PHP Introduction

Définition

PHP est un langage de programmation créé par Rasmus Lerdof en 1995. Il a ensuite été repris par Andi Gutmans et Zeev Suraski qui ont fondé la société Zend. PHP est un acronyme qui a d'abord signifié "Personal HomePage" puis "Hypertext PreProcessor" (acronyme récursif). Ces deux noms indiquent bien que PHP a été conçu à l'origine pour le web.

La dernière version de PHP est la version 5.6, la version 6 a été abandonnée et la version 7 devrait être disponible début 2016.

A quoi sert PHP

PHP est principalement utilisé pour réaliser des sites web. Il peut également servir à programmer des applications console. En revanche, son architecture et son contexte d'exécution ne le destine pas à la création de jeux ou d'application multimédia.

PHP est très utilisé sur le web et permet donc de coder des blogs, des sites marchands, des gestionnaires de contenu…

De nombreuses applications célèbres telles que WordPress, Drupal, Prestashop, Magento ou Facebook sont développées avec PHP.

Contexte d'exécution

PHP est un langage interprété ce qui signifie que le code PHP est contenu dans un fichier texte et qu'il faut passer par un logiciel d'interprétation pour exécuter ce code. Dans un contexte classique, ce logiciel d'interprétation est installé sur le serveur, il est lié à un logiciel serveur web (http) qui déclenche l'interprétation des pages possédant une extension ".php".

media/600px-Php_arch_shema.png)

Les outils

Pour tester nos applications web écrite en PHP, il faudra donc installer les composants suivants.

- un serveur web (Apache, Nginx, IIS)
- un interpréteur PHP
- éventuellement un SGBD

Il existe des solutions intégrées qui permettent d'installer l'ensemble de ces composants en une seule opération.

- XAMPP (multi-plateforme)
- WAMP (Windows)
- MAMP (Mac OS)

> Attention : Ces outils sont certes bien pratiques, mais il faut les réserver aux tests de nos applications soit en local sur notre ordinateur de travail soit en réseau sur un serveur de développement (éventuellement partagé avec nos collègues). Pour la mise en place d'un serveur d'exploitation, il faudra soit installer et configurer chacun des composants séparément soit recourir à un hébergeur qui fournira une configuration standard.

Le serveur web interne de PHP

Depuis la version 5.5, PHP dispose d'un serveur web interne (pour le test uniquement). Pour lancer ce serveur, il faut ouvrir une fenêtre de terminal (cmd.exe ou powerShell sous Windows) et saisir la commande suivante :

```
php -S adresse-ip-du-serveur:port -t chemin-du-dossier-racine
```

exemple :

```
php -S localhost:8000 -t /test/www
```

ou sous Windows

```
console
php -S localhost:8000 -t c:/test/www
```

Ensuite pour tester un script `page.php` dans le dossier `test/www`, il faut saisir l'adresse suivante dans le navigateur :
`http://localhost:8000/page.php`

Les outils pour coder en PHP

Les script PHP sont de simples fichiers texte, interprétés au niveau du serveur. N'importe quel éditeur de texte peut donc suffire pour coder en PHP. Cependant, l'utilisation d'un logiciel un peu évolué, gérant la coloration syntaxique et la présentation du code est fortement recommandée. Pour aller plus loin dans l'automatisation, il existe des environnement de développement intégrés (IDE) qui peuvent considérablement augmenter notre productivité.

Éditeurs de texte

- Notepad++ (Windows - gratuit)
- Bracket (Multi-plateforme - gratuit)
- TextMate (Mac OS - gratuit)

IDE

- NetBeans (Multi-plateforme - gratuit)
- Aptana Studio (Multi-plateforme - gratuit)
- PHPStorm (Multi-plateforme - payant)

Premier code

Intégration du code PHP dans HTML

PHP a été conçu pour la réalisation de site web et il s'intègre parfaitement dans nos fichiers html existants. Pour insérer du contenu dynamique PHP, il suffit d'intégrer des balises particulières dans notre code html. Ces balises sont

les suivantes : `<?php` et `?>`.

Tout code encadré par ces deux balises sera donc traité par l'interpréteur PHP, le résultat de cette interprétation sera ensuite combiné au contenu HTML avant d'être renvoyé au serveur web qui se chargera de transmettre la réponse au client web (le navigateur).

PHP détermine la fin d'une instruction par la présence du caractère ";". **Il faut donc prendre garde à terminer chaque ligne d'instruction par un point-virgule.**

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Première page PHP</title>
</head>
<body>
    <h1>
        Bonjour nous sommes le
        <?php
            echo date('d/m/Y');
        ?>
    </h1>
</body>
</html>
```

Commenter le code PHP

type de commentaire	syntaxe
commentaire mono ligne	//
commentaire multi ligne	/* suivi de */

```
//un commentaire mono ligne
$var = 5; //commentaire à la fin d'un instruction

/* commentaire multi ligne
*  les * à l'intérieur du commentaire
*  sont facultative
*  mais améliorent la lisibilité
```

```
*/
```

Tester le code PHP

Pour tester notre code PHP, il nous faut un serveur web, soit le serveur intégré à PHP, soit un serveur que nous aurons nous même installé. Il convient ensuite d'enregistrer nos fichiers php dans un dossier accessible à ce serveur. En effet, un serveur web ne rend pas disponible pour les internautes l'intégralité des dossiers de son serveur physique. Pour des raisons évidentes de sécurité, il ne travaillera qu'avec un dossier spécifique nommé "Document Root" ou dossier racine. Selon les configurations et les systèmes d'exploitations, ce dossier pourra porter des noms divers, mais les trois plus courants sont les suivants :

- htdocs
- wwww
- public_html

Dans l'exemple ci-dessous, nous testons sur l'hôte local (localhost ou 127.0.0.1). Le serveur utilisé est le serveur interne de PHP d'où l'ajout du port (:8000).

> **A vous :** Réalisez votre première page PHP est testez la dans un navigateur

Les variables et les expressions

Les variables

Définition

Une variable est une donnée stockée temporairement dans la mémoire vive de l'ordinateur. Cette variable possède un nom qui nous permet d'y faire référence et une valeur qui constitue la donnée proprement dite. Il s'agit donc d'une boîte dans laquelle nous mettrons des données que nous pourrons par la suite manipuler avec les instructions du langage.

Types de variables

Dans certains langages tels que Java, nous ne pouvons pas mettre n'importe quoi dans une variable. Il nous faut déclarer le type de données attendue (du texte, des nombres, des dates). Ce n'est pas le cas avec PHP qui est très permissif. Le type de la variable découle de son contenu et change avec celui-ci.

Nomenclature

Un nom de variable PHP commence toujours par le caractère "$" suivi d'une lettre ou d'un caractère "_". Le nom peut ensuite comporter une suite de caractères alphanumériques (des lettres et de chiffres) mais aucun espaces, accents ou caractères spéciaux.

Pour séparer les mots dans un nom de variable et améliorer la lisibilité, il est d'usage d'utiliser le caractère "_" ou de débuter chaque nouveau mot par une majuscule (sauf le premier qui commencera toujours par une minuscule).

```
$nomDeMaVariable;
$nom_de_ma_variable;
```

> **Attention :** En PHP les noms de variables sont sensibles à la casse, ce qui veut dire que `$mavariable` et `$maVariable` font référence à deux variables différentes.

Les noms de variables doivent être les plus explicites possibles, il ne faut pas craindre de déclarer des noms trop long (dans la limite du raisonnable).

Déclaration et affectation

Pour déclarer une variable et lui affecter une valeur nous utilisons la syntaxe suivante :

```php
$nom = 'Sebastien';
$age = 5;
$prixUnitaire = 12.85;
$estMajeur = false;
```

> **Note :** En PHP, le texte est encadré par des apostrophes ou guillemets simples (simple quotes). Nous verront qu'il existe des notations alternatives.

Afficher le contenu d'une variable

Pour afficher le contenu d'une variable nous disposons des instructions `echo` et `print`. Contrairement à `print`, l'instruction `echo` permet d'afficher plusieurs variables en les séparant par des virgules.

```php
echo $nom;
print $nom;

echo 'bonjour ', $nom, ' vous avez ', $age, ' ans';
```

Concaténation

La concaténation consiste à mettre bout à bout des variables et d'éventuelles valeurs littérales (les chaînes de caractères fixes ‘bonjour ’, ‘ vous avez ’ et ‘ ans’ de notre précédent exemple). Pour réaliser une concaténation, nous avons vu qu'il est possible d'utiliser l'instruction `echo` en séparant les éléments par

une virgule. Il est également possible d'utiliser le caractère "." et ce également avec l'instruction `print`.

```
echo 'bonjour '.$nom.' vous avez '.$age.' ans';
print 'bonjour '.$nom.' vous avez '.$age.' ans';
```

Les guillemets double

PHP nous permet d'encadrer les chaînes de caractères avec des guillemets doubles. Cette notation est intéressante, car elle permet d'intégrer les variables dans la chaîne de caractère sans recourir à un opérateur de concaténation. L'interpréteur réalise une interpolation et remplace les variables par leur valeurs. Avec les précédentes versions de PHP, cette facilité se payait par une moindre performance ce n'est plus le cas depuis la version 5.3.

```
echo "bonjour $nom vous avez $age ans";
```

Échappement des caractères

Si les caractères guillemets simples ou doubles délimitent les chaîne de caractère, il nous faut un moyen de coder ces caractères pour les intégrer dans un texte. Pour cela et pour coder d'autres type de caractères, nous utiliserons l'anti-slash ().

```
echo "Il m'a dit : \"PHP c'est mortel\"";
echo 'Il m\'a dit : "PHP c\'est mortel"';
```

Quelques caractères d'échappement utiles

Caractère	Description
\t	tabulation horizontale
\n	saut de ligne
\r	retour à la ligne
\	un caractère \

Obtenir des informations sur les variables

PHP nous propose un outils très pratique pour debugger les variables, la fonction `var_dump` qui affiche le contenu et le type de données.

```
var_dump($nom, $age, $prixUntaire, $estMajeur);
```

```
//Donne le résultat suivant
string 'Sebastien' (length=9)

int 5

float 12.85

boolean false
```

Transtyper une variable

Le type d'une variable PHP découle de son contenu, cependant, il est des cas ou nous déirerons convertir ce contenu vers un autre type. En informatique on parle alors de transtypage. Pour ce faire PHP propose des opérateurs qui se placent avant la variable.

opérateur	opération
(int)	conversion en entier
(double)	conversion en réel
(string)	conversion en chaîne de caractères
(boolean)	conversion en booléen
(array)	conversion en tableau
(object)	conversion en objet

```
$chaine = '5 pommes et 6 poires';

var_dump($chaine);
var_dump((int)$chaine);
```

```
var_dump((double)$chaine);
var_dump((boolean)$chaine);
```

Il est également possible d'utiliser la fonction `settype` pour obtenir le même effet. Dans ce cas, l'appel de la fonction modifie le type de la variable d'origine.

```
settype($chaine,'int');
var_dump($chaine);
```

Tester la valeur d'une variable

Avant de réaliser un traitement, il est prudent de valider les paramètres sur lesquels nous travaillons afin d'éviter de générer des erreurs. Ceci est d'autant plus important si ces paramètres proviennent de l'extérieur (saisie de l'utilisateur, lecture d'un fichier…).
Pour cela nous disposons d'une série de fonctions.

fonction	Description
isset($var)	vrai si $var existe et que sa valeur n'est pas null
empty($var)	vrai si la valeur de $var est NULL ou “”
is_bool($var)	vrai si $var est de type booléen (boolean)
is_int($var)	vrai si $var est de type entier (integer)
is_string($var)	vrai si $var est de type chaîne de caractères (string)
is_float($var)	vrai si $var est de type nombre décimal (float)
is_numeric($var)	vrai si la valeur de $var est numérique
is_null	vrai si la valeur de $var est null
is_scalar	vrai si le type de $var est scalaire (string, boolean, integer, float)
gettype($var)	retourne le type de $var

Les expressions

Une expression est une formule qui une fois évaluée retourne un résultat. Par exemple, `2+3` est une expression qui retourne 5. Une expression est composée de deux élements :

- un ou plusieurs opérateurs : les opérateurs indiquent les opérations qui seront effectuées sur les opérandes
- des opérandes : les opérandes sont les valeurs sur lesquels s'effectue les opérations, elles peuvent constituées de valeurs littérales, d'appels à de fonctions ou de référence à des variables

Les types d'expressions

Les expressions arithmétiques

Ces expressions manipulent et retournent des nombres. Les opérateurs arithmétiques devraient nous évoquer quelques souvenirs de cours de mathématiques élémentaires.

opérateur	Description
+	addition
-	soustraction ou négation
/	division
*	multiplication
%	modulo (reste de la division entière)
**	exponantiation (uniquement depuis PHP 5.6)

```php
$a = 2;
$b = 3;

echo $a + $b;       // donne 5
echo -$a;           // donne -2
echo $a - $b;       // donne -1
echo $b / $a;       // donne 1.5
echo $a * $b;       // donne 6
echo $b % $a        // donne 1
                        // (la division entière de 3 par 2 est 1
```

```
                        // il reste donc : 3 - (1 x 2) = 1
écho $a ** $b       // donne 8 (2 puissance 3)
```

Les expressions d'affectation

Une affectation consiste à donner (affecter) une valeur à une variable. Depuis le début de ce cours, nous n'avons cessé de recourir à ce type d'expression puisque nous avons déclaré et initialisé des variables. Cependant, nous ne connaissons qu'un opérateur d'affectation, il en existe d'autres.

opérateur	Description
=	affectation simple
+=	affectation addition `$a += 2;` équivaut à `$a = $a + 2;`
-=	affectation soustraction `$a -= 2;` équivaut à `$a = $a - 2;`
/=	affectation division `$a /= 2;` équivaut à `$a = $a / 2;`
*	affectation multiplication `$a *= 2;` équivaut à `$a = $a * 2;`
%	affectation modulo `$a %= 2;` équivaut à `$a = $a % 2;`
.=	affectation concaténation `$a .= ' ';` équivaut à `$a = $a.' ;`
++	incrémentation `$a++;` équivaut à `$a = $a + 1;`
–	décrémentation `$a--;` équivaut à `$a = $a - 1;`

Il est possible de placer les opérateurs d'incrémentation et de décrémentation avant ou après l'opérande pour indiquer à quel moment doit s'effectuer l'opération (post ou pré incrémentation/décrémentation). Cela n'a d'importance que si cette incrémentation se trouve au sein d'une autre expression.

