PHP 7

Nouveautés et migration

Formation Stéphane Brunet

www.chapoulougne.com

PHP 7 Nouveautés et migration

Sommaire

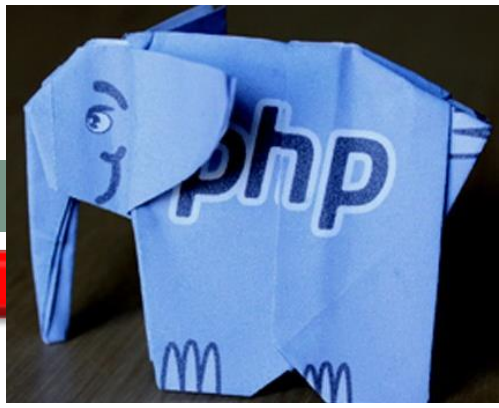
- Evolution de base
- Le langage
- Erreurs et Assertions
- Closure et générateur
- POO
- Migrer

PHP 7 Nouveautés et migration

Sommaire

- ❑ **Evolution de base**
- ❑ Le langage
- ❑ Erreurs et Assertions
- ❑ Closure et générateur
- ❑ POO
- ❑ Migrer

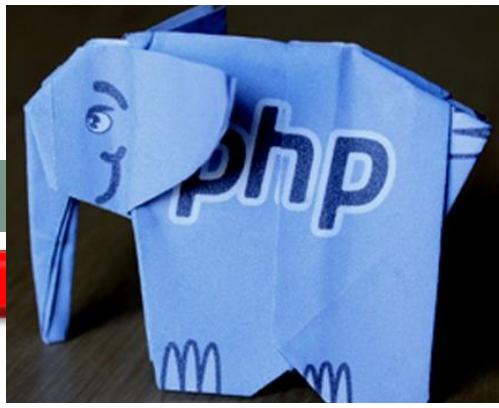
- Le cycle de release.
- Les éléments dépréciés.
- La syntaxe des variables.
- Intégration de l'Unicode au langage.
- Refonte du moteur, optimisation des performances.



PHP 7

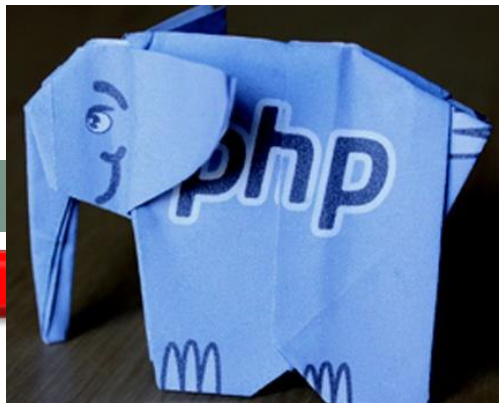
Le développement de PHP suit son court et il a été décidé en 2014 qu'il sauterait une version pour passer de la version 5.x à la version 7. Le processus de développement est relativement long.

Amélioration des performances
PHP 7 sera basé sur PHPNG (PHP Next-Gen) qui a été initialement développé par Zend pour améliorer son framework.
Le gain de performance est énorme. Il va de 25% à 70% selon les applications que vous utilisez. Ce gain se fait sans modifier une seule ligne de code. Il suffira de mettre à jour la version PHP de votre serveur pour en bénéficier.



