Développement web S1
– 4. Fonctions

Christophe BLANC

– BUT MMI –
IUT Clermont Auvergne
Université Clermont Auvergne
www.christophe-blanc.fr

2023-2024

- ✔ A l'instar des différents langage de programmation, PHP offre la possibilité de définir ses propres fonctions 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 un instruction.

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

Syntaxe

```
function nom_fonction([parametres])
{
        instructions;
}
```

- ✔ nom_fonction : nom de la fonction (doit respecter les règles de nommage)
- ✔ parametres : paramètres éventuels de la fonction exprimés sous forme d'une liste de variables : $param1, $param2,...
- ✔ instructions : ensemble des instructions qui composent la fonction.

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

Fonctions

Déclaration et appel : exemple

```php
<?php
//fonction sans parametre qui affiche " Bonjour !" sans valeur de retour
function afficher_bonjour(){
        echo "Bonjour !</br>";
}
//fonction avec 2 parametres qui retourne le produit des deux parametres
function produit($param1,$param2){
        return $param1 * $param2;
}
//utilisation de la fonction afficher_bonjour
afficher_bonjour();
//utilisation de la fonction produit dans une affectation
$resultat=produit(2,4);
echo "$resultat</br>";
//utilisation de la fonction produit dans une comparaison
if (produit(10,12)>100){
        echo "Le resultat est superieur a 100.</br>";
}
//utilisation de la fonction produit dans une affectation et une comparaison
if (($resultat = produit(10,12))>100){
        echo "$resultat est superieur a 100.</br>";
}
?>
```

Bonjour !
8
Le resultat est supérieur à 100.
120 est supérieur à 100.

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

```
<?php
echo produit(5,5);

function produit($param1,$param2)
{
        return $param1 * $param2;
}
?>
```

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
//definition de la fonction produit
function produit($param1,$param2){
        return $param1 * $param2;
}
?>
```

Script utilisant les fonctions définies dans *fonctions.inc* :

```
<?php
//inclusion du fichier contenant la definition des fonctions
include ("fonctions.inc");
echo produit(5,5);
?>
```

PHP propose "une fonction variable" qui permet de stocker un nom de fonction dans une variable et d'appeler la variable dans une instruction comme si c'était une fonction, avec la notation *$variable()*. sur une telle écriture, PHP remplace la variable par sa valeur et cherche à exécuter la fonction correspondante (qui doit ,bien entendu, exister).

```
<?php
function produit($param1,$param2){
        return $param1 * $param2;
}
function somme($param1,$param2){
        return $param1 + $param2;
}
function calculer($operation,$param1,$param2){
        return $operation($param1,$param2);
}
echo calculer("produit",3,5)."</br>";
echo calculer("somme",3,5)."</br>";
?>
```

Les paramètres éventuels de la fonction sont définis sous la forme d'une liste de variables. Nous allons étudier les possibilités suivantes :

- ✔ définir une valeur par défaut pour un paramètre
- ✔ passer un paramètre par référence
- ✔ utiliser une liste de paramètres

Il est possible d'indiquer qu'un paramètre possède une valeur par défaut grâce à la syntaxe suivante :

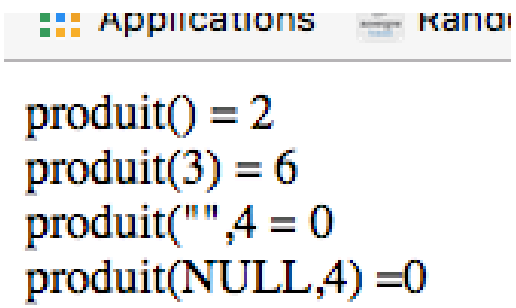
```
$param = expression litterale
```

- ✔ La valeur par défaut d'un paramètre doit être une expression littérale et ne peut être ni une variable, ni une fonction, ni une expression composée.
- ✔ La valeur par défaut est utilisée comme valeur d'un paramètre lorsque la fonction est appelée, sans mentionner de valeur pour le paramètre en question.
- ✔ Ne pas donner de valeur à un paramètre ayant une valeur par défaut n'est possible qu'en partant de la droite.
- ✔ Passer un nombre insuffisant de paramètres et ne pas avoir de valeur par défaut génère une erreur. Passer trop de paramètres ne génère pas d'erreur ; les paramètres en trop sont ignorés.

Fonctions

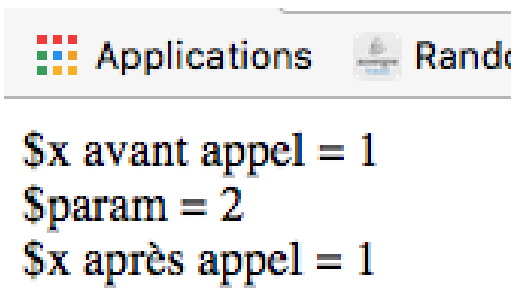
Paramètres : valeur par défaut - exemple

