Développement web S1
– 2. Constantes, variables, types, opérateurs et structures de contrôle –

Christophe BLANC

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

2023-2024

Constantes

Définition

La fonction *define* permet de définir une constante.
Une constante est une zone mémoire identifiée par un nom qui contient une valeur initiale mais non modifiable dans le programme.

Syntaxe

```
booleen define(chaine nom, mixte valeur, booleen sensibleCasse)
```

- ✔ nom : nom de la constante
- ✔ valeur : valeur de la constante
- ✔ sensibleCasse : indique si le nom de la constante est sensible à la casse (TRUE valeur par défaut) ou non (FALSE)

Tout type de données scalaires peut être utilisé comme type de donnée d'une constante.

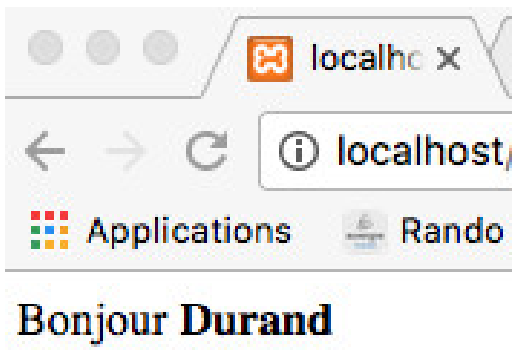
Exemple

```php
<?php
define("NOMCONSTANTE", "valeurConstante");
define("PI", 3.1415);
?>
```

- ✔ Le nom d'une constante ne doit pas commencer par un $ car ce préfixe est réservé au nom des variables. Définir une constante dont le nom commence par un $ ne génère pas d'erreur. Cependant, à l'utilisation, la constante sera vue comme une variable non initialisée.
- ✔ Une fois créée, une constante n'est plus modifiable, ni par un nouvel appel à define , ni par une affectation directe.
- ✔ Utiliser une constante non définie ou tenter de redéfinir une constante déjà définie génère une erreur.

La portée d'une constante est le script dans lequel elle est définie. Une constante peut donc être définie dans une première section de code PHP et utilisée dans une autre section de code PHP.

```
<?php
// definition d'une constante
define("NOM","Durand");
?>
<HTML>
<BODY>
Bonjour <B><?php echo NOM; ?></B>
</BODY>
</HTML>
```



La fonction *defined* permet de savoir si une constante est définie ou non. Syntaxe

```
booleen defined(chaine nom)
```

- ✔ nom : nom de la constante

defined retourne *TRUE* si la constante est définie et *FALSE* sinon.

Exemple

```
<?php
$ok = defined("NOMCONSTANTE");
if ($ok)
{
        echo "NOMCONSTANTE est definie.<BR>";
}
else
{
        echo "NOMCONSTANTE n'est pas definie.<BR>";
}
define("NOMCONSTANTE", "valeurConstante");
$ok = defined("NOMCONSTANTE");
if ($ok)
{
        echo "NOMCONSTANTE est definie.<BR>";
}
else
{
        echo "NOMCONSTANTE n'est pas definie.<BR>";
}?>
```

Cet exemple utilise la structure de contrôle *if* (vu plus tard dans le cours) qui permet de tester une condition et d'agir en conséquence.

La fonction *constant* retourne la valeur d'une constante dont le nom est passé en paramètre.

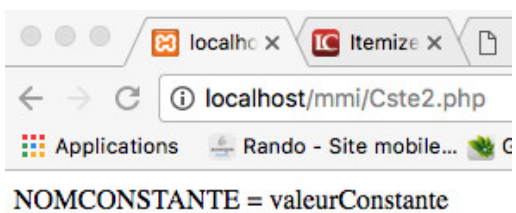
Syntaxe

```
valeur constant(chaine nom)
```

- ✔ nom : nom de la constante

Cette fonction est pratique pour récupérer la valeur d'une constante dont le nom n'est pas connu a priori. Exemple

```
<?php
$nomConstante = "NOMCONSTANTE";
define($nomConstante,"valeurConstante");
echo $nomConstante," = ",constant($nomConstante);
?>
```



Variables

Définition

Une variable est une zone mémoire identifiée par un nom qui contient une valeur lisible ou modifiable dans le programme

- ✔ Les variables sont identifiées par le préfixe $ suivi d'un nom qui respecte les règles de nommage.
- ✔ Le nom des variables est sensible à la casse : $nom et $Nom sont vues par PHP comme deux variables différentes. En cas d'utilisation d'une mauvaise sytaxe, une nouvelle variable vide est créée avec une simple erreur qui n'est pas forcément affichée. Il est donc nécessaire d'adpoter une convention de nommage :
  1. tout en minuscules ($nom)
  2. première lettre en majucule et le reste en minuscules ($Nom)
  3. première de chaque mot en majuscule et le reste en minuscules ($NomDeFamille)

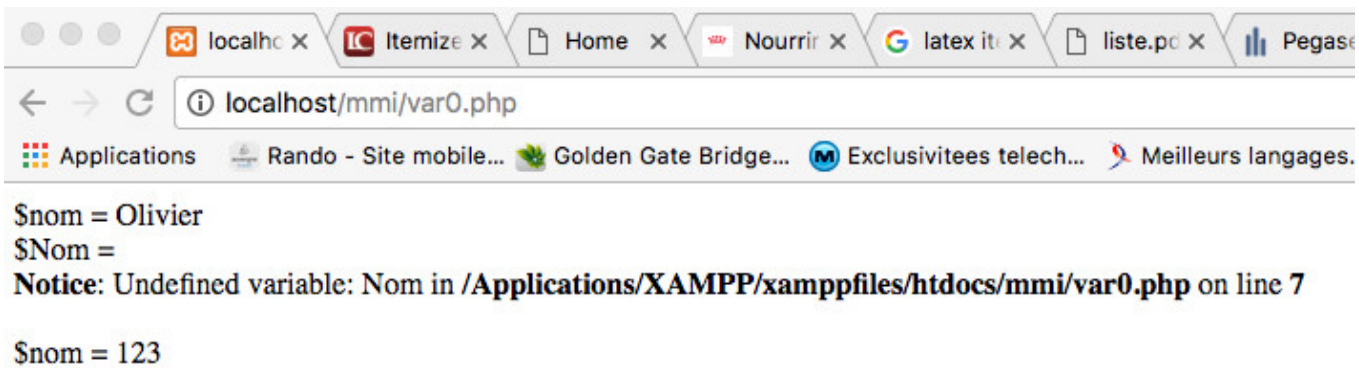
Exemple

```
<?php
$var; $_var; $Var; //Nom de variables valides
$123var; $456 ;$my-test; //Pas valide
?>
```

