

Conception de sites web dynamique

***HTML – CSS – JAVASCRIPT – PHP
BASE DE DONNÉES***

IF04U050

David Bouchet

david.bouchet.paris5@gmail.com

PHP ***2^e partie***



Tableaux

- La **taille** d'un tableau est **dynamique** :
elle est fixée par le nombre d'éléments entrés
- Un tableau peut contenir des **valeurs de types différents**
 - ◆ Exemple :
\$tab[0]= "1er élément";
\$tab[1]= FALSE;
\$tab[2]= 120;
- Les tableaux peuvent être :
 - ◆ **Indicés** :
valeurs référencées par leur **positions** en partant de 0
 - ◆ **Associatifs** :
valeurs référencées par des **noms ou clés**

Tableaux indicés

- **L'affectation d'un indice** à un nouvel élément **est automatique** :
il correspond à la première cellule vide
 - ❖ Exemple :

```
$tab2[] = "1er élément";  
$tab2[] = false;  
$tab2[] = 120;
```
- L'initialisation d'un tableau peut aussi se faire via la fonction **array**
 - ❖ Exemple

```
$tab3 = array("1er élément", false, 120);
```

Les tableaux

Parcours d'un tableau indicé :

```
$nombres = array(3, 6, 9);  
for($i = 0; $i < count($nombres); $i++)  
    echo $i." : ".$nombres[$i]."<br>";
```

ou

```
$nombres = array(3, 6, 9);  
foreach($nombres as $n)  
    echo $n."<br>";
```

Tableaux associatifs

- Indices numériques remplacés par des chaînes de caractères = **clés**
- Pour un tableau donné, toutes les clés doivent être différentes

Exemple d'initialisation :

```
// Méthode 1
```

```
$tab = array
```

```
(
```

```
    "Paris" => "Île-de-France",
```

```
    "Ivry-sur-Seine" => "Île-de-France",
```

```
    "Rennes" => "Bretagne"
```

```
);
```

```
// Méthode 2
```

```
$tab["Toulouse"] = "Languedoc-Roussillon-Midi-Pyrénées";
```

```
$tab["Bordeaux"] = "Aquitaine-Limousin-Poitou-Charentes";
```

```
$tab["Lille"] = "Nord-Pas-de-Calais-Picardie";
```

Parcourir un tableaux associatif

Utilisation de la boucle *foreach*

```
foreach($tableau as $key => $value)  
    echo "La case $key vaut $value";
```

Parcourir un tableau associatif

- Autres fonctions

- ◆ **reset(\$tab)**

- Positionne le pointeur du tableau sur le premier élément et renvoie sa valeur

- ◆ **end(\$tab)**

- Positionne le pointeur du tableau sur le dernier élément et renvoie sa valeur

- ◆ **next(\$tab)**

- avance le pointeur du tableau de 1 et renvoie la nouvelle valeur courante

- ◆ **prev(\$tab)**

- recule le pointeur du tableau de 1 et renvoie la nouvelle valeur courante

Parcourir un tableaux associatif

- On peut utiliser un **curseur**, qui indique l'élément actuellement visé :
 - Curseur initialement sur le premier élément
 - Déplacement par **next()** et **prev()**
 - **key()** → clé de l'élément courant
 - **current()** → valeur de l'élément courant

Exemple :

```
do
{
    echo key($tab)." => ".current($tab)."<br>";
} while (next($tab))
```

Fonctions de manipulation des tableaux

- **array_push**(\$tab, "apple");
 - Empile un ou plusieurs éléments à la fin d'un tableau
- **array_unshift**(\$tab, "banana");
 - Empile un ou plusieurs éléments au début d'un tableau
- \$dernier = **array_pop**(\$tab)
 - Retourne et supprime le dernier élément du tableau
- \$premier = **array_shift**(\$tab);
 - Retourne et supprime le premier élément du tableau

Fonctions de manipulation des tableaux

- **implode("sep", \$tab);**

- ➔ Convertit un tableau en une chaîne de caractères en insérant le séparateur "sep" entre deux éléments consécutifs

```
$tab = array("rouge", "vert", "bleu") ;  
$chaîne = implode(":", $tab);
```

```
// équivaut à :  
$chaîne = "rouge:vert:bleu";
```

