4 Les variables

PLAN

4.1 Notions sur les variables
4.2 Les procédures var_dump() et var_export()
4.3 Le changement explicite de type
4.4 Les variables dynamiques
4.5 La portée des variables
4.6 Quelques fonctions sur les variables

OBJECTIFS

- Comprendre le fonctionnement des variables, la déclaration implicite et les mécanismes de changement de type, automatique ou explicite.
- Comprendre la notion de portée des variables et les variables de session.

4.1 Notions sur les variables

Qu'est-ce qu'une variable ?

Une variable est un espace mémoire que le programme réserve lors de son exécution. Sa déclaration est soit implicite, soit explicite. Une variable possède un nom et un type. Le nom donne accès au contenu, le type précise la taille de l'espace mémoire réservé et le codage binaire utilisé pour lire et écrire la donnée.

La valeur initiale d'une variable

La déclaration d'une variable réserve un espace en mémoire, mais ne « nettoie » pas la zone réservée qui peut contenir des valeurs binaires non nulles, selon que cet espace a été utilisé par un logiciel précédent. La valeur de la variable correspond à ce qui « traîne en mémoire » au moment de la réservation. Sa valeur juste après sa déclaration est indéfinie ! Sa première utilisation doit consister à lui affecter une *valeur initiale*, c'est l'*initialisation* des variables.

Les règles de nommage

En PHP, le nom d'une variable commence toujours par le caractère « $ ». Il est suivi par au moins un caractère puis un nombre quelconque :

- de lettres (A-Z, a-z) ;
- de chiffres (0-9) ;
- du caractère souligné (_).

Voici des exemples de noms de variables valides et non valides :

`$Compteur` : valide ;

`$2Compteur` : non valide ;

`$Compteur_2 :` valide.

Nommez les variables en minuscules, avec éventuellement une majuscule afin de ne pas les confondre avec les constantes. Choisissez des noms basés sur des mots français afin d'éviter d'éventuels conflits avec les mots réservés du langage PHP.

La déclaration des variables

Implicite

En PHP, aucune déclaration explicite n'est nécessaire. Il suffit de nommer une nouvelle variable pour la créer. Son type dépend de son utilisation. Le fait de lui affecter un entier la définit comme une variable entière. Par exemple :

```
$i = 10 ; // Déclaration implicite d'une variable entière $i
```

Selon le contexte, la nature des variables change. Ainsi, il est possible de concaténer un entier avec une chaîne de caractères, ce qui est généralement impossible dans les autres langages. Ce typage automatique possède deux inconvénients :

1. Le type attribué automatiquement n'est pas nécessairement le type désiré. Il faudra alors utiliser une règle de *transtypage explicite* pour convertir la variable.
2. Le type de la variable peut changer en fonction de son usage, et ce, à chaque ligne du programme.

Enfin pour déclarer une variable *non typée*, il faut lui affecter la valeur *null* :

```
$v = null ; // Variable non typée
```

Explicite dans une classe

Avec la POO (Programmation orientée objet), les variables (propriétés) sont déclarées explicitement, *via* les mots-clefs *var*, *private, protected* et *public*.

Le programme `variables_class_shell.php` déclare deux variables `variable1`, `variable2` dans la classe `Traitement`. La fonction *constructeur* leur affecte les

valeurs 1 et 2. Le programme principal affiche leur contenu *via* les fonctions (méthodes) `get1()` et `get2()`. La syntaxe objet est détaillée au chapitre 9.

Listing 4.1 – Programme variables_class_shell.php

```
<?php
 class Traitement
 { // Déclaration de deux propriétés
   private $variable1;
   public  $variable2;
   // Constructeur
   function __construct(){
    $this->variable1=1;
    $this->variable2=2;
   }
  public function get1() {return $this->variable1;}
  public function get2() {return $this->variable2;}
 } // fin de la classe
 // -- Création de l'objet --
 $MesV = new Traitement();
 // -- Affichage --
 echo "Première valeur  : ".$MesV->get1()."\n";
 echo "Deuxième valeur  : ".$MesV->get2()."\n";
?>
```

Le typage automatique

Le type d'une variable change selon la nature des traitements comme le montre le programme `var_typage_automatique_shell.php`.