Variables

Exemple

```php
<?php
//initialiser une variable $nom
$nom = "Olivier";
//afficher la variable $nom
echo "\$nom = ",$nom,"<BR>";
//afficher la variable $Nom
echo "\$Nom = ",$Nom,"<BR>";
//modifier la valeur (et le type) de la variable $nom
$nom = 123;
//afficher la variable $nom
echo "\$nom = ",$nom,"<BR>";
?>
```



- ✔ La portée d'une variable est le script dans lequel elle est définie. Une variable peut donc être définie dans une première section de code PHP et utilisée dans une autre section de code PHP.
- ✔ La durée de vie d'une variable est le temps d'exécution du script. Lorsque le script se termine, les variables sont supprimées. Si le même script est appelé plus tard, de nouvelles variables sont définies.

```php
<?php
// Declaration
$foo = "hello";
$bar = 12;

// Concatenation
$boom = $foo . $bar;//$boom is equals to "hello12"
$baz = $foo . "world";//$baz is equals to "helloworld

// Destruction
unset($foo);//$foo doesn't exist anymore
?>
```

PHP propose un certain nombre de fonction utile sur les variables :

Nom	**Rôle**
empty	Indique si une variable est vide ou non
isset	Indique si une variable est définie ou non
unset	Supprime une variable
var_dump	Affiche des informations sur une variable (type et valeur)

La fonction *empty* permet de tester si une variable est vide ou non.

Syntaxe

```
booleen empty(mixte variable)
```

- ✔ variable : variable à tester

empty retourne *TRUE* si la variable est considérée comme vide et *FALSE* dans le cas contraire.
Une variable est considérée comme vide si elle n'est pas affectée ou si elle contient une chaîne vide (""), une chaîne égale à 0("0"), 0, *NULL* ou *FALSE*

La fonction *isset* permet de tester si une variable est définie ou non.

Syntaxe

```
booleen isset(mixte variable)
```

- ✔ variable : variable à tester

isset retourne *TRUE* si la variable est définie et *FALSE* dans le cas contraire.
Une variable est considérée comme non définie si elle n'a pas été affectée ou si elle contient *NULL*. A la différence de la fonction *empty*, une variable qui contient une chaîne vide (""), une chaîne égale à 0 ("0"), un 0 ou *FALSE*, n'est pas considérée comme non définie.

La fonction *unset* permet de supprimer une variable.

Syntaxe

```
unset(mixte variable,...)
```

- ✔ variable : variable à supprimer (éventuellement plusieurs séparées par des virgules)

Après suppression, la variable se trouve dans le même état que si elle n'avait jamais été affectée. L'utilisation de la fonction *isset* sur une variable supprimée retourne *FALSE*.

PHP propose quatre types de données scalaires (ne pouvant contenir qu'une valeur), deux types composés (pouvenant contenir plusieurs valeurs) et deux types spéciaux :

- ✔ Types scalaires :
  1. nombre entier
  2. nombre à virgule flottante
  3. chaîne de caractère
  4. booléen
- ✔ Types composés :
  1. tableaux (vu dans un prochain cours)
  2. objet (vu dans un prochain cours)
- ✔ Types spéciaux :
  1. NULL
  2. Ressource

- ✔ Le type entier permet de stocker un nombre entier signé sur 32 bits (ou 64 bits : dépendant du système), soit des valeurs comprises entre -2 147 483 648 (-2^{31}) et 2 147 483 648 (2^{31}).
- ✔ En cas de dépassement de capacité dans un calcul, le résultat est automatiquement converti en nombre en virgule flottante.

```php
<?php
$foo = 12;
?>
```

- ✔ Le type nombre à virgule flottante (float) permet de stocker un nombre décimal sur une plage de valeurs dépendante de la plate-forme (nombre maximal 1.8e308 avec une précision sur 14 chiffres sur architecture 64 bits)
- ✔ Un tel nombre peut être exprimé en notation décimale x,y (par exemple 123.456) ou en notation scientifique x,yEz (par exemple 1.23456E2)
- ✔ En cas de conversion d'un nombre à virgule flottante en entier, le nombre est tronqué (pas arrondi) à l'entier le plus proche (1.9 donne 1).

