# PHP

## CM1-1 : PHP

Mickaël Martin Nevot

V5.0.1

# PHP

# Notions générales

- **Opérateurs**
- Données :
  - **Variables ≠ attributs**
  - Constantes
  - **Valeurs littérales** (`"hello!"`, `3`, `2.75`, `true`)
  - Expressions (et expressions « parenthésées »)
- **Affectation ≠ condition**
- **Instructions**
- **Fonction** ≠ procédure ≠ **méthode** (signature)
- **Paramètres** (formels) ≠ **arguments** (paramètres effectifs)
- **Exécution séquentielle**

# Rappel : Web

# PHP/PHP 5


- Historique :
  - 1994-1995 : PHP par Rasmus Lerdorf
  - 2004 : PHP version 5
  - 2020 : PHP version 8
- Principales caractéristiques :
  - Langage de programmation de **script** interprété
  - Conçu pour le développement d'**applications Web**
  - Exécuté **côté serveur** (code non accessible côté client)
  - **Génération de pages Web dynamiques**
  - Possibilité d'être couplé à une **base de données**
  - Langage **objet faiblement typé**
  - *Open source*

# Pour tester

- Interpréteur en ligne : https://3v4l.org/
- Gestion des erreurs :

```
declare(strict_types=1);
ini_set('display_errors', 'on');
ini_set('error_reporting', E_ALL);
```

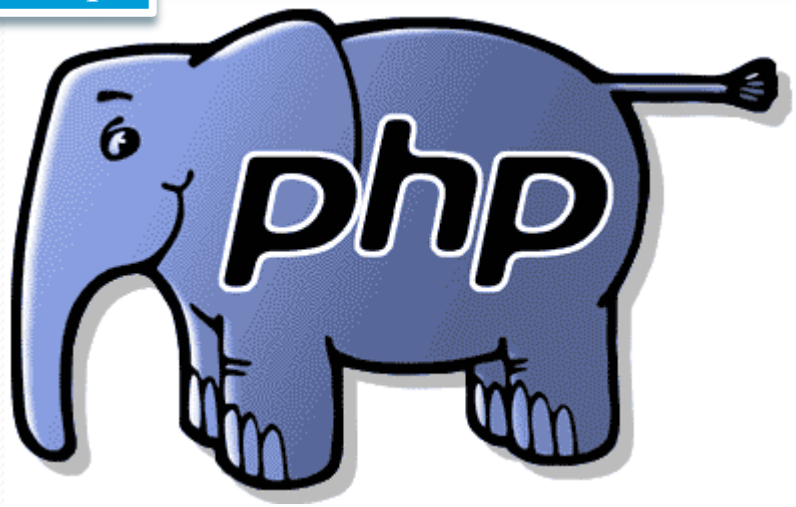
# Utilisation

- Fichier texte avec l'extension : .php
- Script PHP : langage PHP et/ou langage HTML
- Balise <?php ?> :

```
<?php
    instruction_1
    instruction_2
    ...
?>
```

Ne pas mettre pour la dernière balise du script

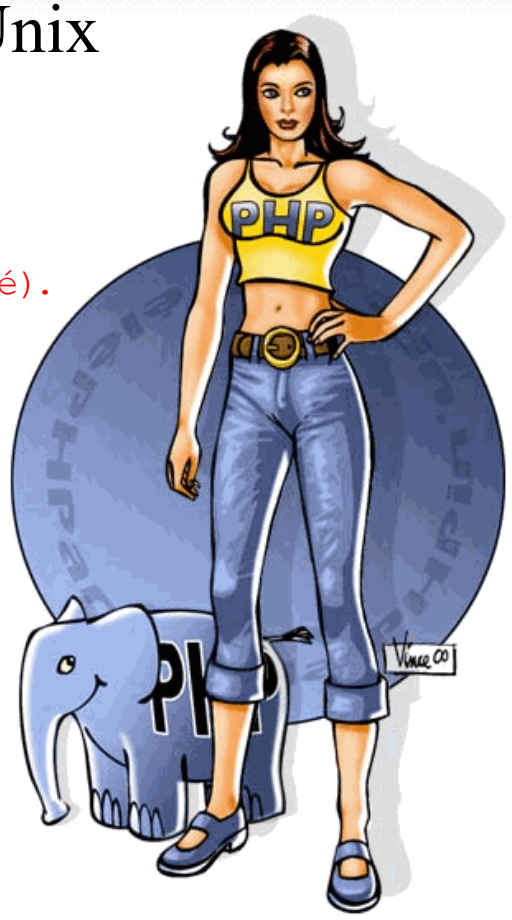
- Affichage : echo ...
- Balise + affichage : <?= ?>



