Cours PHP & JS

L2 MI - UE Développement Web

David Lesaint

Université d'Angers

Décembre 2022

Plan

CM UE : Programme

Programme

Période 8 : PHP

- 5 CTD + 7 TP
- CC1 de 1h30 sur machine (coefficient 3/10)

Période 9 : Javascript

- 5 CTD + 7 TP
- CC2 de 1h30 sur machine (coefficient 3/10)

Période 10 : JS / PHP

- 5 CTD + 7 TP
- CC3 de 1h30 sur machine (coefficient 4/10)

Groupes CM et TP

1 groupe de CTD

6 groupes de TP

- I1 : B. Garreau
- I2 : C. Vasconcellos
- I3 : A. Bakki
- I4 : A. Jamin
- I5 : C. Béhuet
- I6 : D. Lesaint

CC sur machine de 1h30

- CC1 : semaine 5
- CC2 : semaine 13
- CC3 : semaine 20

Supports

Espace Moodle : n° 8818, clé nwrbhm

- Groupe, calendrier, planning détaillé
- CM : livret (diaporama) + annexes
- TD et TP : livret des énoncés + annexes + corrections
- CC : énoncés + annexes + dépôts

CM PHP : Introduction à PHP

Développer pour le Web suppose de ...

Maîtriser différents langages et paradigmes de programmation

- Contenu et style des pages web : HTML, CSS, DOM, SVG ...
- Traitements côté client : Javascript, VBScript, ActionScript ...
- Traitements côté serveur : PHP, Python, Ruby, JSP, ASP ...
- Echange de données : XML, JSON, MySQL ...

Se conformer aux pratiques du génie logiciel

- Conception objet (patrons), développement piloté par les tests, intégration continue, ...

S'adapter à un ensemble d'outils en constante évolution

- Librairies (Bootstrap, jQuery, AngularJS ...).
- Cadriciels (Laravel, Symfony, CodeIgniter ...).
- Outils de développement (PHPUnit, PHPDocumentor ...).

Le langage PHP

Origines

- PHP (acronyme pour Php Hypertext Processor) est un langage de scripts interprété.
- Créé en 1994 par Rasmus Lerdorf.
- Utilisé pour produire “facilement” des pages Web dynamiques.

Un langage impératif, orienté objet, fonctionnel

- Structure proche du langage C.
- Typage dynamique des variables.
- Tableaux associatifs, objets, ressources, . . .

Site et documentation officiels

`http://www.php.net/manual/fr`

Objectifs du cours PHP

Maîtriser les bases du langage PHP version 7 :

- Variables, constantes, types
- Instructions de contrôle
- Chaînes de caractères
- Tableaux
- Formulaires
- Fonctions

- ~~Dates~~
- ~~Programmation objet~~
- ~~Images dynamiques~~
- Fichiers
- Cookies, sessions, ~~emails~~
- Bases de données
- ~~XML~~

Cycle de vie d'une page PHP

Trois étapes :

1. Envoi d'une requête HTTP par le client (eg. navigateur) du type

   `http://www.server.com/codephp.php`

2. Interprétation par le serveur du code PHP contenu dans la page demandée (ici `codephp.php`)
   - l'interpréteur renvoie le code HTML après évaluation du code PHP rencontré
3. Envoi par le serveur d'un fichier dont le contenu n'est que du HTML (+ CSS + JavaScript)
   - voir l'onglet `Code source de la page` sous Firefox

Structure des fichiers HTML5

Une page dynamique PHP est un document HTML envoyé au client par le serveur.

Structure de document HTML 5 (pagehtml.html)

```
1  <!DOCTYPE html>
2  <html lang="fr">
3   <head>
4    <meta charset="UTF-8" />
5    <title>Titre de la page</title>
6   </head>
7   <body>
8    <h2>Bienvenue sur le site PHP 7 </h2>
9   </body>
10 </html>
```

On pourrait aussi utiliser le suffixe `.htm` ou `.php`.

Première page PHP

codephp.php

```
1  <!DOCTYPE html>
2  <html lang="fr">
3   <head>
4    <meta charset="UTF-8" />
5    <title>Une page PHP</title>
6   </head>
7   <body>
8    <?php
9     echo "<h3> Aujourd'hui le ". date('d / M / Y H:m:s')."</h3><hr />";
10    echo "<h2>Bienvenue sur le site PHP 7</h2>";
11   ?>
12  </body>
13 </html>
```

- Les marqueurs <?php et ?> délimitent un script PHP.
- On peut inclure autant de scripts PHP que l'on veut dans un document HTML.

Inclusion de fichiers externes

Séparer le HTML du PHP pour plus de modularité/réutilisabilité

Pour pouvoir utiliser dans un script `a.php` des variables/fonctions/... stockées dans un script `b.php`, on utilise l'une des instructions suivantes :

- `include("b.php")` : importe le contenu de `b.php` dans `a.php` sans générer d'erreur si `b.php` n'existe pas.
- `require("b.php")` : idem mais génère une erreur fatale et met fin au script `a.php` en cas d'absence de `b.php`.
- `include_once` et `require_once` se comportent comme `include` et `require` respectivement mais importent une seule fois le fichier demandé.

principal.php

Extension de fichiers PHP externes

On peut utiliser pour extension de fichiers PHP :

- `.inc` : le navigateur affichera le contenu intégral du fichier.
- `.inc.php` : si le fichier ne contient que des affectations de variables (eg, paramètres sensibles), le serveur ne renverra rien au navigateur.

test.inc

```
1 $password = "WILL show in browser";
2 echo $password;
```

test.inc.php

```
1 $password = "WON'T show in browser";
2 echo $password;
```

Organisation de PHP

Organisation modulaire

- Module standard : accès aux types, instructions et fonctions élémentaires.
- Modules additionnels : ajout de fonctionnalités particulières (eg. accès et gestion de diverses bases de données).

Pour connaître la liste des modules disponibles :

- pour PHP en ligne de commande (CLI) :

```
$ php -m
```

- pour PHP déployé sur votre serveur, chargez le script :

```
<?php phpinfo(); ?>
```

Installation d'un serveur web

Elements d'un serveur web :

- Serveur Apache.
- Interpréteur PHP.
- Bases de données pour MySQL, pour SQLite, ...
  - Utilitaires PHPMyAdmin pour créér et gérer bases et tables de données MySQL, SQLiteManager pour SQLite, ...

Serveurs web pour différents systèmes d'exploitation

- XAMPP
- LAMP pour Linux
- WAMP pour Windows
- MAMP pour Mac

CM PHP : Variables et types

Les commentaires

Plusieurs types de commentaires :

- `// sur une seule ligne`
- `# sur une seule ligne`
- ```
  /* sur
  plusieurs
  lignes */
  ```
- ```
  /**
  * sur plusieurs lignes
  * pour génération automatique
  * de documentation (PHPDoc)
  */
  ```

Les variables

Une variable prend l'un des types suivants et peut en changer en cours d'exécution :

- entiers
- flottants
- chaînes de caractères
- booléens
- tableaux
- objets
- ressources
- nul

Identifiants de variables

Un identifiant de variable commence par le caractère $ et est suivi du nom de la variable

- Le nom de la variable commence par une lettre ou le tiret-bas _
- Les caractères suivants sont des lettres, des chiffres ou _

Identifiants corrects

```
$maVar
$_maVar
$MaVar2
$_123
```

Identifiants incorrects

```
maVar
$1maVar
$+maVar
$maVar*
```

Les identifiants de variables sont sensibles à la casse.

Déclaration et initialisation des variables

On déclare les variables où l'on veut dans un script.

Déclarer une variable sans l'initialiser est permis mais sans intérêt.

- Une variable non initialisée est de type `NULL`.

non-initialisation.php

```php
$var;
echo $var; // PHP Notice: Undefined variable: var in /Users/davidlesaint/...
echo gettype($var); // NULL
```

Affectation de variable par valeur et par référence

L'affectation de variable consiste à attribuer une valeur à une variable

- Elle détermine le type de la variable (*typage dynamique*).
- Elle s'effectue par valeur ou par référence.

Affectation par valeur

- Syntaxe : `$var=exp` où `exp` dénote une expression PHP (littéral, variable, appel de fonction ...)
- Sémantique : `$var` prend la valeur de `exp`.

Affectation par référence avec & (esperluette)

- Syntaxe : `$var1=&$var2` où `$var2` dénote une variable.
- Sémantique : `$var1` est un alias de `$var2`.

Affectation de variable par valeur et par référence

On peut réaffecter une même variable au cours d'un script

- Si `$var=exp` et `exp` est une variable, toute réaffectation de `exp` n'aura aucune incidence sur `$var`.
- Si `$var1=&$var2`, toute réaffectation par valeur de l'une des variables sera répercutée sur l'autre mais les 2 variables seront dissociées si l'une est réaffectée par référence.

affectation.php

L'affectation d'un objet ou d'une ressource à une variable se fait systématiquement par référence. Il est possible de cloner un objet.

Les variables superglobales

Variables prédéfinies de type tableau, accessibles à partir de tout script, et contenant des informations sur le serveur et les données transitant entre client et serveur : données de formulaires, fichiers, cookies, sessions ...

Identifiant	Description
`$_GET`	Contient nom et valeur des champs de formulaires envoyés par HTTP GET. Les noms (attribut `name`) des champs du formulaire sont les clés du tableau.
`$_POST`	Contient nom et valeur des champs de formulaires envoyés par HTTP POST. Les noms (attribut `name`) des champs du formulaire sont les clés du tableau.
`$_COOKIE`	Contient nom et valeur des cookies enregistrés sur le poste client. Les noms des cookies sont les clés du tableau.
`$_FILES`	Contient les noms des fichiers téléversés à partir du poste client.
`$_REQUEST`	Contient l'ensemble des superglobales `$_GET`, `$_POST`, `$_COOKIE` et `$_FILES`.
`$_SERVER`	Contient les informations liées au serveur : contenu des en-têtes HTTP, nom du script en cours d'exécution, ...
`$GLOBALS`	Contient nom et valeur des variables globales du script. Les noms des variables sont les clés du tableau. Exemple : `$GLOBALS["var"]` récupère la valeur de `$var`.
`$_ENV`	Contient nom et valeur des variables d'environnement.
`$_SESSION`	Contient l'ensemble des noms des variables de session et leur valeurs.

Superglobale $_SERVER : quelques éléments

Identifiant	Description
`$_SERVER["DOCUMENT_ROOT"]`	La racine web sous laquelle le script est exécuté (eg. /var/www).
`$_SERVER["PHP_SELF"]`	Chemin du script en cours d'exécution par rapport à la racine web.
`$_SERVER["REQUEST_METHOD"]`	Méthode de requête HTTP utilisée pour accéder à la page (`GET`, ...).
`$_SERVER["QUERY_STRING"]`	Chaîne de requête utilisée, si elle existe, pour accéder au script.
`$_SERVER["HTTP_ACCEPT"]`	Contenu de l'en-tête `Accept:` de la requête courante.
`$_SERVER["HTTP_ACCEPT_CHARSET"]`	Contenu de l'en-tête `Accept-Charset` de la requête courante (eg. 'iso-8859-1,*,utf-8').
`$_SERVER["HTTP_ACCEPT_ENCODING"]`	Contenu de l'en-tête `Accept-Encoding` de la requête courante (eg. 'gzip').
`$_SERVER["HTTP_ACCEPT_LANGUAGE"]`	Contenu de l'en-tête `Accept-Language` de la requête courante (eg. 'fr').
`$_SERVER["HTTP_USER_AGENT"]`	Contenu de l'en-tête `User_Agent` de la requête courante.
`$_SERVER["HTTPS"]`	Valeur vide si le script n'a pas été appelé par HTTPS.