```php
<?php
function produit($param1=1,$param2=2){
        return $param1 * $param2;
}
//appel sans parametre
echo "produit() = ".produit()."</br>";
//appel avec un seul parametre = forcement le premier
echo "produit(3) = ".produit(3)."</br>";
//echo "produit(,4) = ".produit(,4)."</br>";
        //interdit
echo "produit(\"\",4 = ".produit("",4)."</br>";
echo "produit(NULL,4) =".produit(NULL,4)."</br>";
?>
```



Par défaut, le passage des paramètres s'effectue par valeur : c'est une copie de la valeur qui est passée à la fonction. En conséquence, la modification des paramètres à l'intérieur de la fonction n'a aucun effet sur les valeurs dans le script appelant.

```
<?php
//definition d'une fonction qui prend un parametre
function par_valeur($param){
        $param++;
        echo "\$param = $param </br>";
}
$x=1;
echo "\$x avant appel = $x </br>";
par_valeur($x);
echo "\$x apres appel = $x </br>";
?>
```

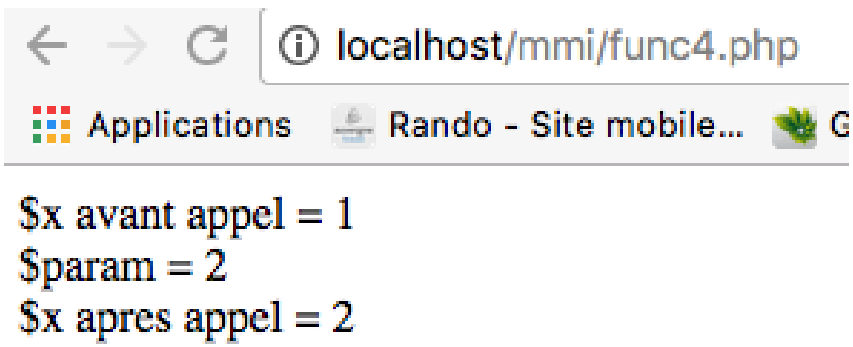


En cas de besoin, il est possible d'avoir un passage par référence en utilisant l'opérateur de référence & devant le nom du paramètre dans la définition de la fonction. Avec une telle définition, c'est une référence vers la variable (plus une copie) qui est passée à la fonction : cette dernière travaille directement sur la variable du script appelant.

Fonctions

Paramètres : passage par référence - exemple

```php
<?php
//definition d'une fonction qui prend un parametre
function par_reference(&$param){
        $param++;
        echo "\$param = $param </br>";
}
$x=1;
echo "\$x avant appel = $x </br>";
par_reference($x);
echo "\$x apres appel = $x </br>";
?>
```



L'utilisation du mot clé *return* à l'intérieur d'une fonction, permet de définir la valeur de retour de la fonction et de stopper son exécution.

Syntaxe

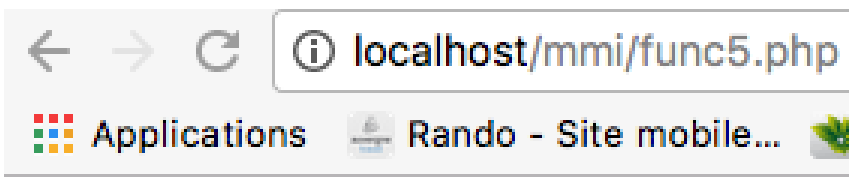
```
return expression
```

- ✔ *expression* : expression dont le résultat constitue la valeur de retour de la fonction.
- ✔ Le résultat d'une fonction peut être de n'importe quel type (chaîne, nombre, tableau,...)
- ✔ 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.

Fonctions

Valeur de retour : exemple