# Commentaires

- Non interprétés
- Commentaires de type C/C++ et *shell* Unix
- Exemples :

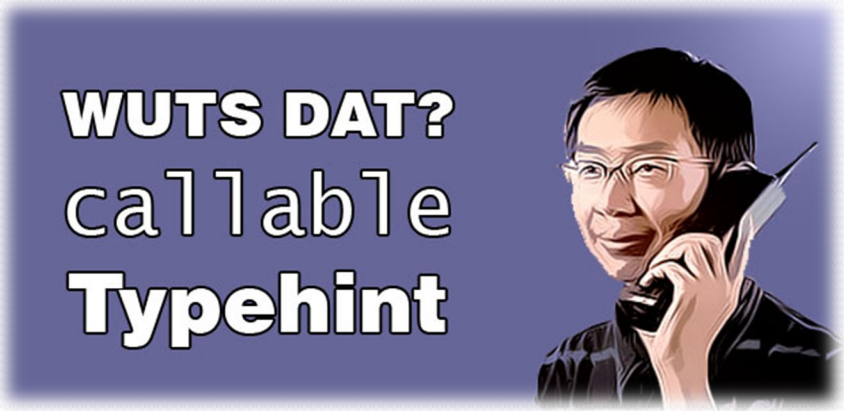
```
// Commentaire PHP (une seule ligne complète).

# Autre commentaire PHP (une seule ligne complète, déprécié).

/* Autre commentaire PHP (une ou plusieurs lignes). */

/*
    Commentaire PHP
    (sur plusieurs lignes).
 */
```



# Définition de type

- Script :
  - **Pas de déclaration explicite** du type d'une variable
  - Le type d'une variable est déterminé par le contexte d'utilisation
  - Conversion automatique
- **Attributs**, fonctions (**argument** et **type de retour**) :
  - `bool`, `int`, `double`, `string`, `array`, `object`, `unset`

*Type Hinting*



# Variables

- Représentées par un signe dollar `$` suivi du nom de la variable : `$x`
- **Sensibles à la casse** : `$x` est différent de `$X`
- Affectation :
  - Simple :

```
$a = 10;
$b = $a;
```

  - Par référence :

```
$a = 10;
$b = &$a; // Ici,  $a et $b pointent sur le même contenu.
```

- Destruction : `unset($var)`

# La valeur NULL

- La valeur spéciale `NULL` représente l'absence de valeur
- Une variable considérée comme `NULL` n'a pas de valeur
- Une variable est considérée comme `NULL` si :
  - On lui a affecté la constante `NULL`
  - Aucune valeur ne lui a été attribuée

# Les types

- Types scalaires (simples) :
  - Booléen (`bool`)
  - Entier (`int`)
  - Nombres décimaux (`float`)
  - Chaîne de caractères (`string`)
- Types composés :
  - Tableau (`array`)
  - Objet (`object`)

# Booléen

- `bool`
- Peut prendre la valeur `true` ou `false`
- Valeurs considérées comme fausses :
  - Le booléen `false` lui-même
  - L'entier `0`
  - Le nombre décimal `0.0`
  - La chaîne de caractères vide et la chaîne de caractères `"0"`
  - Le tableau vide
  - La valeur spéciale `NULL`
  - **Toutes** les autres valeurs sont considérées comme `true`
- Opérateurs de conversion : `(bool)`, `(boolean)`

A cause de la définition de type

# Entier

- `int`
- Un type entier est un entier naturel
- Les entiers peuvent être spécifiés en base 10, 8 ou 16
- Les entiers peuvent être optionnellement précédés par le signe plus ou moins (+ ou −) :

```
$a = 1234; // Nombre entier en base 10.
$a = -123; // Nombre entier négatif.
$a = 0123; // Nombre entier en base 8, octale (83 en base 10).
$a = 0x12; // Nombre entier en base 16, hexadécimale (18 en base 10).
```

- Conversion :
  - Opérateurs de conversion : `(int)`, `(integer)`, `intval()`
  - Lors de la conversion entre un nombre décimal et un entier, le nombre sera arrondi :
    - À la valeur inférieure s'il est positif, à la supérieure s'il est négatif