Les constantes personnalisées

Sont sensibles à la casse et se définissent

- Avec `const nom = valeur;`
- Ou par appel à `define(`*`string`*` nom, mixed valeur)`
- `defined(`*`string`*` nom)` teste si la constante de nom `nom` existe.

constantes.php

```php
// Définition sensible a la casse
const PI = 3.1415926535;
// Utilisation
echo "La constante PI vaut ",PI,"<br />";
// Vérification de l'existence
if (defined( "PI")) echo "La constante PI est déjà définie","<br />";
// Vérification de l'existence et utilisation
if(define("site","http://www.funhtml.com")) {
 echo "<a href=",site,">Lien vers mon site</a>";
}
```

Les constantes prédéfinies

Les constantes prédéfinies sont nombreuses !

```
<?php print_r(get_defined_constants()); ?>
```

Quelques exemples :

Nom	Description
PHP_VERSION	Version de PHP installée sur le serveur.
PHP_OS	Nom du système d'exploitation du serveur.
DEFAULT_INCLUDE_PATH	Chemin d'accès aux fichiers par défaut.
__FILE__	Nom du fichier en cours d'exécution.
__LINE__	Numéro de ligne du fichier en cours d'exécution.

Les types de données

Type abstrait	Type	Description
Scalaires	`integer`	entiers en base 2, 8, 10 ou 16
	`float ou double`	nombres décimaux
	`string`	chaînes de caractères
	`boolean`	les booléens `TRUE` et `FALSE`
Composés	`array`	tableaux
	`object`	objets
Spéciaux	`resource`	(références à) ressources externes
	`null`	type nul

Les pseudo-types de données

Mots-clés utilisés uniquement pour documenter prototypes de fonctions, constructeurs du langage, mots-clés, ...

- Type d'arguments ou de valeur-retour attendus.
- Absence d'arguments ou de valeur-retour.
- Nombre d'arguments variables ...

Pseudo-type	**Description**
`mixed`	le paramètre peut accepter plusieurs types
`number`	le paramètre est de type `integer` ou `float`
`callable`	le paramètre est une fonction de rappel (alias callback)
`array|object`	le paramètre est de type `array` ou `object`
`$...`	nombre indéfini d'arguments
`void`	pas d'arguments ou de valeur-retour

Détermination du type d'une variable

`get_type(mixed $var):string`

Retourne le type de `$var` sous forme de `string`.

Tests de type (fonctions booléennes) :

- `is_scalar($var)` : teste si `$var` est un scalaire.
- `is_numeric($var)` : teste si `$var` est un nombre (`integer` ou `float`).
- `is_int($var)`, `is_float($var)`, `is_string($var)`, `is_bool($var)`.
- `is_array($var)`, `is_object($var)`.
- `is_resource($var)`, `is_null($var)`.

`var_dump($var)` : affiche type et valeur de `$var`.

Conversion de type (alias transtypage, coercition, cast)

- Par affectation combinée à une coercition

  `$var2 = (type_désiré) $var1;`

- En utilisant la fonction `settype()`

  `boolean settype($var,type_désiré)`

coercition.php

```php
$x="3.14 radians";
echo "\$x is string ".$x."<br/>"; // $x is string 3.14 radians
settype($x,"double");
echo "\$x is double ".$x."<br/>"; // $x is double 3.14
$y = (integer) $x;
echo "\$y is integer ".$y."<br/>"; // $y is integer 3
settype($y,"boolean");
echo "\$y is boolean ".$y."<br/>"; // $y is boolean 1
```

Utile en particulier pour traiter les données de formulaire qui sont par défaut de type `string`

Contrôler l'état d'une variable

- `isset($var)` renvoie `FALSE` ssi `$var` n'existe pas, ou n'est pas initialisée, ou vaut `NULL`.
- `empty($var)` renvoie `TRUE` ssi `$var` n'existe pas, ou n'est pas initialisée, ou vaut `NULL`, `FALSE`, `0`, `""`, `"0"`, ou `[]`.

controle-etat.php

```php
$unini; $null=NULL;
echo (int) isset($undef)."<br/>"; // 0
echo (int) isset($unini)."<br/>"; // 0
echo (int) isset($null)."<br/>"; // 0
echo (int) empty(NULL)."<br/>"; // 1
echo (int) empty(FALSE)."<br/>"; // 1
echo (int) empty(0)."<br/>"; // 1
echo (int) empty(0.0)."<br/>"; // 1
echo (int) empty('')."<br/>"; // 1
echo (int) empty('0')."<br/>"; // 1
echo (int) empty([])."<br/>"; // 1
```

Utile pour tester si un champ de formulaire a bien été saisi

Les entiers (type `integer`)

Codage

- Généralement sur 32 bits (varie selon les plateformes).
- Valeur comprise dans $\{-2^{31}, \ldots, 2^{31} - 1\}$.
- Une variable entière est automatiquement convertie en `double` si sa valeur est en dehors de l'intervalle.

Représentations littérales

- Décimale : `$x=1789; $y=-476;`
- Binaire : `$x=0b11011111101; $y=-0b111011100;`
- Octale : `$x=03375; $y=-0734;`
- Hexadécimale : `$x=0x6FD; $y=-0X1DC;`

Les entiers sont affichés en base 10 par `echo`.

Les flottants (type `double`)

Le type `double` représente les nombres décimaux avec une précision de 14 chiffres

- Voir librairie `BCMath` pour permettre des calculs plus précis.

Représentations littérales

- Notation décimale avec le point comme séparateur : `$x=101.23;`
- Notation scientifique avec `E` ou `e` : `$x=1.0123E2;`

Les flottants sont affichés sous forme décimale si le nombre a moins de 15 chiffres, sous forme exponentielle sinon.

Les opérateurs numériques

S'appliquent aux opérandes de type numérique

Opérateur	**Description**
+	addition
-	soustraction
*	multiplication
**	exponentiation/puissance (associatif à droite)
/	division
%	modulo (s'applique aux flottants par restriction aux parties entières)
--	pré-décrementation (`--$x`) ou post-décrémentation (`$x--`)
++	pré-incrémentation (`++$x`) ou post-incrémentation (`$x++`)

Les opérateurs d'affectation combinée

Affectent le résultat à l'opérande de gauche.

Opérateur	Description
`+=`	`$x += $y` équivaut à `$x = $x + $y`. `$y` peut être une expression complexe dont la valeur est un nombre.
`-=`	`$x -= $y` équivaut à `$x = $x - $y`. `$y` peut être une expression complexe dont la valeur est un nombre.
`*=`	`$x *= $y` équivaut à `$x = $x * $y`. `$y` peut être une expression complexe dont la valeur est un nombre.
`**=`	`$x **= $y` équivaut à `$x = $x^$y`. `$y` peut être une expression complexe dont la valeur est un nombre.
`/=`	`$x /= $y` équivaut à `$x = $x / $y`. `$y` peut être une expression complexe dont la valeur est un nombre différent de 0.
`%=`	`$x %= $y` équivaut à `$x = $x % $y`. `$y` peut être une expression complexe dont la valeur est un nombre différent de 0.
`.=`	`$x .= $y` équivaut à `$x = $x . $y`. `$y` peut être une expression complexe dont la valeur est une chaîne de caractères.

Les fonctions mathématiques

De nombreuses fonctions sont disponibles par défaut :

- Conversions : `integer hexdec(string) ...`
- Arrondis : `double ceil(double) ...`
- Trigonométrie : `double cos(double) ...`
- Logarithmes : `double log(double X,double B) ...`
- Génération de nombres aléatoires : `integer rand(integer min, integer max) ...`
- ...

Les booléens (type `boolean`)

Le type `boolean` contient uniquement les valeurs `TRUE` et `FALSE`

Evaluation d'expression en contexte booléen (par exemple, instructions conditionnelles)

- Expression à valeur booléenne
  - `$x<10` vaut `TRUE` ssi `$x` est de valeur inférieure à 10.
- Expression. à valeur non booléenne :
  - `$x` vaut `TRUE` ssi `$x` est initialisée à une valeur *non nulle*.

Pour l'affichage, PHP assimile `TRUE` à `"1"` et `FALSE` à `""`

Règles d'évaluation booléenne d'expressions

Expressions évaluées à `FALSE`	Une expression logique fausse utilisant un ou plusieurs opérateurs de comparaison
	Le mot-clé `FALSE`
	La valeur entière 0
	La valeur décimale 0.0
	Les chaînes "" et "0"
	Une variable de type `null`
	Une variable non initialisée
	Un tableau vide
	Un objet sans propriétés ni méthodes
Expressions évaluées à `TRUE`	Toute autre possibilité dont : - Les entiers strictement positifs ou négatifs - La chaîne `"FALSE"` - Les variables de type `resource`

Règles d'évaluation booléenne d'expressions

evaluation-booleenne.php

```php
// Expressions fausses
if(!FALSE)      var_dump(FALSE);     // bool(false)
if(!0)          var_dump(0);         // int(0)
if(!0.0)        var_dump(0.0);       // float(0)
if(!"")         var_dump("");        // string(0) ""
if(!"0")        var_dump("0");       // string(1) "0"
if(!null)       var_dump(null);      // NULL
if(!isset($x))  echo "1";            // 1
if(![])         var_dump([]);        // array(0) {}
// Expressions vraies
if(-1)          var_dump(-1);        // int(-1)
if("FALSE")     var_dump("FALSE");   // string(5) "FALSE"
$f = fopen(__FILE__,"r");
if($f)          var_dump($f);        // resource(5) of type (stream)
```

Les opérateurs booléens

Se divisent en

- Opérateurs de comparaison (binaires).
- Opérateurs logiques de composition (unaires/binaires).

Opérateurs de comparaison

Opérateur	Description
==	égalité
!= ou <>	inégalité
<	strictement inférieur
<=	inférieur ou égal
>	strictement supérieur
>=	supérieur ou égal
<=>	$x<=>$y vaut -1 ssi $x<$y, 0 ssi $x=$y, 1 ssi $x>$y ("spaceship operator" introduit en PHP 7)
===	égalité des valeurs **et des types**
!==	inégalité des valeurs **ou des types**

Comparaison et coercition implicite

La comparaison via == d'arguments de types différents transtype l'un des arguments en appliquant les règles de transtypage PHP

operateur-comparaison.php

```php
var_dump(FALSE==0); // TRUE
var_dump(FALSE==0.0); // TRUE
var_dump(FALSE==""); // TRUE
var_dump(FALSE=="0"); // TRUE
var_dump(FALSE==NULL); // TRUE
var_dump(FALSE==[]); // TRUE
var_dump(0.0==0);  // TRUE - conversion (double) 0
var_dump(0==""); // TRUE - conversion (integer) ""
var_dump(41=="41ok12");  // TRUE - conversion (integer) "41ok12"
var_dump(41==" 41ok12");  // TRUE - conversion (integer) " 41ok12"
var_dump(0=="a41ok12");  // TRUE - conversion (integer) "a41ok12"
var_dump(NULL==""); // TRUE
var_dump(NULL==[]);  // TRUE
/////
var_dump(""=="0");  // FALSE
var_dump(""==[]); // FALSE
var_dump(0==[]);  // FALSE
```

Comparaison stricte

Pour éviter les désagréments, privilégier les opérateurs === et !==

operateur-comparaison-stricte.php

```php
var_dump(TRUE===1);   // FALSE
var_dump(""===NULL);  // FALSE
var_dump(1e2==="100"); // FALSE
```

Les opérateurs booléens

Opérateurs logiques

Opérateur	Description
`OR`	disjonction
`||`	équivaut à `OR` mais n'a pas la même priorité
`XOR`	disjonction exclusive
`AND`	conjonction
`&&`	équivaut à `AND` mais n'a pas la même priorité
`!`	négation

Priorité des opérateurs en PHP

Les chaînes de caractères

Suites de caractères alphanumériques encadrées par

- des apostrophes : `'PHP7 et MySQL'`
- des guillemets : `"PHP7 et MySQL"`

Traitement d'une variable apparaissant dans une chaîne

- Son identifiant est affiché si la chaîne est encadrée par des apostrophes.
  - `<?php $x="A"; echo '$x'; ?>` affiche `$x`.
- Sa valeur est affichée si la chaîne est encadrée par des guillemets.
  - `<?php $x="A"; echo "$x"; ?>` affiche `A`.