Manipulation de tableaux

Affichage :

- ❖ Le tableau peut être affiché en entier par la fonction ***print_r()*** :
`print_r($tab);`
- ❖ Affichage sur plusieurs lignes : `echo nl2br(print_r($tab, true));`

```
$tab = array("rouge", "vert", "bleu");  
print_r($tab); // Affichage 1  
echo nl2br(print_r($tab, true)); // Affichage 2
```

Affichage 1

```
Array ( [0] => rouge [1] => vert [2] => bleu )
```

Affichage 2

```
Array  
(  
[0] => rouge  
[1] => vert  
[2] => bleu  
)
```

Tri de tableau

- Tri selon l'ordre des éléments

◆ `sort()`

→ tri **selon l'ordre des codes ASCII** :

- « a » est après « Z » et « 10 » est avant « 9 »)
- **Le tableau initial est modifié**
et non récupérable dans son ordre original
- **Pour les tableaux associatifs, les clés seront perdues**
et remplacées par un indice commençant à 0

◆ `rsort()` idem mais en ordre inverse des codes ASCII.

Tri

- Tri selon l'ordre des éléments (suite) :
 - ❖ `asort()` trie également les valeurs selon le critère des codes ASCII, mais en préservant les clés pour les tableaux associatifs
 - ❖ La fonction `arsort()` idem mais en ordre inverse des codes ASCII

Tri

- Tri selon l'ordre des clés
 - ❖ **ksort()** trie les clés du tableau selon l'ordre des codes ASCII et préserve les associations clé /valeur
 - ❖ **krsort()** idem mais en ordre inverse des codes ASCII

```
<?php
$tab = array ("1622"=>"Molière", "1802"=>"Hugo", "1920"=>"Vian") ;
ksort ($tab);
echo "<h3 > Tri sur les clés de \$tab2 </h3>" ;
foreach ($tab as $cle => $valeur)
    echo "L'élément a pour clé : $clé; et pour valeur : $valeur<br />";
?>
```

Manipulation de tableaux (suite)

array_key_exists('cle', \$array);

Exemple :

```
$coordonnees = array ( 'prenom' => 'François',  
                        'nom' => 'Dupont',  
                        'adresse' => '3 Rue du Paradis',  
                        'ville' => 'Marseille');  
  
if (array_key_exists('nom', $coordonnees))  
    echo 'La clé "nom" se trouve dans les coordonnées !';
```


Manipulation de tableaux (fin)

in_array(valeur,tableau) → booléen

EX :

```
$fruits = array ('Banane', 'Pomme', 'Poire', 'Cerise', 'Fraise',  
'Framboise');
```

```
if (in_array('Myrtille', $fruits))  
    echo 'La valeur "Myrtille" se trouve dans les fruits !';
```

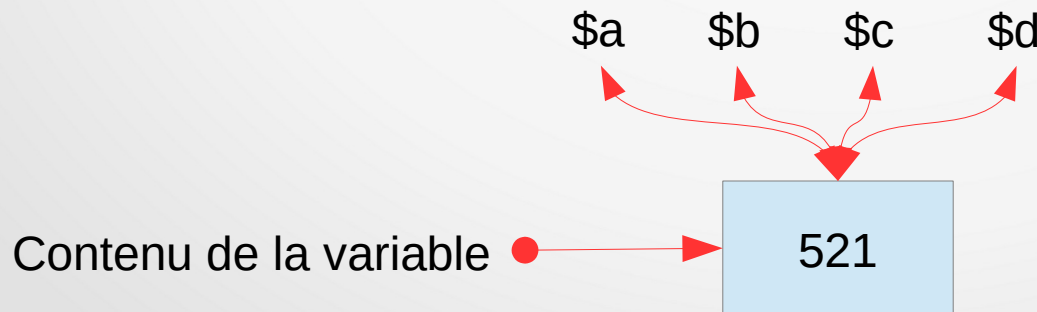
array_search(valeur, tableau)

renvoie

- l'indice ou la clé si la valeur existe
- false sinon

Les références (1)

- Une référence permet d'accéder au contenu d'une même variable en utilisant plusieurs noms.
- Une référence n'est rien d'autre qu'un alias.
- Dans le langage PHP, le nom d'une variable et son contenu sont deux notions distinctes.
- Plusieurs noms pour un même contenu.