# Nombres décimaux

- `double`
  - `float`, `double` et `real` sont des alias (par défaut : `double`)
- Aussi appelés :
  - *Double*
  - *Float*
  - Nombres réels :

```
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
```

- Conversion en nombre décimal :
  - Opérateurs de conversion : `(real)`, `(double)` et `(float)`
  - Fonctions `floatval()` et `doubleval()`

# Chaîne de caractères

- `string`
- Séquence de caractères
- Une chaîne peut être spécifiée de trois manières :
  - Guillemets doubles `"`
  - Guillemets simples `'`
  - Syntaxe *heredoc* (plus de détails : documentation PHP) :

```
$a = "hello world";
$b = '$a : ';
$c = <<<EOT
    Hello world
EOT;
```

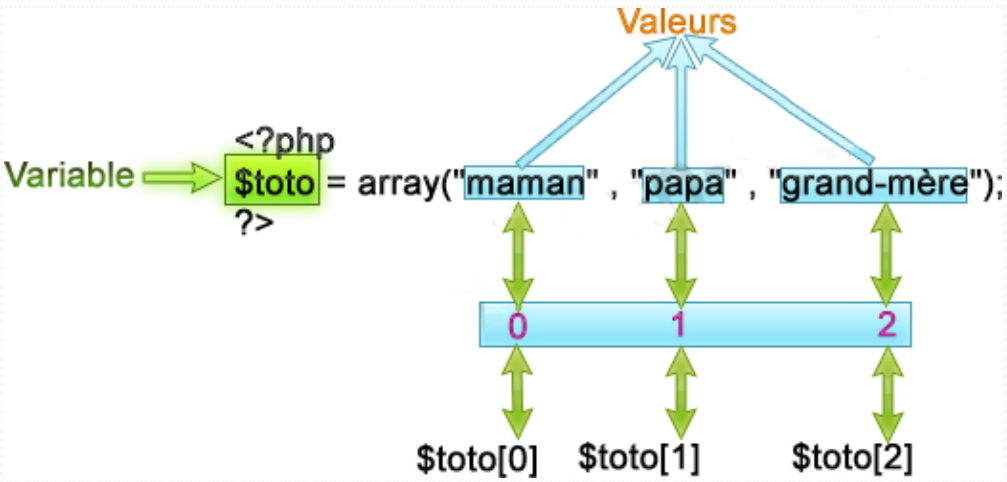
# Chaîne de caractères

- Guillemets doubles / syntaxe *heredoc* :
  - Variables présentes dans la chaîne évaluées
  - Caractères spéciaux pris en compte :
    - Nouvelle ligne : `\n`
    - Retour à la ligne : `\r`
    - Tabulation horizontale : `\t`
    - *Antislash* : `\\`
    - Caractère `$` : `\$`
    - Guillemets doubles : `\"`
- Guillemets simples :
  - Variables présentes dans la chaîne **non** évaluées
  - Caractère spécial pris en compte : `\'` guillemet simple

# Tableau

- array
- Syntaxe :

```
// array(clef => valeur , ...)
// clef ne peut être que :
//  - entier (tableau indexé) ;
//  - chaîne de caractères (tableau associatif).
// Valeur peut être de n'importe quel type.
$arr = array('foo' => 'bar', 12 => true);

echo $arr['foo'];  // bar
echo $arr[12];     // 1
```



# Tableau

- Syntaxe (création/modification avec des crochets) :

```
// $arr[clef] = valeur;
// $arr[] = valeur;
// clef peut être un entier ou une chaîne de caractères.
// valeur peut être n'importe quel type.
$arr = array(5 => 1, 12 => 2);
$arr[] = 56;    // Identique à $arr[13] = 56; à cet endroit du script.
$arr['x'] = 42; // Ceci ajoute un nouvel élément au
                // tableau avec la clef 'x'.
```

- Fonction `count()` :

```
count($arr);  // Retourne le nombre d'éléments du tableau.
```

- Fonction `unset()` :

```
unset($arr[5]);  // Ceci efface un élément du tableau.
unset($arr);     // Ceci efface tout le tableau.
```

# Objet

À voir en détail dans le cours dédié

- `object`
- Instancié avec `new` :

```
class Foo
{
    function do_foo($a)
    {
        return $a / 2;
    }
}
$bar = new \Foo();
$b = $bar->do_foo(2);
```

- Opérateurs de conversion : `(object)`
  - Conversion en un tableau : les propriétés sont les clefs
  - Conversion dans un autre type : membre `scalar`

```
$obj = (object) 'hello';
echo $obj->scalar; // Affiche : hello.
```

