Chapitre 4
Écrire des fonctions et des classes PHP

1. Fonctions

1.1 Introduction

À l'instar des différents langages de développement, PHP offre la possibilité de définir ses propres fonctions (appelées fonctions "utilisateur") avec tous les avantages associés (modularité, capitalisation...). Une fonction est un ensemble d'instructions identifiées par un nom, dont l'exécution retourne une valeur et dont l'appel peut être utilisé comme opérande dans une expression. Une procédure est un ensemble d'instructions identifiées par un nom qui peut être appelé comme une instruction.

1.2 Déclaration et appel

Le mot-clé `function` permet d'introduire la définition d'une fonction.

Syntaxe

```
function nom_fonction([paramètre]) [: type]{
  instructions;
}
```

`nom_fonction` Nom de la fonction (doit respecter les règles de nommage présentées dans le chapitre Introduction à PHP - Structure de base d'une page PHP). Ce nom n'est pas sensible à la casse (pour PHP, les fonctions `unefonction` et `UneFonction` sont les mêmes).

`paramètre` Paramètres éventuels de la fonction exprimés sous forme d'une liste de variables (cf. section Fonctions - Paramètres) : `$paramètre1, $paramètre2, ...`

`type` Déclaration du type de données retourné par la fonction. Valeurs possibles : `int`, `float`, `string`, `bool`, `array`, `callable`, `iterable`, `object`, `mixed`, `void`, un nom de classe ou d'interface (cf. dans ce chapitre la section Classes), ou une union de types. Le nom du type peut être précédé d'un point d'interrogation (`?`) qui indique que la fonction peut retourner une valeur `NULL`. Voir le chapitre Les bases du langage PHP pour la définition des types de données (section Les bases du langage PHP - Types de données).

`instructions` Ensemble des instructions qui composent la fonction.

Le nom de la fonction ne doit pas être un mot réservé PHP (nom de fonction native, d'instruction) ni être égal au nom d'une autre fonction préalablement définie.

Une fonction utilisateur peut être appelée comme une fonction native de PHP : dans une affectation, dans une comparaison, etc.

Si la fonction retourne une valeur, il est possible d'utiliser l'instruction `return` pour définir la valeur de retour de la fonction.

Syntaxe

```
return expression;
```

`expression` Expression dont le résultat constitue la valeur de retour de la fonction (`NULL` par défaut).

Le résultat d'une fonction peut être de n'importe quel type (chaîne, nombre, tableau, etc.).

L'instruction `return` stoppe l'exécution de la fonction et retourne le résultat de `expression` à l'appelant. Si plusieurs instructions `return` sont présentes dans la fonction, c'est la première rencontrée dans le déroulement des instructions qui définit la valeur de retour et provoque l'interruption de la fonction. Si la fonction ne comporte aucune instruction `return` (ou si aucune instruction `return` n'est exécutée), la valeur de retour de la fonction est `NULL`.

Exemple

```
<?php
// Fonction sans paramètre qui affiche "Bonjour !"
// Pas de valeur de retour.
function afficher_bonjour() {
  echo 'Bonjour !<br />';
}
// Fonction avec deux paramètres qui retourne le produit
// des deux paramètres.
function produit($valeur1,$valeur2) {
  return $valeur1 * $valeur2;
}
// Appel de la fonction afficher_bonjour.
afficher_bonjour();
// Utilisations de la fonction produit :
// - dans une affectation
$résultat = produit(2,4);
echo "2 x 4 = $résultat<br />";
// - dans une comparaison
if (produit(10,12) > 100) {
  echo '10 x 12 est supérieur à 100.<br />';
}
?>
```

Résultat

```
Bonjour !
2 x 4 = 8
10 x 12 est supérieur à 100.
```

Remarque

Dans le langage PHP, il n'existe pas à proprement parler de procédure. Pour définir quelque chose d'équivalent à une procédure, il suffit de définir une fonction qui ne retourne pas de valeur et d'appeler la fonction comme s'il s'agissait d'une instruction (comme la fonction `afficher_bonjour` par exemple). Une fonction qui ne retourne rien peut explicitement être déclarée avec le type de retour `void`.