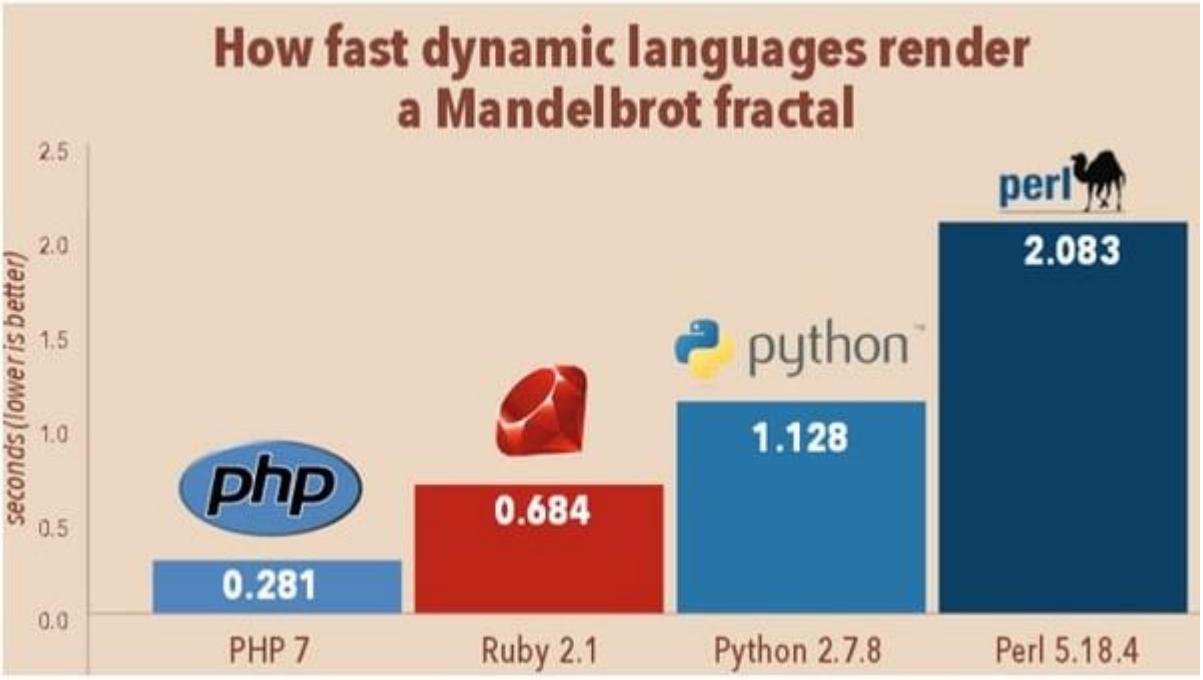
PHP 7

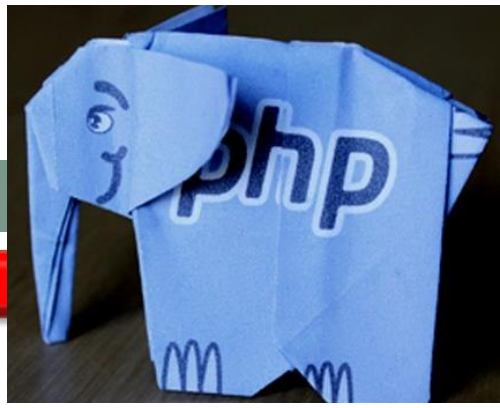
Les performances sont en constante évolution.
Quand PHPNG a été publié pour la première fois, il fallait environ 9.4 milliards d'instructions CPU pour afficher la page d'accueil de WordPress.
Actuellement, il n'en faut déjà plus que 2.6 milliards.
C'est 72% de moins.



PHP 7

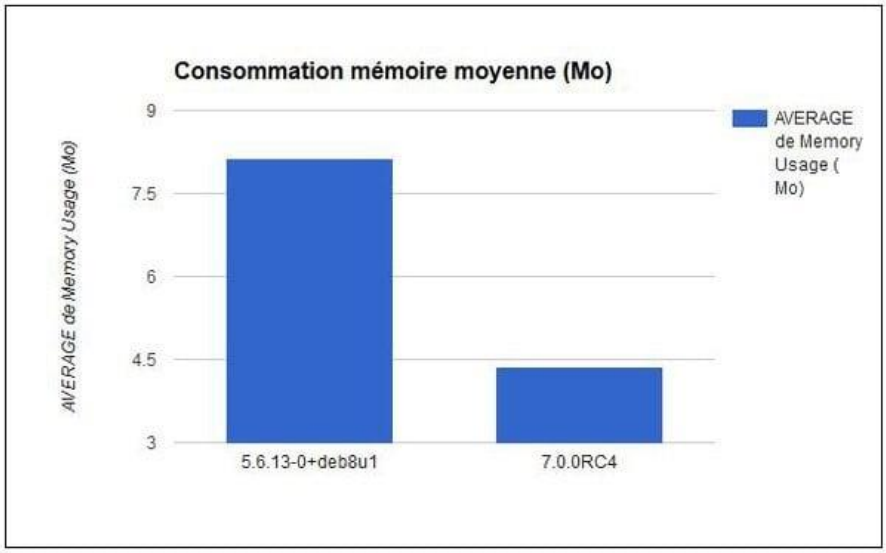
Au final, la mise à jour des applications vers PHP 7 peut engendrer d'après les équipes de Zend un surcroît de performance de 25% à 70%

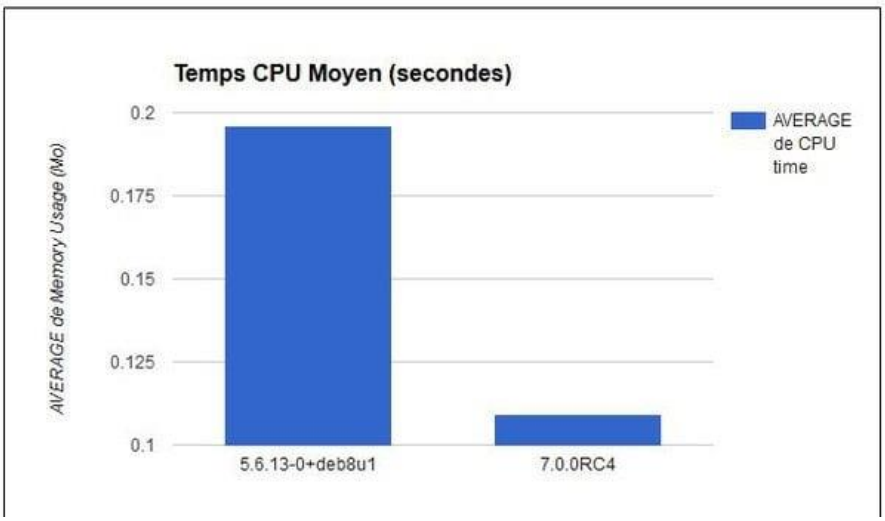


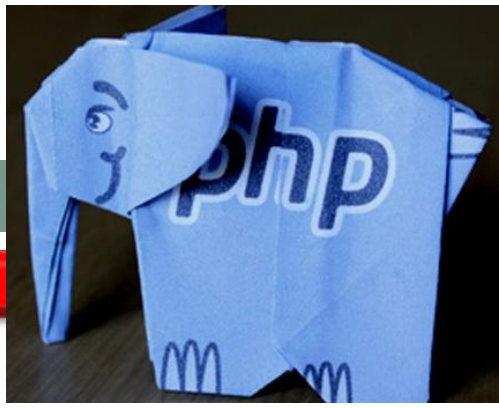


PHP 7

"C'est au-dessus de ce que nous avions anticipé. Je n'ai jamais vu un tel différentiel de performance lors des précédentes montées de version", note Xavier Leune,