```
$a = 1;
/*
* affiche 2 car l'incrémentation
* à lieu après l'évaluation de l'expression
*/
```

```
echo $a++ *2;
```

```
$a = 1;
/*
* affiche 4 car l'incrémentation
* à lieu avant l'évaluation de l'expression
*/
echo $a++ *2;
```

Les expressions de comparaison

Une expression de comparaison retourne un résultat booléen (vrai/faux).

opérateur	Description
==	égal
===	identique (même valeur et même type)
>	supérieur
>=	supérieur ou égal
<	inférieur
<	inférieur ou égal
!=	non égal
!==	non identique

Les expressions logiques

Une expression logique est la combinaison de plusieurs expressions booléennes avec des opérateurs logiques pour retourner un résultat booléen.

Par exemple, "je suis authentifié ET j'ai les droits administrateur" est une expression logique.

opérateur	Description
and, &&	ET logique

or,	
xor	OU exclusif
!	négation logique

var 1	var2	opérateur	résultat
vrai	vrai	AND	vrai
vrai	faux	AND	faux
faux	vrai	AND	faux
faux	faux	AND	faux
vrai	vrai	OR	vrai
vrai	faux	OR	vrai
faux	vrai	OR	vrai
faux	faux	OR	faux
vrai	vrai	XOR	faux
vrai	faux	XOR	vrai
faux	vrai	XOR	vrai
faux	faux	XOR	faux

Les constantes

Comme son nom le suggère, une constante référence une donnée qui ne pourra être modifiée au cours de l'exécution du programme. Par convention, les constantes sont écrites en majuscules.
La déclaration d'une constante s'effectue avec la fonction `define` qui admet deux arguments, le nom de la constante et sa valeur.

```
define('MAIL_ADMIN','admin@monsite.com');
define('PI',3.14116);

//Utilisation d'une constante
```

```php
$perimetre = (2 * PI) * ($rayon ** 2);
```

> Notons que, contrairement aux variables, les constantes ne sont pas précédées du signe $

Il est possible de tester l'existence d'une constante avec la fonction `defined` qui retourne un booléen.

```php
define('PI',3.14116);

defined('PI'); // true
defined('PI2');// false
```

Enfin, la conction `constant` permet de récupérer la valeur d'une constante dont le nom est passé en argument sous la forme d'une chaîne de caractère. Cette fonction est utile si le nom de la constante à récupérer dépend du résultat d'une fonction ou d'un chemin d'exécution du prgramme.

```php
define('PI',3.14116);
define('PI_PRECIS',3.14159265358979323846264338327950288841971);

//Définition du nom de constante à utiliser
//en fonction de la valeur de $precision (true/false)
$constantePI = $precision?'PI_PRECIS':'PI';

$perimetre = (2 * constant($constantePI)) * ($rayon ** 2);
```

Les constantes sont utiles lorsque l'on souhaite référencer des données qui ne doivent jamais être modifiées au cours de l'exécution du programme. Des données de configuration de l'application telles que l'adresse du serveur de base de données sont de bons candidats à cet usage.

Exercices

- à partir d'une variable $hauteur et d'une variable $largeur afficher l'aire d'un rectangle.
- Connaissant le prix unitaire et la quantité d'un produit commandé, calculer le total HT et TTC. Afficher sous la forme suivante : La commande de {qt} produits à {PU} l'unité donne un total {HT} hors taxe soit {TTC} TTC.

- Calculer et afficher la circonférence d'un cercle à partir d'une variable $diametre et d'une constante PI.

Les structures algorithmiques

Les conditions

Structure algorithmique élémentaire, les conditions définissent de blocs d'instructions qui ne seront exécutés que si une expression est vraie. Cette structure permet donc des branchements ou prise de décision qui affecteront le déroulement du programme.

Dans l'exemple ci-dessous, l'expression testée est `$age >= 18`, il s'agit d'une expression booléenne dont l'évaluation retournera soit true soit false. Notons que la condition se trouve entre parenthèse et que les instructions sont encadrées par des accolades. Notons également qu'il n'y a pas de point-virgule en début et fin de condition.

```
if ($age >= 18) {
    echo 'vous êtes majeur';
}
```

La clause else

La clause `else` (sinon) définit un bloc d'instruction qui sera exécuté si la condition n'est pas vérifiée. Cette clause ne peut exister sans une structure `if` préalable.

```
if ($age >= 18) {
    echo 'vous êtes majeur';
} else {
    echo 'vous êtes mineur';
}
```

La clause elseif

La clause `elseif` (sinon si) se comporte comme la clause `else`, mais permet d'ajouter une nouvelle condition. Cette clause est donc très utile pour

tester différentes expressions au sein de la même condition.

```
if ($facture > 100){
    $remise = 0.05;
} elseif($anciennete >5){
    $remise = 0.10;
} else {
    $remise = 0;
}
```

Conditions multiples

Il est possible de tester plusieurs expressions booléennes dans une même condition en séparant ces expressions par des opérateurs logiques.

```
if ($motDePasse == 'test' and $login =='admin'){
    echo 'vous avez accès';
}
```

Dans le cas ou deux expressions sont séparées par un opérateur logique `and`, si la première expression est fausse, la deuxième ne sera pas évaluée, car l'ensemble ne peut être que faux (false and true = false).

l'affectation ternaire

Il s'agit d'un raccourcis permettant une affectation conditionnelle. Sa syntaxe est la suivante :

```
$variable = condition?valeur si vrai:valeur si faux;
```

```
$majorite = $age>=18?'majeur':'mineur';
```

La structure switch

Cette structure est une alternative à la clause `if`, elle permet de définir une suite d'action associée à chaque valeur possible d'une variable.

```
switch(true){
```

```
    case ($age <=5):
        echo 'bébé';
        break;
    case ($age>5 and $age <=12):
        echo 'enfant';
        break;
    case ($age >12 and $age <=17):
        echo 'adolescent';
        break;
    case ($age >17 and $age <= 26):
        echo 'jeune adulte';
        break;
    case ($age >26 and $age <= 70):
        echo 'adulte';
        break;
    case ($age >70 and $age <= 90):
        echo 'viel adulte';
        break;
    case ($age >90):
        echo 'adulte d\'âge canonique';
        break;
}
```

Il est possible de définir un comportement par défaut avec la clause `default`, il est également possible de combiner plusieurs valeurs ensemble lorsqu'elles doivent être associées au même comportement.

```
switch($score){
    case 1:
        $message = 'Vraiment nul';
        break;
    case 2:
    case 3:
        $message = 'Pas terrible du tout';
        break;
    case 4:
    case 5:
        $message = 'Ok mais peut mieux faire';
        break;
    case 6:
    case 7:
        $message = 'C\'est bien, bon score' ;
        break;
    case 8:
    case 9:
```

```
        $message = 'C\'est très bien' ;
        break;
    case 10:
        $message = 'Excellent !';
        break;
    default:
        $message = 'Vous avez triché';
}

echo $message. '<br>';
```

Les boucles

Une boucle permet d'exécuter la même suite d'instructions plusieurs fois. Elle évite ainsi de multiples répétitions fastidieuses.

La boucle for

Cette boucle intègre une variable compteur. Sa syntaxe comporte trois éléments séparés par des point-virgules.

1. Déclaration et initialisation du compteur
2. Condition pour rester dans la boucle
3. Ajustement de la valeur du compteur à chaque tour de boucle

L'exemple ci-dessous présente une boucle qui initialise un compteur $i à la valeur 1. Cette boucle tournera tant que $i sera inférieur ou égal à 10 et enfin à chaque tour de la boucle, la valeur de $i est augmentée de 1.

```
for ($i = 1; $i <= 10; $i++) {
    echo 'ligne' . $i . '<br>';
}
```

Il est également possible de compter à l'envers

```
<h3>compte à rebours</h3>
<?php
for ($i = 10; $i >= 1; $i--) {
    echo $i . ' ';
```

```
}
echo 'ignition <br>';
```

La boucle while

La boucle while n'inclue pas de compteur par défaut, elle ne requière donc qu'une condition et tournera tant que cette condition restera vérifié.

Cette structure est intéressante lorsque nous ne connaissons pas a priori le nombre d'itérations de boucle nécessaires.

```
/*
* Combien puis-je acheter de produits
* à 12.50 l'unité avec 68 euros dans mon portefeuille
*/
$compteur = 1;
$fortune = 68;
$prix = 12.5;
$depense = ($compteur * $prix);
$jaiDuPeze = $depense <= $fortune;
while( $jaiDuPeze){
    echo $depense. " ($compteur) <br>";
    $compteur ++;
    $depense = ($compteur * $prix);
    $jaiDuPeze = $depense <= $fortune;
}
echo 'je peux commander '.($compteur-1).' produits <br>';
```

> La boucle while est souvent utilisée pour parcourir le contenu d'un fichier ou d'une requête, car ces deux objets fournissent souvent un curseur ou une méthode indiquant si l'on se trouve à la fin du fichier ou du jeu d'enregistrement.

Interruption de boucle

PHP propose deux instructions d'interruption qui permettent de sortir prématurément d'une boucle.

instruction	Description
`break`	sortie définitive de la boucle

`continue`	élude les instructions de l'itération en cours

exemples

```
//Sortie d'une boucle lorsque le total
//est supérieur ou égal à la limite
$prix = 5;
$limite = 35;
for($i=0; $i <= 10; $i++){
    if($prix*$i >= $limite){
        //sortie de la boucle
        break;
    }
}
```

```
//Affichage des nombre pairs de 1 à 10
for($i=0; $i <= 10; $i++){
    if($i > 0 && $i %2 > 0){
        continue;
    }
    //Cette instruction n'est exécutée
    //que si $i est pair
    echo "$i <br>";
}
```

Les instructions break et continue pendent être remplacées par une condition supplémentaire dans l'initialisation ou dans le corps de la boucle.

```
$prix = 5;
$boucle = true;

for($i=0; $boucle && $i <= 10; $i++){
    if($prix*$i >= 35){
        $boucle = false;
    }
}
```

```
//Affichage des nombre pairs de 1 à 10
for($i=0; $i <= 10; $i++){
    if($i >0 && $i%2 == 0){
        echo "$i <br>";
```

```
    }
}
```

Exercices

- Ecrire un code qui détermine si un nombre donné est premier (uniquement divisible par 1 et par lui même).
- Afficher une pyramide constituée de n balise `<hr>` de taille croissante.
- Toujours avec des balises `<hr>`, afficher un losange (une pyramide suivie d'une pyramide inversée).
- Encore avec `<hr>` afficher un sablier (une pyramide inversée suivie d'une pyramide).
- Créer une table de multiplication de 1 à 10 sous forme de tableau html en coloriant les lignes et colonnes paires.

Les tableaux

Un tableau est une variable particulière qui peut stocker de multiples valeurs. Chacune de ces valeurs est identifiée soit par un indice numérique soit par une clef unique, dans ce dernier cas l'on parle de tableaux associatifs.

Les tableaux indicés

Déclaration et affectation

Pour déclarer une variable de type tableau, il existe deux syntaxes, la plus courante fait appel au mot-clef `array` :

```php
//Déclaration d'un tableau vide
$listeVilles = array();
//remplissage du tableau
$listeVilles[0] = 'Paris';
$listeVilles[1] = 'Marseille';
$listeVilles[2] = 'Lyon';

//Déclaration et affectation de valeurs
$listeVilles = array('Paris','Marseille','Lyon');


//Depuis PHP 5.5 il est également possible
//de déclarer les tableaux comme ceci

//Tableau vide
$listeVilles = [];

//Déclaration et affectation de valeurs
$listeVilles = ['Paris','Marseille','Lyon'];
```

> **Attention :** Notons que les indices de tableau commencent par zéro et non un.

Il est ensuite possible d'appeler une des valeurs du tableau en faisant référence à son indice.

```php
echo $listeVilles[0];//affiche Paris
```

Parcourir les éléments d'un tableau

Il est très fréquent de vouloir parcourir l'ensemble des éléments d'un tableau soit pour les afficher, soit pour réaliser un autre traitement sur ces valeurs. Pour réaliser cette opération, nous utiliserons le plus souvent une boucle.

A cet effet, il sera utile de connaître le nombre d'éléments de notre tableau. La fonction `count` nous donnera cette information.

```php
$liens = ['Accueil','Produits','contact'];
$nbElements = count($liens)
for($i=0; $i < $nbElements; $i++){
    echo $liens[$i].'<br>";
}
```

> **Note :**
> Le stockage du nombre d'éléments du tableau dans la variable `$nbElements` évite d'appeler la fonction `count` à chaque tour de boucle. Sur un tableau de faible dimension tel que celui de notre exemple, la différence de performance est insignifiante. Avec l'augmentation du nombre d'éléments, cette différence se fera de plus en plus sensible.

Boucle foreach

La boucle `foreach` parcoure tous les éléments d'un tableau et affecte la valeur de l'élément courant à une variable définie dans l'initialisation de la boucle.

```php
$liens = ['Accueil','Produits','contact'];

foreach($liens as $element){
    echo $element.'<br>";
}
```

Trier un tableau

Les fonction `sort` et `rsort` effectuent un tri ascendant ou descendant sur les éléments d'un tableau. Ces deux fonctions renvoient un booléen qui indique le succès ou l'échec de l'opération.