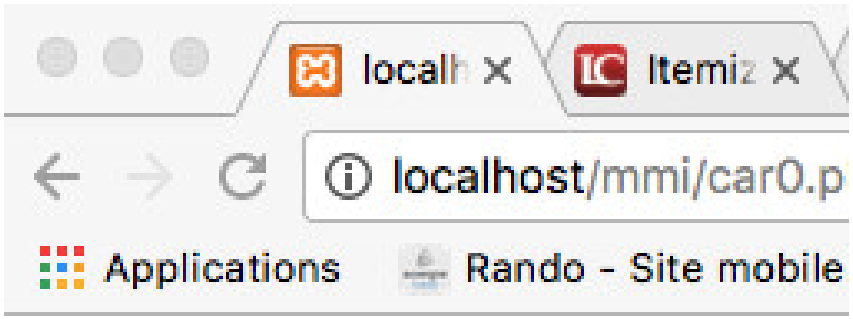
```php
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

- ✔ Le type chaîne de caractères (*string*) permet de stocker toute séquence de caractères sur un octet (code ascii entre 0 et 255), sans limite de taille.
- ✔ Une expression littérale de type chaîne de caractères peut être spécifiée entre guillemets ou entre apostrophes avec des différences de comportement.
- ✔ Les guillemets présents dans une chaîne délimitée par des guillements ou les apostrophes présents dans une chaîne délimitée par des apostrophes doivent être "échappés", c'est-à-dire précédés du caractère anti-slash (\). En complément, un anti-slash présent en fin de chaîne, juste avant le guillemt ou l'apostrophe finaux, doit lui aussi être échappé par un anti-slash.

Types

Chaîne de caractères - Exemple

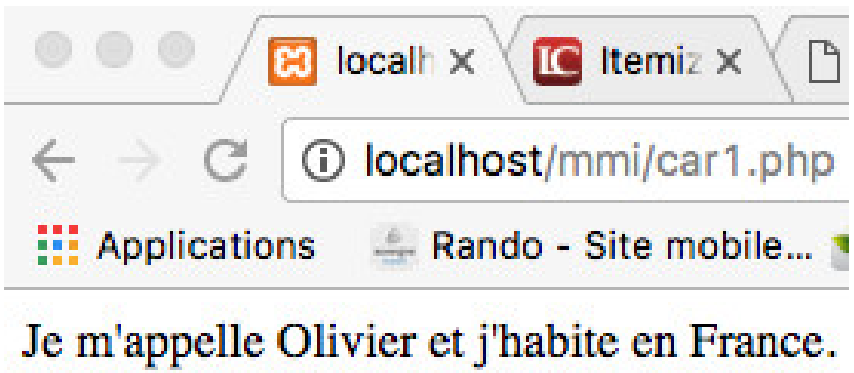
```php
<?php
echo "c'est␣l'ete␣=>␣pas␣de␣probleme.<BR>";
echo 'je␣dis␣"bonjour"␣=>␣pas␣de␣probleme.<BR>';
// echo 'c'est l'ete => erreur de compilation.<BR>';
echo 'c\'est␣l\'ete␣=>␣probleme␣corrige.<BR>';
// echo "je dis "bonjour" => erreur de compilation.<BR>";
echo "je␣dis␣\"bonjour\"␣=>␣probleme␣corrige.<BR>";
// echo "c:\"," => "erreur de compilation.<BR>";
echo "c:\\","␣=>␣probleme␣corrige.<BR>";
// echo 'c:\',"erreur de compilation.<BR>";
echo 'c:\\',"␣=>␣probleme␣corrige.<BR>";
?>
```



Une chaîne peut être saisie sur plusieurs lignes.

```
<?php
$chaine = "Je␣m'appelle␣Olivier
et␣j'habite␣en␣France.";
echo $chaine;
?>
```

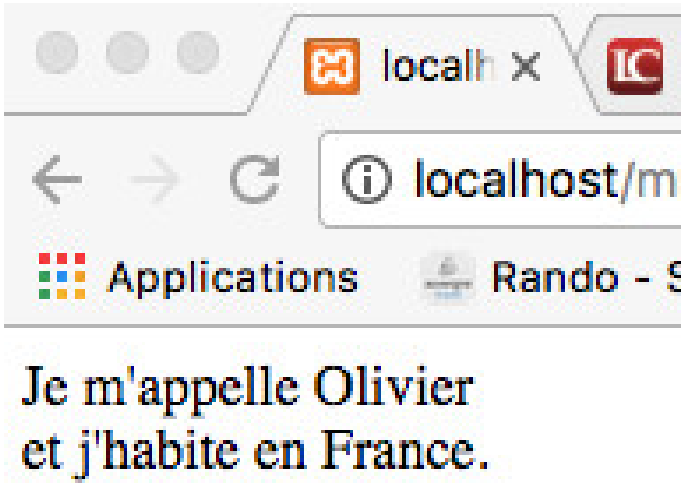
Navigateur



Source

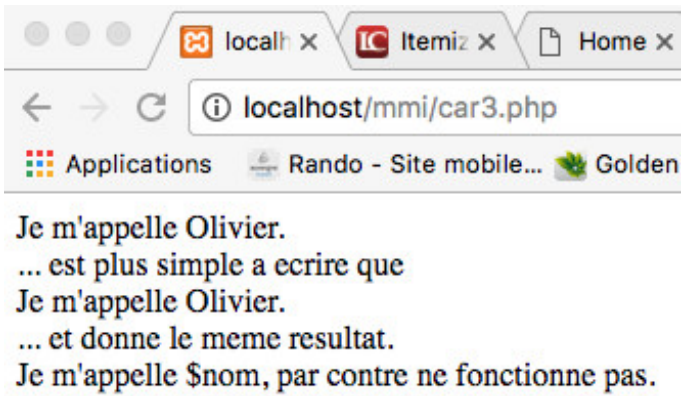
Le retour à la ligne présent dans la chaîne initiale est retrouvé dans le source de la page mais n'est pas interpété comme tel par le navigateur. Il faut une balise pour obtenir un retour à la ligne dans le résultat affiché.

```
<?php
$chaine = "Je␣m'appelle␣Olivier<BR>
et␣j'habite␣en␣France.";
echo $chaine;
?>
```



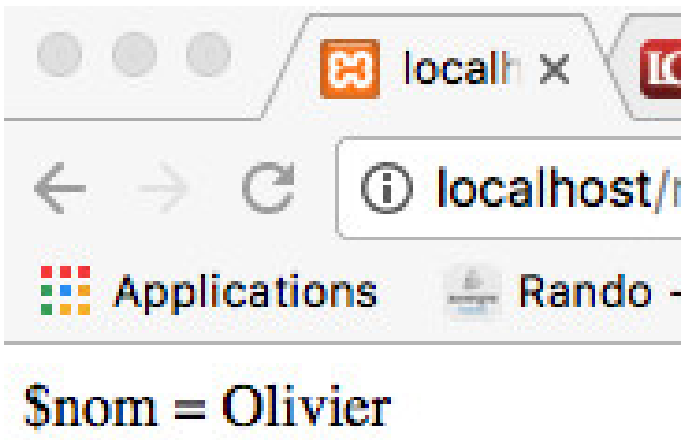
Lorqu'une chaîne est délimitée par des guillemets, toute séquence de caractères commençant par le signe $ est interprétée comme une variable et remplacée par la valeur de cette variable : c'est le mécanisme de substitution des variables par leur valeur. Cette fonction, très pratique, ne fonctionne pas avec les chaînes délimitées par des apostrophes.