Si on modifie \$a, alors \$b, \$c et \$d sont modifiés de la même façon.

Les références (2)

```
$a = 24;  
$b = &$a;  
  
echo '$a = ' . $a;      // Affiche : $a = 24  
echo '$b = ' . $b;      // Affiche : $b = 24  
  
$a++;  
  
echo '$a = ' . $a;      // Affiche : $a = 25  
echo '$b = ' . $b;      // Affiche : $b = 25  
  
$b = "Hello World";  
  
echo '$a = ' . $a;      // Affiche : $a = Hello World  
echo '$b = ' . $b;      // Affiche : $b = Hello World
```

Les références (3)

Intérêt :

- Modifier des variables passées en argument.
- Éviter de faire des copies de gros tableaux ou de gros objets (car le passage par valeur nécessite une copie).

Fonctions

Exemple :

```
function writeName($nom)
{
    echo "Mon nom est $nom";
}
writeName("Bond");
```

Si la fonction doit renvoyer une valeur : **return**

```
function addition($x, $y)
{
    $somme= $x + $y;
    return $somme;
}
```

return arrête l'exécution de la fonction

Arguments par défaut

Exemple :

```
function langagePrefere($lang = "PHP") {  
    echo "Je préfère $lang";  
}  
langagePrefere();  
langagePrefere("Javascript");
```

→ Je préfère PHP
→ Je préfère Javascript

Attention : Les arguments avec valeur par défaut doivent être **après** les arguments sans valeur par défaut,

Ex :

```
function patisserie($type = "gateau", $parfum) {  
    return $type." -> ".$parfum;  
}  
echo patisserie("fraise");
```

→ fraise -> + warning

Nombre d'arguments variable (1)

Exemple :

```
function patisserie($type)
{
    if (func_num_args() == 1)
        echo "Je veux un $type";

    else if (func_num_args() == 2)
    {
        $args = func_get_args();
        echo "Je veux un $type aux $args[1]";
    }
}
```

Nombre d'arguments

Tableau des arguments

```
patisserie("gâteau");           // Je veux un gâteau
patisserie("gâteau", "abricots"); // Je veux un gâteau aux abricots
```

Nombre d'arguments variable (2)

À partir de **PHP 5.6**, le mot clé « ... » apparaît.

→ Les arguments sont passés sous forme de tableau.

```
function test(...$args)
{
    // Le tableau $args contient tous les arguments.

    echo "Nombre d'arguments : ".count($args)."<br>";
    echo "Les arguments sont :<br>";

    foreach ($args as $a)
        echo $a."<br>";
}
```


Passage par valeur et par référence (1)

Par défaut, les arguments sont passés par valeur :

```
function swap($a, $b)
{
    $c = $a;
    $a = $b;
    $b = $c;
}

$a = "Hello";
$b = "World";

echo $a." ".$b."<br>"; // Affiche "Hello World".
swap($a, $b);         // Pas de changement.
echo $a." ".$b."<br>"; // Affiche "Hello World".
```

Passage par valeur et par référence (2)

Il est possible de passer les arguments par référence :

```
function swap(&$a, &$b)
{
    $c = $a;
    $a = $b;
    $b = $c;
}

$a = "Hello";
$b = "World";

echo $a." " ".$b."<br>"; // Affiche "Hello World".
swap($a, $b);           // Changement.
echo $a." " ".$b."<br>"; // Affiche "World Hello".
```

Variables statiques

- Une variable statique conserve sa valeur d'un appel à l'autre de la fonction :

```
function test()  
{  
    static $a = 0; // Exécutée une seule fois.  
    echo $a;  
    $a++;  
}  
  
// Affiche : 012  
test(); test(); test();
```

- Les variables statiques restent locales à la fonction et ne sont pas réutilisables à l'extérieur

Les objets

Définition d'une classe en PHP 5

```
class Personne {  
    // attributs :  
    var $nom;  
    var $age;  
    // constructeur :  
    function __construct($nom, $age) {  
        $this->nom = $nom;  
        $this->age = $age;  
    }  
    // méthodes diverses :  
    function affiche() {  
        echo "$this->nom a $this->age ans.";  
    }  
}
```

Instanciación et manipulation d'un objet

```
$p = new Personne("Tintin", 30);  
$p->affiche();
```

Classes sans objets

Il est possible d'utiliser une méthode d'une classe sans instancier d'objet.