```
$data = [5,8,2];
//Donne [2,5,8]
sort($data);
var_dump($data);

//Donne [8,5,2]
rsort($data);
var_dump($data);
```

Les flags

Les fonctions de tri admettent un deuxième argument sous la forme de constantes qui indiquent le mode de comparaison utilisé pour effectuer le tri.

- **SORT_REGULAR** - compare les éléments normalement (pas de changement de type)
- **SORT_NUMERIC** - compare les éléments numériquement
- **SORT_STRING** - compare les éléments sous forme de chaînes de caractères
- **SORT_LOCALE_STRING** - compare les éléments sous forme de chaînes de caractères, en se basant sur la locale courante. La fonction utilise les locales, et elles peuvent être modifiées en utilisant la fonction setlocale()
- **SORT_NATURAL** - compare les éléments sous forme de chaînes de caractères, en utilisant le "tri naturel", comme le fait la fonction natsort()
- **SORT_FLAG_CASE** - peut être combiné (avec le mot clé OR) avec SORT_STRING ou SORT_NATURAL pour trier les chaînes sans tenir compte de la casse

```
$liens = [
    'image1',
    'image20',
    'image2',
    'image12'
];
var_dump($liens);
/*
```

```
array (size=4)
  0 => string 'image1' (length=6)
  1 => string 'image20' (length=7)
  2 => string 'image2' (length=6)
  3 => string 'image12' (length=7)
*/

sort($liens);
var_dump($liens);
/*
array (size=4)
  0 => string 'image1' (length=6)
  1 => string 'image12' (length=7)
  2 => string 'image2' (length=6)
  3 => string 'image20' (length=7)
*/

sort($liens, SORT_NATURAL);
var_dump($liens);
/*
array (size=4)
  0 => string 'image1' (length=6)
  1 => string 'image2' (length=6)
  2 => string 'image12' (length=7)
  3 => string 'image20' (length=7)
*/
```

Les tableaux associatifs

Un tableau associatif utilise des chaînes de caractères (appelées clefs) au lieu d'indices numériques pour identifier chaque case du tableau. Il va de soit que ces clefs doivent être uniques au sein du tableau.

Déclaration et affectation

```
//Déclaration d'un tableau associatif
$scores = array(
    'Pierre'    => 10,
    'Isabelle'  => 8,
    'Jeanne'        => 11
);

//Ajout d'une nouvelle clef
```

```php
$scores['Paul'] = 7;

//Redéfinition d'une clef existante
$scores['Isabelle'] = 9;

//Récupération de la valeur d'une clef
echo $scores['Paul'];
```

Parcourir un tableau associatif

Le parcours d'un tableau associatif s'effectue généralement avec une boucle foreach de la manière suivante :

```php
$scores = array(
    'Pierre'    => 10,
    'Isabelle'  => 8,
    'Jeanne'        => 11
);

foreach($scores as $key=>$value){
    echo "$key = $value <br>";
}
```

Trier un tableau associatif

Pour trier un tableau associatif, PHP propose plusieurs fonctions

fonction	description
asort	trie les valeurs du tableau par ordre croissant
arsort	trie les valeurs du tableau par ordre décroissant
ksort	trie les clefs du tableau par ordre croissant
krsort	trie les clefs du tableau par ordre décroissant

Tout comme la fonction `sort`, ces fonctions admettent un deuxième argument flag qui détermine le ode de comparaison.

Liste des flags

- **SORT_REGULAR** - compare les éléments normalement (pas de changement de type)
- **SORT_NUMERIC** - compare les éléments numériquement
- **SORT_STRING** - compare les éléments sous forme de chaînes de caractères
- **SORT_LOCALE_STRING** - compare les éléments sous forme de chaînes de caractères, en se basant sur la locale courante. La fonction utilise les locales, et elles peuvent être modifiées en utilisant la fonction setlocale()
- **SORT_NATURAL** - compare les éléments sous forme de chaînes de caractères, en utilisant le "tri naturel", comme le fait la fonction natsort()
- **SORT_FLAG_CASE** - peut être combiné (avec le mot clé OR) avec SORT_STRING ou SORT_NATURAL pour trier les chaînes sans tenir compte de la casse

Les tableaux à n dimensions

Pour définir un tableau à n dimensions, il suffit d'imbriquer les tableaux, c'est à dire de déclarer que la valeur d'une case de tableau est elle même un tableau. Ce type de structure est fréquemment utilisé pour modéliser une table qui est un tableau indicé de tableaux associatifs.

Pour accéder aux données, il suffit alors d'utiliser autant de crochet qu'io y a de dimensions dans le tableau.

```
$table = array (
    array (
        'nom'           => 'Isabelle',
        'age'           => 32,
        'profession'    => 'journaliste'
    ),
    array (
        'nom'           => 'Paul',
        'age'           => 38,
        'profession'    => 'photographe'
    ),
);
```

Dans l'exemple ci-dessus, nous accédons à l'âge de la première ligne avec la syntaxe suivante : `$table[0]['age']`.

Boucle sur un tableau à n dimensions

Voici un exemple simple de boucle sur un tableau à deux dimensions.

```php
//Affichage du tableau

//Compte du nombre de lignes
$nbLignes = count($table);
echo "<table>";

//Affichage de l'en-tête
echo "<thead><tr>";
echo "<th>Nom</th>";
echo "<th>Age</th>";
echo "<th>Profession</th>";
echo "</tr></thead><tbody>";

//Affichage du contenu
for ($i = 0; $i < $nbLignes; $i++) {
    $ligne = $table[$i];
    echo "<tr>";
    echo "<td>" . $table[$i]['nom'] . "</td>";
    echo "<td>" . $table[$i]['age'] . "</td>";
    echo "<td>" . $table[$i]['profession'] . "</td>";
    echo "</tr>";
}

//fin du tableau
echo "</tbody></table>";
```

L'exemple ci-dessus fonctionne, mais il n'est pas très souple, en effet s'il nous arrivait d'ajouter une colonne, nous aurions à modifier notre code. Le script suivant est plus générique.

```php
//Affichage du tableau

//Compte du nombre de lignes
$nbLignes = count($table);
echo "<table>";

//Affichage de l'en-tête
if ($nbLignes > 0) {
    echo "<thead><tr>";
    //Liste des clefs d'un tableau associatif
```

```
    //retourne un tableau indicé des clefs
    $cols = array_keys($table[0]);
    foreach ($cols as $label) {
        echo "<th>$label</th>";
    }
    echo "</tr></thead>";
}

// Affichage du corps du tableau
echo "<tbody>";

//Affichage du contenu
for ($i = 0; $i < $nbLignes; $i++) {
    echo "<tr>";
    //Affichage des données
    foreach ($cols as $label) {
        echo "<td>" . $table[$i][$label] . "</td>";
    }
    echo "</tr>";
}

//fin du tableau
echo "</tbody></table>";
```

Quelques opérations sur les tableaux

PHP propose de nombreuses fonctions de manipulation de tableaux. La liste qui suit n'est pas exhaustive, pour de plus amples informations nous nous référerons à la documentation officielle.

Rechercher une valeur dans un tableau

PHP propose des fonctions de recherche au sein des tableaux

fonction	description
in_array	indique si une valeur se trouve dans le tableau
array_search	recherche une valeur dans un tableau et retourne la clef ou l'indice
array_key_exists	indique si une clef existe dans un tableau associatif

array_keys	le second argument est un critère de recherche qui filtre les clefs
isset	retourne vrai si la clef existe et si elle n'est pas nulle

```php
$tableau = array (5,8,9,8);
$associatif = array(
    'Quingey'           => '25440',
    'Abbans-dessus'     => '25440',
    'Chouzelot'         => '25440',
    'Lons-le-Saunier'   => '39000',
    'Besançon'          => '25000',
    'Inconnu'           => null
);

//true
var_dump(in_array(8,$tableau));

//false
var_dump(in_array(12,$tableau));

//retourne 1 (la position de la première occurrence
var_dump(array_search(8,$tableau));

//retourne un tableau de toutes les clefs
//correspondant au critère
var_dump(array_keys($tableau,8));

//même chose avec un tableau associatif
var_dump(array_keys($associatif,'25440'));

//retourne true
var_dump(array_key_exists('Chouzelot', $associatif));

//retourne true
var_dump(array_key_exists('Inconnu', $associatif));

//retourne false car la valeur est null
var_dump(isset($associatif['Inconnu']));
```

Obtenir un tableau depuis un chaîne

Cette opération est un grand classique. Souvent, le seul moyen d'échanger des données entre deux programmes passe par la transmission de chaînes de

caractères.
L'idée ici est de découper une chaîne de caractère selon un délimiteur qui est constitué d'un ou plusieurs caractères. A chaque occurrence du délimiteur, nous aurons une nouvelle case de tableau. Avec PHP, nous utiliserons la fonction `explode` pour réaliser cette opération.

```php
$str = "Jean;Sophie;Isabelle;Anna;Boris;Philippe";
//$liste est un tableau
$liste = explode(';', $str);
var_dump($liste);
```

Convertir un tableau en chaîne

L'opération inverse est également intéressante, elle permet d'éviter l'écriture d'une boucle pour des cas simples. Pour réaliser cette conversion nous utiliserons la fonction `implode` qui logiquement est l'inverse de la fonction `explode`.
Cette fonction concatène l'ensemble des cases d'un tableau en ajoutant un délimiteur (passé en argument) entre chacune des cases.

```php
$liste = array('Jean','Sophie','Isabelle');
$message = '<ul><li>';
$message .= implode('</li><li>', $liste);
$message .= '</li></ul>';

echo $message;
```

Dédupliquer les valeurs d'un tableau

La fonction `array_unique` retourne un tableau sans doublons.

```php
$tags = array('PHP','Java','PHP');
$uniqueTags = array_unique($tags);
```

Obtenir les clef et les valeurs d'un tableau associatif

Les fonctions `array_keys` et `array_values` permettent d'extraire respectivement les clefs et la valeurs d'un tableau associatif.

```php
$produit = array(
    'designation'   => 'Peluche Tux',
    'code'          =>  'TUX-001',
    'prix'          => 8.5
);

var_dump(array_keys($produit));
var_dump(array_values($produit));
```

Opérations ensemblistes sur les tableaux

Les opérations ensemblistes sont l'union, la différence et l'intersection.

fonction	description
array_merge	fusion des tableaux passés en argument
array_diff	différence des tableaux passés en argument
array_intersect	intersection des tableaux passés en argument

```php
$tab1 = array(1,2,3);
$tab2 = array(5,6,7);
$tab3 = array(7,1,10);

/*
retourne la fusion (avec doublons)
des tableaux $tab1, $tab2 et $tab3
array (size=9)
  0 => int 1
  1 => int 2
  2 => int 3
  3 => int 5
  4 => int 6
  5 => int 7
  6 => int 7
  7 => int 1
  8 => int 10
*/
var_dump(array_merge($tab1,$tab2,$tab3));
```

```php
/*
retourne les éléments de $tab1 qui n'existent pas dans $tab3
array (size=2)
  1 => int 2
  2 => int 3
*/
var_dump(array_diff($tab1,$tab3));

/*
retourne les éléments communs entre $tab1 et $tab3
array (size=1)
  0 => int 1
*/
var_dump(array_intersect($tab1,$tab3));
```

Les variables super globales

PHP dispose de tableaux associatifs dit super globaux. Ces variables sont disponibles dans tous les contextes d'exécution (fonction, classe ou script global).

tableau	description
$_GET	QueryString (paramètres passés dans l'url)
$_POST	Données postées par un formulaire
$_COOKIE	Données des cookies
$_REQUEST	Regroupement des tableaux $_GET$,_POST et $_COOKIE
$_SERVER	Informations relatives au serveur web
$_ENV	Informations relatives à l'environnement d'exécution
$_FILES	Données concernant le fichiers téléchargés (upload)
$_SESSION	Variables de session

```php
//initialisation d'une session
//pour éviter une erreur lors de l'affichage
//de la superglobale $_SESSION
session_start();
```

```php
//Affichage de l'ensemble des variables superglobales
$globalsNames = array(
    '_GET','_POST',
    '_COOKIE','_REQUEST',
    '_SERVER','_ENV','_FILES',
    '_SESSION'
);

foreach($globalsNames as $varName){
    echo "<h2>$varName</h2>";
    var_dump($$varName);
}
```

Exercices

Cadavre exquis

A partir de tableaux noms, verbes et complements, réaliser un programme de cadavre exquis (constitution d'une phrase aléatoire).

> La fonction `rand(min,max)` génère un entier aléatoire entre min et max.

```php
$nom = [
    "le chien",
    "le chat",
    "mon ami",
    "le développeur",
    "le chef de projet",
    "le client",
    "le formateur",
    "le gateau",
    "l'ordinateur",
    "la voiture",
    "le vélo",
    "le médecin",
    "l'enfant",
    "le graphiste",
    "l'oiseau",
    "le programme",
    "le jeu"
];
```

```php
$verbe = [
    "mange",
    "regarde",
    "code",
    "fabrique",
    "admire",
    "aime",
    "déteste",
    "rencontre",
    "examine",
    "évalue",
    "percute",
    "ausculte",
    "dessine",
    "brocarde",
    "exécute",
    "lance"

];

$complement = [
    "dans la voiture",
    "avec lenteur",
    "sur la tour Eiffel",
    "dans la cabane",
    "pour avoir une bonne note",
    "pour la bonne raison",
    "avec rapidité",
    "aujourd'hui",
    "sans hésitation",
    ""
];
```

Table html

A partir d'un tableau indicé de tableaux associatifs, afficher les données dans une table html.