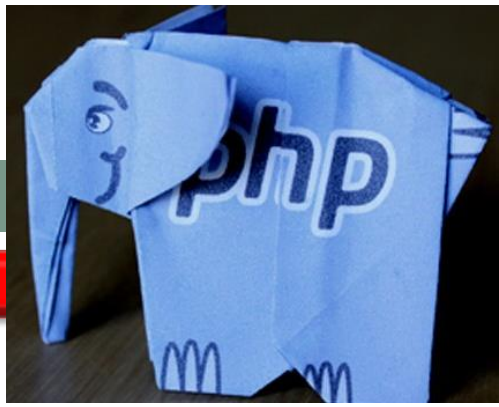




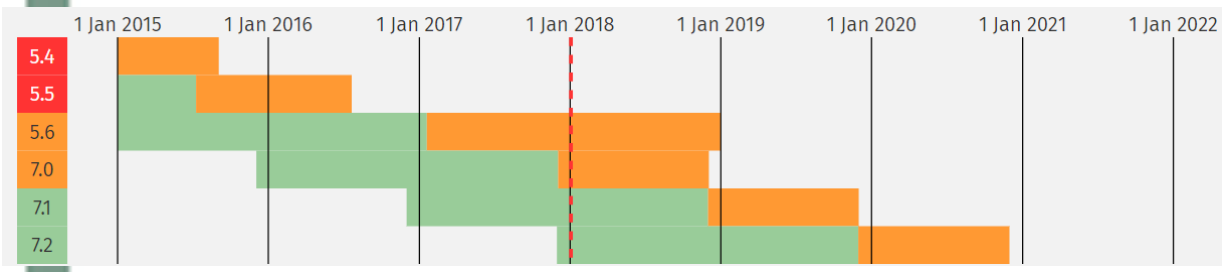


Versions

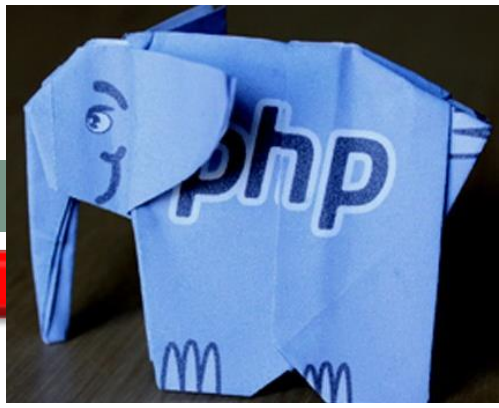
Chaque version de publication de PHP est entièrement prise en charge pendant deux ans à partir de sa version stable initiale.
Au cours de cette période, les bogues et les problèmes de sécurité qui ont été signalés sont corrigés et sont publiés dans des versions ponctuelles régulières.
Après cette période de support actif de deux ans, chaque branche est ensuite prise en charge pour une année supplémentaire uniquement pour les problèmes de sécurité critiques.
Les mises à jour pendant cette période sont effectuées selon les besoins: il peut y avoir plusieurs versions ponctuelles, ou aucune, selon le nombre de rapports.


php

Versions

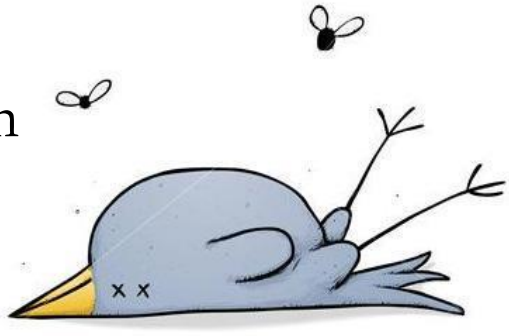

1 Jan 2015
1 Jan 2016
1 Jan 2017
1 Jan 2018
1 Jan 2019
1 Jan 2020
1 Jan 2021
1 Jan 2022
5.4
5.5
5.6
7.0
7.1
7.2


Active support
Security fixes only
End of life



Versions

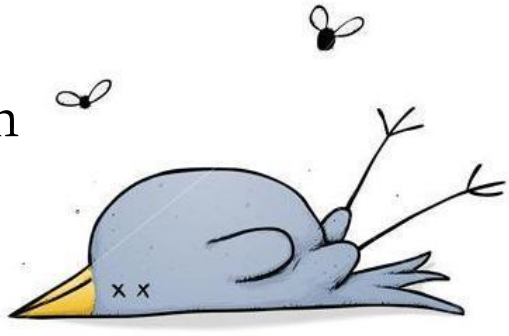
Depuis juin 2011 et le nouveau processus de livraison de PHP, le cycle de livraison de PHP est d'une mise à jour annuelle comportant des changements fonctionnels importants, la durée de vie d'une branche est de 3 ans et trois branches stables sont maintenues.
Cela signifie que lorsqu'une nouvelle version de PHP 5.x sort, la version 5.x-3 n'est plus supportée.
Depuis la sortie de PHP 5.6, la configuration est :
branche de développement : 7.1 ;
branches encore supportées (corrections de bugs et de la sécurité) : 7.0, 5.6 ;
Plus aucune branche < 5.6 n'est supportées.