```
<?php
$nom = "Olivier";
echo "Je␣m'appelle␣$nom.<BR>";
echo "...␣est␣plus␣simple␣a␣ecrire␣que<BR>";
echo "Je␣m'appelle␣",$nom, ".<BR>";
echo "...␣et␣donne␣le␣meme␣resultat.<BR>";
echo 'Je␣m\'appelle␣$nom,␣par␣contre␣ne␣fonctionne␣pas.<BR>';
?>
```



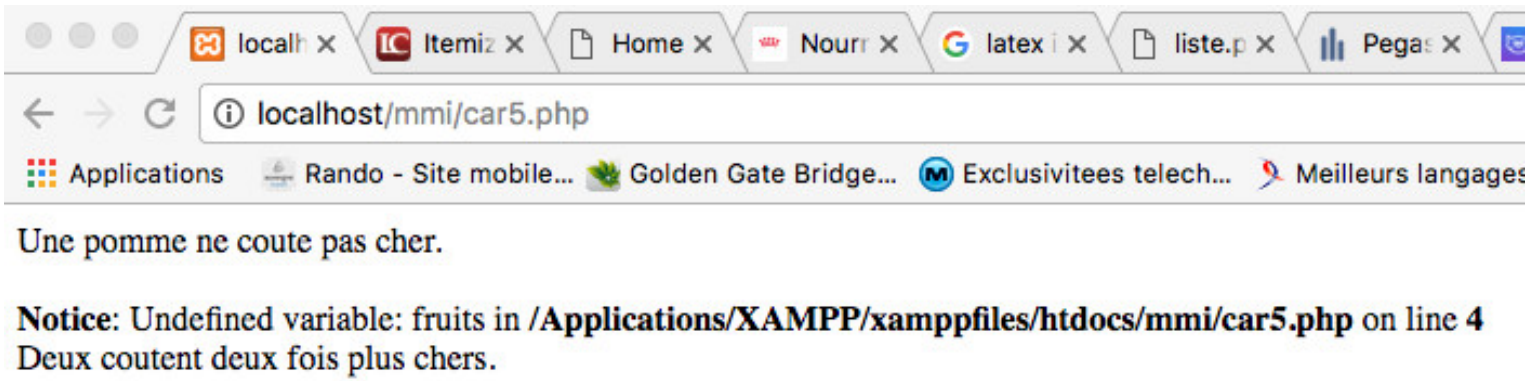
Dans certains cas, ce comportement peut ne pas être désiré. Il suffit d'échapper le signe avec l'anti-slash pour qu'il se comporte comme un $.

```
<?php
$nom = "Olivier";
echo "\$nom = $nom";
?>
```



Dans d'autres cas, le comportement peut être souhaité avec le besoin d'accoler du texte complémentaire derrière le nom de la variable.

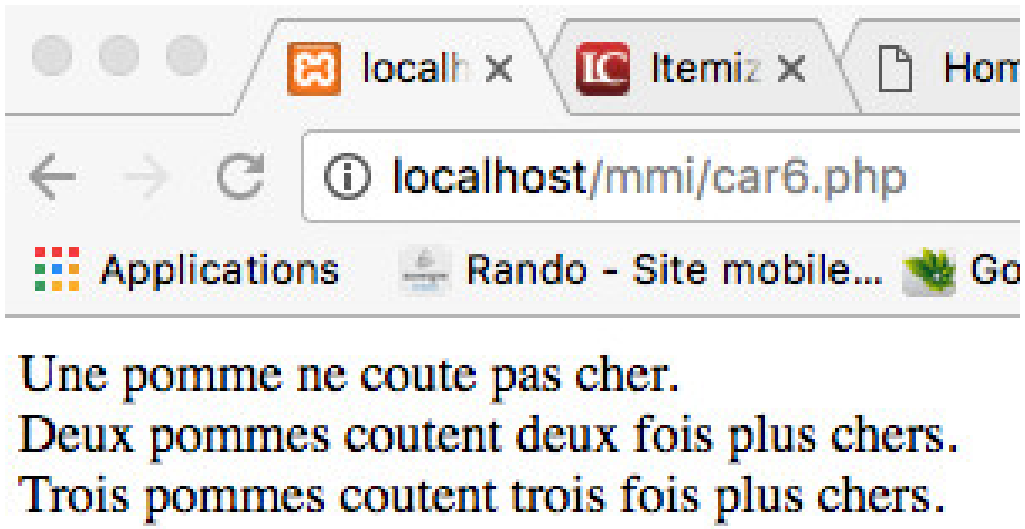
```
<?php
$fruit = "pomme";
echo "Une␣$fruit␣ne␣coute␣pas␣cher.<BR>";
echo "Deux␣$fruits␣coutent␣deux␣fois␣plus␣chers.<BR>";
?>
```



Sur cet exemple, le "s" du pluriel est interprété par PHP comme appartenant à la séquence de caractères située derrière le $; c'est donc la variable $fruits qui est reconnue et remplacée par sa valeur (vide puisque la variable n'a jamais été intialisée).

La solution consiste à délimiter le nom de la variable par des accolades sous la forme {$variable} ou ${variable}.