```php
$catalogue = array(
    array(
        'designation'   => 'La peste'
        'prix'          => 10.80
        'categorie'     => 'Roman'
        'auteur'            => 'Albert Camus'
```

```
        'editeur'       => 'Gallimard'
    ),
    array(
        'designation'   => 'PHP pour les nuls'
        'prix'          => 15.80
        'categorie'     => 'Informatique',
        'auteur'            => 'Janet Valade'
        'editeur'       => 'First Interact'
    ),
    array(
        'designation'   => 'Essais'
        'prix'          => 38.70
        'categorie'     => 'Philosophie',
        'auteur'            => 'Montaigne'
        'editeur'       => 'Pocket'
    ),
    array(
        'designation'   => 'Winston Churchill'
        'prix'          =>  28.30
        'categorie'     => 'Biographie',
        'auteur'            => 'François Kersaudy'
        'editeur'       => 'Tallandier'
    ),
    array(
        'designation'   => 'Barcelone week end'
        'prix'          => 9.00
        'categorie'     => 'Guide de voyage',
        'auteur'            => 'Michelin'
        'editeur'       => 'Michelin'
    ),

);
```

Menu déroulant

A partir d'un tableau, afficher une liste déroulante.

```
$professions = array(
    array(
        'code'      => 7,
        'libelle'   => 'Infographiste'
    ),
    array(
        'code'      => 32,
        'libelle'   => 'Architecte logiciel'
```

```
    ),
    array(
        'code'      => 18,
        'libelle'   => 'Développeur front-end'
    ),
    array(
        'code'      => 19,
        'libelle'   => 'Développeur back-end'
    ),
    array(
        'code'      => 22,
        'libelle'   => 'Développeur full-stack'
    ),
);
```

Les inclusions

Les instructions d'inclusions effectuent un copier-coller d'un fichier dans un autre au niveau du serveur. La première utilité de ces instructions est donc d'éviter les répétitions que tout bon programmeur doit avoir en horreur. Un autre intérêt réside dans la possibilité d'organiser le code en unités logiques et d'assembler ces unités pour réaliser un programme. Enfin, grâce aux inclusions, nous pourrons développer des bibliothèques de fonctions que nous rendrons accessible à tout ou partie de notre application.

Les instructions d'inclusions

PHP propose plusieurs instructions pour réaliser des inclusions.

instruction	Description
include	inclusion
include_once	inclusion unique
require	inclusion impérative
require_once	inclusion impérative unique

L'inclusion impérative soulève une erreur fatale (qui stoppe l'exécution du programme) si le fichier à inclure n'existe pas.

L'inclusion unique s'assure que le fichier n'est inclus qu'une seule fois.

Toutes ces fonctions admettent un argument qui est le chemin vers le fichier inclu.

Ces instructions retournent un booléen qui indiquent le succès (true) ou l'échec (false) de l'inclusion.

```
//inclusion du fichier config.php
//se trouvant dans le même dossier
include 'config.php';
```

Les chemins d'inclusion

Lorsque PHP recontre une instruction d'inclusion, il analyse le chemin passé en argument et effectue les opérations suivantes :

- Si le chemin ne contient aucune référence à une arborescence, mais seulement un nom de fichier, alors PHP ira chercher ce fichier d'abord dans les dossiers de l'include_path, puis dans le dossier du script apellant.
- Si le chemin contient des références à une arborescence alors l'include_path sera ignoré.

```
//inclusion du fichier lib se trouvant :
//soit dans un des dossier de l'include_path
//soit dans le dossier courrant
include 'lib.php';

//Chemin relatif, fait référence à un sous dossier library
//du dossier courant
include 'library/lib.php';

//Chemin relatif, fait référence à dossier library
//se trouvant un niveau au dessus du dossier courant
include '../library/lib.php';

//Chemin absolu sur linux
include '/home/username/public_html';
```

> **Attention :** Dans le cas d'inclusions imbriquées (inclusion d'un fichier comportant lui même des inclusions), c'est le dossier du fichier appellant (le premier fichier de la chaîne d'inclusion) qui sert de réference pour la résolution des chemins relatifs.
> Pour simplifier les choses, nous éviterons donc les chemins relatifs et enregistrerons le chemin absolu vers le dossier racine du site afin d'y faire référence dans toutes nos inclusions.

Quelques fonctions relatives aux chemins

- `getcwd()`
  retourne le chemin absolu du dossier contenant le script qui exécute cette

fonction

- `realpath(chemin)`
  retourne le chemin absolu d'un chemin relatif
- `__DIR__`
  retourne le chemin du fichier courant
- `dirname(chemin)` retourne le chemin du dossier parent

Considérons un fichier php dont les noms et chemins sont les suivants : `/var/web/site/index.php`

```php
// affiche /var/web/site
$cwd = getcwd();
echo $cwd.'<br>';

// affiche /var/web
echo realpath($cwd.'/../').'<br>';

// affiche /var/web/site
echo __DIR__.'<br>';

// affiche /var/web
echo dirname(__DIR__).'<br>';
```

Exercices

- Découper une page html en quatre fichiers (en_tete.php, navigation.php, corps.php et pied.php). Assembler ces fichiers avec des inclusions dans un fichier index.php.

Les entrées

La chaîne de requête

Un URL est constitué de plusieurs éléments, le protocole, l'adresse et la requête. Ce dernier élement permet de passer des paramètres aux pages PHP. Ces paramètres apparaissant en clair dans la barre d'adresse du navigateur, ils sont lisible et modifiables par l'utilisateur. Nous ne **transmettrons donc pas d'informations sensibles** par ce biais et nous ne **validerons toujours les paramètres ainsi récupérés** avant de les utiliser.

Anatomie d'un URL

exemple d'URL avec une chaîne de requête (QueryString)
`https://duckduckgo.com/?q=php&ia=about`

élément	exemple
protocole	https://
adresse	duckduckgo.com/
requête	**?**q=php**&**ia=about

- La chaîne de requête est séparée du reste de l'URL par un caractère "?".
- Cette chaîne de requête contient des paires clef=valeur séparées par un caractère "&".

Ainsi dans notre exemple, nous pouvons décomposer les paramètres de la façon suivante :

clef	valeur
q	php
ia	about

Récupérer un paramètre

Les paramètres passés dans l'URL sont stockés dans un tableau super global `$_GET`.

```php
//Test des paramètres passés en url
var_dump($_GET);

//Affichage du paramètre age
echo $_GET['age'];
```

Les formulaires

Les formulaires permettent également de transmettre des paramètres à un script PHP. Pour cela, il est important de préciser deux attributs de la balise `form`.

- action : il s'agit du script vers lequel les données du formulaire seront envoyées
- method : il s'agit de la méthode de transmission des données du formulaire (GET ou POST)
    - GET : Les données sont transmises dans l'url sous la forme d'une chaîne de requête (comme vu précédemment). Il est donc possible d'enregistrer cette transmission dans un signet. Par contre, les données apparaissent en clair dans la barre d'adresse du navigateur et sont très facilement modifiables. La taille de la requête est également limitée. Cette limite dépend du navigateur de l'internaute. Pour éviter les problèmes, il est conseillé de ne pas dépasser les 2000 caractères pour l'ensemble de l'url.
    - POST : Les données sont transmises dans l'en-tête http de la requête, elles n'apparaissent donc pas en clair et ne sont pas limitées en taille. Le fait que les données soient masquées ne rend pas la méthode POST beaucoup plus sûre. En effet, il existe quantité d'outils permettant de modifier et transmettre simplement une requête http.

Un formulaire simple

Voici un formulaire simple qui demande la saisie d'un age. Ce formulaire envoie

ses données à un script `traitement-formulaire.php` avec la méthode POST. Ce formulaire contient un champ `input` dont l'attribut `name` est égal à "age".
Ce nom sera repris en tant que clef dans le tableau associatif `$_POST`.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Formulaire age</title>
</head>
<body>

<!--
l'attribut novalidate désactive la validation côté client
et permet de tester le formulaire avec des données non valides
-->
<form method="post" action="traitement-formulaire.php" novalida
te>
    <label for="age">Votre Age</label>
    <!--
    l'attribut name de la balise input
    correspond à la clef du tableau $_POST
    -->
    <input type="number" id="age" name="age">
    <button type="submit">Valider</button>
</form>

</body>
</html>
```

Le script de traitement se déroule en trois temps :

1. La récupération des données
2. Le traitement des données
3. l'affichage du résultat

```
<?php
//Affichage des données transmises
var_dump($_POST);
```

```php
//Récupération des données postées
if(isset($_POST['age'])){
    $age = $_POST['age'];
} else {
    $message = "Vous devez passer par le formulaire pour saisir
 un age";
}

//Traitement des données
if(! is_numeric($age)){
    $message = "Vous devez saisir un nombre";
}elseif ($age <18){
    $message = "vous avez $age ans, vous êtes mineur";
} else {
    $message = "vous avez $age ans, vous êtes majeur";
}

//Affichage du message
echo $message;
```

Données non scalaires (valeurs multiples)

Certains contrôles de formulaires permettent des choix multiples parmi une série de valeurs. Dans ce cas, la variable transmise est un tableau des valeurs séléctionnées.
Parfois, pour des raisons d'organisation, nous souhaiterons également récupérer une suite de choix ou de saisie sous la forme d'un tableau.

Pour ce faire, le nom de la variable récupérée doit être suivie des caractères `[]`.

Voic une illustration sous la forme d'une formulaire permettant le choix de multiples centres d'intérêt. Ici, l'utilisation d'un tableau n'est pas obligatoire, mais il simplifie le traitement et évite d'avoir à modifier le code lors de l'ajout éventuel de nouvelles cases à cocher.

```php
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
```

```
    <title>Centres d'intérêt</title>
</head>
<body>

<form method="post" action="traitement-formulaire.php" novalida
te>
    <h3>Vos centres d'intérêt</h3>

    <div>
        <input type="checkbox" id="photo" name="interet[]" value
="photo">
        <label for="photo">photo</label><br>
        <input type="checkbox" id="dessin" name="interet[]" val
ue="dessin">
        <label for="dessin">dessin</label><br>
        <input type="checkbox" id="programmation" name="interet
[]" value="programmation">
        <label for="programmation">programmation</label><br>
        <input type="checkbox" id="ceramique" name="interet[]"
value="ceramique">
        <label for="ceramique">céramique</label><br>
    </div>

    <button type="submit">Valider</button>
</form>

</body>
</html>
```

Voici le traitement de ce formulaire. Encore une fois, le fait d'utiliser un tableau nous évite d'avoir à récupérer la saisie de chaque contrôle individuellement ce qui rend notre code plus simple et évolutif.

```
<?php
//Affichage des données transmises
var_dump($_POST);

//Récupération des données postées
if(isset($_POST['interet'])){
    $interets = $_POST['interet'];
    //Test de validité des données
    if(! is_array($interets)){
        $message = "Vous devez choisir au moins un centre d'int
érêt";
    } else {
```

```
        //Concaténation des choix dans une liste html
        $message = "<ul><li>". implode('</li><li>', $interets).
'</li></ul>';
    }
} else {
    $message = "Vous devez passer par le formulaire pour choisi
r vos centres d'intérêt";
}

//Affichage du message
echo $message;
```

Organisation des contrôles de formulaire

L'organisation des contrôles de formulaire en tableau peut nous également permettre de regrouper les données en ensembles cohérents. De cette façon nous pourrons récupérer et traiter chaque partie du formulaire séparément.

Dans l'exemple ci-dessous, nous définissons deux groupes (contact et adresse), ces ensembles pourrons ensuite être traités de façon à générer des enregistrements dans deux tables de base de données correspondantes.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title></title>
</head>
<body>

<form method="post" action="traitement-formulaire.php">
    <fieldset>
        <legend>Contact</legend>

        <label>Civilité</label><br>
        <input type="radio" id="civF" name="contact[civilite]"
value="madame">
        <label for="civF">Madame</label><br>
        <input type="radio" id="civM" name="contact[civilite]"
value="monsieur">
        <label for="civM">Monsieur</label><br>

        <label for="nom">Nom</label>
        <input type="text" id="nom" name="contact[nom]"><br>
```

```
    </fieldset>

    <fieldset>
        <legend>Adresse</legend>

        <label for="voie">Adresse</label>
        <input type="text" id="voie" name="adresse[voie]"><br>
        <label for="code_postal">Code postale</label>
        <input type="text" id="code_postal" name="adresse[code_
postal]"><br>
        <label for="ville">Ville</label>
        <input type="text" id="ville" name="adresse[ville]"><br>

    </fieldset>

    <button type="submit">Valider</button>
</form>

</body>
</html>
```

La récupération des données s'effectue de la façon suivante.

```php
<?php
//Récupération des données du contact
$contact = $_POST['contact'];
var_dump($contact);

//récupération des données de l'adresse
$adresse = $_POST['adresse'];
var_dump($adresse);
```

Filtrage et validation des données

PHP propose une série de fonctions destinées au filtrage et à la validation des données. Ces deux opérations sont cruciales pour le développement d'applications web. En effet, les données, qu'elle soient transmise avec les méthodes GET ou POST sont très facilement modifiable. Il ne faut donc absolument jamais utiliser ces données sans les avoir préalablement filtrées et/ou validées.

Fonction filter_input

Cette fonction offre une méthode simple et rapide pour récupérer et filtrer les valeurs contenues dans une variable superglobale. Sa syntaxe est la suivante :

`filter_input(source,clef,[filtres],[options])`

source
: Une constante indiquant le tableau à filtrer, les valeurs admises sont `INPUT_GET`, `INPUT_POST`, `INPUT_COOKIE`, `INPUT_SERVER` et `INPUT_ENV`.
clef
: Le nom de la variable à récupérer
filtre
: Une constante indiquant le filtre à appliquer. La liste des filtres se trouve ici.
flags
: Une ou plusieurs constantes modifiant les comportement du filtre. Si nous souhaitons appliquer plusieurs flags, ceux-ci seront séparés par des caractères `|`.

Cette fonction retourne la valeur de la variable éventuellement filtrée selon le filtre spécifié. Si le filtre échoue, la fonction retourne `false`, si la variable n'est pas définie, la fonction retourne `null`.