# Constante

- **Ni modifiable, ni effaçable**
- Ne commence pas par un `$`
- Par convention : écrite en majuscule
- **Accessible globalement**
- Valeur scalaire uniquement
- Définie par `define()` :

```
define('FOO', 'something');      // Valide.
define('2FOO', 'something');     // Invalide.
define('__FOO__', 'something'); // A éviter (à cause de __...__).
```

# Variables prédéfinies

- Variables prédéfinies « superglobales » :
  - `$GLOBALS` : toutes les variables du contexte global
  - `$_SERVER` : variables serveur et d'exécution
  - `$_GET` : HTTP GET
  - `$_POST` : HTTP POST
  - `$_SESSION` : variables de session
  - `$_COOKIE` : cookies HTTP
  - `$_FILE` : fichier chargé
- Variables globales et configuration : `phpinfo()`

# Opérateurs

- Unaires
  - Signes : +, –
  - Négation : `!`
  - Incrémentation/décrémentation (pré et post) : ++, ––
  - Conversion : `(int)`, `(bool)`, etc.
  - Clonage : `clone`
  - Contrôle d'erreur : `@`

```
$a = -0.5;
--$a;
$b = (bool) $a;
$c = !$b;
```

# Opérateurs

- Binaires
  - Arithmétiques : +, –, *, /, %
  - Concaténation de chaîne : .
  - Affectations : =, +=, .=, *=, etc.
  - Comparaisons : ==, >, >=, <, !=, <>, ===, !==, etc.
  - Logiques : &&, ||, xor, etc.
  - Binaires : &, |, ~, ^, <<, >>

Plus performant

```php
$a = 5;
$b = 10;
$c = '$a : ' . $a;  // Pas de caractère '+' !
$d = ($a == $b) && ((0 != $b % $a) || ($b < 20));
```

# Structures conditionnelles

## Structure conditionnelle :

```php
if (0 == $a)
{
    ++$a;
}
else if (1 == $a)
{
    --$a;
}
else
{
    $a = 0;
}
```

Ou : `if () : endif;...`

## Branchement conditionnel :

```php
switch($i)
{
   case 'Jambon' :
      echo 'Salé';
      break;
   case 'Tarte' :
   case 'Bonbon' :
      echo 'Sucré';
      break;
   default :
      echo 'Autre';
}
```

# Boucles

- Boucle `for` :

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

- Boucle `foreach` :

```
$arr = array(1, 2, 3, 4, 5);

foreach ($arr as $key => &$val)
{
    $val *= 2;
}
// $arr = (2, 4, 6, 8, 10).
```

- Boucle `while` :

```
while($i < 5)
{
    ++$i;
}
```

De 0 à n fois

- Boucle `do ... while` :

```
do
{
    --$i;
} while ($i > 0);
```

De 1 à n fois

# Break/continue

## Break

- Sortir d'une boucle ou d'un branchement conditionnel :

```
while(1)
{
   if (10 == $i)
   {
      break;
   }
   ++$i;
}
// Exemple de break dans un
// switch sur la diapositive
// de structures conditionnelles.
```

## Continue

- « Sauter » à l'itération suivante d'une boucle :

```
$a = 0;

for  ($i = 0 ; $i < 10 ; ++$i)
{
    if (0 == $i % 2)
    {
        continue;
    }
    $a += $i;
}
// ?
```

# Fonctions

```php
function foo(int $a, int $b, int $c = 2): int
{
    // Rappel : $a, $b et $c sont des paramètres.
    // = 2 : valeur par défaut (uniquement de type scalaire).
    // Un paramètre par défaut est optionnel.
    ++$a;
    --$b;
    // Termine la fonction en retournant la valeur calculée.
    return ($a * $b) / $c;
    // Le code éventuellement placé ici n'est pas exécuté.
}
…
$a = 5;
$b = foo(3, $a);  // Rappel : 3 et $a sont des arguments.
// $a = 5$b = 8.
…
function bar(): void { … }
function baz(?string $name): ?bool { … }
function qux(int|float $name): string|array|bool { … }
```

Type de retour

Pas de type de retour

Type *nullable* : type indiqué ou NULL ¶

Types d'unions

# Portée des variables

- Globale : définie dans un script PHP
- Locale : définie dans une fonction

```php
$a = 1; // Portée globale.

function test()
{
    echo $a; // Portée locale : rien n'est affiché à l'écran.
}
test();
```