Affichage mixte à l'aide de séquences d'échappement

`<?php $x="A"; echo "\$x vaut $x"; ?>` affiche `$x vaut A`.

Les tableaux

Un tableau stocke des *éléments* (valeurs) indépendants

- Les éléments peuvent prendre n'importe quel type.
- Les éléments peuvent être de types différents.

On peut créer des tableaux multi-dimensionels avec des tableaux de tableaux.

Deux types de tableaux :

- Tableau *indicé* : les éléments sont repérés par des indices numériques.
- Tableau *associatif* : les éléments sont repérés par des *clés* (chaîne ou variable de type chaîne).

Les types spéciaux

Le type `resource`

- Une valeur de type `resource` est une référence vers une ressource externe : fichier ouvert, connexion à base de données (BDD), image, ...
- Libérées automatiquement par le ramasse-miettes sauf cas des connexions persistantes à BDD.

Le type `null` (ou `NULL`)

Attribué à une variable sans contenu ou qui a été explicitement initialisée à la valeur `NULL`.

- `""` et `"0"` ont le type `string`.
- `0` a le type `integer`.

CM PHP : Structures de contrôle

Les instructions de contrôle

Instructions de contrôle des scripts

Syntaxe et sémantique proches du C/C++.

Trois types

- Instructions conditionnelles :
  - `if` et `if...else`
  - `?` et `??`
  - `switch...case`
- Instructions de boucle :
  - `for`
  - `while` et `do...while`
  - `foreach`
  - `break`, `continue` , ~~`goto`~~
- Gestion des erreurs :
  - `error_reporting`
  - ~~`try...catch...finally`~~

L'instruction `if`

`if(exp) instruction;`

`instruction` est exécutée si `exp` est évaluée à `TRUE`.

`if(exp) {bloc}`

Le `bloc` d'instructions est exécuté si `exp` est évaluée à `TRUE`.

Différentes types d'expressions possibles :

- Booléenne (`$x>2`) ou non (`$y` avec `$y="abc"`).
- Atomique (`$x>2`) ou composite (`$x>2 || $y`).

Confer cours 2

- Tableau des règles d'évaluation booléenne d'expression.
- Liste des opérateurs logiques de comparaison et composition.

L'instruction if...else

if(exp) ins1; else ins2;

Exécute ins1 si exp est évaluée à TRUE ou ins2 sinon.

if(exp) {bloc1} else {bloc2}

Exécute bloc1 si exp est évaluée à TRUE ou bloc2 sinon.

exemple3-1.php

```
1  $prix=55;
2  if($prix>100)
3  {
4   echo "<b>Pour un montant d'achat de $prix &#8364;, la remise est de 10 % </b><br>";
5   echo "Le prix net est de ",$prix*0.90;
6  }
7  else
8  {
9   echo "<b>Pour un montant d'achat de $prix &#8364;, la remise est de 5 %</b><br>";
10  echo "<h3>Le prix net est de ",$prix*0.95," &#8364;</h3>";
11 }
```

Les if imbriqués avec if...elseif...else

exemple3-2.php

```php
// ********************if...elseif...else****************
$cat="PC";
$prix=900;
if($cat=="PC")
{
 if($prix>= 1000)
 {
  echo "<b>Pour l'achat d'un PC d'un montant de $prix &#8364;, la remise est de 15 %</b><br>";
  echo "<h3> Le prix net est de : ",$prix*0.85, "&#8364; </h3>";
 }
 else
 {
  echo "<b>Pour l'achat d'un PC d'un montant de $prix &#8364;, la remise est de 10 %</b><br>";
  echo "<h3> Le prix net est de : ",$prix*0.90, "&#8364; </h3>";
 }
}
elseif($cat=="Livres")
{
 echo "<b>Pour l'achat de livres la remise est de 5 %</b><br />";
 echo "<h3> Le prix net est de : ",$prix*0.95, "&#8364; </h3>";
}
else
{
 echo"<b>Pour les autres achats la remise est de 2 %</b><br>";
 echo "<h3> Le prix net est de : ",$prix*0.98, "&#8364; </h3>";
}

```

L'opérateur ternaire ?

`exp ? val1 : val2`

Renvoie `val1` si `exp` est évaluée à `TRUE` ou `val2` sinon.

Usage : `$var = exp ? val1 : val2;`

Equivaut à : `if(exp){$var=val1;} else {$var=val2;}`

exemple3-3.php

```php
$ch = "Bonjour ";
$sexe="M";
$ch .= ($sexe=="F")?"Madame":"Monsieur";
echo "<h2>$ch</h2>";
$nb = 3;
$pmu ="Il faut trouver ".$nb;
$mot = ($nb==1)?" cheval":" chevaux";
echo "<h3> $pmu $mot </h3>";
```

L'opérateur binaire *null coalescent* ?? (PHP 7)

`$var ?? exp`

Renvoie `$var` si `$var` initialisée à non `NULL` ou `exp` sinon.

Usage : `$x = $y ?? exp;`

Equivaut à : `$x = isset($y) ? $y : exp;`

operateur-coalescing.php

```php
// 3 instructions équivalentes
$x = $_GET['user'] ?? 'aie';
$x = isset($_GET['user']) ? $_GET['user'] : 'aie';
if(isset($_GET['user'])) $x = $_GET['user']; else $x = 'aie';
```

Chaînage possible

```php
$x = $_GET['user'] ?? $_POST['user'] ?? 'aie';
```

L'instruction `switch...case`

Alternative à `if...elseif...else`

Pour simplifier l'écriture de branchements conditionnels imbriqués.

Syntaxe

```
switch(exp) {
   case val1:
      bloc1;
      break;
   ...
   case valN:
      blocN;
      break;
   default:
      blocD;
      break;
}
```

Sémantique

- Si `exp` vaut `val1`, le `bloc1` est exécuté et l'exécution passe à la fin du bloc `switch`.
- Sinon, la procédure est réitérée sur les valeurs `val2` ...`valN` jusqu'à concordance.
- Si aucune concordance n'est trouvée, le `blocD` est exécuté.

L'instruction switch...case

exemple3-4.php

```php
$dept=75;
switch($dept) {
//Premier cas
    case 75:
    case "Capitale":
        echo "Paris";
        break;
//Deuxième cas
    case 78:
        echo "Hauts de Seine";
        break;
//Troisième cas
    case 93:
    case "Stade de France":
        echo "Seine Saint Denis";
        break;
//la suite des départements...
//Cas par défaut
    default:
        echo "Département inconnu en Ile de France";
        break;
```

La boucle for

Syntaxe

`for(exp1;exp2;exp3){instruction;}`
ou `for(exp1;exp2;exp3){bloc}`

Sémantique

1. `exp1` est évaluée.
2. `exp2` est évaluée en contexte booléen : si elle vaut `TRUE`, l'`instruction` (ou le `bloc` d'instructions) est exécutée, sinon on sort du bloc `for`.
3. `exp3` est exécutée et la boucle reprend à l'étape (2) jusquà ce que `exp2` soit évaluée à `FALSE`.

exemple3-5.php

```
1 for($i=1;$i<7;$i++) {
2     echo "<h$i> $i :Titre de niveau $i </h$i>";
3 }
```

Boucle à plusieurs variables

Sous-expressions séparées par des virgules

- Multiples initialisations (eg, plusieurs compteurs).
- Multiples conditions.
- Multiples in-/dé-crémentations.

exemple3-6.php

```php
for($i=1,$j=9;$i<10,$j>0;$i++,$j--) {
//$i varie de 1 à 9 et $j de 9 à 1
    $css="style=\"border-style:double;border-width:3;\"";
    echo "<span ".$css.">$i + $j=10</span>";
}
```

Boucles imbriquées

exemple3-7.php

```
1  echo "<h2>Révisez votre table de multiplication!</ h2>";
2  //Début du tableau HTML
3  echo "<table border=\"2\" style=\"background-color:yellow\"> <th> X
 </th>";
4  //Création de la première ligne
5  for($i=1;$i<10;$i++)
6   echo "<th> $i </th>";
7  //Création du corps de la table
8  //Boucles de création du contenu de la table
9  for($i=1;$i<10;$i++) {
10  //Création de la première colonne
11  echo "<tr><th> $i </th>";
12  //Remplissage de la table
13  for($j=1;$j<10;$j++) {
14   echo "<td style=\"background-color:red;color:white\">   
<b>". $i*$j."   </td>";
15  }
16  echo "</b></tr>";
17 }
18 echo "</table>"
```

La boucle while

Alternative à for

Quand on ne connaît pas a priori le nombre limite d'itérations.
Exemple : afficher les résultats d'une requête sur une BDD.

while(exp){instruction;} ou while(exp){bloc}

Tant que exp est évaluée à TRUE, exécute l'instruction (ou le bloc d'instructions).

exemple3-8.php

```php
$n=1;
while($n%7!=0)
{
 $n = rand(1,100);
 echo $n,"  /";
}
```

La boucle do...while

`do {bloc} while(exp);`

- Similaire à `while` mais `bloc` est au moins exécutée une fois avant d'évaluer `exp`.
- Les variables évaluées dans `exp` peuvent être initialisées dans `bloc`.

exemple3-9.php

```
1 do
2 {
3  $n = rand(1,100);
4  echo $n,"  / ";
5 }
6 while($n%7!=0);
```

Sortie anticipée de boucle avec `break`

`break`

Arrête une boucle `for`, `foreach` ou `while` avant son terme

- N'arrête pas le script contrairement à `exit`.
- Arrête uniquement la boucle qui le contient dans le cas de boucles imbriquées.
- `break n;` arrête les `n` boucles les plus internes imbriquant l'instruction.

Sortie anticipée de boucle avec break

exemple3-13.php

```php
//Création d'un tableau de noms
$tab[1]="Basile"; $tab[2]="Conan";
$tab[3]="Albert"; $tab[4]="Vincent";
//Boucle de lecture du tableau
for($i=1;$i<count($tab);$i++) {
 if ($tab[$i][0]=="A") {
  echo "Le premier nom commençant par A est le numéro $i: ",$tab[$i];
  break;
 }
}
```

- `$tab[$i][n]` ($n \geq 0$) correspond à la `n`+1-ième lettre de la chaîne `$tab[$i]`.
- `$i<count($tab)` évitera une boucle infinie éventuelle si aucun mot de `$tab` ne démarre avec `A`.

Avancement de boucle avec continue

continue

Arrête l'itération en cours, pas la boucle.

exemple3-14.php

```php
//Interruption d'une boucle for
for($i=0;$i<20;$i++)
{
 if($i%5==0) { continue;}
 echo $i,"<br />";
}
//Interruption d'une boucle foreach
$tab[1]="Ain";
$tab[2]="Allier";
$tab[27]="Eure";
$tab[28]="Eure-et-Loir";
$tab[29]="Finistère";
$tab[33]="Gironde";
foreach($tab as $cle=>$valeur)
{
 if($tab[$cle][0]!="E") { continue;}
 echo "code $cle : département ",$tab[$cle],"<br />";
```

Avancement de boucle avec continue

`continue n;`

Arrête les n-1 boucles les plus internes l'imbriquant et l'itération courante de la n-ième.

exemple3-15.php

```php
for ($i=0;$i<10;$i++)
{
 for ($j=0;$j<10;$j++)
 {
  for ($k=0;$k<10;$k++)
  {
   if(($i+$j+$k)%3==0) continue 3;
   echo "$i : $j : $k <br /> ";
  }
 }
}
```

Gestion des erreurs

Filtrage des messages d'erreur renvoyés au poste client

- Nomenclature (partielle) des erreurs en PHP :

Constante	Valeur	Niveau d'affichage
`E_ERROR`	1	Erreur fatale (eg. appel de fonction inexistante) : le script s'arrête.
`E_WARNING`	2	Avertissement (eg. division par 0) : le script se poursuit.
`E_PARSE`	4	Erreur de syntaxe : le script s'arrête.
`E_NOTICE`	8	Avis de problème simple.
...		
`E_ALL`	2^{15}	Toute erreur.