Listing 4.2 – Programme var_typage_automatique_shell.php

```
<?php
$var = "0";     // $var est une chaîne de caractères
echo '$var type chaîne : '.$var."\n";
$var += 5;     // $var est maintenant un entier (5)
echo '$var type entier : '.$var."\n";
$var = $var + 1.3;  // $var devient un nombre réel (6.3)
echo '$var type réel   : '.$var."\n";
$var = 5 + "10 ordinateurs"; // $var devient un entier (15)
echo '$var type entier : '.$var."\n";
?>
```

Voici l'exécution de ce programme :

Listing 4.3 – Exécution de var_typage_automatique_shell.php

```
$ php var_typage_automatique_shell.php
$var type chaîne : 0
$var type entier : 5
$var type réel   : 6.3
$var type entier : 15
```

4.2 Les procédures var_dump() et var_export()

var_dump()

La déclaration implicite et le changement de type selon les traitements peuvent induire en erreur le développeur sur le vrai type de la variable.

La procédure `var_dump()`indique quels sont le type et la valeur de la variable au moment de l'exécution.

Le programme `var_dump_shell.php` présente sa mise en œuvre.

Listing 4.4 – Programme var_dump_shell.php

```
<?php
 $var1="texte";
 echo '$var1="texte"   : ';
 var_dump($var1);
 $var2=18;
 echo '$var2=18        : ';
 var_dump($var2);
 $var3=3.1415926535;
 echo '$var3=3.1415926 : ';
 var_dump($var3);
 $var4="-26";
 echo '$var4="-26"     : ';
 var_dump($var4);
 echo '$var4=$var4+0   : ';
 $var4=$var4+0;
 var_dump($var4);
?>
```

Son exécution montre que la variable :

- `$var1` est reconnue comme une chaîne de caractères (string) ;
- `$var2` est défini comme un entier (int) ;
- `$var3` est de type réel (float) ;
- `$var4` est une chaîne de caractères (string) après son affectation de "–26" ;
- `$var4` devient un entier (int) après l'opération arithmétique `$var4=$var4+0` ;

Listing 4.5 – Exécution de var_dump_shell.php

```
$ php var_dump_shell.php
$var1="texte"   : string(5) "texte"
$var2=18        : int(18)
$var3=3.1415926 : float(3.1415926535)
$var4="-26"     : string(3) "-26"
$var4=$var4+0   : int(-26)
```

var_export()

La procédure `var_export()` est identique à `var_dump()` mais l'information affichée se présente sous la forme d'une syntaxe PHP valide. Ainsi les syntaxes :

```
$var1=18;
var_export($var1);
echo "\n";
$var2=3.1415926535;
var_export($var2);
echo "\n";
$var3="-26";
var_export($var3);
echo "\n";
```

afficheront :

```
18
3.1415926535000001
'-26'
```

4.3 Le changement explicite de type

Avec settype()

Même si le type d'une variable est défini implicitement, il est possible de le forcer *via* la fonction `settype()`. La syntaxe est :

```
settype($i, "float") ;
```

Les types autorisés sont :

- `"boolean"`, ou `"bool"` depuis PHP 4.2.0 ;
- `"integer"`, ou `"int"` depuis PHP 4.2.0 ;
- `"float"`, uniquement depuis PHP 4.2.0. Avant il fallait utiliser `"double"` ;
- `"string"` ;
- `"array"` ;
- `"object"` ;
- `"NULL"`, depuis PHP 4.2.0.

Voici quelques exemples :

```
$i = "7 nains"; // chaîne de caractères
$j = 2.45     ; // réel
$k = true     ; // booléen
settype($i, "integer"); // $i vaut maintenant 7   (integer)
settype($j, "int")    ; // $j vaut maintenant 2   (integer)
settype($k, "string") ; // $k vaut maintenant "1" (string)
```

Par transtypage

La fonction `settype()` modifie le type de la variable. Mais si on désire modifier l'interprétation de la variable dans un calcul, sans en changer son type, il suffit de faire précéder son nom par le type désiré entre parenthèses.

Dans l'exemple suivant, le type de `$x` reste inchangé, alors que `$i` reçoit *l'interprétation entière* de `$x`, ce qui la définit comme une variable entière.

```
$i = (int) $x ;
```

Les préfixes autorisés sont :

- `(int)`, `(integer)` : modification en entier (integer) ;
- `(bool)`, `(boolean)` : modification en booléen (boolean) ;
- `(float)`, `(double)`, `(real)` : modification en réel (float) ;
- `(string)` : modification en chaîne de caractères (string) ;
- `(array)` : modification en tableau (array) ;
- `(object)` : modification en objet (object) ;
- `(unset)` : modification en NULL (en PHP 5) ;
- `(binary)` : modification en binaire (en PHP 5). On peut aussi utiliser le préfixe b. pour les chaînes de caractères.

