BACK #1319 - PHP

3 - Fonctions - Partie 1

> Une fonction encapsule un morceau de code, généralement indépendant, écrit pour exécuter une action spécifique.

Optionnellement, des paramètres peuvent lui être passés et une valeur peut être retourné.

PHP offre deux types de fonctions :

- les fonctions dites "natives", qui sont fournies par PHP ou ajoutées à l'aide de modules d'extension,
- les fonctions utilisateur que sont créées lors de l'écriture du code source d'une application.

3.1 - Fonctions natives

> PHP propose un grand nombre de fonctions. Leur description complète peut être trouvée sur le site php.nrt (https://www.php.net/manual/fr/functions.internal.php)

3.2 - Fonctions utilisateur

Déclaration d'une fonction :

> La définition d'une fonction s'appelle **déclaration** et peut se faire à n'importe quel endroit dans le code.

- Un nom de fonction est insensible à la casse
- La déclaration d'une fonction se fait avec le mot clé `function` suivi de son nom , d'une paire de parenthèses contenant les éventuels paramètres et d'une paire d'accolades contenant l'ensemble des instructions à exécuter
- Le nom de la fonction ne doit pas être un mot réservé de PHP et doit respecter les règles de nommage suivantes :
    - il commence par une lettre ou un underscore,
    - il n'est composé que de caractères alphanumériques (lettres et chiffres) et d'underscores

Les paramètres (ou arguments) sont les valeurs qui doivent être transmises à la fonction

```php
function nom_de_la_fonction( $arg1, $arg2 ) {
    // instructions
}
```

Appel de fonction

> Pour exécuter le code d'une fonction, il faut l'appeler, en lui passant les paramètres nécessaires.

Certains arguments peuvent être optionnels, lorsqu'une valeur par défaut leur a été attribuée à la déclaration de la fonction.

```
function nom_de_la_fonction( $arg1, $arg2 = 'Valeur par defaut') {
    // instructions
    return $une_valeur;
}

$valeur_de_retour = nom_de_la_fonction( 'argument obligatoire', 'argument
optionnel' );
```

Paramètres (arguments) :

> Une fonction peut déclarer une liste de paramètres attendus, séparés par des virgules.

Cette liste peut être :

- vide,
- fixe (arguments déclarés et attendus),
- variable (tous les arguments ne sont pas déclarés),

Les arguments peuvent :

- être typés
- avoir une valeur par défaut,
- être passés par valeur ou par référence,

Valeur par défaut :

> Il est possible de donner une valeur par défaut aux paramètres d'une fonction

Cette valeur sera alors utilisée par la fonction si celle-ci n'est pas précisé lor de l'appel

```
function nom_de_la_fonction( $arg1 = 'Valeur par defaut') {
    // instructions
}
```

Si un argument possède une valeur par défaut, il est dit `optionnel`ou `facultatif`, `obligatoire` sinon.

> Les arguments obligatoires doivent être placés avant les arguments optionnels.

```
function nom_de_la_fonction( $arg_obligatoire, $arg_optionnel = 'Valeur') {
    // instructions
}
```

> Les arguments par défaut doivent être de type constant : `$arg = 1+4` fonctionne mais `$arg=1+$a` ne fonctionne pas

Valeur de retour

> La fonction peut renvoyer une valeur de retour grâce au mot-clé `return`

Lorsque l'instruction `return` est rencontrée :

- la valeur spécifiée est retournée au code appelant
- l'exécution de la fonction est stoppée.

La déclaration d'une valeur de retour est facultative. Si la fonction n'utilise pas l'instruction `return`, la valeur retournée est `null`

```php
function nom_de_la_fonction( $arg1 = 'Valeur par defaut') {
    // instructions
    return $une_valeur;
}
```

Déclaration de type (Type hinting)

> Le **type hinting** consiste à indiquer à une fonction le type des arguments attendus et le type de la valeur qui doit être retournée

Ces déclarations permettent à PHP de vérifier que les valeurs transmises sont effectivement du type prévu.

En programmation procédurale, les types possibles sont :

Type	
array	tableau
bool	booléen
float	flottant
int	entier
string	chaîne de caractères
mixed	object, resource, array, string,int, float,bool, null (PHP >= 8.0.0)

Attention : l'utilisation d'autres alias ne sera pas traitée correctement. Ces alias seront considérés comme des *noms de classe*

Type nullable

> Les déclarations de type peuvent être marquées comme nullable en préfixant le nom du type avec un point d'interrogation (`?`).

Ceci signifie que la fonction retourne soit le type spécifié soit `null`.

Valeur de retour

> Le type de la valeur de retour peut être contrôlé :

```
function retour(): int
{
    $b = 5;
    return $b;
}
```

Pour le type de retour uniquement, le type `void`est disponible. Il précise que la fonction re retourne pas de valeur.

Nombre d'arguments variable

> PHP supporte l'appel de fonctions avec un nombre d'arguments variable

Pour réaliser cette implémentation, on peut utiliser le token `...`

Le token `...`indique à PHP que la fonction peut recevoir un nombre d'arguments variable et que ceux-ci sont disponible sous la forme d'un tableau dont le nom est indiqué juste après le token

```
function somme(...$nombres) {
    $total = 0;
    foreach($nombres as $nombre) {
        $total += $nombre;
    }
    return $total;
}

echo (somme(1,2,3,4,5));
```

Avant l'apparition de ce token, il était cependant possible d'utiliser un nombre variable d'arguments avec les fonctions `func_num_args()`, `func_get_args()`et `func_get_arg()`

```
function somme() {
    $total = 0;
    foreach(func_get_args() as $nombre) {
        $total += $nombre;
    }
    return $total;
}

echo (somme(1,2,3,4,5));
```

Portée des variables

La portée d'une variable dépend du contexte dans lequel la variable est définie.

Pour la majorité des variables, la portée concerne la totalité d'un script PHP. Mais, lorsque vous définissez une fonction, la portée d'une variable définie dans cette fonction est locale à la fonction.

En déclarant globales les variables locales d'une fonction, toutes les références à ces variables concerneront les variables globales. Il n'y a aucune limite au nombre de variables globales qui peuvent être manipulées par une fonction.

```php
$a = 1;
$b = 2;

function multiplie() {
    global $a, $ab;
    $a++;
    return $a + $b;
}

var_dump(multiplie());
var_dump($a);
```