- Le type d'erreurs relayées est prédéfini et configurable dans le fichier de configuration `php.ini`
- On peut l'ajuster pour chaque script en y insérant `error_reporting(n);` au début ce qui n'affichera que les erreurs dont les valeurs correspondent aux bits positionnés dans `n`.

CM PHP : Chaînes

Les chaînes de caractères

Suites de caractères encadrées par des

- Apostrophes : `'PHP 8 et MySQL'`
- Guillemets : `"PHP 8 et MySQL"`

Importance des chaînes de caractères

- Constituent l'essentiel du contenu des pages Web.
- Sont manipulées pour créér des pages à partir de fichiers ou de BDD.
- Sont le type des données envoyées par formulaire, des cookies, des données de sessions . . .

Affichage des chaînes de caractères

Avec `echo` ou la fonction `print()`

Traitement des variables apparaissant dans une chaîne

- Affichage litéral de son identifiant si la chaîne est encadrée par des apostrophes.
  - `<?php $x="A"; echo '$x'; ?>` affiche `$x`.
- Interpolation (affichage de sa valeur) si la chaîne est encadrée par des guillemets.
  - `<?php $x="A"; echo "$x"; ?>` affiche `A`.

Affichage mixte à l'aide de séquences d'échappement

`<?php $x="A"; echo "\$x vaut $x"; ?>` affiche `$x vaut A`.

Séquences d'échappement

Pour afficher

- Des caractères protégés du langage (eg, $).
- Des caractères de contrôle (eg. un retour à la ligne).
- Des caractères à partir de leur code ASCII octal ou hexadécimal.

Séquence	**Signification**
`\'`	affiche une apostrophe
`\"`	affiche un guillemet droit
`\$`	affiche le symbole $
`\\`	affiche une barre oblique inversée \ (alias backslash)
`\n`	nouvelle ligne (code ASCII 0x0A)
`\r`	retour chariot (code ASCII 0x00)
`\t`	tabulation (code ASCII 0x09)
`\[0-7] {1,3}`	affiche le caractère dont le code ASCII en octal correspond à la séquence de caractères (séquence de 1 à 3 chiffres pris entre 0 et 7). `echo "\101";` affiche `A`
`\x[0-9 A-F a-f] {1,2}`	affiche le caractère dont le code ASCII en hexadécimal correspond à la séquence de caractères (séquence de 1 à 2 caractères). `echo "\x4A";` affiche `J`

Concaténation de chaînes

Avec l'opérateur point (.)

```php
$str1 = "AA";
$str2 = "BB";
$str3 = $str1.$str2."<br/>";
echo $str3; // affiche AABB<br/>
```

Avec la virgule (,) pour echo

```php
$str1 = "AA";
$str2 = "BB";
echo $str1,$str2,"<br/>";   // affiche AABB<br/>
```

Syntaxes alternatives

La syntaxe Heredoc permet de définir de longues chaînes (eg, fragment HMTL) et fonctionne comme les guillemets

```php
$p = 2;
$str=<<<IDF
hello variable \$p $p
IDF;
// !!! AUCUN caractère avant et après "IDF;"
echo $str;
```

La syntaxe Nowdoc permet de définir de longues chaînes et fonctionne comme les apostrophes

```php
$p = 2;
$str=<<<'IDF'
hello variable \$p $p
IDF;
// !!! AUCUN caractère avant et après "IDF;"
echo $str;
```

Longueur, Accès, Codage ASCII

`strlen(string):int`

Donne le nombre de caractères de la chaîne.

`empty(mixed):boolean`

Teste si une chaîne est vide.

Accès aux caractères avec `[]`

Les chaîne de caractères sont assimilables à des tableaux indicés (le premier caractère est d'indice 0).

`ord(string l):int` et `chr(int c):string`

- `ord` retourne le code ASCII du caractère `l`.
- `chr` retourne le caractère de code ASCII `c`.

Modification de la casse

Minuscules / Majuscules / Capitalisation

- `strtolower(string):string` : tout en minuscules.
- `strtoupper(string):string` : tout en majuscules.
- `ucfirst(string):string` : 1ère lettre en majuscule.
- `ucwords(string):string` : 1ère lettre de chaque mot en majuscule.

Suppression des caractères invisibles

- `trim(string $ch [, charlist]):string` supprime tout espace, tabulation, retour chariot et caractère nul apparaissant au début et en fin de `$ch` ou bien tout caractère dans `charlist`
- `ltrim(...)` supprime en début de chaîne.
- `rtrim(...)` supprime en fin de chaîne.

Entités HTML et caractères spéciaux

`htmlspecialchars(string $ch [, ...]):string`

- Remplace les caractères & " ' < > par leurs entités HTML.
- Utile pour créer des noeuds texte HTML ou XML contenant des caractères spéciaux HTML//XML.

`htmlspecialchars_decode(string $ch [int $flags]):string`

- Fonction inverse.

`htmlentities(string $ch [, ...]):string`

- Remplace tous les caractères éligibles (Unicode>128) par leurs entités HTML.

`html_entity_decode(string $ch [, ...]):string`

- Fonction inverse.

Entités HTML et caractères spéciaux

`addslashes(string $ch) :string`

- Echappe (ajoute \ devant) toute occurrence de ' " \ et du caractère NUL dans `$ch`.
- Utile pour enregistrer des données envoyées par formulaire en BDD (sans requêtes préparées).

`stripslashes(string $ch):string`

- Fonction inverse.

`quotemeta(string $ch) :string`

- Echappe les caractères réservés des regex . \ + * ? [ ] ( ) $ ^

Recherche de sous-chaînes

`substr_count(string $ch1, string $ch2):int`

- Renvoie le nombre d'occurrences de `$ch2` dans `$ch1`.

`strstr(string $ch1, string $ch2):string`

- Renvoie `FALSE` si `$ch2` n'apparaît pas dans `$ch1` ou sinon tous les caractères allant de la 1ère occurrence de `$ch2` dans `$ch1` jusqu'à la fin de `$ch1`.

`substr(string $ch, int i [, int N]):string`

- Renvoie la sous-chaîne de `$ch` commençant à la position `i` et de longueur maximale `N`.

Recherche de sous-chaînes

`str_replace(string $ch1, string $ch2, string $ch [, string $n]):string`

- Remplace les occurrences de `$ch1` par `$ch2` dans `$ch`. `$n` passée par référence est le nombre de remplacements effectués.

exemple4-5.php

```php
$ch = "Perette et le pot au lait. C'est pas de pot!" ;
$ssch = substr($ch, 8, 9) ;
echo $ssch,"<br />" ;
$ssch = substr($ch,8);
echo $ssch ,"<br />";
$ch2="pot";
$nb=substr_count($ch,$ch2);
echo "Le mot $ch2 est présent $nb fois dans $ch <br />";
$ch3=str_replace('pot','bol',$ch);
echo $ch3,"<br />" ;
```

Recherche de position ou d'existence d'un mot

`strpos(string $ch1, string $ch2 [, int p]):int`

- Donne la position de la chaîne `$ch2` dans `$ch1` en commençant au début de la chaîne ou à la position `p`. Renvoie `FALSE` si le motif n'est pas trouvé.

strpos.php

```php
$chn="do you go to Togo ?";
$pattern="go";
$pos=0;
while (true) {
   $pos=strpos($chn,$pattern,$pos);
   /* ATTENTION : l'usage du == au lieu du === ne donnerait pas le
    * fonctionnement "attendu" car la position 0 serait automatiquement
    *  convertie à FALSE et le test réussirait */
 if ($pos === false) break;
 echo "found pattern at $pos\n";
 ++$pos;
}
```

Comparaison de chaînes

== compare une chaîne à un nombre en la convertissant en nombre (ses premiers caractères numériques sont retenus)

- Conversion identique pour les opérations arithmétiques (+, -, *, /, %) entre chaîne et nombre.

<, >, <= et >= comparent les chaînes selon le code ASCII

Alternatives :

- `strcmp(string $ch1, string $ch2):int` : sensible à la casse.
- `strcasecmp(string $ch1, string $ch2):int` : insensible à la casse.

strcmp.php

```php
$ch1 = "Blanc"; $ch2 = "Bleu"; $ch3 = "blanc";
echo strcasecmp($ch1,$ch3)."<br/>"; // affiche 0
echo strcmp($ch1,$ch2)."<br/>"; // affiche -4
echo strcmp($ch1,$ch3)."<br/>"; // affiche -32
```

Les expressions régulières

Expression régulière (alias motif, masque, regex, modèle)

Chaîne de caractères qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles.

- Adresses e-mail.
- Numéros de téléphone.
- Code INSEE ...

Encadrement des motifs avec / et /

`/motif/` pour la regex `motif`.

Caractères spéciaux (alias méta-caractères)

`\ + * ? [ ] ( ) $ ^`

Motifs élémentaires

Recherche de chaînes précises

- `/angers/` : `angers` apparaît dans la chaîne analysée.
- `/\.fr/` : `.fr` apparaît.
- `/a|b/` : `a` ou `b` apparaît.

Recherche d'un ou plusieurs caractères

- `/[axz]/` : l'un des caractères `a`, `x`, `z`.
- `/[b-p]/` : l'une des minuscules entre `b` et `p`.
- `/[A-Z]/` : une majuscule.
- `/[0-9]/` : un chiffre.

Recherche de méta-caractères

A échapper avec l'antislash.

Classes de caractères prédéfinies

Classe	Recherche
`[[:alnum:]]`	Tous les caractères alphanumériques : [a-zA-Z0-9].
`[[:alpha:]]`	Tous les caractères alphabétiques : [a-zA-Z].
`[[:blank:]]`	Tous les caractères blancs : espaces, tabulations . . .
`[[:ctrl:]]`	Tous les caractères de contrôle.
`[[:digit:]]`	Tous les chiffres : [0-9]
`[[:print:]]`	Tous les caractères imprimables sauf caractères de contrôle.
`[[:punct:]]`	Tous les caractères de ponctuation.
`[[:space:]]`	Tous les caractères d'espace : espaces, tabulations, sauts de ligne, . . .
`[[:upper:]]`	Tous les caractères en majuscules : [A-Z].
`[[:xdigit:]]`	Tous les caractères en hexadécimal.

Restriction de caractères

Avec ^ entre crochets

- `/[^axz]/` : un caractère différent de `a`, `x`, `z`.
- `/[^A-Z]/` : un caractère qui n'est pas une majuscule.

Début et fin de chaîne :

Avec ^ hors crochets

- `/^axz/` : toute chaîne démarrant par `axz`.

Avec $

- `/axz$/` : toute chaîne finissant par `axz`.

Motifs généraux

N'importe quel caractère

Avec `.` (non échappé)

- `/a.z/` : toute chaîne contenant `a` suivi d'un caractère suivi de `z`.

Répétition de caractères

Avec `?` (0 ou 1 fois), `+` (au moins une fois), `*` (0 ou plus)

- `/ab?/` : présence d'un `a` non suivi d'un `b` ou suivi d'un seul `b`.
- `/ab+/` : présence d'un `a` suivi d'une suite d'au moins un `b`.
- `/ab*/` : présence d'un `a` suivi ou non d'une suite de `b`.

Motifs généraux

Sous-motifs

Regroupement avec `()`

- `/(ab)+/` : toute chaîne contenant une série de `ab`.

Contraintes de cardinalité

Avec `{n}` (n répétitions)

- `/(ab){5}/` : série de 5 `ab`.

Avec `m,n` (entre m et n répétitions) :

- `/(ab){3,5}/` : série de 3 à 5 `ab`.

Avec `m,` (au moins m répétitions) :

- `/(ab){3,}/` : série d'au moins 3 `ab`.