Comme nous l'avons déjà évoqué, le contenu d'un tableau peut être transformé en liste de paramètres dans un appel de fonction grâce à l'opérateur `...` (points de suspension).

Exemple

```
<?php
// Fonction avec trois paramètres qui retourne la somme
// des trois paramètres.
function somme($valeur1,$valeur2,$valeur3) {
  return $valeur1 + $valeur2 + $valeur3;
}
// Transformation du contenu d'un tableau en
// liste de paramètres.
$valeurs = [1,2,3];
echo '1 + 2 + 3 = ',somme(...$valeurs),'<br />';
// La même chose pour une partie seulement des paramètres
// avec un tableau défini directement dans l'appel.
echo '1 + 2 + 4 = ',somme(1,...[2,4]),'<br />';
?>
```

Résultat

```
1 + 2 + 3 = 6
1 + 2 + 4 = 7
```

Lorsqu'une fonction retourne un tableau, il est possible d'accéder directement à un élément du tableau lors de l'appel à la fonction avec une syntaxe du type `fonction(...)[clé]`.

Exemple

```
<?php
// Définition d'une fonction qui retourne un tableau.
function qui() {
  return ['Olivier','Heurtel'];
}
// Appel de la fonction et récupération directe du prénom stocké
// à l'indice 0 du tableau retourné.
$prénom = qui()[0];
echo "qui()[0] = $prénom<br />";
?>
```

Résultat

```
qui()[0] = Olivier
```

Cette technique fonctionne aussi lorsque la fonction retourne un tableau multidimensionnel avec une syntaxe du type `fonction(...)[clé1][clé2]`.

Il est possible d'utiliser une fonction avant de la définir.

Exemple

```
<?php
// Utilisation de la fonction produit.
echo produit(5,5);
// Définition de la fonction produit.
function produit($valeur1,$valeur2) {
  return $valeur1 * $valeur2;
}
?>
```

Résultat

```
25
```

Il n'y a donc aucun problème pour définir des fonctions qui s'appellent entre elles.

Remarque

Une fonction est utilisable uniquement dans le script où elle est définie. Pour l'employer dans plusieurs scripts, il faut, soit recopier sa définition dans les différents scripts (vous perdez l'intérêt de définir une fonction), soit la définir dans un fichier inclus partout où la fonction est nécessaire.

Exemple

– Fichier `fonctions.inc` contenant des définitions de fonctions :

```
<?php
// Définition de la fonction produit.
function produit($valeur1,$valeur2) {
  return $valeur1 * $valeur2;
}
?>
```

– Script utilisant les fonctions définies dans `fonctions.inc` :

```
<?php
// Inclusion du fichier contenant la définition des fonctions.
include('fonctions.inc');
// Utilisation de la fonction produit.
echo produit(5,5);
?>
```

Déclaration du type de retour

Il est possible de définir le type de données retourné par une fonction.

Lorsque c'est le cas, dans le mode de fonctionnement par défaut (à opposer au mode strict présenté ci-dessous), PHP effectue si besoin une conversion automatique de la valeur retournée dans le type de données déclaré.

Exemple

```
<?php
// Déclaration de deux fonctions qui retournent le produit
// des deux paramètres, la deuxième spécifiant un type
// de données "entier" pour la valeur de retour.
function produit1($valeur1,$valeur2) {
  return $valeur1 * $valeur2;
}
function produit2($valeur1,$valeur2) : int {
  return $valeur1 * $valeur2;
}
// Appel des deux fonctions avec les mêmes paramètres
echo 'produit1(20,1/7) => ',var_dump(produit1(20,1/7)),'<br />';
echo 'produit2(20,1/7) => <b>',var_dump(produit2(20,1/7)),'</b><br />';
?>
```

Résultat

```
produit1(20,1/7) => float(2.8571428571428568)
produit2(20,1/7) => int(2)
```

Sur cet exemple, nous voyons bien que la valeur retournée par la deuxième fonction a été convertie en entier par PHP (avec les règles de conversion évoquées dans le chapitre Introduction à PHP - Les bases du langage PHP - Types de données).