```php
<?php
//definition d'une fonction avec deux appels a return
function valeur_retour($param){
        if($param==1){
                return "Premier return";
        }
        return "Deuxieme return";
}
//appel a la fonction
echo "valeur_retour(1) =".valeur_retour(1)."</br>";
echo "valeur_retour(0) =".valeur_retour(0)."</br>";
?>
```



- ✔ Les variables utilisées à l'intérieur d'une fonction sont locales : elles sont non définies en dehors de la fonction et initialisées à chaque appel de la fonction.
- ✔ Réciproquement, une variable définie en dehors de la fonction (dans le script appelant) n'est pas définie à l'intérieur de la fonction.

Fonctions

Variables utilisées dans les fonctions : variables locales - exemple

```php
<?php
function variable_locale(){
        $x = 0;
        $z = 3;
        echo "Valeur de \$x dans la fonction = $x</br>";
        echo "Valeur de \$y dans la fonction = $y</br>";
        echo "Valeur de \$z dans la fonction = $z</br>";
}
$x = 1;
$y = 2;
variable_locale();
echo "Valeur de \$x dans le script = $x</br>";
echo "Valeur de \$y dans le script = $y</br>";
echo "Valeur de \$z dans le script = $z</br>";
?>
```

Valeur de $x dans la fonction = 0

Notice: Undefined variable: y in **/Applications/XAMPP/xamppfiles/htdocs/mmi/func6.php** on line **6**
Valeur de $y dans la fonction =
Valeur de $z dans la fonction = 3
Valeur de $x dans le script = 1
Valeur de $y dans le script = 2

Notice: Undefined variable: z in **/Applications/XAMPP/xamppfiles/htdocs/mmi/func6.php** on line **14**
Valeur de $z dans le script =

- ✔ PHP propose une notion de variable globale pour accéder, dans une fonction, aux variables définies dans le contexte du script appelant
- ✔ Pour cela, à l'intérieur de la fonction, il faut déclarer les variables globales que la fonction utilise avec l'instruction *global*.

Syntaxe

```
global $variable,...
```

- ✔ *$variable* : variable du script appelant que la fonction souhaite utiliser. Plusieurs variables peuvent être mentionnées, en les séparant par des virgules.

Fonctions

Variables utilisées dans les fonctions : variables globales - exemple

```php
<?php
function variable_globale(){
        global $x;
        echo "Valeur de \$x au debut la fonction = $x</br>";
        $x = 0;
        $z = 3;
        echo "Valeur de \$x a la fin de la fonction = $x</br>";
        echo "Valeur de \$y dans la fonction = $y</br>";
        echo "Valeur de \$z dans la fonction = $z</br>";
}
$x = 1;
$y = 2;
variable_globale();
echo "Valeur de \$x dans le script = $x</br>";
echo "Valeur de \$y dans le script = $y</br>";
echo "Valeur de \$z dans le script = $z</br>";
?>
```

Valeur de $x au debut la fonction = 1
Valeur de $x a la fin de la fonction = 0

Notice: Undefined variable: y in **/Applications/XAMPP/xamppfiles/htdocs/mmi/func7.php** on line **8**
Valeur de $y dans la fonction =
Valeur de $z dans la fonction = 3
Valeur de $x dans le script = 0
Valeur de $y dans le script = 2

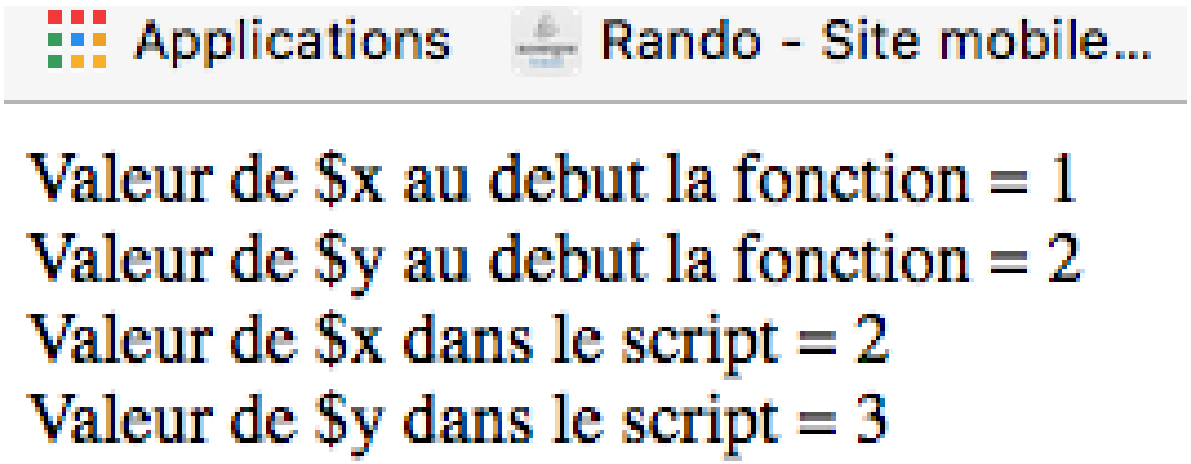
Notice: Undefined variable: z in **/Applications/XAMPP/xamppfiles/htdocs/mmi/func7.php** on line **16**
Valeur de $z dans le script =

Il est possible, sans déclaration, d'accéder aux variables globales à l'intérieur d'une fonction, en utilisant un tableau associatif *$GLOBALS* géré par PHP. Dans ce tableau associatif, la clé est égale au nom de la variable globale (sans le $) et la valeur, à la valeur de la variable globale

Fonctions

Variables utilisées dans les fonctions : variables globales - exemple