Les fonctions de recherche

`preg_match(string $motif, string $ch [, array $tab]):int|false`

- Recherche une sous-chaîne de `$ch` satisfaisant la regex `$motif`. Retourne 1 le cas échéant, 0 sinon ou `FALSE` si une erreur survient.
- `$tab` est un tableau indicé contenant `$ch` comme premier élément puis toutes les sous-chaînes satisfaisant à `$motif` comme éléments suivants.

preg-match.php

```php
$ch1="une date : 30-12-1970";
$ch2="4-01-18";
$motif="/([0-9]{1,2})-([0-9]{2})-([0-9]{2,4})/i";
preg_match($motif,$ch1,$result);
print_r($result); //Array([0] => 30-19-1970 [1] => 30 [2] => 19 [3] => 1970)
preg_match($motif,$ch2,$result); echo "<br/>";
print_r($result);//Array([0] => 4-01-18 [1] => 4 [2] => 01 [3] => 18)
```

Les fonctions de recherche

`preg_replace(string $motif, string $rep, string $ch):string|array|null`

- Remplace toute occurrence du motif `$motif` dans `$ch` par `$rep` et retourne la chaîne résultat.

preg-replace.php

```php
$chn="deux dates : 30-12-1970 et 19-05-1900";
$motif="/(\d+)-(\d+)-(\d{2,4})/i";
$remplace="$3/$2/$1";
$str=preg_replace($motif,$remplace,$chn);
echo "chaîne de départ : $chn<br/>";
echo "chaîne résultat  : $str<br/>";
```

Autres fonctions pour les expressions régulières

- `preg_split`
- `preg_grep`
- `preg_filter`
- `preg_match`
- `preg_match_all`

CM PHP : Tableaux

Les tableaux

Un tableau stocke des *éléments* (valeurs) indépendants

- Les éléments peuvent prendre n'importe quel type.
- Les éléments peuvent être de types différents.

Deux types de tableaux :

- Tableau *indicé* : les éléments sont repérés par des indices numériques.
- Tableau *associatif* : les éléments sont repérés par des *clés* (chaîne ou variable de type chaîne).

Les tableaux indicés

Création d'éléments avec []

- `$tab[]=exp;` ajoute l'élément `exp` en fin du tableau `$tab`, c.a.d. à l'indice qui suit l'indice maximum ou 0 si `$tab` est vide.
- `$tab[n]=exp;` initialise/remplace l'élément d'indice `n` du tableau `$tab` à/avec l'expression `exp`.
- `print_r` formatte l'affichage de tableaux.

tableau-indice-creation1.php

```php
$tab[0]="UFR"; // équivalent ICI à $tab[]="UFR";
$tab[1]="Sciences"; // équivalent ICI à $tab[]="Sciences";
$tab[49]="Angers";
$tab[]="janvier"; // équivalent ICI à $tab[50]="janvier";
$i = 3;
$tab[$i]="Université";   // équivalent ICI à $tab[3]="Université";
$tab[]="2017";   // équivalent ICI à $tab[51]="2017";
echo count($tab),"<br/>"; // affiche 6
print_r($tab);
```

Les tableaux associatifs

Création d'éléments avec []

Les clés (littéral ou variable `string`) remplacent les indices.

- Les clés ne comportent pas d'espace, sont encadrées par apostrophes/guillemets, et sont sensibles à la casse.
- Encadrer tout élément de tableau utilisé dans une chaîne par des accolades
  - `"{$tab['alpha']}"` et non pas `"$tab['alpha']"`.

tableau-associatif-creation1.php

```php
$tab["zero"]="UFR";
$tab[1]="Sciences";
$tab["forty-nine"]="Angers";
$tab[]="janvier"; // équivalent ICI à $tab[2]="janvier";
$tab["3"]="Université";
$tab[51]="2017";
echo count($tab),"<br/>";
print_r($tab);
```

Les tableaux associatifs

exemple2-3.php

```
 1 <!DOCTYPE html>
 2 <html lang="fr">
 3  <head>
 4   <meta charset="UTF-8" />
 5  </head>
 6  <body>
 7  <?php
 8  //création des éléments du tableau
 9  $tab["php"] = "php.net";
10  $tab["mysql"] = "mysql.com";
11  $tab["html"] = "w3.org";
12  //création des liens
13  echo"<h2> Mes liens préférés </h2>";
14  echo "<ul><li><a href=\" http://www.{$tab['php']}\" title=\"Le site php.net\">  PHP </a>
</li>";
15  echo "<li><a href=\" http://www.{$tab['mysql']}\" title=\"Le site mysql.com\">  MySQL </a>
</li>";
16  echo "<li><a href=\" http://www.{$tab['html']}\" title=\"Le site du W3C\">  HTML </a>
</li></ul>";
17  ?>
18  </body>
19 </html>
```

Création de tableaux avec `array()` ou `[]`

```
$tab=[v0,...,vN];
$tab=array(v0,...,vN);
```

Crée un tableau indicé à `N+1` éléments :

- Le 1er élément de valeur `v0` a pour indice 0.
- Accès au `i+1`-ième élément : `$tab[i]`.

```
$tab=["k1"=>v1,...,"kN"=>vN];
$tab=array("k1"=>v1,...,"kN"=>vN);
```

Crée un tableau associatif à `N` éléments :

- Chaque élément est une paire clé `"k"` - valeur `v`.
- Pas de notion d'ordre entre les éléments.
- Accès par clé aux éléments : `$tab["k"]`.

Tableaux multidimensionnels par tableaux de tableaux

```
$tab=[[...],...,[...]];
$tab=array(array(...),...,array(...));
$tab=["k1"=>[...],...,"kN"=>[...]];
$tab=array("k1"=>array(...),...,"kN"=>array(...));
```

Accès : `$tab[a][b]` où `a` et `b` sont des indices ou clés selon le cas.

exemple5-1.php

```php
$tab=[  ["A-0","A-1","A-2"],
        ["B-0","B-1","B-2"],
        ["C-0","C-1","C-2"],
        ["D-0","D-1", "D-2"]];
echo "<h3>Tableau multidimensionnel</h3><table> <tbody>";
for ($i=0;$i<count($tab);$i++) {
    echo "<tr>";
    for($j=0;$j<count($tab[$i]);$j++)
        echo "<td><h3> .. ",$tab[$i][$j]," .. </h3></td>";
    echo "</tr>";
}
echo " </tbody> </table> ";
print_r($tab);
```

Suites de nombres/lettres

`range(int m, int M):array`

- Crée un tableau indicé contenant les entiers de `m` à `M`.

`range(char c, char C):array`

- Crée un tableau indicé contenant les caractères de `c` à `C` selon le code ASCII.

exemple5-2.php

```php
// Suite de nombres de 1 à 10
$tabnombre= range(1,10);
print_r($tabnombre);
echo "<hr>";
// Suite de lettres de a à z avec une boucle
for($i=97;$i<=122;$i++) {
    $tabalpha[$i-96]=chr($i);
}
print_r($tabalpha);
echo "<hr>";
//Suite des caractères de Y à b avec range()
$tabalpha2 = range("Y","b");
print_r($tabalpha2);
```

Transformations de chaînes en tableaux

`explode(string sep, string $ch [, int N]):array`

- Décompose la chaîne `$ch` en tableau de mots (sous-chaînes) selon le séparateur `sep` fourni. `N` est le nombre maximum de mots recherchés.

`implode(string sep, array $tab):string`

- Fonction inverse.

Transformations de chaînes en tableaux

Exemple4-8.php

```php
//Passage chaîne -> tableau
$ch1="L'avenir est à PHP7 et MySQL";
$tab1=explode(" ",$ch1);
echo $ch1,"<br />";
print_r($tab1);
echo "<hr />";
$ch2="C:\wampserver\www\php7\chaines\string2.php";
$tab2=explode("\\",$ch2);
echo $ch2,"<br />";
print_r($tab2);
echo "<hr />";
//Passage tableau -> chaîne
$tab3[0]="Bonjour";
$tab3[1]="monsieur";
$tab3[2]="Rasmus";
$tab3[3]="Merci!";
$ch3=implode(" ",$tab3);
echo $ch3,"<br />";
```

Décompte des éléments

`count(array $tab)` ou `int sizeof(array $tab):int`

- Renvoie le nombre d'éléments.

Pour tableau "muti-dimensionnel"

```php
// Comptage du nombre d'éléments
$tab=array("Bonjour","Web",array("1-0","1-1","1-2"),
     1970,2013,array("3-0","3-1","3-2","3-3"));
echo "Le tableau \$tab a ",count($tab)," éléments <br />";
//ou encore: echo "Le tableau \$tab a ",sizeof($tab)," éléments <br />";
// Comptage du nombre de valeurs
$nb_val=0;
for ($i=0;$i<count($tab);$i++) {
 if(gettype($tab[$i])=="array") {
  $nb_val+=count($tab[$i]);
 } else {
  $nb_val++;
 }
 }
echo "Le tableau \$tab a ",$nb_val," valeurs <br />";
```

Décompte des valeurs

`array_count_values(array $tab):array`

- Renvoie un tableau associatif ayant pour clés les valeurs de `$tab` et pour valeur le nombre d'occurrences de chaque valeur dans `$tab`.
- Ne s'applique qu'aux tableaux de nombres et/ou chaînes.

exemple5-4.php

```
1 $tab= ["PHP","JavaScript","PHP","ASP","PHP","ASP"];
2 $result=array_count_values($tab);
3 echo "Le tableau \$tab contient ",count($tab)," éléments <br>";
4 echo "Le tableau \$tab contient ",count($result)," valeurs différentes
<br>";
5 print_r($result);
```

Lecture des éléments avec boucles for et while

Avec boucle for

```php
$montab=array("Paris","London","Brüssel");
for ($i=0;$i<count($montab);$i++) {
 echo "L'élément $i est {$montab[$i]}<br />";
}
```

Avec boucle while

```php
$montab=array("Paris","London","Brüssel");
$i=0;
while(isset($montab[$i])) {
 echo "L'élément $i est $montab[$i]<br />";
 $i++;
}
```

Lecture des éléments avec `foreach()`

`foreach($tab as $val){bloc}`

- Pour tableaux indicés.
- La variable `$val` contiendra successivement chacune des valeurs du tableau `$tab`.

`foreach($tab as $key=>$val){bloc}`

- Pour tableaux associatifs.
- La paire de variables (`$key`,`$val`) correspondra successivement à chaque élément (clé,valeur) du tableau `$tab`.

Exemple avec foreach

exemple5-12.php

```php
//*********************************************************
//Lecture de tableau indicé sans récupération des indices
//*********************************************************
$tab=array("Paris","London","Brüssel");
echo "<H3>Lecture des valeurs des éléments </H3>";
foreach($tab as $ville) {
 echo "<b>$ville</b> <br>";
}
echo"<hr>";
//*********************************************************
//Lecture de tableau indicé avec récupération des indices
//*********************************************************
echo "<h3>lecture des indices et des valeurs des éléments </h3>";
foreach($tab as $indice=>$ville) {
 echo "L'élément d'indice <b>$indice</b> a la valeur <b>$ville</b><br>";
}
echo"<hr>";
//*********************************************************
//Lecture de tableau associatif avec récupération des clés
//*********************************************************
$tab2=array("France"=>"Paris","Great Britain"=>"London","België"=>"Brüssel");
echo "<h3>lecture des clés et des valeurs des éléments</h3>";
foreach($tab2 as $cle=>$ville) {
 echo "L'élément de clé <b>$cle</b> a la valeur <b>$ville</b> <br>";
}
echo"<hr>";
```

Ajout et retrait d'éléments

`array_push($tab,v1,...,vN):int`

- Ajoute `v1,...,vN` en fin de `$tab` et en retourne la taille.
- Indexation de `v1,...,vN` à partir de `count($tab)`.

`array_pop($tab):mixed`

- Supprime et retourne le dernier élément de `$tab` s'il existe, `NULL` sinon.

`unset(mixed $tab [, mixed $...]):void`

Utilisé avec un élément de tableau comme argument :

- Supprime l'élément.
- Sans conséquence sur les indices/clés des éléments restants.

Ajout et retrait d'éléments

`array_unshift($tab,v1,...,vN):int`

- Ajoute `v1,...,vN` en début de `$tab` et en retourne la taille.
- Ré-indexation à partir de 0 avec décalage-droite pour éléments indicés, clés inchangées pour les autres.