```
<?php
$fruit = "pomme";
echo "Une␣$fruit␣ne␣coute␣pas␣cher.<BR>";
echo "Deux␣{$fruit}s␣coutent␣deux␣fois␣plus␣chers.<BR>";
echo "Trois␣${fruit}s␣coutent␣trois␣fois␣plus␣chers.<BR>";
?>
```



- ✔ Le type booléen peut prendre deux valeurs : *TRUE* (ou *true*) et *FALSE* (ou *false*).
- ✔ Ce type de données est principalement utilisé dans les structures de contrôle pour tester une condition (voir suite du cours)

PHP est capable de convertir tout type de donnée en booléen selon les règles suivantes :

Valeur	Réultat conversion
nombre entier 0	FALSE
nombre décimal 0.000...	FALSE
chaîne vide (""")	FALSE
chaîne égale à 0 ("0")	FALSE
tableau vide	FALSE
objet vide	FALSE
NULL	FALSE
tout le reste	TRUE

PHP propose plusieurs fonctions utiles relatives au type de variables :

- ✔ *is_** : indique si la variable est du type donné par *
- ✔ *strval* : convertit une variable en chaîne
- ✔ *floatval* ou *doubleval* : convertit une variable en nombre à virgule flottante
- ✔ *intval* : convertit une variable en entier

La fonction *is_** permet de tester si une variable est d'un type donnée. Syntaxe

```
booleen is_*(mixte variable)
```

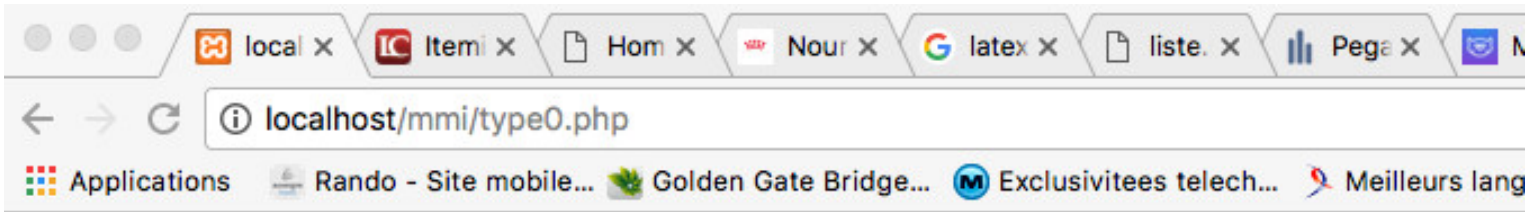
variable : variable à tester
Les déclinaisons sont les suivantes :

- ✔ *is_array* : type tableau ?
- ✔ *is_bool* : type booléen ?
- ✔ *is_double*, *is_float*, *is_real* : type nombre à virgule flottante ?
- ✔ *is_int*, *is_integer*, *is_long* : type entier ?
- ✔ *is_null* : type NULL ?
- ✔ *is_numeric* : entier ou nombre à virgule flottante ou chaîne contenant un nombre
- ✔ *is_object* : type objet ?
- ✔ *is_string* : type chaîne ?
- ✔ *is_resource* : type ressource ?
- ✔ *is_scalar* : type scalaire ?

Types

Fonctions utiles : exemple *is_**

```php
<?php
if (is_null($x))
{
        echo "Pour l'instant , \$x est du type NULL.<BR>";
}
$x = (1<2);
if (is_bool($x))
{
        echo "\$x = (1<2) est du type booleen.<BR>";
}
$x = "123abc";
if (is_string($x))
{
        echo "\$x = \"123abc\" est du type chaine...<BR>";
}
if (is_numeric($x))
{
        echo "...mais aussi est du type <I>numeric</I>.<BR>";
}
else
{
        echo "...mais pas du type <I>numeric</I>.<BR>";
}
$x="1.23e45";
if (is_numeric($x))
{
        echo "Par contre , \$x = \"1.23e45\" est du type <I>numeric</I>.<BR>";
}
?>
```



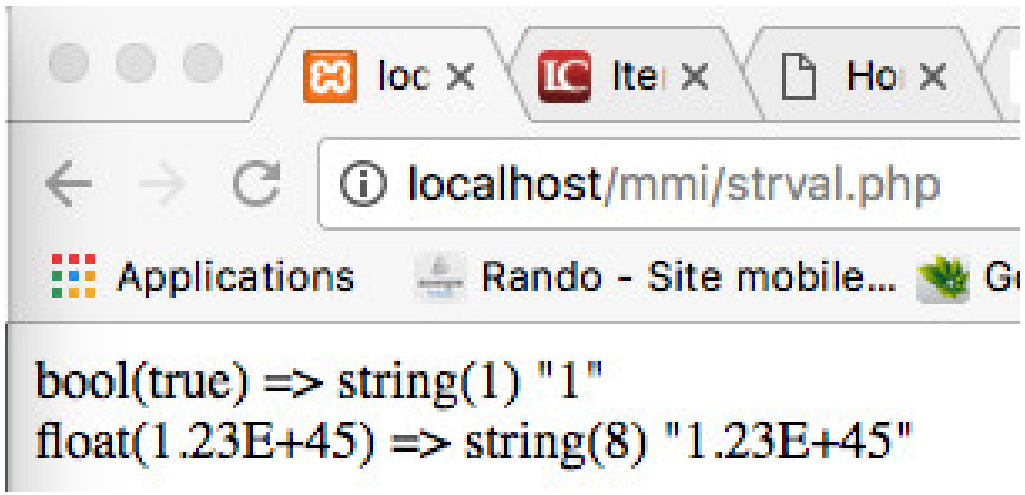
Notice: Undefined variable: x in **/Applications/XAMPP/xamppfiles/htdocs/mmi/type0.php** on line **2**
Pour l'instant, $x est du type NULL.
$x = (1<2) est du type booleen.
$x = "123abc" est du type chaine...
...mais pas du type *numeric*.
Par contre, $x = "1.23e45" est du type *numeric*.

La fonction *strval* retourne la valeur d'une variable après la conversion en chaîne. Syntaxe

```
chaine strval(mixte variable)
```

variable : variable à traiter
Cette fonction ne s'applique qu'aux variables de type scalaire (non valable pour les tableaux, ni les objets).
Le type de la variable reste inchangé.