Suppression de tag

Certains tags vont être supprimés.

<% tag d'ouverture
<%= tag d'ouverture avec echo
%> tag de fermeture
(<script\s+language\s*=\s*(php| »php »|'php')\s*>)i tag d'ouverture
(</script>)i tag de fermeture

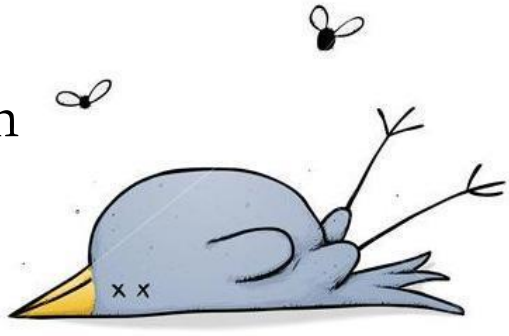


Appels statique de méthodes

Les appels statiques aux méthodes qui ne sont pas déclarées avec le mot-clé **static** sont dépréciés et peuvent être supprimés dans le futur.
Le fait de déclarer des propriétés ou des méthodes comme statiques vous permet d'y accéder sans avoir besoin d'instancier la classe. On ne peut accéder à une propriété déclarée comme statique avec l'objet instancié d'une classe

```
<?php
class foo {
function bar() {
 echo 'Je ne suis pas statique !'; }
}
foo::bar();
?>
```

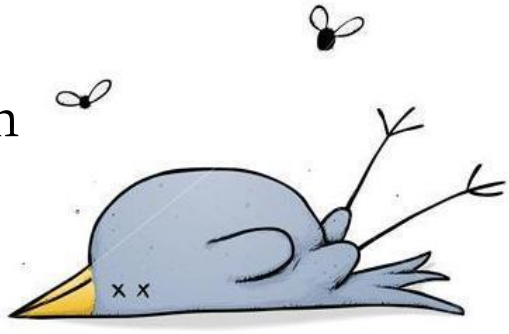
```
<?php
class  Ouistatic {
 static function bar() {
    echo 'Je suis statique !';
      }
  }
Ouistatic::bar();
?>
```



Constructeurs PHP 4

Les constructeurs de style PHP 4 (méthodes ayant le même nom que la classe dans laquelle elles sont définies) sont obsolètes et seront supprimés

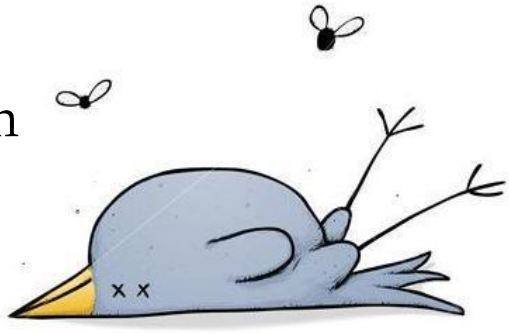
```
<?php
  class foo {
     function foo() {
        echo 'Je suis le constructeur';
      }
   }

?>
```



salt de la fonction password_hash()

L'option salt de la fonction password_hash() est dépréciée pour épargner aux développeurs de générer leurs propres salts (habituellement non sécurisés). La fonction elle-même génère cryptographiquement un salt sécurisé en l'absence d'un salt fourni par le développeur. Donc, la génération d'un salt sur mesure ne sera plus nécessaire.

```
<?php $options = [
   'cost' => 10,
    'salt' => 'salt-deprecated-salt-deprec',
];
echo password_hash("rasmuslerdorf", PASSWORD_BCRYPT, $options);
?>
```

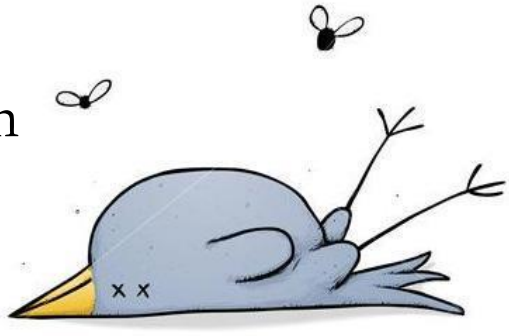


LDAP_SORT

La fonction ldap_sort() est obsolète

La fonction trie les entrées d'un résultat LDAP

```
<?php
ldap_sort($link, $result, 'givenname');
ldap_sort($link, $result, 'sn');

?>
```



