE DECLARATION
    print("$A\n");
    $A = 20;
    print("$A\n");
}
afficher($A);                // CONTEXTE D'UTILISATION
print("$A\n");
```

- Inspiré du langage C++ : passage sur la pile de la référence à la variable (équivalent à une adresse non modifiable)
- Ajouter le caractère **&** au nom du paramètre
- Affiche : **10 20 20**
- Par exemple, utile pour modifier un tableau existant

NB: peu utilisé et à éviter car ce n'est pas de la programmation 'fonctionnelle'

Passage de paramètres : **par référence**

```
function exchange(int &$A, int &$B) {  
    $C = $B;  
    $B = $A;  
    $A = $C;  
}  
  
$X = 10;  
$Y = 20;  
exchange($X,$Y);
```

- Echange des 2 variables
- Attention : limiter l'usage des références, programmer plutôt de manière purement fonctionnelle.

Paramètres par défaut

```
function nouvelle_personne(string $nom,  
                           string $prenom,  
                           string $adresse='adresse inconnue'  
                           ) : array {  
    $Personne[ 'Nom' ] = $nom;  
    $Personne[ 'Prenom' ] = $prenom;  
    $Personne[ 'Adresse' ] = $adresse;  
    return $Personne;  
}
```

```
$moi = nouvelle_personne("Jean-Pierre", "Chevallet");
```

```
var_dump($moi);
```

- Valeurs par défaut : les derniers paramètres
- Ici usage d'un **tableau** comme un **objet** ! (sans les méthodes)

Fonctions, portée et durée de vies des variables

```
$A = 10;
function afficher() {
    $A = 20;
    print("$A\n");
}

afficher();
print("$A\n");
```

- Une variable dans le bloc d'une fonction **est locale** à ce bloc.
- Les variables globales sont **invisibles** depuis un bloc
- La durée de vie des variables d'une fonction est celle du temps de l'exécution de la fonction.
- Les variables d'une fonction sont donc **sur une pile** (comme dans la plupart des langages de programmation)
- Affiche : **20**, puis **10**

Fonctions, portée et durée de vies des variables

```
$A = 10;
function afficher() {
    global $A;
    $A = 20;
    print("$A\n");
}
afficher();
print("$A\n");
```

- Accès à une variable globale : ajouter la déclaration **global**
- Choix :
 - Parce qu'on ne *déclare* pas les variables en PHP
 - Pour éviter les **collisions involontaires** avec les variables globales.
- Affiche : 20, puis 20

NB: conseillé **d'initialiser** les variables en début de code *pour faire comme* une déclaration en Java ou en C++

Fonctions, portée et durée de vies des variables

```
$A = 10;
function afficher() {
    static $A = 20;
    $A++;
    print("$A\n");
}

print("$A\n");
afficher();
print("$A\n");
afficher();
```

- Variable rémanente : **static**
- Initialisée puis garde sa valeur lors de l'appel suivant
- Affiche : 10, puis 21, à nouveau 10, puis 22

Variables super-globales

```
$GLOBALS : Toutes les variables globales
$_SERVER : Infos sur l'exécution du script PHP
$_GET     : Liste des paramètres passés par l'URL
$_POST    : Liste des valeurs passées par POST (HTTP)
$_FILE    : Valeurs des données envoyées par POST (ex: fichiers)
$_COOKIE  : Liste des cookies
$_SESSION: Données sauvée pour la session
$_REQUEST: fusion des GET, POST, et COOKIE
$_ENV     : variables d'environnement
```

- Visible même à l'intérieur des blocs sans avoir à spécifier le mot clé **global**
- Utile pour accéder aux données de l'environnement : protocole HTTP, variables du Shell, état de la connexion, du serveur, etc

Tableaux super-globaux

`$GLOBALS`

Contient l'ensemble des variables globales définies.

`$_SERVER`

Permet de connaître le détail de la requête en cours (nom, chemin de la page,...) et les éléments spécifiques au serveur web (nom, version) et à la connexion (IP paramètres du navigateur,...)

`$_COOKIE`

Contient la liste des variables transmises par le navigateur via les cookies

`$_GET`

Contient la liste des variables transmises via la méthode GET (par l'URL) dans un formulaire

`$_POST`

Contient la liste des variables transmises via la méthode POST dans un formulaire

Tableaux super-globaux

`$_REQUEST`

Ce tableau agrège `$_POST` `$_GET` `$_COOKIE` en un seul tableau. Les valeurs des variables ont été transmises à l'interpréteur PHP via le réseau, depuis le navigateur du client et donc ce sont des variables « sensibles » (leur contenu peut être nocif)

`$_FILES`

Dans un formulaire, il est possible de télécharger des fichiers. Ce tableau décrit ces fichiers.

`$_ENV`

Contient les variables de l'environnement d'exécution de l'interpréteur PHP

`$_SESSION`

Ce tableau permet de manipuler directement les données de session utilisateur

La query string

Envoi au serveur d'un ensemble de couples *nom=valeur* séparés par **&**

nom1=val1&nom2=val2&nom3=val3...

Les caractères spéciaux sont codés par une suite de 1 à 4 octets (UTF-8)

Exemples de chaîne:

ps=toto&nm=Lagaffe&pn=Gaston

nom=Jules&yeux=bleu+clair&nomcinema=Le+M%C3%A9li%C3%A8s

Exemples : € devient **%E2%82%AC**, é devient **%C3%A9**, è devient **%C3%A8**,...

Récupération et décodage de la query string

Elles peuvent être :

- intégrées au langage (dans PHP par exemple)

`$_GET` `$_POST` `$_REQUEST`

- ou disponibles dans les variables d'environnement du système (du shell).
- utilisation de bibliothèques pour les récupérer

Encodage / décodage de la query string