```php
$nom = filter_input(INPUT_GET,'nom');

//La première ligne de code
//équivaut au code suivant
if (!isset($_GET['nom'])) {
    $a = null;
} elseif (!is_string($_GET['nom'])) {
    $a = false;
} else {
    $a = $_GET['nom'];
}
```

Les filtres

Les filtres appliquent un traitement sur les variables lors de leur récupération et augmentent ainsi l'utilité des fonctions de filtrage des données.

Filtres de nettoyage

Il s'agit ici de supprimer des caractères indésirables dans le contenu de la variable. Nous utiliserons ce type de filtre pour éliminer les balises html d'une saisie et nous prémunir ainsi contre une éventuelle attaque de type XSS (insertion de code javascript dans une saisie).

```php
$nom = filter_input(INPUT_GET,'nom',FILTER_SANITIZE_STRING);
$age = filter_input(INPUT_GET,'age',FILTER_SANITIZE_NUMBER_INT)
;

//La première ligne de code
//équivaut à
if (!isset($_GET['nom'])) {
    $a = null;
} elseif (!is_string($_GET['nom'])) {
    $a = false;
} else {
    $a = $_GET['nom'];
}
```

FILTER_SANITIZE_NUMBER_INT

Supprime tout ce qui n'est pas chiffres ou signes +- dans l'entrée.

valeur	résultat
5	5
5.1	51
5AV6	56

FILTER_SANITIZE_NUMBER_FLOAT

Même comportement que le précédent. Cependant, il est possible d'accepter des caractères supplémentaires en utilisant des options.

option	caractère
`FILTER_FLAG_ALLOW_FRACTION`	.
`FILTER_FLAG_ALLOW_THOUSAND`	,

`FILTER_FLAG_ALLOW_SCIENTIFIC`	e ou E

Il est évident que ce filtre n'a de sens qu'avec des flags, car sinon il réalise exactement le même traitement que `FILTER_SANITIZE_NUMBER_INT`.

FILTER_SANITIZE_STRING

Supprime les balises dans l'entrée.

valeur	résultat
`sébastien`	`sébastien`
`<h1>sébastien</h1>`	`sébastien`

FILTER_SANITIZE_SPECIAL_CHARS

Encode les caractères spéciaux (tous caractère dont le code ascii est inférieur à 32 ainsi que les caratères suivants : `<>&`).

valeur	résultat
`sébastien`	`sébastien`
`<h1>sébastien</h1>`	`&#60;h1&#62;sébastien\n&#60;/h1&#62;`

FILTER_SANITIZE_FULL_SPECIAL_CHARS

Encode tous les caractères spéciaux et accentués avec des entités html.

valeur	résultat
`sébastien`	`s&eacute;bastien`
`<h1>sébastien</h1>`	`<h1>s&eacute;bastien\n</h1>`

Filtres de validation

Les filtres de validation testent la valeur d'une variable selon les critères définis au sein du filtre. Si le test échoue, alors la fonction de filtre retourne `false`.

liste de filtres de validation

Filtrer toutes les variables

La fonction `filter_input` permet donc de filtrer les paramètres un à un, c'est bien, mais PHP va plus loin et nous permet de filtrer une série de paramètres en une seule fois. La fonction `filter_input_array` admet en argument une constante définissant la source des données et un tableau associatif où les clef sont les noms des variables et les valeurs les filtres à appliquer. Cette fonction retourne un tableau associatif contenant les valeurs filtrées.

```
$args = array(
    'nom' => FILTER_SANITIZE_STRING,
    'age' => FILTER_SANITIZE_NUMBER_INT,

);

$params = filter_input_array(INPUT_GET,$args);

var_dump($params);
/* Retourne un tableau associatif
*
* array (size=2)
*  'nom' => string 'TY' (length=2)
*  'age' => string '5' (length=1)
*/
```

Il est également possible d'ajouter des paramètres aux filtres, dans ce cas, il faudra utiliser un tableau associatif contenant les clefs suivantes

- filter : le nom du filtre à appliquer
- flags : le ou les flags éventuels
- options : un tableau associatif des options éventuelles

```
$args = array(
    'nom'   => FILTER_SANITIZE_STRING,
    'prix' => array('filter'    => FILTER_SANITIZE_NUMBER_FLOAT
,
                    'flags'     => FILTER_FLAG_ALLOW_FRACTION,
                ),
    );

$params = filter_input_array(INPUT_GET,$args);
```

```
var_dump($params);
/* Retourne un tableau associatif
*
* array (size=2)
*  'nom' => string 'TY' (length=2)
*  'age' => string '5' (length=1)
*/
```

La fonction `filter_input_array` admet également un troisième argument booléen facultatif `add_empty`. Lorsque cet argument vaut `true` et que des clefs du tableau de paramètre sont absente de la source, ces clefs sont alors ajoutées au résultat avec la valeur `null`. Si `add_empty` vaut false, alors seules les clefs présentes sont traitées. Par défaut, la valeur de `add_empty` est `true`.

```
//execution avec cet url
//http://localhost:8000/test.php?prix=5
$args = array(
    'nom'   => FILTER_SANITIZE_STRING,
    'prix'  => array('filter'   => FILTER_SANITIZE_NUMBER_FLOAT
,
        'flags'     => FILTER_FLAG_ALLOW_FRACTION,
    ),
    'ville' => FILTER_SANITIZE_STRING
);

$params = filter_input_array(INPUT_GET,$args, false);

var_dump($params);
/* Retourne un tableau associatif
*  les clefs manquantes sont ignorées
*   array (size=1)
*   'prix' => string '5' (length=1)
*
* avec omission de l'argument ou add_empty à true
* le résultat est le suivant
*
*   array (size=3)
*   'nom' => null
*   'prix' => string '5' (length=1)
*   'ville' => null
*/
```

Vérifier l'existence d'une clef

La fonction `filter_has_var` permet de vérifier l'existence d'une clef dans une superglobale. Son intérêt réside dans sa performance légèrement plus élevée que les fonctions `isset` et `array_key_exists`.

Notons également que `filter_has_var` travaille sur les données transmises et ne prend pas en compte les éventuelles modifications directe des superglobales en cours de script (notons toutefois que modifier les variables superglobales dans nos scripts est une très mauvaise idées).

```
$_GET['inclusion'] = true;

//retourne false
var_dump(filter_has_var(INPUT_GET,'inclusion'));

//retourne true
var_dump(isset($_GET['inclusion']));

//retourne true
var_dump(array_key_exists('inclusion',$_GET));
```

Autres fonctions de filtres

Il est possible d'utiliser les filtres avec d'autres sources que les variables superglobales. Pour cela, nous utiliserons les fonctions `filter_var` et `filter_var_array` qui se comportent comme les fonctions `filter_input` et `filter_input_array` à la différence que leur premier argument est une variable et non une constant représentant une supeglobale.
Grâce à ces fonctions, nous pourrons par exemple facilement valider les arguments passés à nos fonctions et obtenir ainsi un code robuste et lisible.

```
function total($prix, $qt){
    //nous sommes pessimistes,
    //nous considérons que la fonction échoue
    //par défaut
    $succes = false;
    $total = 0;

    //Validation des arguments
```

```
    $prix = filter_var($prix, FILTER_VALIDATE_FLOAT);
    $qt = filter_var($qt, FILTER_VALIDATE_INT);

    //Traitement uniquementsi les arguments sont valides
    //l'opérateur !== teste la non identité à false
    //ceci pour éviter que la valeur 0 soit considérée comme fa
lse
    if($prix !== false && $qt !== false) {
        $total = $prix*$qt;
        $succes = true;
    }

    //retour sous la forme d'un tableau associatif
    //afin de transmettre à la fois le résultat
    //et le succès ou l'échec de la fonction
    return array(
        'succes' => $succes,
        'total'  => $total
    );
}

//Utilisation de la fonction

var_dump(total('a', 'b'));
/*
array (size=2)
  'succes' => boolean false
  'total' => int 0
*/

var_dump(total(5.5, 8));
/*
array (size=2)
  'succes' => boolean true
  'total' => float 44
*/
```

Couplage du nettoyage et de la validation

Pour appliquer à la fois un nettoyage et une validation, il faut passer par la fonction `filter_input`, puis par `filter_var` sur le résultat de la première fonction.

```
$sanitationArgs = array(
    'nom'   => FILTER_SANITIZE_STRING,
```

```
    'prix' => array(
                    'filter'    => FILTER_SANITIZE_NUMBER_FLOAT
,
                    'flags'     => FILTER_FLAG_ALLOW_FRACTION,
    ),
);

$validationArgs = array(
    'prix' => FILTER_VALIDATE_FLOAT,
    'nom' => null //pas de règle de validation à appliquer
);

//Récupération et nettoyage donne un tableau associatif
$params = filter_input_array(INPUT_GET,$sanitationArgs);

//validation sur le résultat de filter_input_array
$params = filter_var_array($params,$validationArgs,true);

var_dump($params);
```

Filtrer les valeurs d'un tableau

Pour filter les valeurs d'un tableau, il faut ajouter la flag `FILTER_REQUIRE_ARRAY`.

```
$notes = array(3,5,9,10,'cinq');

$notes = filter_var(
    $notes,
    FILTER_VALIDATE_INT,
    FILTER_REQUIRE_ARRAY
);

var_dump($notes);
```

L'opposé de ce flag est `FILTER_REQUIRE_SCALAR` qui requiére une variable scalaire (non composite) en entrée et retournera `false` si tel n'est pas le cas.

Les fonctions

Une fonction est un sous programme qui regroupe une série d'instructions, il effectue un traitement et peut éventuellement renvoyer un résultat. l'intérêt des fonctions est multiple :

- Les fonctions **réduisent la duplication du code** puisqu'elle déportent les instruction récurrentes au sein du corps de la fonction qui sera appelée ensuite de multiples fois dans notre programme. Cela facilite la **maintenance** et la **clarté** de notre code.
- Le fonctions permettent de **décomposer un problème complexe en unités distinctes**. Chaque fonction est chargée de résoudre un problème simple, l'ensemble des appels de fonctions permet d'appréhender le problème complexe en faisant abstraction des détails de l'implémentation et donc d'obtenir une vision plus synthétique de la résolution du problème. Cette gestion des niveau d'abstraction est très importante car l'une des difficulté de la programmation réside dans le constant va et vient entre vision macroscopique et vision microscopique. On retrouvera ce même principe développé et étendu dans la programmation orientée objet.
- Les fonctions **améliorent la fiabilité du code**, ceci est un effet de la réutilisation. Si je fais appel à une fonction que j'ai déjà utilisé et testé dans de multiples projet, je possède un plus grand niveau de certitude quant à son bon fonctionnement.
- Enfin, les fonctions, en automatisant les traitements répétitifs **améliorent notre productivité**. Au fil du temps et des projets, nous nous constituons des bibliothèques de fonctions chargées de résoudre des problèmes récurrents et pouvons nous concentrer sur l'essentiel, la résolution des problématiques métier du client.

Déclaration d'une fonction

Pour déclarer une fonction, nous utilisons la syntaxe suivante :

```
function nomDeLaFonction ([argument1, argument2...]){
    instructions...
```

```
}
```

Par exemple ici une fonction chargée d'afficher bonjour

```php
//fonction sans arguments
function bonjour(){
    echo 'bonjour';
}

//Utilisation de la fonction
bonjour();
```

La même fonction peut admettre un argument pour la rendre plus générique.

```php
//fonction sans arguments
function bonjour($nom){
    echo "bonjour .$nom";
}

//utilisation de la fonction
bonjour('Seb');
```

> **Notes :**
> Les noms de fonction suivent les mêmes règles que les noms de variables.
> - le nom doit commencer par une lettre
> - un nom de fonction peut comporter des lettres, des chiffres et les caractères _ et & (les espaces ne sont pas autorisés!)
>
> Un nom de fonction, comme celui des variables est sensible à la casse (différenciation entre les minuscules et majuscules)
> - Les arguments sont facultatifs, mais s'il n'y a pas d'arguments, les parenthèses doivent rester présentes
> - Il ne faut pas oublier de refermer les accolades

Les retours

Une fonction exécute une suite d'instructions, mais elle peut également renvoyer un résultat. C'est d'ailleurs le cas de nombre de fonctions intégrées à

PHP que nous avons déjà utilisées.
Pour définir un retour, il suffit d'utiliser le mot clef `return` suivi de la valeur qui doit être retournée (variable ou littérale).

```php
function lien($texte, $url){
    return "<a href=\"$url\">$texte</a>";
}
```

> **Attention :** l'apparition de `return` met fin à l'exécution de la fonction, tout code se situant après une instruction `return` ne sera donc pas exécuté.
> Généralement, une fonction ne contient qu'un seul retour et ce retour est la dernière instruction de la fonction.

Les retours multiples

Si notre fonction doit retourner plusieurs valeurs, alors il faudra utiliser une structure de données composite telle qu'un tableau ou un objet.

```php
//Cette fonction admet un tableau en argument
//elle retourne les valeurs maximum et minimum
function minMax(array $source){
    //tri du tableau
    sort($source);

    //retourne les premiers et derniers éléments
    return array(
        'min' => $source[0],
        'max' => $source[count($source]-1
        )
    );
}

//appel de la fonction

var_dump(minMax(array(5,8,2)));
```

Les arguments

Nous avons vu qu'une fonction peut définir des arguments qui seront utilisés

dans les instructions de la fonction. Ces arguments sont identifiés par une liste de variables déclarée au début de la fonction. Cette liste d'arguments constitue la signature de la fonction. Lors de l'appel de celle-ci, le nombre et l'ordre des arguments doit correspondre à cette signature.

Les valeurs par défaut

Il est possible de définir des valeurs par défaut pour certains arguments de fonction. Attention toutefois, les arguments possédant des valeurs par défaut doivent impérativement se trouver après les autres arguments dans la signature de la fonction.

```php
function bonjour($nom = 'world') {
    echo "Hello $nom";
}

//appels de la fonction
bonjour(); //affiche Hello world

bonjour('you');//affiche Hello you
```

```php
function bonjour($nom, $lang = 'fr'){
    $hello = $lang=='fr'?'Bonjour':'Hello';
    echo "$hello $nom";
}

//appels de la fonction

bonjour('Seb');//affiche Bonjour Seb

bonjour('Seb','en');//affiche Hello Seb
```

Le typage des arguments

PHP est un langage de programmation faiblement typé. Cela veut dire que nous ne pouvons pas déclarer le type de données attendues pour les arguments d'une fonction, du moins pour les variables scalaires, car cette possibilité existe pour les variables composites telles que les tableaux ou les objets.