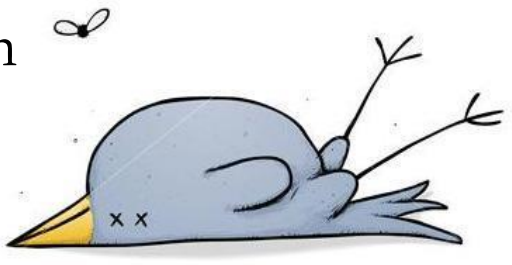
Elements déprécies

Variables nom dynamiques

Des variables variables dans global

La syntaxe n'est pas la même et donc l'interprétation

Syntax	Old Interpretation	New Interpretation
`$$var['key1']['key2']`	`${$var['key1']['key2']}`	`($$var)['key1']['key2']`
`$var->$prop['key']`	`$var->{$prop['key']}`	`($var->$prop)['key']`
`$var->$prop['key']()`	`$var->{$prop['key']}()`	`($var->$prop)['key']()`
`Class::$var['key']()`	`Class::{$var['key']}()`	`(Class::$var)['key']()`



Hexadécimal ne sont plus des valeurs numériques

C'est maintenant des caractères

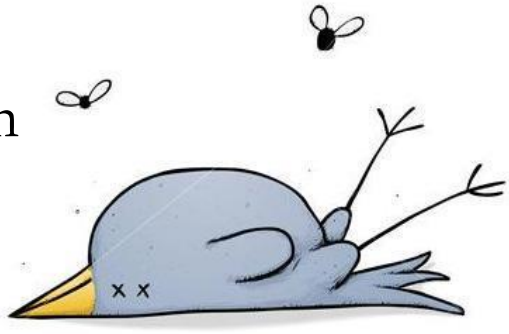
```
<?php
var_dump("0x123" == "291");
var_dump(is_numeric("0x123"));
var_dump("0xe" + "0x1");
var_dump(substr("foo", "0x1"));

?>
```

```
bool(true)
bool(true)
int(15)
string(2) "oo"
```

5.6

```
bool(false)
bool(false)
int(0)
Notice: A non well formed numeric
value encountered in /tmp/test.php
on line 5
string(3) "foo"
```

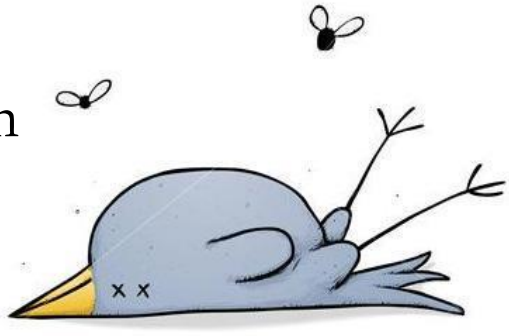
7



Fonction mcrypt

Les fonctions suivantes sont désormais obsolètes :
mcrypt_cbc()
mcrypt_cfb()
mcrypt_ecb()
mcrypt_ofb()

libmcrypt est un projet mort, non maintenu depuis plus de 8 ans, la dernière version **2.5.8** a été publiée en **février 2007 !**... et malgré les nombreux tickets ouverts, aucune activité.
La cryptographie est un élément essentiel de la gestion de la sécurité.
Si on regarde en arrière, les failles découvertes et corrigées dans divers logiciels, la nécessite d'augmenter les standards de sécurités et d'abandonner les vieux algorithmes, comment peut-on imaginer utiliser une vieillerie de plus de 8 ans ?
Elle est remplacée par openSSL



Switch case

Avant, il était tout à fait possible d'avoir plusieurs '**default**' dans un switch case.
Aucune erreur n'était affichée alors que maintenant, votre script lèvera une erreur de type E_COMPILE_ERROR.

```
<?php
switch ($expr) {
 default: jamaisExecute();
 break;
default: execute();
}
?>
```

PHP 7 Nouveautés et migration

Sommaire

- Evolution de base
- **Le langage**
- Erreurs et Assertions
- Closure et générateur
- POO
- Migrer

- Les opérateurs de fusion et de comparaison (spaceship).
- Les constantes.
- Le déballage d'objets.
- Les nouvelles fonctions : array_column.
- Les expressions régulières.
- La cryptographie.
- Les fonctions qui évoluent.
- Le typage d'entrée et de retour..

PHP 7

PHP 7

Nouvel opérateur <=>

Il sera possible de définir le type de valeur que doit retourner la fonction..

derrière ce nom étrange **"spaceship"**

Un nouvel opérateur, <=>, d'où le nom vaisseau spatial (ce n'est pas une nouveauté PHP, il existe en ruby pour n'en citer qu'un) pour comparer deux valeurs et qui renvoie un entier strictement inférieur à 0, nul ou strictement supérieur à 0 suivant que son opérande de gauche soit, respectivement, inférieur, égal ou supérieur à celui de droite.