```
//Accès direct à la query string
$q = $_SERVER['QUERY_STRING'];
// Décodage de la query string
echo urldecode($q);
// Encodage de 10€
echo urlencode("10€");

// Encodage complet d'une query string depuis un tableau
$data['article']='téléphone portable';
$data['prix']='236€';
$q = http_build_query($data);
```

- Accès à la query string par les variables super globales : `$_SERVER` ou `$_GET`
- Fonctions de codage et décodage
 - **http_build_query** : construction à partir d'un tableau

Paramètre de type tableau

```
declare(strict_types = 1);

function moyenne(array $t) : float {
    $total = 0.0;
    $nb_notes = 0;
    foreach($t as $note) {
        $total += $note;
        $nb_notes += 1;
    }
    return $total/$nb_notes;
}

$evaluation['Alice'] = 11.5;
$evaluation['Mohamed'] = 17;
$evaluation['Julien'] = 9;

printf("Moyenne = %f\n",moyenne($evaluation));
```

Nombre de paramètres variables : array

```
function affiche(array $personne) {  
    if ($personne['Prenom']) {  
        print $personne['Prenom'].' '  
    }  
    if ($personne['Nom']) {  
        print $personne['Nom'].' '  
    }  
    if ($personne['Adresse']) {  
        print $personne['Adresse'].' '  
    }  
    print "\n";  
}  
  
$VOISINE['Nom'] = 'Varde';  
$VOISINE['Prenom'] = 'Melba';  
$VOISINE['Adresse'] = '9 rue Baratin';  
// Plusieurs paramètres simulé par un tableau  
affiche($VOISINE);
```

Nombre de paramètres variables : array

```
function affiche(array $personne) {  
    foreach($personne as $valeur) {  
        print "$valeur ";  
    }  
    print "\n";  
}  
  
$VOISINE['Nom'] = 'Varde';  
$VOISINE['Prenom'] = 'Melba';  
$VOISINE['Adresse'] = '9 rue Baratin';  
  
affiche($VOISINE);
```

Nombre de paramètres variables

```
function affiche(string ... $args) {  
    foreach($args as $valeur) {  
        print "$valeur ";  
    }  
    print "\n";  
}  
$Nom = 'Varde';  
$Prenom = 'Melba';  
$Adresse = '9 rue Baratin';  
  
affiche($Nom,$Prenom,$Adresse)
```

- Les arguments sont récupérés sous la forme d'un tableau

NB: peu utilisé.

Tableau : à retenir

- Tableaux associatifs généralisés
- Taille variable : pas d'allocation mémoire fixe
- Beaucoup de fonctions
- Tableau super globaux et la query string
- Peut remplacer les types :
 - vecteur (ADA, et tableau en C, vector en C++)
 - matrice
 - dictionnaire associatif
 - liste : mais SANS insertion (sauf si tri par les clés)
 - structure dynamique : avec utilisation des clés comme nom de champs
 - fonction avec nombre variable de paramètres
 -
- Structure de base du langage
 - forcément moins efficace qu'une "vrai" structure de vecteurs
 - chaines plus rapides et plus compactes que les tableaux => binaire brut

Fonction : à retenir

- Déclaration des paramètres : très proche du C++
 - Types de paramètres : int, string, float, array, et tout les types de classes
 - Types de retour, void et type nullable
 - Utiliser la vérification stricte des types : `declare(strict_types=1);`
 - Passage par référence :
 - peu utile car l'appel par valeur est optimisé : si on ne modifie pas la valeur passée (ex: un tableau), alors il n'est pas copié !
- Portée des variables
 - Attention : les variables globales ne sont pas visible par défaut dans un bloc comme c'est le cas en C et C++. Il faut utiliser le mot clé global pour les rendre visibles.