Si PHP n'est pas en mesure d'effectuer la conversion (types de données non convertibles entre eux), une exception `TypeError` est levée ; cette exception interrompt le script si elle n'est pas gérée (cf. dans ce chapitre la section Classes - Exceptions).

Exemple

```
<?php
// Déclaration et appel d'une fonction qui doit retourner un
// tableau mais qui retourne une chaîne de caractères.
function qui() : array {
  return 'Olivier Heurtel';
}
echo 'qui()[0] = ',qui()[0];
?>
```

Résultat

```
qui()[0] =
Fatal error: Uncaught TypeError: Return value of qui() must be of the
type array, string returned in /app/scripts/index.php:5 Stack trace: #0
/ app/scripts/index.php(7): qui() #1 {main} thrown in/app/scripts/
index.php on
line 5
```

Une fonction déclarée avec un type de retour autre que `void` doit retourner une valeur non `NULL`. Si ce n'est pas le cas, une erreur est retournée, différente selon les cas :

Absence d'instruction `return`

```
Fatal error: Uncaught TypeError: Return value of MaFonction() must be of
the type int, none returned in ...
```

Instruction `return` vide

```
Fatal error: A function with return type must return a value in ...
```

Instruction `return NULL`

```
Fatal error: Uncaught TypeError: Return value of MaFonction() must be of
type int, null returned in ...
```

Pour autoriser une fonction à retourner une valeur `NULL`, il faut faire précéder le nom du type (autre que `void`) d'un point d'interrogation (`?`).

```
<?php
// Déclaration et appel d'une fonction qui spécifie un
// type de donnée de retour qui peut être NULL
function cube($valeur) : ?int {
  if (is_null($valeur)) {
    return NULL;
  } else {
    return $valeur ** 3 ;
  }
}
echo 'cube(2) => <b>',var_dump(cube(2)),'</b><br />';
echo 'cube(NULL) => <b>',var_dump(cube(NULL)),'</b><br />';
?>
```

Résultat

```
cube(2) => int(8)
cube(NULL) => NULL
```

Même avec cette option, la fonction doit avoir une instruction `return` non vide. Si ce n'est pas le cas, une erreur est retournée, différente selon les cas :

Absence d'instruction `return`

```
Fatal error: Uncaught TypeError: Return value of MaFonction() must be of
type ?int, none returned in ...
```

Instruction `return` vide

```
Fatal error: A function with return type must return a value (did you mean
"return null;" instead of "return;"?) in ...
```

À l'inverse, il est possible de déclarer qu'une fonction ne retourne rien, en utilisant `void` comme nom de type. Dans ce cas, la fonction doit omettre l'instruction `return` ou mettre une instruction `return` vide, sans valeur (même pas `NULL`).

Chapitre 4
Sécurité et gestion des utilisateurs

1. Introduction

Savez-vous qu'une bonne partie des attaques informatiques les plus médiatisées concernent les bases de données ? En y réfléchissant, c'est parfaitement logique : les parties les plus sensibles d'un système informatique sont les données personnelles (identifiants, mots de passe, nationalité, salaire, historique de santé…) qui sont stockées dans une base de données. Il est donc impératif de veiller à la sécurisation de vos serveurs MySQL si vous voulez éviter la mauvaise publicité liée à une fuite de données.

Ce chapitre a pour but de vous donner les clés pour assurer la sécurité de votre système MySQL. Notez que nous nous limitons aux éléments spécifiques à MySQL et que d'autres éléments tout aussi importants tels que la sécurisation du serveur hôte ou du réseau ne sont pas abordés.