Ces transtypages sont détaillés avec les types simples et structurés.

4.4 Les variables dynamiques

Il s'agit d'une variable qui contient le nom d'une autre variable. Si la variable `$plat` contient le nom d'une variable, alors l'accès au nom de cette variable se note : `$plat`, et l'accès à sa donnée se note : `$$plat`.

Le programme `variables_dynamiques_shell.php` en présente un exemple :

Listing 4.6 – Programme variables_dynamiques_shell.php

```
<?php
// Définition des variables
$viande=25;
$poisson=30;
// Définition de la variable dynamique
$plat='viande';
// Affichage
echo "Nom du plat : ".$plat."\n";
echo "Prix du plat : ".$$plat."\n";
?>
```

Voici son exécution :

Listing 4.7 – Exécution de variables_dynamiques_shell.php

```
$ php variables_dynamiques_shell.php
Nom du plat : viande
Prix du plat : 25
```

4.5 La portée des variables

La durée de vie ou la visibilité (portée) d'une variable dépend de sa nature. Généralement, elle est créée et visible durant l'exécution du programme où elle est déclarée et supprimée à la fin du programme. Elle est locale au programme. Elle peut également être locale au contexte de sa création (une fonction, une classe d'objet) et être invisible aux autres parties du programme.

Une variable est : locale, globale, ou locale statique. Le langage PHP propose également les variables super globales et les variables de session.

Variables locales

Une variable locale est déclarée dans un contexte, comme le bloc principal, ou un sous-programme, et elle n'est connue que de ce contexte. Elle est créée à l'appel du sous-programme et détruite à la fin de celui-ci. À chaque nouvel appel du sous-programme, elle est de nouveau créée. Le programme `variables_locales_shell.php` présente cette notion.

Listing 4.8 – Programme variables_locales_shell.php

```
<?php
 $a=1;
 $b=2;
 echo 'Dans programme : $a='.$a.'  $b='.$b."\n";
 echo "Retour de la fonction somme()=".somme()."\n";
 echo '$loc1='.$loc1."\n";
 function somme()
 {echo "----Dans la fonction---\n";
  $result=$a+$b;
  $loc1=10;
  echo '$a='.$a.' et $b='.$b.' result='.$result."\n" ;
  echo 'et $loc1='.$loc1."\n";
  echo "-----------------------\n";
  return $result;
 }
?>
```

Lors de l'exécution, les variables `$a` et `$b` déclarées dans le programme sont connues du programme mais inconnues de la fonction `somme()`. De même, `$loc1` est connue de la fonction où elle est déclarée, et inconnue du programme.

Listing 4.9 – Exécution de variables_locales_shell.php

```
$ php variables_locales_shell.php
Dans programme : $a=1  $b=2
----Dans la fonction---
$a= et $b= result=0
et $loc1=10
-----------------------
Retour de la fonction somme()=0
$loc1=
```

Variables locales statiques

Une variable locale statique est une variable locale, elle n'est connue que dans son contexte de déclaration. Elle est créée au premier appel du sous-programme, mais à la différence des variables locales « simples », elle n'est pas supprimée à la fin du sous-programme et conserve sa valeur lors des appels successifs. Sa déclaration est précédée du mot `static`. Le programme `variables_statiques_shell.php` présente son traitement.

Listing 4.10 – Programme variables_statiques_shell.php

```
<?php
 $a=1; $b=2;
 echo 'Somme('.$a.','.$b.')='.somme($a,$b)."\n" ;
 $a=10; $b=20;
 echo 'Somme('.$a.','.$b.')='.somme($a,$b)."\n" ;
 function somme($a,$b)
 {static $compteur=1;
  echo "Appel : ".$compteur."\n";
  $result=$a+$b;
  $compteur++;
  return $result;
 }
?>
```

Son exécution montre que la variable `$compteur` existe toujours lors du deuxième appel de la fonction `somme()` et que sa valeur progresse à chaque appel.