PHP 7

un nouvel opérateur : *expr* <=> *expr*

Il retourne 0 si les 2 expressions sont égales, 1 si l'expression de gauche est plus grande et -1 si l'expression de droite est plus grande.

```
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1
echo "a" <=> "a"; //  0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1
echo [] <=> []; // 0
echo [1, 2, 3] <=> [1, 2, 3]; // 0
echo [1, 2, 3] <=> []; // 1
echo [1, 2, 3] <=> [1, 2, 1]; // 1
echo [1, 2, 3] <=> [1, 2, 4]; // -1
```

PHP 7

un nouvel opérateur : *expr* <=> *expr*

Il peut notamment être utilisé pour classer une liste d'éléments.

```
if (($handle = fopen("people.csv", "r")) !== FALSE) {
                while (($row = fgetcsv($handle, 1000, ",")) !== FALSE) {
                                $data[] = $row;
                }
                fclose($handle);
}
usort($data, function ($left, $right) {
                return $left[1] <=> $right[1];
});
```

PHP 7

un nouvel opérateur : *expr* <=> *expr*

Pour comparer

```
$data = [
  ['name' => 'Ado', 'cars' => 2],
  ['name' => 'Tony', 'cars' => 4],
  ['name' => 'Ramirond', 'cars' => 3],
  ['name' => 'Woloski', 'cars' => 12]
];
function sortByCars($x, $y) {
      return $x['cars'] <=> $y['cars'];
 }
usort($data, 'sortByCars');
print_r($data);
```

PHP 7

PHP 7

un nouvel opérateur : *??*

Pour remplacer toutes vos ternaires dont la condition est un isset

```
$page = isset($_GET['page']) ? $_GET['page'] : 1
```

5.6

```
$page = $_GET['page'] ?? 1
```

7

PHP 7

un nouvel opérateur : *??*

Autre illustration, le contrôle de droits, avec pour grade
0 = non connecté, 1 = membre et 2 = administrateurs :

```
if (($_SESSION['grade'] ?? 0) >= MEMBRE) { /* ok */ }.
```

7

L'équivalent pour une version antérieure est :

```
if (isset($_SESSION['grade']) && $_SESSION['grade'] >= MEMBRE).
```

5.6

C'est un peu plus concis et évite une répétition de la variable concernée.

PHP 7

un nouvel opérateur : *??*

Pour remplacer toutes vos ternaires dont la condition est un isset
L'opérateur ?? n'émet pas d'erreur de type E_NOTICE si la variable (ou clé) n'existe pas et vous pourriez parfaitement en chaîner plusieurs
$langue = $_POST['langue'] ?? $_GET['langue'] ?? $_COOKIE['langue'] ?? 'en';
de façon à récupérer la première valeur non nulle en les parcourant de gauche à droite (post > get > cookie > valeur par défaut 'en').

```
$langue = $_POST['langue'] ?? $_GET['langue'] ?? $_COOKIE['langue'] ?? 'en';
```

PHP 7

Nouvelle fonction

fonction PHP 7 nouveau IntDiv d'addition () accepte deux paramètres, la valeur de retour est la valeur du premier argument divisé par le second argument et l'arrondissement.

```
<?php
echo IntDiv (9, 3), PHP_EOL;
echo IntDiv (10, 3), PHP_EOL;
echo IntDiv (5, 10), PHP_EOL;
?>
```

PHP 7

Noms réservés

PHP comporte de nouveaux mots réservés.
Attention donc, vous ne pouvez plus utiliser, pour nommer vos classes, namespaces ou traits,
les mots suivants :
- **int, float, bool, string, true, false, null, resource, object, scalar, mixed, numeric**.

PHP 7

Unicode

Si vous aimez placer des caractères Unicode dans vos chaines de caractères, cette nouveauté est pour vous : PHP 7.0 permet d'utiliser la syntaxe \u{xxxx} pour spécifier un caractère à partir de son code :

```
<?php
'echo "\u{1F408} \u{1F431} \u{1F638} \u{1F639} \u{1F63A}
\u{1F63B} \u{1F63C} \u{1F63D} \u{1F63E} \u{1F63F}
\u{1F640} - so cute!!!\n";'
?>
```

```
<?php
function getMoney() {
   echo "\u{1F4B0}";
}

getMoney();
?>
```

PHP 7

Visibilité des constantes

Les constantes de classe étaient d'office jusque là publiques,
ce qui n'était pas toujours souhaitable quand elles étaient destinées,
par exemple, à définir des valeurs fixes à usage purement interne et dont
la valeur est susceptible d'évoluer d'une version à l'autre.
PHP 7.1 permet d'en restreindre la visibilité en ajoutant le mot clé
private ou protected devant const, comme nous le ferions pour une méthode.
public est également possible de façon à rendre une telle constante
explicitement accessible de tous mais, bien entendu, par compatibilité,
si aucune visibilité n'est mentionnée (public, protected ni private),
elle sera implicitement publique

```
class Example {
const CONSTANTE_1=33; // public, implicite
public const CONSTANTE_2="";
protected const CONSTANTE_3= array();
private const CONSTANTE_4= 3.75;
                                }
```

PHP 7

Les tableaux de constantes à l'aide de define()

Les tableaux (Array) constants peuvent maintenant être définis avec la fonction **define().**
Dans PHP 5.6, ils pouvaient être défini seulement avec **const.**

```
<?php
define('LANGAGES', [
    'php',
    '.net',
    'jsp'
]);

echo  LANGAGES[1]; // affiche  " .net"
?>
```

PHP 7

Nouvelles constantes globales

Constantes predéfinies **PHP_INT_MIN**
Gd **IMG_WEBP**
Json **JSON_ERROR_INVALID_PROPERTY_NAME,**
JSON_ERROR_UTF16
LibXML **LIBXML_BIGLINES**
Posix
Zlib