2. Sécurisation du serveur

Les questions liées à la sécurité doivent être posées dès l'installation du serveur de base de données. Dans la philosophie de MySQL l'utilisateur doit pouvoir télécharger, installer et utiliser le SGBDR (système de gestion de bases de données relationnelles) sans aucune difficulté. Cette simplicité est l'un des points forts de MySQL, car d'une part cela a permis de démocratiser l'utilisation d'une base de données en rendant accessible cet univers (beaucoup de développeurs ont connu les bases de données grâce à MySQL) et d'autre part, cela donne la possibilité de tester très simplement son application. Mais historiquement, l'installation par défaut a longtemps été beaucoup trop permissive, ce qui explique pourquoi on trouve encore nombre de serveurs mal sécurisés. De gros progrès ont été réalisés à partir notamment de MySQL 5.7.

2.1 Sécurisation de l'installation

Le but de cette section n'est pas de revenir sur l'installation du serveur qui est détaillée au chapitre Installation du serveur, mais simplement de rappeler quelques points cruciaux concernant la sécurité. Vous devrez certainement les adapter en fonction de votre type d'installation et de votre système d'exploitation.

2.1.1 Contrôler les droits

Vous devez créer un groupe et un utilisateur dédié pour lancer l'instance `mysqld` (cette étape est effectuée automatiquement si vous utilisez un installeur ou votre gestionnaire de paquets) :

```
$ groupadd mysql
$ useradd -g mysql mysql
```

Le répertoire de données contient les informations sensibles, il doit donc être préservé d'actes de malveillance ou de la visualisation par des personnes non autorisées :

```
$ cd /usr/local/mysql/
$ chown -R root .
$ chown -R mysql data
$ chgrp -R mysql .
```

`mysqld` ne doit en aucun cas être exécuté avec l'administrateur système `root` sous UNIX (ou `administrateur` sous MS Windows). Un utilisateur avec par exemple le droit `FILE` pourrait créer des fichiers en tant que `root`.

2.1.2 Ajouter un mot de passe au compte utilisateur root

C'est le superutilisateur de MySQL (à ne pas confondre avec l'administrateur système `root` sous UNIX) ; il a donc tous les droits sur le serveur ; il doit être protégé par un mot de passe. Ceci est valable quel que soit le système d'exploitation.

```
$ mysql –u root # connexion au serveur avec l'utilisateur root sans
mot de passe
mysql> ALTER USER root@localhost IDENTIFIED BY 'm0T2p4ss3';
```

Comme vous allez le voir un peu plus loin, un utilisateur MySQL est composé d'un nom « user » et du nom de la machine à partir de laquelle l'utilisateur se connecte « host ». Vous pouvez donc avoir un compte `'root'@'localhost'` qui permet de se connecter au serveur avec l'utilisateur `root` à condition que le client soit sur la même machine que le serveur MySQL (en local) et par exemple un compte `'root'@'123.45.67.89'` qui, lui ne permet de se connecter en `root` qu'à partir de la machine qui a comme adresse IP 123.45.67.89. Ce sont bien deux comptes différents, qui peuvent avoir des droits et des mots de passe différents. Cette possibilité offerte par le serveur peut permettre une gestion des droits très fine. Cependant, par expérience, nous vous conseillons de n'avoir qu'une seule occurrence par utilisateur, par souci de simplification de la gestion des droits et donc pour diminuer les risques d'erreurs. En d'autres termes, ne gardez que l'utilisateur `'root'@'localhost'`. Si vous avez besoin d'administrer votre serveur à distance, au lieu d'avoir `'root'@'%'` (qui permet de se connecter avec l'utilisateur `root` depuis n'importe quelle machine), utilisez le protocole de communication sécurisée SSH ou un équivalent.

Exemple : Si vous trouvez plusieurs comptes `root` sur votre système, alors que seul le compte `root@localhost` est utile

```
mysql> SELECT user, host FROM mysql.user WHERE user = 'root' \G
*************************** 1. row ***************************
user: root
host: localhost
```

```
*************************** 2. row ***************************
user: root
host: 123.456.78.9

Vous pouvez supprimer le compte inutile :
mysql> DROP USER 'root'@'123.456.78.9' ;

mysql> SELECT user, host FROM mysql.user WHERE user = 'root' \G
*************************** 1. row ***************************
user: root
host: localhost
```