`array_shift($tab):mixed`

- Supprime et retourne le premier élément de `$tab` s'il existe, `NULL` sinon.
- Ré-indexation à partir de 0 avec décalage-gauche pour éléments indicés, clés inchangées pour les autres.

`array_unique($tab):array`

Supprime les valeurs en doublons et ne conserve que la "dernière" occurrence.

- Sans conséquence sur les indices/clés des éléments restants.

Exemple

exemple5-15.php

```php
$tab= array(800,1492, 1515, 1789); print_r($tab);
echo "<hr />";
// Ajout au début du tableau
$poitiers=732;
$nb=array_unshift($tab,500,$poitiers);
echo "Le tableau \$tab a maintenant $nb éléments <br>"; print_r($tab);
echo "<hr />";
// Ajout à la fin du tableau
$armi=1918;
$newnb=array_push($tab,1870,1914,$armi);
echo "Le tableau \$tab a maintenant $newnb éléments <br>"; print_r($tab);
echo "<hr />";
// Suppression du dernier élément
$suppr= array_pop($tab);
echo "Le tableau \$tab a perdu l'élément $suppr <br>"; print_r($tab);
echo "<hr />";
// Suppression du premier élément
$suppr= array_shift($tab);
echo "Le tableau \$tab a perdu l'élément $suppr <br>"; print_r($tab);
echo "<hr />";
// Suppression de l'élément d'indice 4
unset($tab[4]);
echo "L'élément d'indice 4 a été supprimé <br>"; print_r($tab);
```

Extraction d'éléments avec `array_slice()`

`array_slice(array $tab, int i, int n):array`

Retourne un sous-tableau de `$tab` différent selon le signe et la valeur de `i` et `n` :

1. `array_slice($tab,2,3)` : les 3 premiers éléments à partir de celui d'indice 2.
2. `array_slice($tab,2,-3)` : tous les éléments à partir de celui d'indice 2 sauf les 3 derniers.
3. `array_slice($tab,-2,3)` : les 3 éléments précédant l'avant-dernier.
4. `array_slice($tab,-3,-2)` : les éléments d'indices virtuels négatifs entre -3 compris et -2 non compris.

exemple5-14.php

Tri d'éléments

Différentes méthodes

- Tri sur tableaux indicés ou sur tableaux associatifs.
- Tri sur les valeurs ou sur les clés.
- Critères de tri prédéfinis (ordre alphabétique, ordre ASCII . . .).
- Critères de tri personnalisés.
- Préservation ou non des indices/clés.

Tri sur tableaux indicés

Selon l'ordre ASCII

- `sort(array &$tab):bool` : tri des valeurs en ordre croissant. Perte des correspondances indices-valeurs.
- `rsort(array &$tab):bool` : tri des valeurs en ordre décroissant. Perte des correspondances indices-valeurs.
- `array_reverse(array $tab):array` : inverse l'ordre des valeurs. Perte des correspondances indices-valeurs.

exemple5-18.php

Tableau indicé d'origine
Array ([0] => Blanc2 [1] => Jaune [2] => rouge [3] => Vert [4] => Bleu [5] => Noir [20] => Blanc10)

Tri en ordre ASCII sans sauvegarde des indices
Array ([0] => Blanc10 [1] => Blanc2 [2] => Bleu [3] => Jaune [4] => Noir [5] => Vert [6] => rouge)

Tri en ordre ASCII inverse sans sauvegarde des indices
Array ([0] => rouge [1] => Vert [2] => Noir [3] => Jaune [4] => Bleu [5] => Blanc2 [6] => Blanc10)

Inversion de l'ordre des éléments
Array ([0] => Blanc10 [1] => Noir [2] => Bleu [3] => Vert [4] => rouge [5] => Jaune [6] => Blanc2)

Tri sur tableaux indicés ou associatifs

Selon l'ordre naturel

- `natsort(array &$tab):bool` : tri des valeurs selon l'ordre "naturel" (a2<a10). Préservation des correspondances indices/clés-valeurs.
- `natecasesort(array &$tab):bool` : comme `natsort` mais insensible à la case.

Boucle `foreach` indispensable pour itérer selon le nouvel ordre.

exemple5-19.php

Tableau indicé d'origine
Array ([0] => Blanc2 [1] => Jaune [2] => rouge [3] => Vert [4] => Bleu [5] => Blanc10 [6] => 1ZZ [7] => 2AA)

Tri en ordre naturel avec sauvegarde des indices
Array ([6] => 1ZZ [7] => 2AA [0] => Blanc2 [5] => Blanc10 [4] => Bleu [1] => Jaune [3] => Vert [2] => rouge)

Tri en ordre naturel insensible à la casse avec sauvegarde des indices
Array ([6] => 1ZZ [7] => 2AA [0] => Blanc2 [5] => Blanc10 [4] => Bleu [1] => Jaune [2] => rouge [3] => Vert)

Tri des valeurs sur tableaux associatifs

Avec préservation des correspondances clés-valeurs

- `asort(array &$tab):bool` : tri ASCII des valeurs en ordre croissant.
- `arsort(array &$tab):bool` : tri ASCII des valeurs en ordre décroissant.

exemple5-22.php

Tableau associatif d'origine
Array ([w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir [w10] => Blanc10)

Tri en ordre ASCII des valeurs avec sauvegarde des clés
Array ([w10] => Blanc10 [w2] => Blanc2 [blu] => Bleu [y] => Jaune [bla] => Noir [g] => Vert [r] => rouge)

Tri en ordre ASCII inverse avec sauvegarde des clés
Array ([r] => rouge [g] => Vert [bla] => Noir [y] => Jaune [blu] => Bleu [w2] => Blanc2 [w10] => Blanc10)

Tri en ordre naturel avec sauvegarde des clés
Array ([w2] => Blanc2 [w10] => Blanc10 [blu] => Bleu [y] => Jaune [bla] => Noir [g] => Vert [r] => rouge)

Tri en ordre naturel insensible à la casse avec sauvegarde des clés
Array ([w2] => Blanc2 [w10] => Blanc10 [blu] => Bleu [y] => Jaune [bla] => Noir [r] => rouge [g] => Vert)

Tri des clés sur tableaux associatifs

Avec préservation des correspondances clés-valeurs

- `ksort(array &$tab):bool` : tri ASCII des clés en ordre croissant.
- `krsort(array &$tab):bool` : tri ASCII des clés en ordre décroissant.

exemple5-23.php

Tableau associatif d'origine
Array ([w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir [w10] => Blanc10)

Tri en ordre ASCII des clés
Array ([bla] => Noir [blu] => Bleu [g] => Vert [r] => rouge [w10] => Blanc10 [w2] => Blanc2 [y] => Jaune)

Tri en ordre ASCII inverse des clés
Array ([y] => Jaune [w2] => Blanc2 [w10] => Blanc10 [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir)

Tri selon la longueur des clés
Array ([w10] => Blanc10 [blu] => Bleu [bla] => Noir [w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert)

CM PHP : Fonctions

Les fonctions natives de PHP

Vérifier la disponibilité de modules/fonctions (eg. chez votre hébergeur)

Liste des modules installés

- `get_loaded_extensions():array`

Liste des fonctions disponibles dans un module d'extension

- `get_extensions_funcs("nom_module"):array`

Tester l'existence d'une fonction

- `function_exists("nom_fonction"):bool`

exemple7-1.php

```php
//Tableau contenant le nom des extensions
$tabext = get_loaded_extensions(); natcasesort($tabext);
foreach($tabext as $cle=>$valeur) {
 echo "<h3>Extension  $valeur </h3> ";
 //Tableau contenant le nom des fonctions
 $fonct = get_extension_funcs($valeur); sort($fonct);
 for($i=0;$i<count($fonct);$i++) {
  echo $fonct[$i],"    \n";
  echo "<hr />";
 }
}
```

Définir ses fonctions

Différentes sortes de fonctions

- Avec ou sans paramètres.
- Avec paramètres par défaut ou non.
- ~~Avec un nombre fixe ou variable de paramètres~~.
- Avec ou sans valeur de retour.
- Avec passage par référence ou non.
- Avec retour par référence ou non.
- ~~Avec variables statiques ou non.~~
- ~~Avec itération sur valeurs de retour (générateur) ou non~~.
- Fonction de rappel (callback).
- Fonction nommée ou anonyme
- ~~Avec capture de l'environnement lexical (fermeture) ou non~~.
- Typée fortement ou faiblement.

Définition de fonction

Déclaration=Définition

- Pas de déclaration séparément de la définition.
- Peut être définie n'importe où dans un fichier.

En-tête

- Mot-clé `function`.
- Nommage : mêmes règles que pour les variables.
- Paramètres : liste de variables.

Appel

- Peut être appelée avant sa définition dans le script.
- Peut être appelée sans arguments même si elle en attend (avertissement émis) !

Fonctions sans valeur retour

exemple7-2.php

```
1  <!DOCTYPE html>
2  <html lang="fr">
3   <head>
4    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
5    <title>Fonctions ne retournant pas de valeur</title>
6   </head>
7   <body>
8  // Fonction sans argument
9  function ladate() {
10   echo "<table ><tr><td> ";
11   echo date("d/m/Y H:i:s");
12   echo "</td></tr></table><hr />";
13 }
14 // Fonction avec un argument
15 function ladate2($a) {
16   echo "<table><tr><td>";
17   echo "$a ",date("d/m/Y H:i:s");
18   echo "</td></tr></table><hr />";
19 }
20 // Appels des fonctions
21 echo ladate();
22 echo ladate2("Bonjour");
23 echo ladate2("Salut");
24 //echo ladate2();//Provoque une erreur fatale
25 </body>
26 </html>
```

Fonctions sans valeur retour

exemple7-3.php

```
<!DOCTYPE html>
<html lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
  <title>Fonction de lecture de tableaux</title>
 </head>
 <body>
// Définition de la fonction
function tabuni($tab,$bord,$lib1,$lib2)
{
 echo "<table border=\"$bord\" width=\"100%\"> <tbody><tr><th>$lib1</th> <th>$lib2
</th></ tr>";
 foreach($tab as $cle=>$valeur) {
  echo "<tr><td>$cle</td> <td>$valeur </td></tr>";
 }
 echo "</tbody> </table><br />";
}
// Définition des tableaux
$tab1 = array("France"=>"Paris","Allemagne"=>"Berlin","Espagne"=>"Madrid");
$tab2 = array("Poisson"=>"Requin","Cétacé"=>"Dauphin","Oiseau"=>"Aigle");
// Appels de la fonction
tabuni($tab1,1,"Pays","Capitale");
tabuni($tab2,6,"Genre","Espèce");
</body>
</html>
```

Typage des paramètres (PHP 7+)

Types valides pour déclarer paramètres et valeur de retour

- `int`, `float`, `string`, `bool`, `array`.
- `callable` : fonctions de rappel.
- Nom de classe ou interface.
  - `self` : Réfère à la classe définissant la méthode.

Valeur de retour inutile ou aucun paramètre en entrée : `void`.

Typage faible vs. typage fort (alias typage strict)

Choix du typage fort en incluant en début de fichier la directive `declare(strict_types=1);`

- Exception `TypeError` levée en cas d'erreur de type à l'appel de fonctions.

Transtypage automatique en cas de typage faible.