```php
<?php
function variable_globale(){
        global $x;
        echo "Valeur de \$x au debut la fonction = $GLOBALS[x]</br>";
        echo "Valeur de \$y au debut la fonction = $GLOBALS[y]</br>";
        $GLOBALS["x"]++;
        $GLOBALS["y"]++;
}
$x = 1;
$y = 2;
variable_globale();
echo "Valeur de \$x dans le script = $x</br>";
echo "Valeur de \$y dans le script = $y</br>";
?>
```



Par défaut les variables locales d'une fonction sont réinitialisés à chaque appel de la fonction. L'instruction *static* permet de définir des variables locales statiques qui ont pour propriété de conserver leur valeur d'un appel à l'autre de la fonction, pendant la durée du script.

Syntaxe

```
static $variable = expression_litterale
```

- ✔ *$variable* : variable concernée
- ✔ *expression_litterale* : valeur initiale affectée à la variable lors du premier appel de la fonction.

Fonctions

Variables utilisées dans les fonctions : variables statiques - exemple

```php
<?php
function variable_statique(){
        static $variable = 0;
        $autre_variable = 0;
        echo "\$variable = $variable</br>";
        echo "\$autre_variable = $autre_variable</br>";
        $variable++;
        $autre_variable++;
}
echo "<B>Premier appel de la fonction :</B></br>";
variable_statique();
echo "<B>Deuxieme appel de la fonction :</B></br>";
variable_statique();
echo "<B>Troisieme appel de la fonction :</B></br>";
variable_statique();
?>
```

Premier appel de la fonction :
$variable = 0
$autre_variable = 0
Deuxieme appel de la fonction :
$variable = 1
$autre_variable = 0
Troisieme appel de la fonction :
$variable = 2
$autre_variable = 0

- ✔ Nous savons que la portée des constantes est le script dans lequel elles sont définies.
- ✔ A la différence des variables, cette portée s'étend aux fonctions appelées dans le script : une constante peut être utilisée à l'intérieur de la fonction sans qu'elle soit déclarée globale.
- ✔ Réciproquement, une constante définie dans une fonction peut être utilisée dans un script, après appel de la fonction.

```php
<?php
define("CONSTANTE_SCRIPT","constante␣script");
function constante(){
        define("CONSTANTE_FONCTION","constante␣fonction");
        echo "Dans␣la␣fonction,␣CONSTANTE_SCRIPT␣=␣".CONSTANTE_SCRIPT."</br>";
}
constante();
echo "Dans␣le␣script,␣CONSTANTE_FONCTION␣=␣".CONSTANTE_FONCTION."</br>";
?>
```

Dans la fonction, CONSTANTE_SCRIPT = constante script
Dans le script, CONSTANTE_FONCTION = constante fonction