- Statique :

```php
function compteur()
{
    static $a = 0; // Prend une valeur scalaire, pas une expression.

    // Qu'affiche cet echo au cinquième appel de la fonction compteur ?
    echo $a;
    ++$a;
}
```

# Instruction

- Se termine par un `;`
- Types d'instruction :
  - Déclaration : `$a;`
  - Affectation : `$a = 10;`
  - Appel de fonction : `myFunction();`
  - Instruction conditionnelle : `if ($a == $b) ... ;`
  - Instruction vide : `;`
  - Bloc (d'instructions) :

```
{
    instruction_1
    instruction_2
    …
}
```

# Inclusion de fichiers

- Inclut et exécute un fichier PHP en remplacement de l'instruction : `include/require`
- `include_once/require_once` : comme leur version sans le suffixe « *_once* » mais suppriment l'instruction sans remplacement si le fichier a déjà été inclus
- Seule différence entre `include` et `require` : la manière dont les erreurs sont produites :
  - `include` : produit une simple alerte
  - `require` : produit une erreur fatale

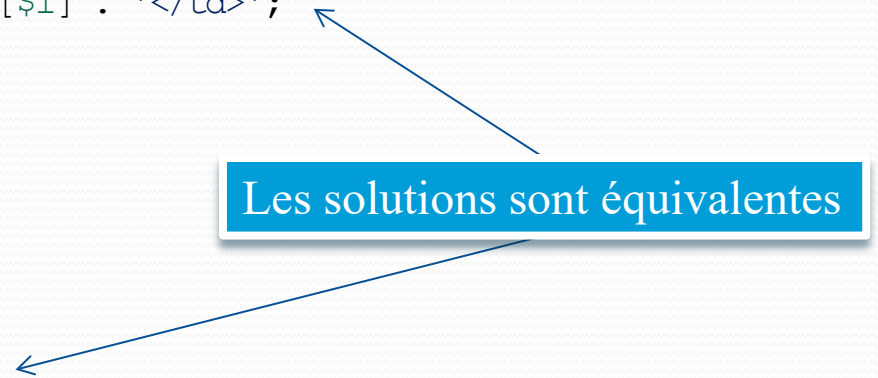
# Exemple

- Solution 1 (« totalement PHP ») :

```php
<?php
$arr = array(1, 2, 3, 4, 5);
echo '<table><tr>';
for  ($i = 0 ; $i < 5 ; ++$i)
{
    // PHP_EOL : retour à la ligne dans le code source (pas dans la page Web)
    echo PHP_EOL . '    <td>' . $arr[$i] . '</td>';
}
echo PHP_EOL . '</tr></table>';
?>
```

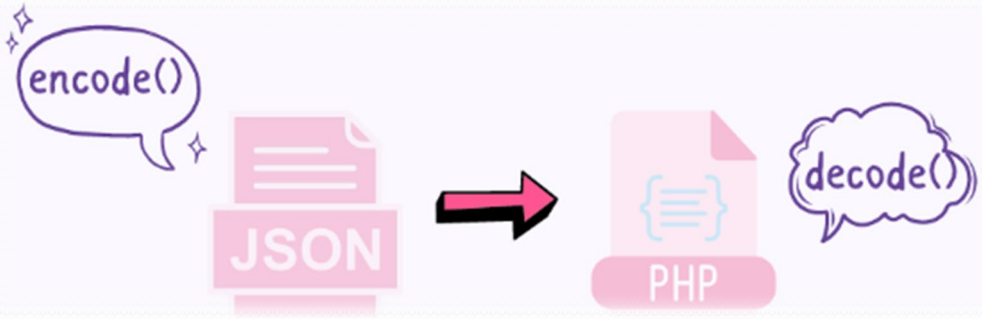
- Solution 2 (mixte) :

```php
<table><tr>
    <?php
    $arr = array(1, 2, 3, 4, 5);
    for  ($i = 0 ; $i < 5 ; ++$i)
    {
    ?>
    <td><?php echo $arr[$i];?></td>
<?php } ?>
</tr></table>
```