Listing 4.11 – Exécution de variables_statiques_shell.php

```
$ php variables_statiques_shell.php
Appel : 1
Somme(1,2)=3
Appel : 2
Somme(10,20)=30
```

Variables globales

Une variable globale est connue de tous les sous-programmes. Elle est *définie* dans le programme et est « *redéfinie* » comme globale dans les sous-programmes, *via* le mot `global` comme dans le programme `variables_globales1_shell.php` :

Listing 4.12 – Programme variables_globales1_shell.php

```
<?php
 $a=1; $b=2;
 echo '$a='.$a."\n";
 echo '$b='.$b."\n";
 echo "Au retour de l'appel somme()=".somme();
 echo ' et $loc1='.$loc1."\n";
 function somme()
 {global $a, $b ;
  echo "-----------------------\n";
  $result=$a+$b;
  $loc1=10;
  global $loc1;
  echo 'Dans somme() $a='.$a.', $b='.$b.', result='.$result ;
  echo ' et $loc1='.$loc1."\n";
  echo "-----------------------\n";
  return $result;
 }
?>
```

Lors de l'exécution, les variables `$a` et `$b`, qui sont déclarées et initialisées dans le programme, sont connues du programme et connues de la fonction `somme()`. À l'inverse, la déclaration de `$loc1` comme `globale` provoque son invisibilité dans la fonction, car elle n'a pas été créée auparavant dans le programme.

Listing 4.13 – Exécution de variables_globales1_shell.php

```
$ php variables_globales1_shell.php
$a=1
$b=2
-----------------------
Dans somme() $a=1, $b=2, result=3 et $loc1=
-----------------------
Au retour de l'appel somme()=3 et $loc1=
```

La déclaration de variables globales peut également se faire grâce au tableau associatif prédéfini `$GLOBAL[ ]`. La fonction somme précédente se réécrit (sans les affichages) :

```
function somme()
{
  $result=$GLOBALS["a"]+$GLOBALS["b"]; // Variables globales
  return $result;
}
```

Variables super globales

Les variables super globales sont des tableaux associatifs. Elles sont internes au langage PHP, et sont *toujours accessibles* quel que soit le contexte. C'est par exemple le tableau `$GLOBALS[ ]` utilisé dans l'exemple précédent.

Les variables prédéfinies du langage ne sont pas toutes super globales.

Les variables super globales sont :

- `$GLOBALS` : variables globales du programme, mises à jour par le développeur ;
- `$_SERVER` : valeurs renvoyées par le serveur ;
- `$_GET` : valeurs envoyées *via* l'URL, ou le formulaire en méthode GET ;
- `$_POST` : valeurs envoyées *via* un formulaire en méthode POST ;
- `$_FILES` : liste des fichiers envoyés par le formulaire précédent ;
- `$_COOKIE` : valeurs des cookies enregistrés sur l'ordinateur du client ;
- `$_SESSION` : variables de session, stockées sur le serveur durant la session ;
- `$_REQUEST` : valeurs de `$_GET`, `$_POST` et `$_COOKIE` ;
- `$_ENV` : valeurs d'environnement renvoyées par le serveur ;

L'utilisation des variables `$_GET` et `$_POST` a été présentée au chapitre 3.

Variables de session

Les **variables de session** sont stockées dans la variable super globale, `$_SESSION[ ]`, qui existe durant toute la session du client (visiteur du site). Ce tableau associatif transmet les données d'une page PHP à une autre page PHP. Le développeur peut y conserver les données individuelles d'un utilisateur qui navigue sur le site, *via* un identifiant unique de session. Le programme `variables_session.php` présente les variables de session :

Listing 4.14 – Programme variables_session.php

```
<?php
 session_start();
?>
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8" />
  <title>Variables de session</title>
 </head>
 <body>
 <!-- Début du programme PHP -->
```

```
  if (!$_SESSION['nom'])
  {
   $_SESSION['nom'] = "Dupont";
   echo "Premier acc&egrave;s au site<br/>";
   echo "Bonjour Monsieur ".$_SESSION['nom']."<br/>";
  }
  else
  {
   echo "Second acc&egrave;s au site<br/>";
   echo "Bonjour Monsieur ".$_SESSION['nom'].", content de vous
revoir<br/>";
   unset($_SESSION['nom']);
   session_destroy();
  }
 ?>
 <!-- Fin du programme PHP -->
 </body>
</html>
<?php
```

L'instruction `session_start()`, démarre la session.

Elle doit impérativement se trouver avant toute syntaxe HTML.