```php
function liste(array $data){
    echo '<ul><li>'. implode('</li><li>',$data).'</li></ul>';
}

//appels de la fonction
//retourne une erreur car 'test' n'est pas un tableau
liste('test');

//Ok car l'argument est un tableau
liste(array('poire','pomme','orange');
```

Les arguments nommés

Certains langages tels que Python utilisent des arguments nommés dans les appels de fonction, cette technique est intéressante car elle permet de connaître immédiatement le rôle de chaque argument sans avoir à vérifier la signature de la fonction (pour peu que les noms soient explicites).
Avec PHP, nous n'avons pas cette possibilité, mais nous pouvons nous en approcher en passant en argument un tableau associatif.

```php
function bonjour(array $args = array('nom'=>'Seb','langue'=>'fr'
)){
    $hello = $args['lang']=='fr'?'Bonjour':'Hello';
    echo "$hello ".$args['nom'];
}

//appels de la fonction

bonjour(array('nom' =>'toto', 'lang' => 'en'));
```

Un nombre variable d'arguments

Parfois, il est pratique de définir des fonctions avec un nombre variable d'arguments. Imaginons une fonction qui fasse la somme de tous les nombres passés en argument. Il est clair que nous n'allons pas déclarer l'ensemble des arguments possibles dans la signature de la fonction. Nous disposons de plusieurs solutions pour régler ce problème.

Utiliser un tableau

La solution la plus évident est de passer un tableau en argument, ainsi nous pourrons boucler sur les éléments du tableau pour réaliser la somme.

```php
function somme(array $nombres){
    $total = 0;
    $nbArguments = count($nombres);

    for($i=0; $i < $nbArgument; $i++){
        $total += (int)$nombre[$i];
    }

    echo "la somme est : $total";
}

//appel de la fonction
somme(array(4,8,12));
```

Utiliser les fonctions d'accès aux arguments (PHP <=5.5)

Depuis PHP 4 (c'est à dire la nuit des temps), nous disposons de trois fonctions pratiques pour accéder aux argument d'une fonction.

fonction	description
`func_num_args`	retourne le nombre d'arguments passés à la fonction
`func_get_args`	retourne les arguments sous forme de tableau indicé
`func_get_arg(pos)`	retourne l'argument à la position pos

Ce qui peut donner le code suivant :

```php
function somme(){
    $total = 0;
    $nbArguments = func_num_args;

    for($i=0; $i < $nbArgument; $i++){
        $total += (int)$func_get_arg($i);
    }

    echo "la somme est : $total";
```

```
}

//appel de la fonction
somme(4,8,12);
```

Ou encore avec la fonction `func_get_args` :

```
function somme(){
    $total = 0;

    foreach(func_get_args() as $nombre){
        $total += (int)$nombre;
    }

    echo "la somme est : $total";
}

//appel de la fonction
somme(4,8,12);
```

Utiliser la notation `...` (PHP >= 5.6)

Depuis PHP 5.6, nous disposons d'une notation alternative pour les arguments variables. L'intérêt ici réside dans la possibilité de mixer des arguments variables et d'autres qui ne le sont pas.

```
function somme($type,...$nombres){
    $total = 0;

    foreach($nombres as $nombre){
        $total += (int)$nombre;
    }

    $accord = $total >0? 's':'';

    echo "il y a $total $type$accord";
}

//appel de la fonction
somme('pomme',4,8,12);
```

Bonnes pratiques : tester les entrées

Il est important de tester le type et la valeurs des arguments passés à nos fonction et de prévoir les cas où ces arguments sont incorrects. ici, nous avons deux possibilités, soit nous tentons de convertir les arguments pour que la fonction effectue son traitement, soit nous retournons une erreur. Nous pouvons également mixer les deux solutions

```php
function total($prixUnitaire, $qt){
    //conversion des arguments
    $prixUnitaire = (float)abs($prixUnitaire);
    $qt = (int)abs($qt);

    echo 'le total est de : '.($prixUnitaire * $qt).'<br>';
}

total('a','b');//affiche 0

total('5', '3');//affiche 15

total(-5, 3);//affiche 15
```

```php
function total($prixUnitaire, $qt){
    //conversion des arguments
    $prixUnitaire = (float)$prixUnitaire;
    $qt = (int)$qt;

    //test de validité
    if($prixUnitaire <= 0){
        echo "Le prix unitaire doit être un nombre positif <br>"
;
    } else if ($qt <= 0){
        echo "La quantité doit être un entier positif <br>";
    } else {
        echo 'le total est de : '.($prixUnitaire * $qt).'<br>';
    }
}

total('a','b');//affiche erreur

total('5', '3');//affiche 15

total(-5, 3);//affiche erreur
```

Le passage des arguments par référence

Par défaut, les arguments d'une fonction sont passés par valeur. Dans le cas de valeurs littérales, cela n'à pas d'importance, mais si l'on passe une variable en argument, cela implique que la valeur de cette variable sera copiée et donc que la variable d'origine ne sera pas affectée par les éventuelles modification opérées au sein du corps de la fonction.
Ce comportement est plutôt intéressant puisqu'il évite les effets de bord, c'est à dire la modification de valeurs en dehors de la fonction.

Cependant, cette sécurité à un coût, en effet, copier les valeurs des arguments peut avoir un impact négatif sur la consommation de mémoire dès lors que ces valeurs sont des tableaux contenant de nombreuses données. On peut donc voir sur le net de nombreux articles qui préconisent de passer les tableaux par référence en PHP. **Ceci est une mauvaise idée**, en effet, le moteur PHP utilise une technique appelée **"copy on write"** qui optimise le passage des arguments par valeur et rend cette opération au moins aussi performante si ce n'est plus que le passage d'arguments par référence. **Ceci à condition que nous ne tentions pas de modifier la valeur des arguments dans le corps de la fonction**.

> **Note :** Les objets sont toujours passés par référence, car la copie profonde d'un objet est une opération non triviale qui peut être très consommatrice de mémoire.

Pour passer un argument par référence, il suffit de faire précéder la déclaration de l'argument du caractère `&` dans la signature de la fonction. A ce moment, l'argument ne sera plus copié. La fonction php `sort` est un bon exemple de fonction utilisant le passage d'argument par référence.

```php
$entier = 2;

function incremente(&$nombre){
    $nombre ++;
    return $nombre;
}

var_dump(incremente($nombre)); //retourne 3

//La variable $nombre étant passée par référence,
```

```
//elle est modifiée par la fonction
var_dump($nombre)//retourne 3
```

De manière générale, il faut considérer les fonctions comme des unités de code indépendantes de leur environnement qui reçoivent des données, effectuent un traitement et retournent un résultat sans référence à un contexte plus global. En faisant cela, nous obtenons des fonctions plus réutilisables, maintenables et testables de façon unitaire. Le passage d'arguments par référence introduit une modification de ce contexte global et doit donc être envisagé avec la plus grande prudence.

La portée des variables

La portée d'une variable indique son niveau d'accessibilité et dépend du contexte de sa déclaration. Une variable déclarée dans un script PHP sera accessible à l'ensemble de ce script. En revanche, une variable déclarée dans le corps d'une fonction ne sera disponible que tant dans le contexte d'exécution de cette fonction, et non pour le reste du script. De même, une variable globale ne sera pas disponible au sein d'une fonction (à moins qu'elle ne soit passée en argument auquel cas elle devient locale pour cette fonction).

```
$globale = 5;

function test(){
    //La variable $globale n'est pas disponible
    echo $globale;
    $locale = 8;
}

test();

//La variable $locale n'est pas disponible
echo $locale;
```

Pour rendre disponible une variable globale dans une fonction, nous utiliserons le mot clef `global` suivi du nom de la variable que nous souhaitons obtenir. De même, ce mot clef nous servira à rendre globalement disponible une variable déclarée localement dans une fonction.

```php
$globale = 5;

function test(){
    global $globale, $locale;
    //La variable $globale est disponible
    echo $globale;// affiche 5

    //réaffectation de $globale
    $globale = 8;
    $locale = 8;
}

test();

//La variable $locale est disponible
echo $locale;//affiche 8

echo $globale;//affiche 8
```

Cette gymnastique est plaisante, mais elle peut induire des effets de bords ainsi qu'une dépendance entre nos fonctions et le contexte global. Ces deux phénomènes sont généralement à éviter si nous souhaitons pouvoir tester et réutiliser nos fonctions. Un des intérêt des fonctions consiste à isoler un traitement spécifique afin de simplifier la résolution d'un problème complexe en le divisant en plusieurs problèmes simples. Cette isolation nous permet également de tester et corriger le code en ne prenant en compte qu'un niveau d'abstraction, celui de la fonction. En ajoutant des variables globales, le déboguage et le test deviennent plus complexes puisqu'il nous faut prendre en compte deux contextes différents et deux niveaux d'abstraction.

Utilisation avancées des fonctions

Tester l'existence d'une fonction

La fonction `function_exists` retoune un booléen indiquant la disponibilité d'une fonction dont le nom est passé en argument. Cela permet de tester l'existence d'une fonction à la fois parmi les fonctions personnalisées et les fonctions internes de PHP. Une utilisation fréquente consiste à tester la disponibilité d'une nouvelle fonction uniquement utilisable avec les dernières

versions de PHP. Si le test échoue, nous définissons alors notre propre implémentation de cette fonction.

```
    /*
     * Redéfinition de la fonction array_column
     * pour les versions de php antérieures à 5.5
     * array_column retourne un tableau contenant
     * toutes les valeurs d'une clé d'un tableau associatif
     */
    if( !function_exists( 'array_column' ) ){

        function array_column( array $input, $column_key, $inde
x_key = null ) {
            $result = array();
            foreach( $input as $k => $v ){
                //Définition de la clef à utiliser pour le résu
ltat
                $resultKey = $index_key ? $v[ $index_key ] : $k
;
                $result[$resultKey] = $v[$column_key];
            }

            return $result;
        }
    }
```

Fonctions dynamiques

PHP permet l'utilisation d'une variable pour invoquer une fonction. Dans ce cas, le nom de la fonction appelée sera la valeur de la variable. Cette astuce nous permet de réaliser des appels de fonction dynamiques qui dépendent du contexte.

```
function printPDF(){
    echo 'impression PDF';
}

function printTXT(){
    echo 'impression texte';
}

$printType = 'TXT';
$printFunction = 'impression_'.$printType;
```

```php
if(function_exists($function)){
    $printFunction();
}else {
    echo "la fonction $printFunction n'existe pas";
}
```

Les en-têtes http

PHP peut envoyer un en-tête http au navigateur avec la fonction `header`. Cette fonction admet trois arguments :

- l'en-tête http sous la forme d'une chaîne de catactère
- Un booléen indiquent si l'en-tête doit replacer les en-têtes existant. Cet argument vaut true par défaut. On utilisera false pour transmettre plusieurs en-têtes.
- Un entier représentant le code de réponse http, cet argument est optionnel.

Un résumé de la spécification

Le protocole http 1.1 est décrit de façon exhaustive dans un document, la RFC2616 dont une traduction en françias se trouve ici. Ce document technique d'une centaine page n'est pas d'une lecture très digeste. Voici donc un résumé sommaire des principaux codes et en-têtes les plus utilisés.

> Attention : la fonction header ne permet de générer que des en-têtes de réponse du serveur http. Il n'est pas possible de générer des en-têtes de requête et donc d'envoyer des données en POST par ce biais.

Les en-têtes de réponse http

Nom de l'en-tête	Description
Content-Encoding	Type de codage du corps de la réponse
Content-Language	Type de langage du corps de la réponse
Content-Length	Longueur du corps de la réponse

Content-Type	Type de contenu du corps de la réponse (par exemple text/html). Voir types MIME
Date	Date de début de transfert des données
Expires	Date limite de consommation des données
Location	Redirection vers une nouvelle URL associée au document

Les codes de status http

codes	description
200-299	succès
300-399	redirection
400-499	erreur du client ou du serveur web
500-599	erreur interne du serveur

Les succès

- 200 – OK : La requête demandée a été traitée avec succès. C'est ce qui est retourné 90% du temps, le comportement normal de l'application qui affiche un contenu correct.

Les redirections

- 301 – Moved Permanently : La ressource demandée a définitivement été déplacée à une autre URL. Le client devrait ainsi mettre à jour l'emplacement de cette ressource pour ne plus la rechercher à l'emplacement courant.
- 302 – Found (HTTP 1.0) : La ressource demandée est accessible temporairement depuis une autre URL. En ce sens, la redirection étant temporaire, le client devrait continuer de solliciter la ressource depuis l'URL qu'il a demandé.
- 303 – See Other (HTTP 1.1) : La réponse pour la ressource demandée est accessible à une autre URL. L'adresse qu'il faut continuer de solliciter pour la ressource est bien celle demandée par le client, mais la réponse est disponible ailleurs, par l'intermédiaire d'une requête GET. Cette redirection

est typiquement recommandée après un POST, si votre site veut indiquer la page de réponse à un autre endroit (RFC 2616).
- 307 – Temporary Redirect (HTTP 1.1) : La ressource demandée est accessible temporairement depuis une autre URL. Pour accéder à cette URL temporaire, le client DOIT soumettre sa requête avec la même méthode qu'il l'a déjà fait (refaire une requête POST si la demande initiale était en POST, PUT pour du PUT, …). A l'avenir, la redirection étant temporaire, le client devrait continuer de solliciter la ressource à l'URL demandée initialement.

302, 303 et 307, quelles différences ?

Le code 302, à son origine, aurait du entraîner les clients à conserver la méthode d'interrogation utilisée au départ (au même titre que le code 307). Dans les faits, certains outils répandus faisaient cette redirection par l'intermédiaire d'une requête de type GET, quel que soit la méthode originelle d'interrogation (POST PUT ou DELETE).

Les codes 303 et 307 sont donc apparus pour lever l'ambiguïté et demander au client de conserver (307) ou pas (303) la méthode d'interrogation initiale de la ressource à son emplacement temporaire.

Les erreurs coté client

- 400 – Bad Request : La syntaxe de la requête semble incorrecte. Pour ma part, je l'utilise par exemple en cas de paramètre (GET ou POST) manquant ou incorrect, comme le fait aussi OAuth.
- 401 – Unauthorized : L'erreur 401 (je ne sais pas qui vous êtes) indique qu'une authentification est nécessaire afin d'accéder à la ressource, et le client est invité à s'authentifier (correctement) s'il veut accéder à la ressource.
- 403 – Forbidden : (Je sais qui vous êtes, vous ne pouvez pas) On indique, comme pour l'erreur 401, qu'une autorisation est nécessaire pour pouvoir accéder à la ressource, et que le niveau de droit courant n'est pas suffisant.
- 404 – Not Found : La ressource est introuvable.
- 410 – Gone : Petite soeur de la page 404, on indique ici que la ressource n'est « plus » disponible à cette adresse (définitivement ou non) sans que le serveur soit en mesure de donner une nouvelle adresse.

- 405 – Method Not Allowed : Le client tente d'accéder à la ressource via une méthode (GET / POST / PUT / DELETE) non autorisée pour cette dernière. La réponse devrait contenir la liste des méthodes autorisées pour la ressource.

Erreurs serveurs

- 500 – Internal Server Error : ça plante, mais pourquoi… ?
- 501 – Not Implemented : Le serveur ne sait pas traiter la demande.
- 503 – Service Unavailable : Le site est en maintenance ou indisponible de façon temporaire.

Opérations fréquentes avec les en-têtes

Redirection

Envoyer un en-tête de redirection est une opération assez courrante. Elle permet de renvoyer vers une page d'erreur en cours de script ou d'éviter la soumission multiple de formulaire en redirigeant vers une autre page après le traitement.

L'en-tête utilisé est `location` suivi de `:` et de l'adresse de redirection.