Typage faible et typage fort

function-typing-weak.php

```php
function sum(int $a, int $b) : int
  { return $a + $b; }
echo sum(1, 2);
// conversion automatique de float vers int
echo '<br/>'.sum(1.5, 2.5);
```

function-typing-strict.php

```php
declare(strict_types=1);
function sum(int $a, int $b) : int { return $a + $b; }
try {
    echo sum(1.5, 2.5);
} catch (TypeError $e) {
    echo 'Erreur : '.$e->getMessage();
}
```

Retourner plusieurs valeurs avec un tableau

exemple7-5.php

```
 1 <!DOCTYPE html>
 2 <html lang="fr">
 3  <head>
 4   <meta charset="UTF-8" />
 5   <title>Nombres complexes</title>
 6  </head>
 7  <body>
 8   <div>
 9 function modarg($reel,$imag) {
10     // $mod= hypot($reel,$imag);
11     // ou encore si vous n'avez pas la fonction hypot()du module standard
12     $mod =sqrt($reel*$reel + $imag*$imag);
13     $arg = atan2($imag,$reel);
14     return array("module"=>$mod,"argument"=>$arg);
15 }
16 // Appels de la fonction
17 $a= 5;
18 $b= 8;
19 $complex= modarg($a,$b);
20 echo "<b>Nombre complexe $a + $b i:</b><br /> module = ", $complex["module"] , "<br />argument
= ",$complex["argument"]," radians<br />";
21 </div>
22 </body>
23 </html>
```

Paramètres par défaut

Passage de paramètres par défaut

Les paramètres "les plus à droite" peuvent avoir des valeurs par défaut.

function-default.php

```php
function f($a, $b=2) { return $a*$b; }
echo f(10);
```

Portée des variables

Déterminée selon le contexte

Portée globale pour toute variable déclarée en dehors d'une fonction ou d'une classe (portée limitée au script et aux scripts inclus).

variable-scope-include.php

```php
echo $a;
```

variable-scope.php

```php
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
</head>
<body>
$a = 1;
include 'variable-scope-include.php';
<div></div>
echo $a;
</body>
</html>
```

Portée des variables

Au sein des fonctions

Portée locale pour toute variable définie dans une fonction ou une classe SAUF si redéclarée avec `global` ou utilisée avec `$GLOBALS[]`.

variable-scope-function.php

```php
$a = $b = $c = 1;
function test() {
    global $b;
    echo $b;
    echo $GLOBALS['c'];
    echo $a; // undefined variable $a
    $x = 2;
    echo $x;
}
test();
```

Paasage par référence

Référence en PHP = alias

- Semblable à la notion de lien sur fichier en LINUX.

Trois usages

- Affectation par référence : opérateur =&
- Passage par référence : `function f(&$a)`
- Retour par référence : `function &f(); $a= &f();`

Affectation par référence - Destruction de référence

reference1.php

```php
// affectation par référence
$b = 10;
$a =& $b;
echo '$a='.$a; //10
$a += 1;
echo '\n$b='.$b; //11
```

reference2.php

```php
// destruction de référence
$b = "b";
$c = "c";
$a =& $b;
echo '\n$a='.$a; // b
$a =& $c;
echo '\n$a='.$a; // c
echo '\n$b='.$b; // b
unset($a);
echo '\n$a='.$a; // Undefined
echo '\n$c='.$c; // c
```

Fonctions : passage et retour par référence

reference3.php

```php
// passage par référence
function foo(&$var) { $var++; }
function bar($var) { $var++; }
$a=0; foo($a);
echo $a; // 1
bar($a);
echo $a; // 1
```

reference4.php

```php
// retour par référence
function &collector() {
   static $collection = array();
   print_r($collection);
   return $collection;
}
// !! préfixer l'appel avec &
$collect = &collector(); //Array()
$collect[] = 'foo';
collector();//Array([0]=>foo)
```

Fonctions de rappel

Fonction de rappel (alias callback)

- Fonction nommée ou anonyme, méthode d'objet ou méthode statique de classe.
- Type `callable`.
- Peut être passée comme argument à d'autres fonctions (eg. fonctions de tri sur tableaux) :
  - Par la chaîne la dénommant.
  - Par la variable objet la représentant si elle est anonyme : on parle alors de fermeture (alias closure).

Fonctions anonymes (alias fermetures)

function-anonymous.php

```php
$tab=array(1,2,3);
// fonction anonyme
$tab1=array_map(function($n) { return $n*$n; },$tab);
print_r($tab1);
```

function-anonymous2.php

```php
$tab=array(1,2,3);
// fermeture (objet Closure)
$f = function($n) {return $n*$n;};
echo $f(10);
$tab1=array_map($f,$tab);
print_r($tab1);
```

Application : tri personnalisé sur tableaux indicés

Sur la base d'une fonction binaire numérique à définir (le critère de tri)

`moncritere(mixed $x, mixed $y):int` doit renvoyer :

- -1 (ou nombre négatif) si `$x` est "inférieur" à `$y`.
- 0 si `$x` est "égal/équivalent" à `$y`.
- 1 (ou nombre positif) si `$x` est "supérieur" à `$y`.

Fonction utilisée comme argument de `usort(array &$tab, callable $f):bool`

```
usort($tab,"moncritere");
```

- Trie `$tab` selon la fonction `moncritere()`.
- Perte des correspondances indices-valeurs.

Exemple

exemple5-20.php

```php
// Définition de la fonction de tri
function long($mot1,$mot2) {
if(strlen($mot1)==strlen($mot2))
    return 0;
elseif(strlen($mot1)>strlen($mot2))
    return -1;
else
    return 1;
}
// Fonction de tri équivalente
function long1($mot1,$mot2) { return strlen($mot2)<=>strlen($mot1); }
// Tableau à trier
$tab=["Blanc","Jaune","rouge","Vert","Orange","Noir","Emeraude"];
// Utilisation de la fonction de tri
echo "Tableau initial<br />";
print_r($tab);
usort($tab,"long");
echo "<br />Tableau trié selon la longueur décroissante des mots<br />";
print_r($tab);
```

Tableau initial
Array ([0] => Blanc [1] => Jaune [2] => rouge [3] => Vert [4] => Orange [5] => Noir [6] => Emeraude)
Tableau trié selon la longueur décroissante des mots
Array ([0] => Emeraude [1] => Orange [2] => Blanc [3] => Jaune [4] => rouge [5] => Vert [6] => Noir)

Tri des valeurs sur tableaux associatifs

Avec préservation des correspondances clés-valeurs

- `uasort(array $tab, callable $f):bool` : tri personnalisé des valeurs avec `$f`.

exemple5-22.php

Tableau associatif d'origine
Array ([w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir [w10] => Blanc10)

Tri en ordre ASCII des valeurs avec sauvegarde des clés
Array ([w10] => Blanc10 [w2] => Blanc2 [blu] => Bleu [y] => Jaune [bla] => Noir [g] => Vert [r] => rouge)

Tri en ordre ASCII inverse avec sauvegarde des clés
Array ([r] => rouge [g] => Vert [bla] => Noir [y] => Jaune [blu] => Bleu [w2] => Blanc2 [w10] => Blanc10)

Tri en ordre naturel avec sauvegarde des clés
Array ([w2] => Blanc2 [w10] => Blanc10 [blu] => Bleu [y] => Jaune [bla] => Noir [g] => Vert [r] => rouge)

Tri en ordre naturel insensible à la casse avec sauvegarde des clés
Array ([w2] => Blanc2 [w10] => Blanc10 [blu] => Bleu [y] => Jaune [bla] => Noir [r] => rouge [g] => Vert)

Tri des clés sur tableaux associatifs

Avec préservation des correspondances clés-valeurs

- `uksort(array $tab, callable $f):bool` : tri personnalisé des clés avec `$f`.

exemple5-23.php

Tableau associatif d'origine
Array ([w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir [w10] => Blanc10)

Tri en ordre ASCII des clés
Array ([bla] => Noir [blu] => Bleu [g] => Vert [r] => rouge [w10] => Blanc10 [w2] => Blanc2 [y] => Jaune)

Tri en ordre ASCII inverse des clés
Array ([y] => Jaune [w2] => Blanc2 [w10] => Blanc10 [r] => rouge [g] => Vert [blu] => Bleu [bla] => Noir)

Tri selon la longueur des clés
Array ([w10] => Blanc10 [blu] => Bleu [bla] => Noir [w2] => Blanc2 [y] => Jaune [r] => rouge [g] => Vert)

Sélection d'éléments

`array_filter($tab, callable $f):array`

Retourne le tableau d'éléments de `$tab` acceptés par la *call-back* `$f`

- `f(mixed $var):bool` est appelée avec chaque élément `$var` de `$tab` et doit retourner `TRUE` ssi elle accepte l'élément.
- Préserve les correspondances clés-valeurs pour les éléments retenus.

exemple5-24.php

```php
1 //Définition du tableau
2 $villes=array("Paimpol","Angers","Pau","Nantes","Lille");
3 //Fonction de sélection
4 function init($ville) {
5  if($ville[0]=="P" || $ville[0]=="p") return $ville;
6 }
7 //Utilisation de array_filter()
8 $select=array_filter($villes,"init"); print_r($select);
```

Transformation de tableaux

`array_walk(array &$tab, callable $f [, mixed $userdata]):bool`

Applique la callback `$f` à chaque élément de `$tab`.

- `f(mixed $v, mixed $c):bool` prend comme arguments la valeur et la clé de chaque élément. Accepte `$userdata` comme 3ème argument si défini.
- Ne peut modifier les valeurs de `$tab` qu'à la condition de spécifier `$v` comme référence :

  `f(mixed &$v, mixed $c):bool`

exemple5-25.php

Accès d'une fermeture à l'environnement lexical

Accès par valeur ou par référence à variable non locale

- Avec syntaxes `use($x)` et `use(&$x)`.

function-closure.php

```php
$diviseur = 10;
$modulo = function ($x) use ($diviseur) {
    return $x % $diviseur;
};
$autremodulo = function ($x) use (&$diviseur) {
    return $x % $diviseur;
};
echo $modulo(12); // 2
echo $autremodulo(12); // 2
$diviseur = 12;
echo $modulo(12); // 2
echo $autremodulo(12); // 0
$tab=array(1,2,3);
$x = 5;
// x + 2n
$tab1=array_map(function($n) use($x) { return $x + 2*$n; },$tab);
print_r($tab1);
```

CM PHP : Formulaires

Rappel sur les formulaires HTML5

Les formulaires

- Base de l'interactivité des sites Web.
- Echanges avec serveur par le protocole HTTP en utilisant méthode `GET` ou, de préférence, `POST`.
  - Envoi de données.
  - Déclenchement d'une requête dans une BDD.
  - Création de page dynamique en réponse.

Composants de formulaires

- Saisie de texte et mots de passe.
- Choix par boutons radio, cases à cocher, listes de sélection.
- Envoi de fichiers ou données cachées.

Création de formulaire

Attributs de l'élément `<form>`

`action="f.php"`

- Désigne le fichier de traitement des données saisies.
- Chemin relatif ou URL.
- Si le fichier `f.php` génère lui-même le formulaire, on peut utiliser le script `action="<?= $_SERVER["PHP_SELF"] ?>"` qui générera le code HTML `action="f.php"`.
  - `<?= $x ?>` est une alternative plus concise à `<?php echo $x; ?>`
- Possibilité d'utiliser d'autres protocoles (`mailto:a@b.c`).

Création de formulaire

Attributs de l'élément `<form>`

`method="get"` ou `method="post"`

- Méthode HTTP d'envoi des données.
- HTTP GET par défaut.
  - Données en clair dans l'URI : le *query-string*
    - Ajouté à l'URL avec ?
    - Affectations de champs séparées par `&`
    - Caractères accentués, symboles de ponctuation . . . encodés en hexadécimal.
    - Exemple : `http://www.abc.fr/f.php?x=bleu&y=12`
  - Limitées à ~2000 caractères.
- HTTP POST recommandée.
  - Encapsule les données dans le corps de la requête.
  - Données de tout type.

Création de formulaire

Attributs de l'élément `<form>`

`name="monformulaire"`

- Utile essentiellement pour JavaScript.

`enctype` définit le type d'encodage

- `"multipart/form-data"` pour le transfert de fichiers.
- `"application/x-www-form-urlencoded"` par défaut.

```
<form method="post" action="f.php"
enctype="application/x-www-form-urlencoded">
...
</form>
```

L'élément <input/>

L'attribut type permet de distinguer différents types de champs

- Texte en clair et mot de passe.
- Email et numéro de téléphone.
- Nombre et date.
- Choix unique et choix multiple.
- Soumission et réinitialisation.
- Fichier et données cachées.