# PHP et JSON

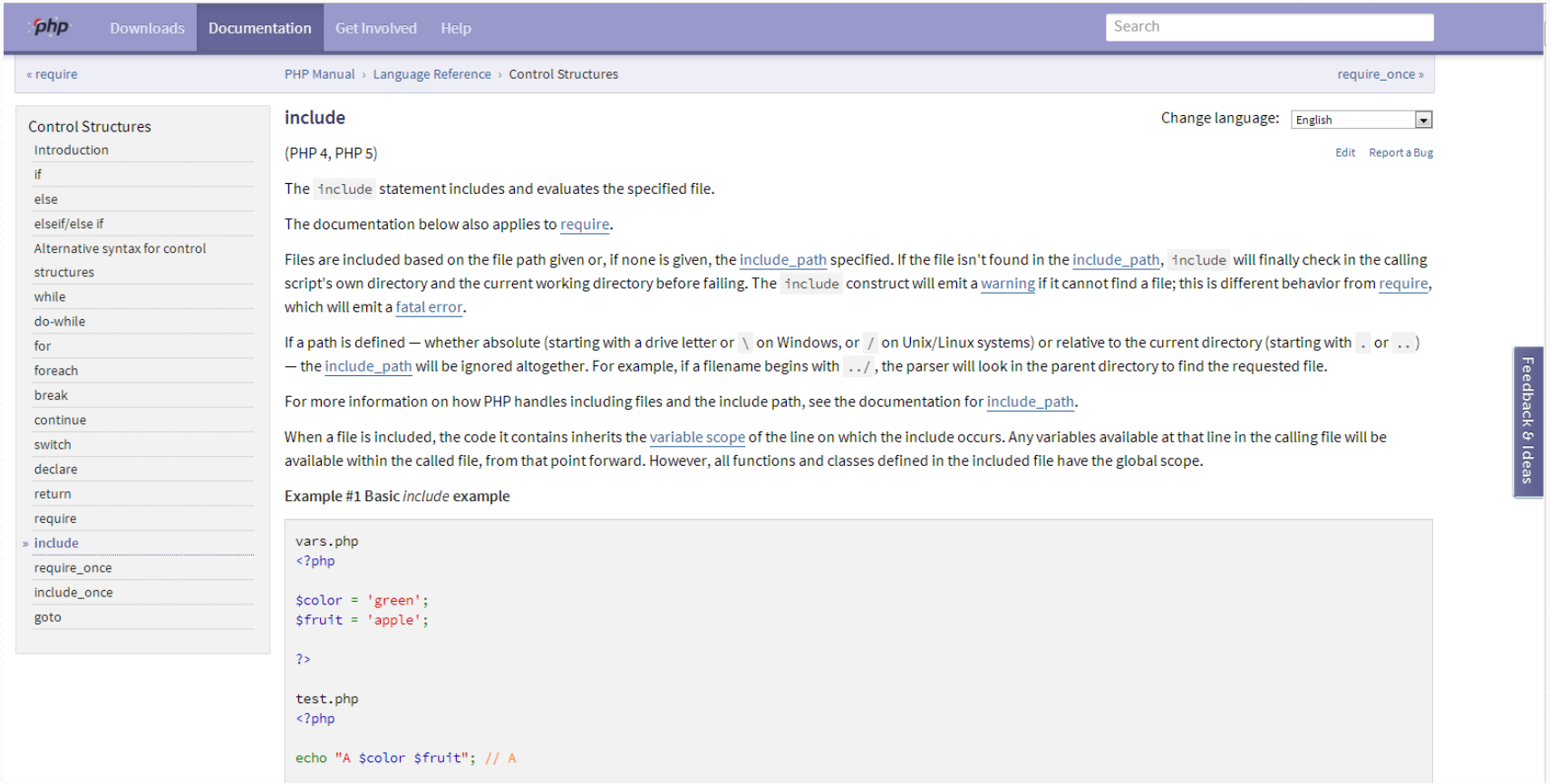
- Exemple :




```
$json = '{
    "menu":
    {
        "id":"file",
        "value":"File",
        "popup":
        {
            "menuitem":
            [
                { "value":"New", "onclick":"CreateNewDoc()" },
                { "value":"Open", "onclick":"OpenDoc()" }
            ]
        }
    }
}';
$jsonObj = json_decode($json);
```

# Manuel PHP

De préférence, en anglais




http://www.php.net

# Convention de codage

- Convention de codage :
  - *PHP standards recommendations* : http://www.php-fig.org/psr/

# Crédits

## Auteur

Mickaël Martin Nevot
mmartin.nevot@gmail.com

Carte de visite électronique

## Relecteurs

- Christophe Delagarde (christophe.delagarde@univ-amu.fr)
- Pierre-Alexis de Solminihac (pa@solminihac.fr)

Cours en ligne sur : www.mickael-martin-nevot.com