```php
//Redirection relative vers un fichier page.php
//se trouvant dans le même dossier
$redirectTarget = "page.php";
header("location:$redirectTarget");

//Redirection relative vers un fichier page.php
//se trouvant un dossier au dessus
$redirectTarget = "../page.php";
header("location:$redirectTarget");

//Redirection absolue vers un fichier page.php
//se trouvant dans le dossier racine du serveur web
//(DOCUMENT_ROOT)
$redirectTarget = "/page.php";
header("location:$redirectTarget");

//Redirection adresse absolue
//vers mon site
```

```php
$redirectTarget = "http:/www.monsite.com/page.php";
header("location:$redirectTarget");

//Redirection adresse absolue
//vers un autre site
$redirectTarget = "http:/www.google.fr";
header("location:$redirectTarget");
```

> **Note :** Il est possible d'ajouter des paramètres à l'URL sous la forme d'une chaîne de requête.
> `header("location:page.php?var1=2&var2=5`

Forcer le téléchargement

Lorsque nous réalisons un lien vers un fichier dont le mimeType est reconnu par le navigateur, ce dernier utilise l'action associé à ce mimeType pour traiter le document. Dans le cas d'un fichier au format PDF, cela se traduit généralement par l'affichage de ce document dans la fenêtre du navigateur. Ce comportement peut parfaitement convenir, mais il peut également arriver que nous souhaitions proposer le téléchargement deu fichier plutôt que son affichage.

```php
$pdfFileName = 'rapport.pdf';
$pdfSourceFile = 'source-rapport.pdf';

header("Content-type:application/pdf");

// filename permet de suggérer un nom de fichier
header("Content-Disposition:attachment;filename='$pdfFileName'"
);

// Lecture du fichier source
readfile($pdfSourceFile);
```

Un autre cas très fréquent consiste à forcer le téléchargement de contenu de type texte CSV ou XML.

```php
$fileName = 'participants.csv';

header("Content-type:application/csv");
```

```php
// filename permet de suggérer un nom de fichier
header("Content-Disposition:attachment;filename='$fileName'");

// Affichage du contenu
echo "nom;age\n";
echo "Pierre;20\n";
echo "Jean;32\n";
echo "Odile;40\n";
```

Déterminer le type de contenu

La plupart du temps, nos pages PHP produiront du contenu html, mais il peut arriver que tel ne soit pas le cas. Dans le cadre du développement d'une application javascript utilisant Ajax, nous pourrions avoir à produire des pages retournant un contenu Json par exemple.

```php
header('Content-Type: application/json');
echo '{"name":"Maloron","lastname":"S\u00e9bastien","age":87}';
```

Pour nous simplifier la vie et éviter d'avoir à encoder nous même les caractères accentués, nous pouvons utiliser la fonction `json_encode`.

```php
header('Content-Type: application/json');
$user = array(
    'name' => 'Maloron',
    'lastname' => 'Sébastien',
    'age' => 87
);

echo json_encode($user);
```

Les cookies

Un cookie est un fichier texte enregistré par le navigateur sur le disque dur de l'internaute. Ce fichier stocke des informations sous la forme d'une paire clef/valeur de la même façon qu'un tableau associatif à une case.

PHP peut demander au navigateur la création ou la modification d'un cookie par le biais d'un en-tête http `set-cookie`. Par la suite, les requêtes http entre le navigateur et le serveur web transmettront le cookie par le biais d'un en-tête `cookie`, ce qui permet à nos scripts PHP de récupérer les informations.

Par mesure de sécurité et pour éviter l'envoi de données inutiles, l'envoi de cookies est limité par le domaine, seul le domaine qui a crée le cookie recevra les informations de ce dernier dans les requêtes http. Ainsi, si un navigateur possède deux cookies, l'un créé par le domaine `wwww.sncf.fr` et l'autre par `www.monsite.com`, chacun de ces sites ne recevra que les cookies associés à son domaine.

Par défaut, les cookies sont supprimés lors de la fermeture du navigateur web. Il est cependant possible de leur attribuer une durée de vie, ce qui permattra de les récupérer d'une session de navigation à l'autre.

Création d'un cookie

Pour demander la création d'un cookie, nous utiliserons la fonction `setcookie` qui admet les arguments suivants :

argument	description
name	nom du cookie, correspond à la clef
value	valeur du cookie
expire	temps après lequel le cookie doit être détruit (timestamp UNIX)
path	chemin sur le serveur associé au cookie

domain	domaine associé au cookie
secure	indique si le cookie est limité au requêtes https
httponly	si true, le cookie ne sera pas disponible pour les langages de script côté client (javascript)

Seul le premier argument `name` est obligatoire, tous les autres sont facultatifs. Cependant, créer un cookie sans valeur, juste avec l'argument `name` aura pour effet de supprimer ce cookie.

La fonction `setcookie` renvoie un booléen indiquant le succès ou l'échec de l'opération.

Création d'un cookie temporaire

```
//Le cookie sera détruit à la fermeture du navigateur
setcookie('visiteur','seb');
```

Création d'un cookie persistant

```
//Le cookie expirera dans 1 heure
setcookie('visiteur','seb', time()+36000);

//le cookie expirera dans 7 jours
setcookie('couleur','#FF0000', time()+(60*60*24*7));
```

Enregistrement d'un tableau

Il est possible d'utiliser la notation des tableaux dans les nomns de cookies, cela générera autant de cookies qu'il y a de clef, mais ces cookies seront ensuite récupérés comme un tableau associatif.

```
setcookie('couleur[fond]','#CCCCEE', time()+(60*60*24*7));
setcookie('couleur[texte]','#442255', time()+(60*60*24*7));
setcookie('couleur[titre]','#FF0000', time()+(60*60*24*7));
```

Lecture d'un cookie

Le récupération des valeurs des cookies est très simple, il suffit d'interroger la superglobale `$_COOKIE`. Attention toutefois à une erreur fréquente, les cookies générés par un script ne seront envoyés qu'avec la prochaine requête http. Ils ne sont donc pas encore disponibles en tant que cookie lors de leur création.

page de création des cookies

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Création de cookies</title>
</head>
<body>
<?php
setcookie('couleurs[fond]','#CCCCEE', time()+(60*60*24*7));
setcookie('couleurs[texte]','#442255', time()+(60*60*24*7));
setcookie('couleurs[titre]','#FF0000', time()+(60*60*24*7));

setcookie('visiteur', 'seb');
?>

<a href="affichage.php">tester les cookies</a>

</body>
</html>
```

Utilisation des cookies

Affichage du nom du visiteur et utilisation de ses couleurs préférées.

```
<?php
$couleurDefaut = array(
    'titre' => '#FF0000',
    'texte' => '#000000',
    'fond'  => '#FFFFFF'
);
```

```php
if (! filter_has_var(INPUT_COOKIE,'couleurs')){
    $couleur = $couleurDefaut;
} else {
    $couleurs = filter_input(INPUT_COOKIE,'couleurs', FILTER_SA
NITIZE_STRING,FILTER_REQUIRE_ARRAY);
}
$visiteur = filter_input(INPUT_COOKIE,'visiteur',FILTER_SANITIZ
E_STRING);
?>

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Utilisation des cookies</title>

    <style>
        h1 {
            color: <?php echo $couleurs['titre']; ?>;
        }

        body {
            background-color: <?php echo $couleurs['fond']; ?>;
            color: <?php echo $couleurs['texte']; ?>;
        }

    </style>

</head>
<body>

<h1>Bonjour <?php echo $visiteur; ?></h1>

<p>Ceci est un test de cookies</p>
</body>
</html>
```

Limites des cookies

Le nombre et la taille des cookies est limité par le navigateur. d'un éditeur à l'autre ces limites sont différentes, il est donc bon de se limiter au moins disant soit :

- 30 cookies par domaine

- 4095 octet par cookie

Même sans atteindre ces bornes, il est bon de se rappeler que les cookies sont transmis dans toutes les requêtes http et qu'une quantité de données importante aura donc un impact sur la consommation de bande passante de notre site ce qui se traduira peut être par une lenteur d'affichage des pages ou une sur-facturation de notre hébergement.

En outre, les cookies étant stockés sur le disque dur de l'internaute, ils peuvent être falsifiés ou récupérés par un développeur malveillant. Il font donc prendre garde à ne pas stocker d'informations sensibles dans un cookie d'une part et d'autre part toujours valider et nettoyer la valeur des cookies récupérés.

A quoi servent les cookies

Les deux principales utilisations des cookies sont les suivantes :

- faire des statistiques sur la fréquentation d'une site (nombre, fréquence et durée des visites)
- stocker les préférences d'un utilisateur

Les sessions

Une session est un espace de stockage de données temporaire qui permet de conserver des informations d'une requête http à l'autre. En effet, le protocole http est dit sans état, ce qui implique que le serveur n'a aucune connaissance des requêtes précédentes lorsqu'il traite la requête en cours. Les sessions offrent un moyen de conserver un état entre deux requêtes provenant du même navigateur.

Pour ce faire, une session enregistre un identifiant de session dans un cookie. Cet identifiant est transmis dans l'en-tête http et permet au serveur web de récupérer les informations correspondant à cet identifiant. Il est également possible de transmettre cet identifiant de session dans l'URL sous la forme d'un paramètre du QueryString.

A première vue, on pourrait penser que cette dernière méthode n'est guère sécurisée et on aurait sans doute raison. Même si la transmission via un cookie

n'est pas beaucoup plus sûre, la transmission de l'indentifiant de session dans l'URL implique des vulnérabilités supplémentaires. Outre le fait que l'URL soit visible lorsque l'internaute se trouve sur notre site, il est également récupérable via la superglobale `$_SERVER[‘HTTP_REFERER’] qui indique l'adresse de la page d'où nous venons. Il est donc possible par ce biais de récupérer un identifiant de session d'un autre site, ce qui constitue une faille de sécurité. Enfin, la possibilité de mettre en signet un URL contenant un identifiant de session peut également poser quelques problèmes.

Session utilisation simple

Pour initialiser une session, nous utilisons la commande `session_start`. Ensuite, la superglobale `$_SESSION` nous donne accès à l'ensemble des variables de sessions.

Création d'une varaible de session dans une page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Création de session</title>
</head>
<body>
<?php
//Initialisation de la session
session_start();

//initialisation et affectation
//d'une variable de session userName
$_SESSION['userName'] = 'seb';
?>

<a href="affichage.php">tester les sessions</a>

</body>
</html>
```

Récupération de la session dans une autre page

```
<?php
```

```php
//Initialisation de la session
session_start();

//Récupération et nettoyage de la variable de session
//Attention, filter_input(INPUT_SESSION...) n'est pas encore im
plémenté (PHP 5.6)
if(isset($_SESSION['userName'])){
    $userName = filter_var($_SESSION['userName'],FILTER_SANITIZ
E_STRING);
} else {
    $userName = 'Inconnu';
}
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Utilisation des sessions</title>

</head>
<body>
<h1>Bonjour <?php echo $userName; ?></h1>
<p>Ceci est un test de sessions</p>
</body>
</html>
```

Durée de vie d'une session

Il convient de distinguer deux choses, la session d'une part qui est enregistrée sur le serveur et d'autre part le cookie qui contient l'identifiant de session et permet de récupérer cette dernières.

Concernant le cookie, sa durée de vie est déterminée par la configuration du serveur web et peut être modifiée par du code PHP. Assez souvent, les serveurs utilisent une durée de vie égale à zéro, ce qui implique que le cookie est automatiquement détruit à la fermeture du navigateur.

Concernant les données liées à l'identifiant de session, elles sont détruites automatiquement par un ramasse miettes (garbage collector) qui fait régulièrement le ménage sur le serveur. Cependant, cette opération consomme des ressources, elle est donc effectuée de manière sporadique. Il est donc très fréquent que la session sur le serveur survive à son cookie. A priori, cela ne

pose pas de problème, puisque le cookie contient l'identifiant qui seul peut donner accès à la session. Certes, mais cet identifiant aurait pu être volé et une tierce personne pourrait ainsi accèder à une session qui ne lui appartiens pas.

Modifier la durée de vie du cookie

Pour modifier la durée de vie des cookies de session, nous avons plusieurs solutions.

- Au niveau global, nous pouvons modifier le fichier `php.ini` et préciser le paramètre `session_cookie_lifetime` qui indique la durée de vie du cookie en secondes.
- Au niveau du script php, il est possible de modifier la configuration de PHP avec la fonction `ini_set`.
- Enfin, la fonction `session_set_cookie_params` permet de modifier les paramètres des cookies de session dans un script PHP. Le premier argument de cette fonction est la durée de vie du cookie en secondes, les autres arguments facultatifs sont identiques aus 4 derniers arguments de la fonction `setcookie` soit le chemin, le domaine, et la limitation aux seules connexion https ainsi que l'inaccessibilité du cookie pour les langages côté client.

Configuration dans php.ini

Cette configuration est valable pour tous les sites hébergés sur le serveur web.