L'attribut name est obligatoire

- Identifie chaque champ côté serveur.
- Ne pas confondre avec l'attribut id utilisé pour CSS et manipulation JS via le DOM.

Champ texte mono-ligne en clair

Attributs de l'élément `<input type="text"/>`

`size="nombre"`

- Largeur de la zone affichée en #caractères.

`maxlength="nombre"`

- Nombre maximum de caractères à saisir.

`value="texte"`

- Texte affiché et transmis par défaut.

```
<input type="text" name="ville" size="30" maxlength="40"
value="Angers"/>
```

Autres champs texte

Adresse e-mail : `<input type="email"/>`
N° de téléphone : `<input type="tel">`

Validation par regex avec l'attribut `pattern`.

Mots de passe : `<input type="password"/>`

Caractères saisis affichés par des *

Nombres : `<input type="number"/>`

Encadrement forcé avec attributs `min` et `max`.
Obligation de saisie avec `required`.

Dates : `<input type="date"/>`

Champ texte au format `AAA-MM-JJ` avec interface selon navigateur.
Encadrement avec attributs `min` et `max`.
Variantes : `time` (heure), `month`, `week`, `datetime-local` (date et heure), `datetime` (date, heure et fuseau horaire).

Boutons radio

Attributs de l'élément `<input type="radio"/>`

`checked="checked"`

- Bouton coché par défaut.

`value`

- Indispensable comme pour champs texte.

Utiliser la même valeur d'attribut `name` entre boutons pour imposer un choix exclusif

```
<label>Homme</label><input type="radio" name="genre"
value="h"/> <label>Femme</label><input type="radio"
name="genre" value="f"/>
```

Cases à cocher

Elément `<input type="checkbox"/>`

Utiliser la même valeur d'attribut `name` concaténée avec `[]` pour permettre un choix multiple :

```
<input type="checkbox" name="lang[]" value="php"/>
<input type="checkbox" name="lang[]"
value="javascript"/>
```

Boutons de soumission et RAZ

Elément `<input type="submit"/>`

Indispensable d'avoir au moins un bouton de soumission.
Attribut `value`

- Le texte affiché du bouton.
- Permet d'effectuer différentes tâches de traitement selon la valeur du bouton pressé.

Elément `<input type="reset"/>`

Réinitialise les champs à leurs valeurs par défaut ($\neq$ effacement).

Fichiers et données masquées

Elément `<input type="file"/>`

Même apparence qu'un champ texte et incorpore un sélecteur de fichier sur le poste client.

Attribut `accept`

- Définit le ou les types de fichiers acceptés.

Attribut `multiple`

- Possibilité de sélectionner et téléverser plusieurs fichiers.

```
<input type="file" name="image" accept=".jpeg, .jpg, .png"
multiple>
```

Elément `<input type="hidden"/>`

Permet d'envoyer des données invisibles pour l'utilisateur (e.g. jeton de sécurité, identifiant de transaction, taille maximum de fichier téléversable).

Zones de texte multilignes

Attributs de l'élément `<textarea>`

`cols`

- Nombre de colonnes de la zone de texte.

`rows`

- Nombre de lignes de la zone de texte.

Liste déroulante

Elément `<select>`

Sélection simple ou multiple d'options définies chacune par un sous-élément `<option>`.

Attribut `size`

- Nombre d'options visibles simultanément.

Attribut `multiple="multiple"`

- Autorise la sélection multiple (avec touche Ctrl).

Attributs de l'élément `<option>`

`value`

- La valeur transmise si l'option est sélectionnée.

`selected="selected"`

- Option présélectionnée par défaut.

exemple6-1.html

Récupération des données de formulaire

Différents cas de figure :

- Les valeurs uniques.
- Les valeurs multiples.
- Les fichiers
- Les boutons d'envoi multiples.

Récupération de valeurs uniques

Contenues dans le tableau superglobal `$_POST` (ou `$_GET`)

exemple6-2.php

Vérification avec `isset()` sur champ texte et boutons radio.
Tous deux `NULL` initialement au chargement de la page.
Si champ non saisi et boutons non cochés à la soumission :

- La valeur du champ est la chaîne vide : vérification plus fine avec `empty()`.
- La valeur des boutons radio reste `NULL`.

Récupération de valeurs multiples

Exemples

La valeur de l'attribut `name` doit se terminer par `[]` :

- Cases à cocher de même attribut `name`.
- Liste de sélection multiple avec attribut `multiple`.

Récupération des valeurs via `$_POST` qui est alors un tableau bi-dimensionel.

exemple6-5.html, exemple6-5.php

Récupération de fichiers

```
<input type="file" name="f".../>
```

Le téléversement renomme le fichier et le place dans un répertoire tampon sur le serveur.
Récupération via tableau associatif `$_FILES["f"]`.
Eléments de `$_FILES["f"]` :

- `"name"` : le nom d'origine.
- `"type"` : le type du fichier.
- `"size"` : la taille du fichier.
- `"tmp_name"` : le nom du fichier sur le serveur.
- `"error"` : code d'erreur en cas d'échec du téléchargement.

`move_uploaded_file(string src, string dest)` permet de renommer et déplacer le fichier.

exemple6-6.php

CM PHP : Fichiers

Notion de flux

Flux (stream)

- Une ressource (type `resource`) accessible linéairement en lecture ou en écriture : fichier, ressource Web, données de BDD, ...
- Identifié avec la syntaxe `scheme://target`
  - `file://*` : accès aux fichiers locaux
  - `http://*` : accès HTTP aux URL
  - `ftp://*` : accès FTP aux URL
  - `zlib://*` : compression de données
  - `phar://*` : archive PHP
  - `ssh2://*` : SSH 2
  - ...

Gestionnaire de protocole (wrapper)

Code PHP pour gérer certains protocoles/encodages liés à un flux.

- eg. le wrapper HTTP convertit une URL en requête HTTP.

Contexte de flux

Contexte de flux

- Un ensemble de paramètres et d'options spécifiques à un wrapper pour contrôler le comportement d'un flux.
- Se crée avec la fonction `resource stream_context_create([array $options [, $params]]`.
- Se passe ensuite en argument aux fonctions de création et d'accès aux flux :
  - `fopen`
  - `file_get_contents`
  - `file_put_contents` ...

Exemple : accès HTTP

Récupération d'une ressource Web par son URL (file_get_contents.php)

```php
// création des options du contexte de flux
$opts = array(
  'http'=>array(
    'method'=>"GET",
    'header'=>"Accept-language: en\r\n" .
              "Cookie: foo=bar\r\n"
  )
);
// création du contexte de flux
$context = stream_context_create($opts);
// accès au flux "http://www.univ-angers.fr"
// avec envoi par le wrapper PHP pour HTTP
// d'une requête HTTP/1.1
// contenant les entêtes HTTP ci-dessus
//$file = file_get_contents('http://www.univ-angers.fr/', false, $context);
$file = file_get_contents('https://leria.univ-angers.fr/', false, $context);
echo $file;
```

Gestion des fichiers

Lecture et écriture de fichiers à l'aide de

- Fonctions : `fopen`, `fread`, `fwrite`, `fclose`, `file_get_contents`, `file_put_contents` ...
- ~~Deux classes SPL : `SPLFileInfo` et `SPLFileObject`~~.

Différentes modalités d'accès

- Lecture seule, écriture seule, ou les deux.
- Ecrasement de fichier ou ajout de données par écriture.
- Lecture intégrale, ligne par ligne, caractère par caractère.
- Lecture automatique de données formattées.
- Positionnement de pointeur de lecture/écriture.
- Verrouillage d'accès.
- Copie, renommage, effacement, méta-données.

Lecture de fichier avec `file_get_contents`

`string file_get_contents(string $f [,...])`

Lit le contenu d'un fichier en totalité et le retourne dans une chaîne.

- `$f` : nom du fichier.
- Drapeaux :
  - recherche ou non du fichier dans le `$PATH`.
  - contexte de flux valide (`NULL` pour les fichiers).
  - positionnement du pointeur de lecture.
  - nombre d'octets à lire ou bien lecture intégrale.

Retourne `FALSE` en cas d'erreur.
Emet un `E_WARNING` si fichier introuvable.

Lecture de fichier avec file_get_contents

Lecture d'un fichier local (file_get_contents1.php)

```php
$filename=__DIR__.'/html.txt';
$str=file_get_contents($filename);
if ($str===false) {
   throw Exception('could not read');
} else {
   echo $str;
}
```

Lecture d'une page HTML (file_get_contents2.php)

```php
$file=file_get_contents("http://www.univ-angers.fr");
echo $file;
```

Ecriture de fichier avec `file_put_contents`

`file_put_contents(string $f, mixed $d [,...])`

Ecrit dans le fichier spécifié par `$f` la donnée `d` :

- `$d` : chaîne, tableau ou ressource de flux.
- Drapeaux :
  - ou-logique entre constantes : `FILE_USE_INCLUDE_PATH` (recherche dans le `$PATH`), `FILE_APPEND` (concaténation vs. écrasement), `LOCK_EX` (verrou exclusif).
  - contexte de flux (`NULL` pour les fichiers)

Crée le fichier s'il n'existe pas, sinon l'écrase ou y ajoute les données selon l'option `FILE_APPEND`.
Retourne le nombre d'octets écrits ou `FALSE` en cas d'erreur.

Ecriture de fichier avec `file_put_contents`

Ecriture dans un fichier local (file_put_contents.php)

```php
$fname=__DIR__.'/file_put_contents1.txt';
$tab=array('odyssée', 'de', 'l\'espace', 2001);
file_put_contents($fname,implode(' ',$tab)."\n");
```

Utilisation de fopen, fwrite, fclose

`resource fopen(string $f, string $mode [,...])`

- Ouvre un fichier ou une URL spécifiée par `$f`.
- Drapeaux :

- `$mode` : lecture seule `"r"`, lecture et écriture `"r+"`, écriture seule avec écrasement `"w"`, lecture et écriture avec écrasement `"w+"`, écriture seule par ajout `"a"`, lecture et écriture par ajout `"a+"`.
- recherche dans le `$PATH`.
- contexte de flux valide (`NULL` pour les fichiers).

`int fwrite(resource $h, string $s [,int $n])`

- Ecrit le contenu de `$s` (maximum `$n` octets) dans le fichier identifié par `$h`.

`bool fclose(resource $h)`

- Ferme le fichier identifié par `$h`.

Utilisation de `fopen`, `fwrite`, `fclose`

Exemple de compression de fichier (fopen.php)

```php
/* création d'un fichier compressé (binaire)
 * Le fichier peut être décompressé
 * et lu avec le wrapper compress.zlib stream
 * ou en ligne de commande 'gzip -d foo-bar.txt.gz'
 */
$fp = fopen("compress.zlib://foo-bar.txt.gz", "wb");
if (!$fp) die("Unable to create file.");
fwrite($fp, "This is a test.\n");
fclose($fp);
```

Autres fonctions

Nom	Description
`flock`	dé-/verrouillage d'accès.
`fgetc`	lecture d'un caractère à la fois.
`fgets`	lecture d'une ligne à la fois (délimiteur `\n`).
`fread`	lecture de blocs de taille prédéfinie.
`fseek`	positionnement du pointeur.
`rewind`	positionner le pointeur en début de fichier.
`ftell`	position du pointeur.
`fgetcsv`	lecture d'une ligne à la fois et conversion sous forme de tableau de mots basée sur séparateur prédéfini.
`readfile`	lecture intégrale et affichage sur la sortie standard.

Autres fonctions

Nom	Description	
`copy`	copie.	
`rename`	renommage.	
`unlink`	suppression.	
`file_exists`	existence.	
`filesize`	taille.	
`filetype`	de type fichier ou répertoire.	
`is_file`	vérification.	
`is_readable`	accessibilité en lecture.	
`is_writable`	accessibilité en écriture.	
`is_uploaded_file`	test de téléversement.	
`fileatime`	dernier accès.	
`filemtime`	dernière modification.	
`filectime`	dernière modification des permissions.	
`realpath`	chemin d'accès.	
`basename`	extraction du nom à partir d'un chemin.	