PHP 7

Classes anonymes

La prise en charge des classes anonymes a été ajouté à travers l'instanciation ***new class***.
Celle-ci peut être utilisée au lieu de définir toute une classe pour des objets "jetables" :

```
<?php
class Logger
{
    public function log($msg)
    {
        echo $msg;
    }
}
$util->setLogger(new Logger());

// PHP 7+ code
$util->setLogger(new class {
    public function log($msg)
    {
        echo $msg;
    }
});

?>
```

PHP 7

Classes anonymes

La prise en charge des classes anonymes a été ajouté à travers l'instanciation ***new class***.
Celle-ci peut être utilisée au lieu de définir toute une classe pour des objets "jetables" :

```
<?php
interface Logger {
    public function log(string $msg);
}
class Application {
    private $logger;
    public function getLogger(): Logger {
        return $this->logger;
    }
    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }
}

$app = new Application;
$app->setLogger(new class implements Logger {
    public function log(string $msg) {
        echo $msg;
    }
});
var_dump($app->getLogger());
?>
```

PHP 7

Indice de chaîne négatifs

fonctions PHP qui effectuent du découpage de chaînes basées sur des indices et/ou des longueurs : toutes ces fonctions gagnent un comportement uniforme dont la référence n'est autre que **substr** en acceptant des indices comme des longueurs négatifs, ce qui n'était pas nécessairement le cas de toutes jusque là.

Un exemple : toutes les fonctions strpos (strpos, mb_strpos, grapheme_strpos, etc) n'acceptaient pas une position de recherche négative. **strpos**("abcdefabc", "abc", -3) émettait une erreur Warning: strpos(): Offset not contained in string et renvoyait false. Maintenant, par -3, elle débutera sa recherche de sous-chaîne 3 "caractères" avant sa fin.

```
<?php
// Nous pouvons chercher le caractère, et ignorer tout ce qui est avant l'
offset
$newstring = 'abcdef ';
$pos = strpos($newstring, 'a', -1); // $pos = 0, non pas 7
?>
```

PHP 7

Indice de chaîne négatifs

Pour avoir un accès direct au X^{ième} caractère d'une chaîne de caractères **en partant de la fin** ?

```
<?php
$newstring = 'abcdef abcdef';
echo $pos = strpos($newstring, 'b', 0);
var_dump($newstring[-2]);
?>
```

PHP 7

Les tableaux et list

List syntaxe a été étendue pour supporter les clés des tableaux associatifs.
Il n'y a donc plus lieu de laisser des paramètres vides pour ignorer une valeur et chaque variable est alors précédée de la clé et d'une flèche.

Avec PHP 5, **list()** assigne les valeurs commençant avec le paramètre le plus à droite.
Avec PHP 7, **list()** commence avec le paramètre le plus à gauche.

```
<?php
  $info = array('coffee', 'brown', 'caffeine');
  list($a[0], $a[1], $a[2]) = $info;
  var_dump($a);
?>
```

```
<?php
$result = $pdo->query("SELECT id, name, salary FROM employees");
while (list($id, $name, $salary) = $result->fetch(PDO::FETCH_NUM)) {
    echo " . <a href=\"info.php?id=$id\">$name</a><br />\n" .
        $salary ";
}
?>
```

PHP 7

Les tableaux et list

List syntaxe a été étendue pour supporter les clés des tableaux associatifs.

Il n'y a donc plus lieu de laisser des paramètres vides pour ignorer une valeur et chaque variable est alors précédée de la clé et d'une flèche.

```
<?php
$data = [
    ["id" => 1, "name" => 'Stéphane'],
    ["id" => 2, "name" => 'Charlotte'],
];

// list() style
list("id" => $id1, "name" => $name1) = $data[0];

// [] style
["id" => $id1, "name" => $name1] = $data[0];

// list() style
foreach ($data as list("id" => $id, "name" => $name)) {
    // logic here with $id and $name
}

// [] style
foreach ($data as ["id" => $id, "name" => $name]) {
    // logic here with $id and $name
}
```

PHP 7 call_user

PHP 7

Passage d'une fonction de rappel

Les méthodes statiques de classe peuvent aussi être passées sans instancier d'objet de cette classe, en passant le nom de la classe au lieu d'un objet à l'index 0.
Il est également possible de passer *'NomDeLaClasse::NomDeLaMethode'*.

```
<?php
// Un exemple de fonction de rappel
function my_callback_function() {
    echo 'hello world!';
}
// Un exemple de méthode de rappel
class MyClass {
    static function myCallbackMethod() {
        echo 'Hello World!';
    }
}
// Type 1 : Fonction de rappel simple
call_user_func('my_callback_function');
// Type 2 : Appel d'une méthode statique de classe
call_user_func(array('MyClass', 'myCallbackMethod'));
// Type 3 : Appel d'une méthode objet
$obj = new MyClass();
call_user_func(array($obj, 'myCallbackMethod'));
// Type 4 : Appel d'une méthode statique de classe (depuis PHP 5.2.3)
call_user_func('MyClass::myCallbackMethod');
```

PHP 7

Passage d'une fonction de rappel

Pour accompagner ce changement, la hiérarchie des classes d'exceptions a été revue, avec l'introduction d'une interface **Throwable** commune à tous les types d'exceptions.