```php
<?php
$x = TRUE;
echo var_dump($x), "␣=>␣", var_dump(strval($x)),"<BR>";
$x = 1.23e45;
echo var_dump($x), "␣=>␣", var_dump(strval($x)),"<BR>";
?>
```

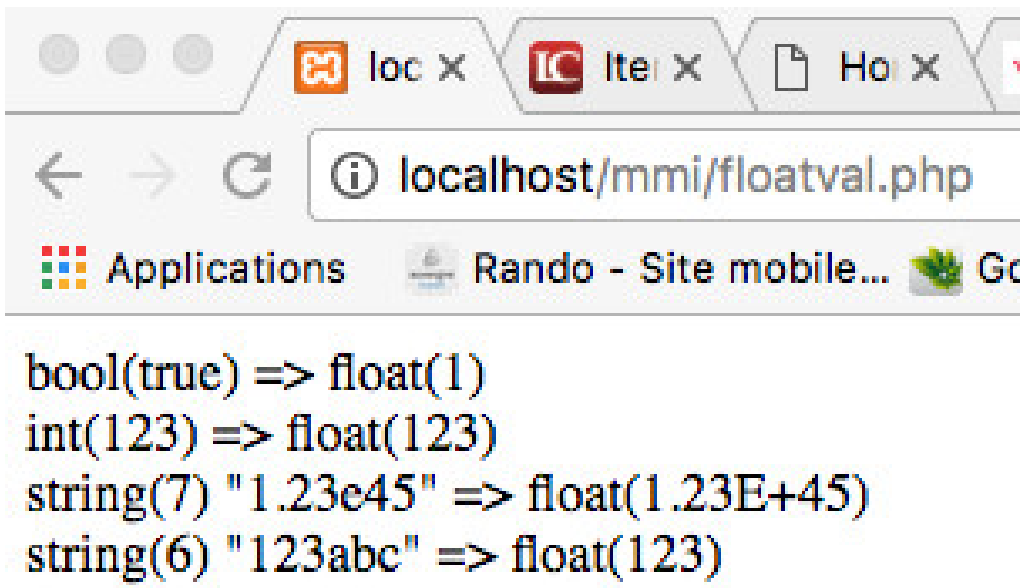


La fonction *floatval* retourne la valeur d'une variable après la conversion en nombre à virgule flottante. La fonction *doubleval* est un alias de la fonction *floatval*. Syntaxe

```
nombre floatval(mixte variable)
```

variable : variable à traiter
Cette fonction ne s'applique qu'aux variables de type scalaire (non valable pour les tableaux, ni les objets).
Le type de la variable reste inchangé.

```php
<?php
$x = TRUE;
echo var_dump($x), " => ", var_dump(floatval($x)),"<BR>";
$x = 123;
echo var_dump($x), " => ", var_dump(floatval($x)),"<BR>";
$x = "1.23e45";
echo var_dump($x), " => ", var_dump(floatval($x)),"<BR>";
$x = "123abc";
echo var_dump($x), " => ", var_dump(floatval($x)),"<BR>";
?>
```

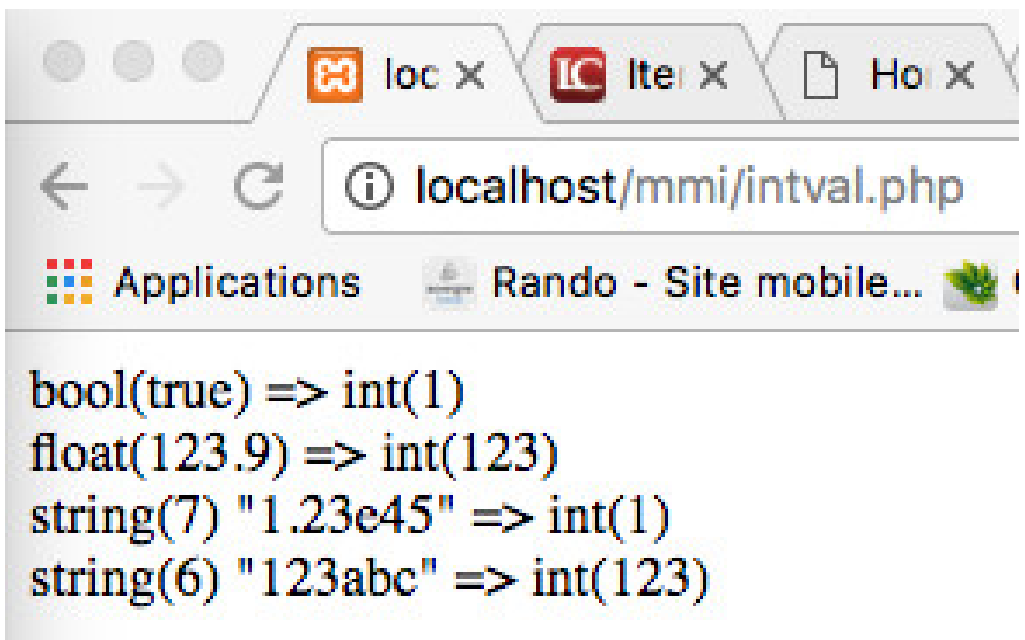


La fonction *intval* retourne la valeur d'une variable après la conversion en entier. Syntaxe

```
nombre intval(mixte variable)
```

variable : variable à traiter
Cette fonction ne s'applique qu'aux variables de type scalaire (non valable pour les tableaux, ni les objets).
Le type de la variable reste inchangé.