Le test `if (!$_SESSION['nom'])` vérifie que l'entrée `nom` n'existe pas dans le tableau associatif `$_SESSION[ ]`. Si c'est le cas, on crée cette entrée et on affiche un message indiquant que c'est le premier accès au site. Si cette entrée existe dans le tableau `$_SESSION[ ]`, alors c'est la deuxième visite, on affiche un message en conséquence, puis on détruit la session *via* `session_destroy()`. La visite suivante sera considérée comme une première visite. Si l'on quitte le navigateur, la session est alors terminée. La figure 4.1 présente les écrans du premier et du second accès au site.

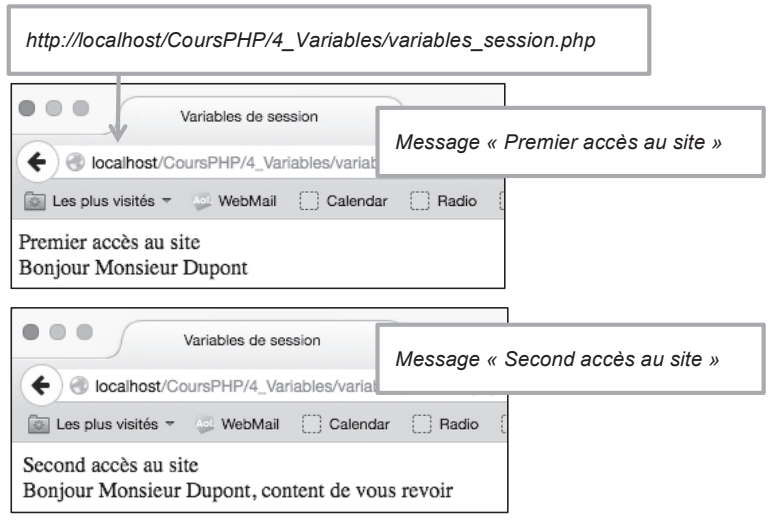

Figure 4.1 – Variable de session. Premier et second accès.

Le programme `session_prog_principal.php` présente un second exemple de gestion de variables de session. Il mémorise dans la variable `$_SESSION[ ]`, le nom, prénom et l'âge d'une personne, et affiche deux liens vers les pages `session_mapage.php` et `session_informations.php`.

Listing 4.15 – Programme session_prog_principal.php

```
<?php
// On démarre la session AVANT d'écrire du code HTML
session_start();
// On crée quelques variables de session dans $_SESSION
$_SESSION['prenom'] = 'Jean';
$_SESSION['nom']    = 'Dupont';
$_SESSION['age']    = 24;
?>
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8" />
  <title>Page principale</title>
 </head>
 <body>
   Bonjour <?php echo $_SESSION['prenom']; ?> !<br />
   Bienvenue sur la page d'accueil de mon site. <br />
   Pour aller sur une page cliquez sur un des liens.<br />
   <a href="session_mapage.php">Lien vers ma page</a><br />
   <a href="session_informations.php">Vos informations</a>
 </body>
</html>
```

Le programme `session_mapage.php` récupère et affiche le **nom** et le **prénom**.

Listing 4.16 – Programme session_mapage.php

```
<?php
session_start(); // On démarre la session AVANT toute chose
?>
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
  <meta charset="utf-8" />
  <title>Ma Page HTML</title>
 </head>
 <body> <!-- Corps de la page HTML -->
  <h3> Ma page HTML</h3>
   Bienvenue sur ma page<br/>
   Monsieur <?php echo $_SESSION['prenom'] . ' ' . $_SESSION['nom'];
?>
 </body>
</html>
```

Le programme `session_informations.php` affiche les informations de la personne.

Listing 4.17 – Programme session_informations.php

```
<?php
session_start(); // On démarre la session AVANT toute chose
?>
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8" />
  <title>Informations</title>
 </head>
 <body>
  <h3>Vos informations :</h3>
   Vous vous appelez <?php echo $_SESSION['prenom'] . ' ' . $_
   SESSION['nom']; ?> !<br/>
   Vous avez <?php echo $_SESSION['age']; ?> ans
 </body>
</html>
```

La figure 4.2 présente l'organisation de ces trois pages.