EX :

```
class A {  
    function bonjour() {  
        Echo "Bonjour !"  
    }  
}  
  
A::bonjour();
```

Utilisation : classes "boîtes à outils" pour des fonctions spécialisées.

EX : fonctions trigonométriques en degrés :

```
Class Degre {  
    function sin($angle) {  
        return sin(deg2rad($angle));  
    }  
}  
  
echo Degre::sin(90);
```

Héritage

Une classe peut être une **extension** d'une autre :
Elle **hérite** de tous les **attributs** et **méthodes** de la classe mère
Et y **ajoute** ses propres **attributs** et **méthodes**

EX :

```
class Etudiant extends Personne {  
    var $annee;  
    function __construct($nom,$age,$annee) {  
        Parent::__construct($nom,$age);    // OU : Personne::__construct($nom,$age);  
        $this->annee = $annee;  
    }  
    function getAnnee() {  
        return $this->annee;  
    }  
}
```

```
$e = new Etudiant("Toto",20,"L2");  
$e->affiche();
```

→ **Toto a 20 ans.**


Héritage

Si la classe fille n'a pas de constructeur,
Elle hérite de celui de la classe mère.

EX :

```
class Etudiant extends Personne
{
    var $annee;
}
```

```
$e = new Etudiant("Toto",20);
```

```
$e->affiche();       Toto a 20 ans.
```

Héritage

Redéfinition des méthodes de la classe mère

```
class Etudiant extends Personne {  
    var $annee;  
    function __construct($nom,$age,$annee) {  
        parent::__construct($nom,$age);  
        $this->annee = $annee;  
    }  
    function affiche() {  
        echo "$this->nom a $this->age ans et est en $this->annee.";  
    }  
}
```

```
$e = new Etudiant("Toto",20,"L2");
```

```
$e->affiche();  Toto a 20 ans et est en L2.
```


Héritage

Redéfinition + ré-utilisation des méthodes de la classe mère

```
class Etudiant extends Personne {  
    var $annee;  
    function __construct($nom, $age, $annee) {  
        parent::__construct($nom, $age);  
        $this->annee = $annee;  
    }  
    function affiche() {  
        parent::affiche(); // OU Personne::affiche();  
        echo " et est en $this->annee."  
    }  
}  
$e = new Etudiant("Toto",20,"L2");  
$e->affiche();
```

Visibilité

- La visibilité d'une propriété ou d'une méthode peut être définie en préfixant sa déclaration avec un mot-clé : *public*, *protected*, ou *private* (à partir de PHP 5).
- Les éléments déclarés comme publics peuvent être utilisés par n'importe quelle partie du programme.
- L'accès aux éléments protégés est limité à la classe elle-même, ainsi qu'aux classes qui en héritent, et à ses classes parentes.
- L'accès aux éléments privés est uniquement réservé à la classe qui les a défini.
- Si une propriété est déclarée en utilisant *var*, elle sera alors définie comme publique (compatibilité avec PHP 4).

Objets et références

Depuis PHP 5, **une variable objet** ne contient plus l'objet en lui-même. Elle **contient seulement un identifiant d'objet**.

Conséquences :

```
EX : class A {public $x = 1 ;}
```

```
$a = new A() ;
```

```
$b = $a ;      → $b copie l'identifiant, donc pointe sur le même objet
```

```
$b->x = 2 ;
```

```
echo $a->x ;    → 2
```

```
function change($obj) {$obj->x = 2 ;}
```

```
change($a) ;
```

```
echo $a->x ;    → 2
```

Conclusion : c'est comme si on manipulait des références sur des objets

Clonage d'objets

Clonage → copie superficielle de toutes les propriétés de l'objet.

EX :

```
class A
{
    public $x = 1;
}

$a = new A();
$b = clone $a;
$b->x = 2;
echo $a->x; // Affiche 1
```

Conclusions :

Le clonage crée un nouvel objet indépendant

Ne pas confondre **\$b = \$a** avec **\$b = clone \$a**