Remarque

Renommer le compte administrateur : vous pouvez renommer votre compte administrateur pour qu'il soit plus difficile à trouver par une personne malveillante : `RENAME USER 'root'@'localhost' TO 'leader'@'localhost';`

2.1.3 Supprimer les comptes anonymes

Des comptes anonymes, c'est-à-dire des comptes ayant le champ `user` à vide, peuvent être présents sur votre système. Certains installeurs créaient automatiquement un tel compte dans le passé avec certaines versions de MySQL. Un compte anonyme permet de se connecter au serveur avec n'importe quel utilisateur, en local et sans mot de passe : pratique, mais franchement problématique au niveau sécurité. Mieux vaut supprimer au plus vite ce genre de comptes, qui peuvent être trouvés avec la requête suivante :

```
mysql> SELECT user, host FROM mysql.user WHERE user = '' \G
*************************** 1. row ***************************
user:
host: localhost
password:
```

Remarque

Depuis MySQL 5.7, plus aucun compte anonyme n'est créé lors de l'installation du serveur.

2.1.4 Supprimer le schéma test

Le schéma `test` est créé par MySQL lors de l'installation pour toutes les versions avant 5.7 et tous les utilisateurs ont le droit d'y accéder en lecture comme en écriture sans qu'il soit nécessaire de spécifier explicitement les droits adéquats. Un utilisateur mal intentionné pourrait alors remplir de données votre disque dur et ainsi empêcher le bon fonctionnement de votre application. Si ce schéma n'a aucune utilité pour votre application, mieux vaut le supprimer :

```
mysql> SHOW SCHEMAS;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| test               |
+--------------------+

mysql> DROP SCHEMA test;
```

À noter que cette recommandation s'applique également à tout schéma `test` qui serait créé par la suite, mais également à tout schéma qui commencerait par le mot `test_`.

2.1.5 Sécuriser votre installation avec l'outil mysql_secure_installation

La mise en œuvre de ces différentes recommandations peut être effectuée avec l'outil `mysql_secure_installation`. Cet outil n'est pas disponible sous MS Windows, cependant l'installeur graphique propose également de sécuriser l'installation.

Il suffit de lancer le script en tant qu'utilisateur Linux `root` et de suivre les indications à l'écran :

```
# mysql_secure_installation
Securing the MySQL server deployment.

Connecting to MySQL using a blank password.

VALIDATE PASSWORD PLUGIN can be used to test passwords
and improve security. It checks the strength of password
```

```
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD plugin?
Press y|Y for Yes, any other key for No:
```

2.2 Chiffrement des données

Pour faire face aux besoins de chiffrement d'informations sensibles (financières, médicales...) ou pour tenter de se prémunir contre le vol de données, contre des administrateurs peu scrupuleux, il peut être nécessaire de crypter ses données. Avant la version 5.7, MySQL ne possédait pas de modules de chiffrement capables par exemple de crypter le contenu d'une colonne ou d'une table. Une possibilité était cependant de crypter/décrypter des données avec les fonctions MySQL telles que `AES_ENCRYPT()` / `AES_DECRYPT()`, `DES_ENCRYPT()` / `DES_DECRYPT()` et `ENCODE()` / `DECODE()`.

Remarque

Pour plus de détails, consultez la page suivante :
https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html

Les données sont alors cryptées sur le disque. Par contre, elles sont en clair, y compris la clé de cryptage dans le code SQL ainsi que sur le réseau. Par exemple :

```
mysql> INSERT INTO t_crypt VALUES (1, AES_ENCRYPT('Phrase cryptée',
'crypt_key'));

mysql> SELECT i, b FROM t_crypt;
+------+------------------+
| i    | data_crypt       |
+------+------------------+
| 1    | ?z#?uŠb?7?G[g,<  |
+------+------------------+

mysql> SELECT i, AES_DECRYPT(b,'crypt_key') AS data_crypt FROM t_crypt;
+------+----------------+
| i    | data_crypt     |
+------+----------------+
| 1    | Phrase cryptée |
+------+----------------+
```