```
#Dans php.ini
#durée de vie du cookie 1 heure
session_cookie_lifetime = 3600
```

Configuration dans un script PHP

Cette configuration ne sera valable que durant l'exécution du script en cours.

```
//Durée de vie du cookie 1 heurs
ini_set('session_cookie_life_time',3600);
```

```php
//Durée de vie du cookie 10 minutes
session_set_cookie_params(600);

//Le cookie de session
//      - a une durée de vie de 10 minutes
//      - est accessible pour tous les dossiers du domaine
//      - est lié au domaine www.monsite.com
//      - est accessible par http et https
//      - n'est pas lisible par javascript
session_set_cookie_param(
    600,
    '/',
    'www.monsite.com',
    false,
    true
);

//Initialisation de la session et envoie du cookie au navigateur

session_start();
```

Configuration du ramasse miettes

Le ramasse miettes ou garbage collector est chargé de faire le ménage, il collecte les références orphelines et les détruit pour libérer de la mémoire. Le rammase miette des sessions peut être configuré dans le fichier `php.ini`, nous disposons des options suivantes :

session_gc_maxlifetime

Indique la durée de vie de la session en secondes. Au dela de cette durée, la session sera considérée comme expirée et sera détruite à la prochaine invocation du rammasse miette. La valeur par défaut est 1440 secondes soit 24 minutes.

session_gc_probabilty et session_gc_divisor

Indique la probabilité d'invocation du ramasse miette. Cette probabilité est testée à chaque requête http. Ces deux paramètres sont des entiers, la formule `session_gc_probabilty / session_gc_divisor` indique la probabilité d'invocation. Avec les valeurs par défaut, respectivement 1 et 100 il y a donc 1% de chance d'invoquer le ramasse miette à chaque requête. Cette

probabilité peut paraître très faible, mais il faut prendre en compte qu'un serveur web peut traiter de nombreuses requêtes par secondes (des dizaines voire des centaines ou des milliers selon les configurations et le traffic). Dans ces conditions, invoquer le ramasse miette en moyenne une fois toutes les 100 requêtes n'est pas si déraisonable.

Exemples d'utilisation des sessions

L'authentification

Les messages flash

La sécurité des sessions

Le principal problème de sécurité concernant les sessions concerne les identifiants de session qui peuvent être récupérés et utilisés par un tiers. Pour rendre plus difficile ce type d'attaque nous pouvons adopter deux stratégies.

- Regénérer un identifiant de session à interval régulier et au minimum à chaque fois que l'utilisateur change de statut (connexion, déconnexion, attribution de droits). De cette manière, le vol d'un identifiant de session aura moins de chance de fonctionner puisque cet identifiant deviendra rapidement caduq.
- Ajouter une variable de contrôle basée sur une information du client connecté et une clef privée au niveau du serveur. Cette varaible de contrôle permettra de valider la session.

Upload

L'upload, ou téléversement consiste à récupérer un fichier envoyé par l'internaute sur notre serveur. Pour cela, il faut utiliser un contrôle html de type `file`. Il faut également ajouter une propriété `enctype` à la balise `form` et lui attribuer la valeur `multipart/form-data` afin de préciser que le formulaire envoie également des données binaires.

Le formulaire

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Upload</title>
</head>
<body>

<form method="post" action="upload.php" enctype="multipart/form
-data">
    <input type="file" name="fichier">
    <button type="submit">Envoyer</button>
</form>

</body>
</html>
```

Récupération et traitement du fichier

Les fichiers envoyés via un formulaire sont référencés dans une superglobale `$_FILES`. C'est donc ce tableau que nous interrogerons pour traiter le téléversement.

```
var_dump($_FILES);
/*
array (size=1)
  'fichier' =>
    array (size=5)
```

```
      'name'        => string 'Four_gaz_4.jpg' (length=14)
      'type'        => string 'image/jpeg' (length=10)
      'tmp_name'    => string '/private/var/folders/
                        80/1n9sjk8j6gvct1n0fn0fxzsh0000gn/T/php
gqktDC' (length=66)
      'error'       => int 0
      'size'        => int 76745
*/
```

Cette superglobale se présente comme un tableau associatif à deux dimensions. Le première dimension référence le fichier transmis, la clef correspond à l'attibut name de la balise `input`. La deuxième dimension correspond aux attributs du fichier transmis.

clef	description
name	nom d'origine du fichier
type	type mime du fichier
tmp_name	nom et chemin du fichier temporaire sur le serveur
error	code d'erreur, 0=succès
size	taille du fichier en octet

Le traitement de l'upload s'effectuera en deux temps, d'abord la vérification de la taille et du type du fichier. Ensuite l'attribution d'un nom unique et le déplacement du fichier dans un dossier adhoc.
La fonction `move_uploaded_file` réalise cette opération elle adement deux arguments, la source (le fichier temporaire) et la cible (la destination du déplacement).

```php
$ok = true;

//Liste des types de fichiers autorisés
$allowedFileTypes = array(
    'jpg' => 'image/jpeg',
    'png' => 'image/png',
    'gif' => 'image/gif',
);

//Récupération des infos sur le fichier
```

```php
//téléversé
if(! isset($_FILES['fichier'])){
    $message = "Aucun fichier à traiter";
    $ok = false;
} else {
    $upload = $_FILES['fichier'];
}

//Contrôle des erreurs
if($ok){
    switch ($upload['error']) {
        case UPLOAD_ERR_OK:
            break;
        case UPLOAD_ERR_NO_FILE:
            $message = "Aucun fichier à traiter";
            $ok = false;
        case UPLOAD_ERR_INI_SIZE:
        case UPLOAD_ERR_FORM_SIZE:
        $message = "Le fichier est trop lourd";
        $ok = false;
        default:
            $message = "Erreur de type inconnu";
            $ok = false;
    }
}

//Contrôle de la taille du fichier
//au cas où l'on souhaiterai une limite
//différente de celle spécifiée dans
//php.ini
if ($ok && $upload['size'] > 1000000) {
    $message = "Le fichier est trop lourd";
    $ok = false;
}

//Contrôle du type de fichier
//la clef mimeType pouvant être falsifiée
//on utilise la fonction finfo pour récupérer
//le type du fichier
if($ok){
    $finfo = finfo_open(FILEINFO_MIME_TYPE);
    $mimeType = finfo_file($finfo, $upload['tmp_name']);

    //recherche dans la liste des types de fichiers autorisés
    //et récupération de l'extension
    $ext = array_search($mimeType,$allowedFileTypes,true);
```

```
    if($ext === false){
        $message = "Le type de fichier est incorrect";
        $ok = false;
    }
}

//Déplacement du fichier dans le dossier cible
if($ok){
    $targetPath = getcwd().'/images/';
    //Attribution d'un nom unique au fichier
    $filePath = $targetPath.uniqid('image_').'.'.$ext;

    if(!move_uploaded_file($upload['tmp_name'], $filePath)){
        $message = "Impossible de déplacer le fichier temporaia
re";
        $ok = false;
    } else {
        $message = "Upload réussi";
    }
}

echo $message;
```

Upload multiples

Pour transmettre plusieurs fichiers à la fois, il faut ajouter l'attribut `multiple` à la balise `input`. Il faut également que la variable transmise soit un tableau et donc qu'elle se terminent par `[]`.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>Upload</title>
</head>
<body>

<form   method="post" action="upload.php"
            multiple enctype="multipart/form-data">
    <input type="file" name="fichiers[]">
    <button type="submit">Envoyer</button>
</form>

</body>
```

```
</html>
```

La structure du tableau `$_FILES` dans le cas d'upload multiples est un peu particulière. Les clefs sont conservées, mais chacune de ces clefs contient un tableau indicé contenant les valeurs pour l'ensemble des fichiers.

```
//var_dump de la transmission de deux fichiers
array (size=1)
  'fichiers' =>
    array (size=5)
      'name' =>
        array (size=2)
          0 => string 'ILTQq.png' (length=9)
          1 => string 'Tutorial.pdf' (length=12)
      'type' =>
        array (size=2)
          0 => string 'image/png' (length=9)
          1 => string 'application/x-download' (length=22)
      'tmp_name' =>
        array (size=2)
          0 => string '/private/var/folders/80/
          1n9sjk8j6gvct1n0fn0fxzsh0000gn/T/phpweagVT' (length=66
)
          1 => string '/private/var/folders/80/
          1n9sjk8j6gvct1n0fn0fxzsh0000gn/T/phpho6dUB' (length=66
)
      'error' =>
        array (size=2)
          0 => int 0
          1 => int 0
      'size' =>
        array (size=2)
          0 => int 41236
          1 => int 951133
```

Le traitement de multiples upload peut être réalisé de la façon suivante. Ici, nous définissons une fonction chargée de traiter chaque fichier séparément puis nous exécutons cette fonction dans une boucle pour traiter l'ensemble des fichiers.

```
//fonction de traitement d'un upload
//dans le contexte d'upload multiples
//$uploadedVarName correspond à la propriété name de l'input
```

```php
//$uploadNumber correspond à l'indice du fichier à traiter
function upload($uploadVarName, $uploadNumber, $allowedFileTypes
, $maxFileSize = 10000)
{
    $ok = true;
    $upload = array ();
    $message = '';

//Récupération des infos sur le fichier
//téléversé
    if (!isset($_FILES[$uploadVarName])) {
        $message = "Aucun fichier à traiter";
        $ok = false;
    } elseif (!isset($_FILES[$uploadVarName]['name'][$uploadNum
ber])) {
        $message = "Le fichier numéro " . ($uploadNumber + 1) .
" n'existe pas";
        $ok = false;

    } else {
        //récupération des données du fichier à traiter
        //dans un tableau $upload
        foreach ($_FILES[$uploadVarName] as $key => $val) {
            $upload[$key] = $val[$uploadNumber];
        }
    }

//Contrôle des erreurs
    if ($ok) {
        switch ($upload['error']) {
            case UPLOAD_ERR_OK:
                break;
            case UPLOAD_ERR_NO_FILE:
                $message = "Aucun fichier à traiter";
                $ok = false;
            case UPLOAD_ERR_INI_SIZE:
            case UPLOAD_ERR_FORM_SIZE:
                $message = "Le fichier est trop lourd";
                $ok = false;
            default:
                $message = "Erreur de type inconnu";
                $ok = false;
        }
    }

//Contrôle de la taille du fichier
//au cas où l'on souhaiterai une limite
```

```php
//différente de celle spécifiée dans
//php.ini
    if ($ok && $upload['size'] > $maxFileSize) {
        $message = "Le fichier est trop lourd";
        $ok = false;
    }

//Contrôle du type de fichier
//la clef type pouvant être falsifiée
//on utilise la fonction finfo pour récupérer
//le type du fichier
    if ($ok) {
        $finfo = finfo_open(FILEINFO_MIME_TYPE);
        $mimeType = finfo_file($finfo, $upload['tmp_name']);

        //recherche dans la liste des types de fichiers autoris
és
        //et récupération de l'extension
        $ext = array_search($mimeType, $allowedFileTypes, true)
;

        if ($ext === false) {
            $message = "Le type de fichier est incorrect";
            $ok = false;
        }
    }

//Déplacement du fichier dans le dossier cible
    if ($ok) {
        $targetPath = getcwd() . '/images/';
        $filePath = $targetPath . uniqid('image_') . '.' . $ext
;

        if (!move_uploaded_file($upload['tmp_name'], $filePath)
) {
            $message = "Impossible de déplacer le fichier tempo
raiare";
            $ok = false;
        } else {
            $message = "Upload du fichier " . $upload['name'] .
" réussi";
        }
    }

    //message final
    $message = 'Fichier : ' . $upload['name'] . ' : ' . $message
;
```

```
    return array (
        'success' => $ok,
        'message' => $message
    );
}

//========================================
//Utilisation de la fonction
//========================================
//Liste des types de fichiers autorisés
$allowedFileTypes = array (
    'jpg' => 'image/jpeg',
    'png' => 'image/png',
    'gif' => 'image/gif',
);

if (!isset($_FILES['fichiers'])) {
    $message = "Aucun fichier à traiter";
} else {
    //Récupération du nombre e fichiers à traiter
    $numberOfUploadedFiles = count($_FILES['fichiers']['name'])
;

    //Boucle sur l'ensemble des fichiers
    //et traitement de chacun d'eux
    for ($i = 0; $i < $numberOfUploadedFiles; $i++) {
        $result = upload('fichiers', $i, $allowedFileTypes, 500
00);
        echo $result['message'] . '<br>';
    }
}
```

Configuration

L'upload peut être configuré dans le fichier `php.ini`.

configuration	description
`file_uploads`	boolean : active ou désactive l'upload (1 actif par défaut)
`upload_tmp_dir`	string : chemin du dossier temporaire

`upload_max_filesize`	int : taille maximale d'un fichier en octet (2Mo par défaut)
`max_file_uploads`	int : nombre maximum de fichiers (20 par défaut)