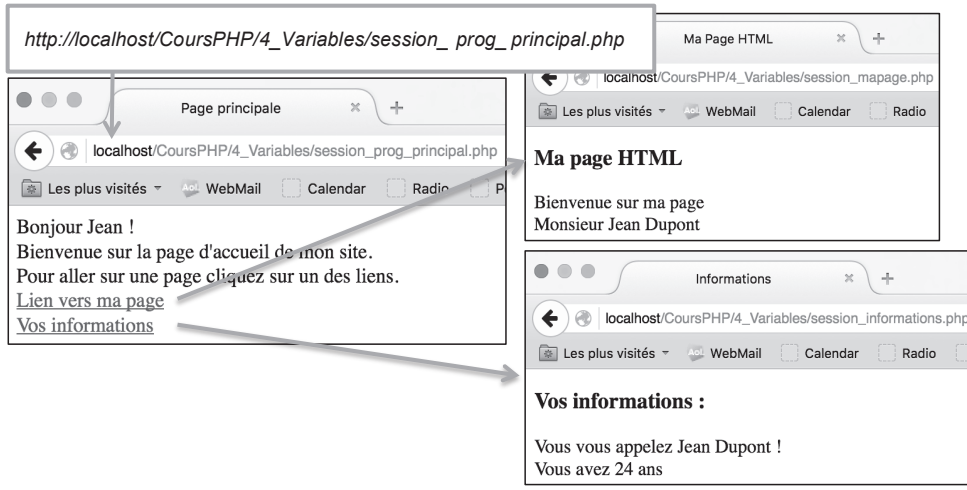

Figure 4.2 – Variable de session. Transmission des informations.

Voici la liste des fonctions gérant les variables de session (source : *http://php.net/manual/fr/ref.session.php*) :

- `session_abort` : supprime les modifications et termine la session ;
- `session_cache_expire` : retourne la configuration actuelle du délai d'expiration du cache ;
- `session_cache_limiter` : lit et/ou modifie le limiteur de cache de session ;
- `session_commit` : alias de `session_write_close` ;
- `session_decode` : décode les données encodées de session ;
- `session_destroy` : détruit une session ;
- `session_encode` : encode les données de session ;
- `session_get_cookie_params` : lit la configuration du cookie de session ;
- `session_id` : lit et/ou modifie l'identifiant courant de session ;
- `session_is_registered` : vérifie si une variable est enregistrée dans la session ;
- `session_module_name` : lit et/ou modifie le module de session courant ;
- `session_name` : lit et/ou modifie le nom de la session ;
- `session_regenerate_id` : remplace l'identifiant de session par un nouveau ;
- `session_register_shutdown` : fonction de fermeture de session ;
- `session_register` : enregistre une variable globale dans une session ;
- `session_reset` : réinitialise le tableau de session avec les valeurs d'origine ;
- `session_save_path` : lit et/ou modifie le chemin de sauvegarde des sessions ;
- `session_set_cookie_params` : modifie les paramètres du cookie de session ;
- `session_set_save_handler` : configure les fonctions de stockage de sessions ;
- `session_start` : démarre une nouvelle session ;
- `session_status` : détermine le statut de la session courante ;
- `session_unregister` : supprime une variable de la session ;
- `session_unset` : détruit toutes les variables d'une session ;
- `session_write_close` : écrit les données de session et ferme la session.

4.6 Quelques fonctions sur les variables

Le tableau 4.1 présente quelques fonctions sur les variables.

Tableau 4.1 – Quelques fonctions sur les variables

Fonction	Signification	Exemple
`isset()`	Retourne vrai si la variable a été affectée faux sinon.	`if (isset($i))...`
`empty()`	Retourne vrai si la variable est vide faux sinon.	`if (empty($i))...`
`is_numeric()`	Retourne vrai si l'argument est un numérique faux sinon.	`if (is_numeric($i)) …`
`is_string()`	Vrai si l'argument est une chaîne de caractères, faux sinon.	`if (is_string($ch)) …`
`is_array()`	Retourne vrai si l'argument est un tableau, faux sinon.	`if (is_array($tab)) …`
`is_bool()`	Retourne vrai si l'argument est un booléen, faux sinon.	`if (is_bool($b1)) …`
`is_float()`	Retourne vrai si l'argument est un réel, faux sinon.	`if (is_float($x)) …`
`is_int()`	Retourne vrai si l'argument est un entier, faux sinon.	`if (is_int($x)) …`
`is_scalaire()`	Vrai si c'est un scalaire (integer, float, string, boolean) faux sinon.	`if (is_scalaire($b1)) …`
`is_null()`	Vrai si la variable est : de valeur « null », sans aucune valeur ou supprimée.	`if (is_null($MaVar)) …`
`is_objet()`	Vrai si c'est un objet, faux sinon.	`if (is_objet($ob)) …`
`is_ressource()`	Vrai si c'est une ressource, faux sinon.	`if (is_ressource($ob)) …`
`unset()`	Supprime la variable.	`unset($var1)`