```php
<?php
$x = TRUE;
echo var_dump($x), " => ", var_dump(intval($x)),"<BR>";
$x = 123.9;
echo var_dump($x), " => ", var_dump(intval($x)),"<BR>";
$x = "1.23e45";
echo var_dump($x), " => ", var_dump(intval($x)),"<BR>";
$x = "123abc";
echo var_dump($x), " => ", var_dump(intval($x)),"<BR>";
?>
```



```php
<?php
$x = TRUE;
echo intval(32),"<BR>";//32
echo intval(3.2),"<BR>";//3
echo intval("032"),"<BR>";//32
echo intval(0x1A),"<BR>";//26
echo floatval("122.64MMI"),"<BR>";//122.64
echo floatval("MMI122.64"),"<BR>";//0
echo strval(32),"<BR>";//"32"
?>
```

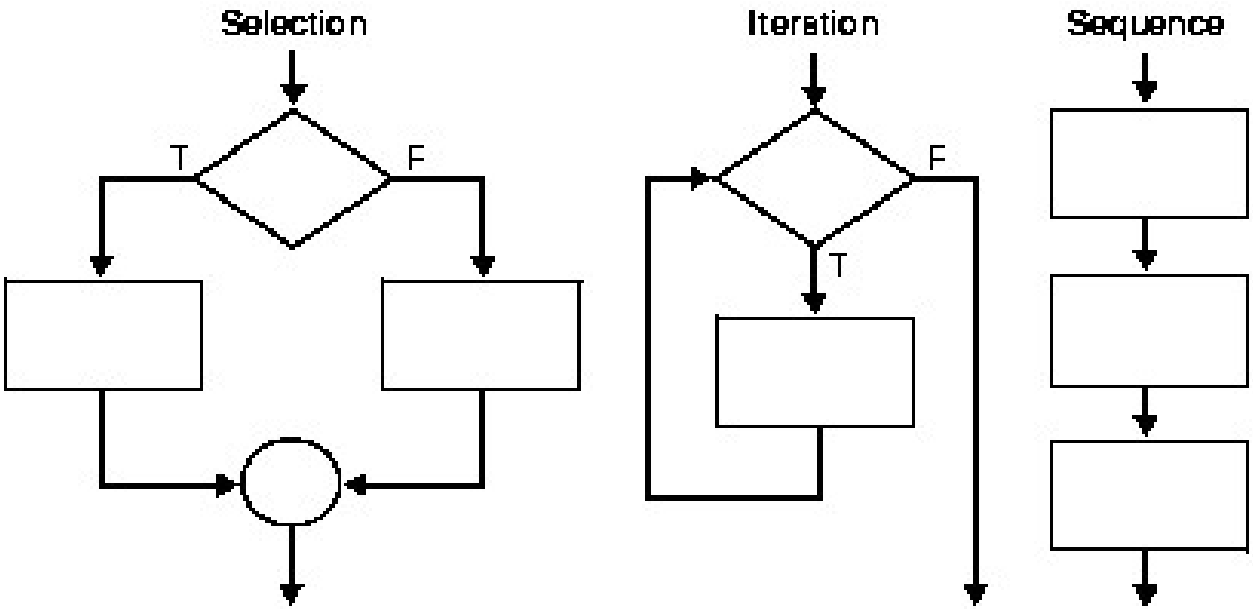
Symbole	Exemple	Description
=	$salaire = 2800 ;	affectation
+	$salaire = $salaire + 2800	addition-concaténation
-	$salaire = $salaire - 2800	soustraction
*	$salaire = $salaire * 2800	multiplication
/	$salaire = $salaire / 2800	division
%	$salaire = $salaire % 2800	modulo

salaire=2800

Symbole	Exemple	Retour	Description
==	$salaire == 2800	true	égalité
==	$salaire == "2800"	true	égalité
===	$salaire === 2800	true	égalité exacte
===	$salaire == "2800"	false	égalité exacte
!=	$salaire != 2800	false	inégalité
!==	$salaire !== "2800"	true	inégalité exacte
>	$salaire > 2800	false	supérieur
>=	$salaire >= 2800	true	supérieur ou égal
<=	$salaire <= 2800	true	inférieur ou égal
<	$salaire < 2800	false	inférieur

Symbole	Exemple	Description
&&	($age==18) && ($salaire > 2800)	ET
||	($age==18) \|\|($salaire > 2800)	OU

Symbole	Exemple	Description
+=	($age+=18)	$age = $age + 18
++	$age++	post incrémentation
++	++$age	pré incrémentation
–	$age–	post décrémentation
–	–$age	pré décrémentation


Selection
T
F
Iteration
F
T
Sequence

- ✔ Le mot clef *if* a besoin d'une condition :

Notation :

- ✔ Notation classique :

```
<?php
if ($str == "Hello")
{
        //code
}
?>
```

- ✔ Notation alternative :

```
<?php
if ($str == "Hello"):
        //code
endif;
?>
```

La notation alternative est valide pour toutes les autres structures de contrôle.

✔ *else* est exécuté si la condition n'est pas vérifiée :

```
<?php
if ($str == "Hello")
{
        //code
}
else
{
        //code
}
?>
```

- ✔ Avec *elseif* il est possible de faire plus qu'un seul test

```php
<?php
if ($str == "Hello")
{
        //code
}
elseif ($str == "Bye")
{
        //code
}
else
{
        //code
}
?>
```

✔ il est possible de mettre plusieurs conditions dans un test :

```
<?php
if ((($str == "Hello") || ($str=="Bye" ))
//if ($str == "Hello" || $str=="Bye" )
{
        //code
}
else
{
        //code
}
?>
```

✔ similaire à *if...else*

(condition) ? instruction si *TRUE* : instruction si *FALSE*

```php
<?php
echo ($int == 1 ? "int = 1" : "int = ?");
// if $int == 1 displays : int = 1
// else displays : int = ?
?>
```

Structure de contrôle

switch

- ✔ similaire à *if...elseif...else*
- ✔ succession de tests sur la variable
- ✔ default est exécuté si tous les tests sont faux

```
switch ($str)
{
case "Hello":
        // Code
        break;
case "Bye":
        // Code
        break;
default:
        // Code
        break;
}
```

- ✔ Le code est exécuté tant que la condition est vraie :

Notation :

- ✔ Notation classique :

```
<?php
$int=0;
while ($int < 10){
        echo $int;
        $int++;
}
?>
```

- ✔ Notation alternative :

```
<?php
$int=0;
while ($int < 10):
        echo $int;
        $int++;
endwhile;
?>
```

Similaire à la structure de contrôle *while* mais :

- ✔ la condition est vérifiée à la fin de la boucle
- ✔ la boucle est exécutée au moins une fois

```
$int=1;
do
{
        echo $int;
}
while ($int<0);
```

Structure de contrôle

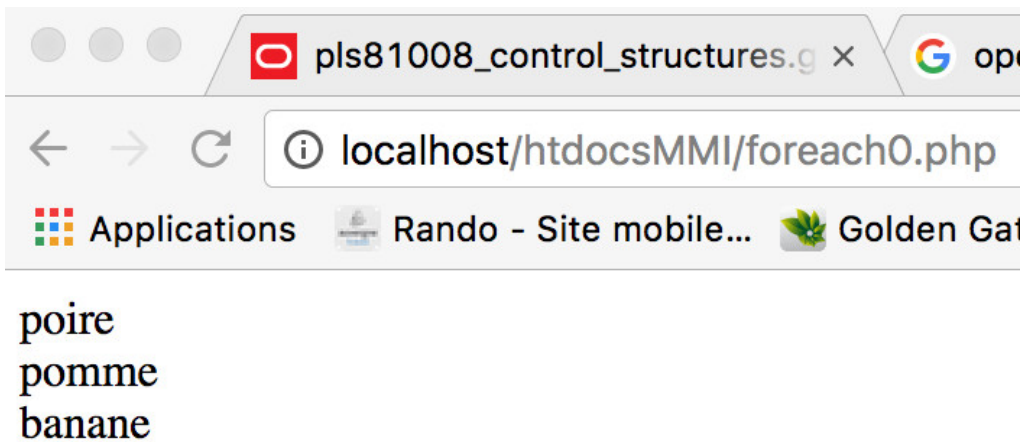
for

- ✔ une autre façon pour faire des boucles
- ✔ prend en paramètres trois expressions :
  1. la variable d'itération
  2. la condition de test
  3. évolution de la variable

```
for ($i=0 ; $i<10 ; $i++)
{
        echo $i;
}
```

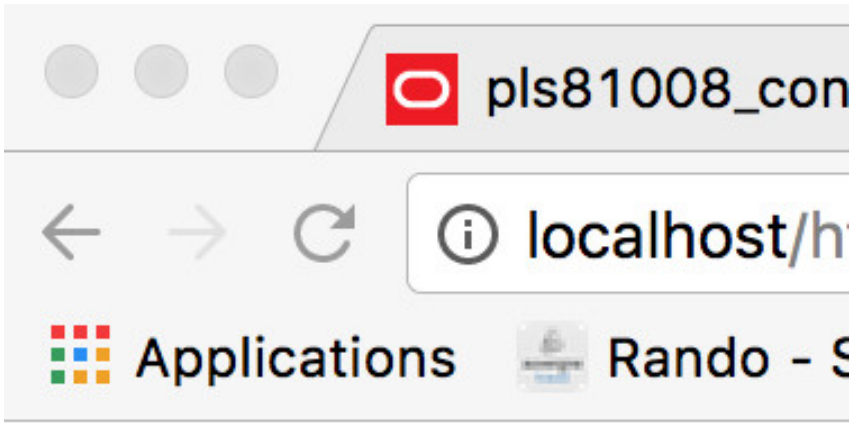
✔ lit chaque élément d'un tableau

```php
$tableau = array('poire','pomme','banane');
foreach ($array as $value)
{
        echo $value . '<br>';
}
```



- ✔ possibilité d'avoir l'indice et la valeur de chaque élément d'un tableau

```
$tableau = array('poire','pomme','banane');
foreach ($tableau as $key => $value)
{
        echo $key . "␣:␣" . $value . '<BR>';
}
```



0 : poire
1 : pomme
2 : banane

Structure de contrôle

break

- ✔ stoppe la structure de contrôle

```
while ($int<10)
{
        //code
        break;
}
```

- ✔ possibilité de stopper plusieurs structures de contrôle

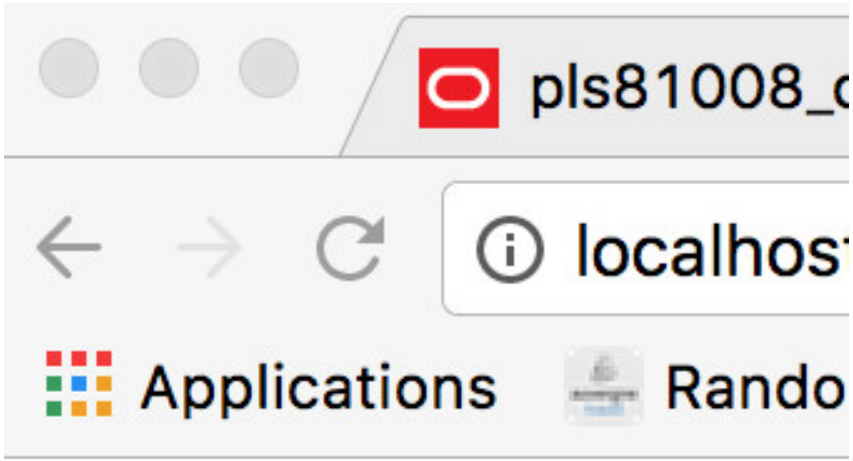
```
while ($int<10)
{
        while (true)
        {
                break 2;
        }
}
```

Structure de contrôle

continue

✔ saut direct à la prochaine itération

```
for ($i = 0 ; $i < 10 ; $i++)
{
        if ($i == 1)
                continue;
        echo $i;
}
```



Structure de contrôle

continue

✔ saut direct à la prochaine itération d'une autre boucle

```
$i = 0;
while ($i++ < 5){
        for ($j = 0 ; $j < 4; $j++) {
                if ($j == 3) continue 2;
                echo $j;
        }
}
```