Les exceptions usuelles continuent d'hériter de **Exception**, alors que les erreurs héritent quant à elles de **Error**, qui est une classe-sœur de **Exception** :

```
Class toto extends Exception{}
Class tata extends Error{}
try{
   throw new toto('message Exception');
 }
catch( Throwable $t)
     { echo $t->getMessage(),PHP_EOL; }
Try
     { throw new tata('message Erreur');
}
catch( Throwable $t){
      echo $t->getMessage(),PHP_EOL;
}
```

PHP 7

PHP annonce la disponibilité de sa version 7.3.0 alpha 1 depuis le 7 juin 2018
Plusieurs bogues ont également été corrigés :

- manque de mémoire dans zend_register_functions() en mode ZTS)
- les exceptions non interceptées ne sont pas formatées correctement lorsque error_log est défini sur syslog)
- les opérations mathématiques convertissent les objets en entier
- erreur fatale au lieu de l'exception erreur lorsque la classe parente n'est pas trouvée
(- la session ne démarre pas après l'envoi des headers

PHP 7 Nouveautés et migration

Sommaire

- Evolution de base
- Le langage
- **Erreurs et Assertions**
- Closure et générateur
- POO
- Migrer

- La nouvelle hiérarchie d'exceptions dans PHP 7.
- Exception Constructeur.
- Exception du Moteur.
- Exception d'Erreur.
- Les Erreurs presque fatales.
- Throwable et Userland.
- Appel de méthode sur non-objet.
- Assertions et Expectations.

PHP 7

vous connaissez les erreurs fatales, les alertes, les erreurs d'analyse ou les notices.
Nous allons découvrir dans ce chapitre une façon différente de les erreurs.
Nous allons, créer nos propres types d'erreurs.
exceptions sont des erreurs assez différentes, qui ne fonctionnent pas de la même façon
Cette nouvelle façon de gérer ses erreurs est assez pratique.
Vous pouvez attraper l'erreur pour l'afficher comme vous voulez que d'avoir un fameux« Warning ».

PHP 7

Intégration des exceptions au moteur

Une refonte énorme a été débutée dans la façon de signaler des erreurs spécifiques : son moteur ne se contente(ra) plus d'émettre systématiquement une simple erreur mais lèvera en lieu et place une Exception de type adapté

PHP 7 erreurs

PHP 7

Nous allons créer une fonction avec la gestion des erreurs
Nous allons « lancer » une exception
Le constructeur demande en paramètre le message, le code d'exception et l'exception précédente, paramètres facultatifs

```
<?php
function additionner($a, $b) {
if ( !is_numeric($a) || !is_numeric($b))
 {
  throw new Exception('Les deux valeurs doivent être num')
  }
return $a + $b;
}
echo additionner (12, 3 ) , '<br />' ;
echo  additionner('azerty' ,  54 )  , '<br />' ;
echo additionner(1, 8);
```

Cela génère une erreur fatale car une exception non *catchée* génère une erreur

PHP 7 erreurs

PHP 7

Attrapons des erreurs

si une exception est lancée, alors on attrapera celle-ci afin qu'aucune erreur fatale ne soit lancée
et que de tels messages ne s'affichent plus
À l'aide de **catch** et **try**

```
<?php
function additionner($a, $b) {
if ( !is_numeric($a) || !is_numeric($b))
 {
  throw new Exception('Les deux valeurs doivent être num')
  }
return $a + $b;
}
try {

echo additionner (12, 3 ) , '<br />' ;
echo  additionner('azerty' ,  54 )  , '<br />' ;
eco additionner(1, 8);
}
catch (Exception $e) {
}
```

PHP 7

Attrapons des erreurs

seul premier calcul s'affiche, il n'y a plus d'erreur !
En revanche les deux autres résultats ne sont pas affichés.
Nous allons afficher le message d'erreur.
Il faut appeler la méthode **getMessage()** .
Pour récupérer le code d'erreur, il faut appeler getCode().

```
<?php
…
catch (Exception $e)
{
   echo 'une exception : ', $e->getMessage();
}
```

PHP 7

Attrapons des erreurs

La troisième instruction du bloc **try** n'a pas été exécuté
C'est normal puisque la deuxième instruction a interrompu la lecture du bloc.
Si vous interceptez les exceptions comme nous l'avons fait,
alors le script n'est pas interrompu.

```
<?php
…
try {

echo additionner (12, 3 ) , ‘<br />’ ;
echo  additionner('azerty' ,  54 )  , ‘<br />’ ;
eco additionner(1, 8);
}
```

PHP 7

Intégration des exceptions au moteur

Une refonte énorme a été débutée dans la façon de signaler des erreurs spécifiques : son moteur ne se contente(ra) plus d'émettre systématiquement une simple erreur mais lèvera en lieu et place une Exception de type adapté

```
try {
    var_dump(3 % 0);
}
   catch (DivisionByZeroError $e) {
       echo "On cheche à deviser par 0 ?";
} finally {
      echo "finally";
}
```

PHP 7

Intégration des exceptions au moteur

On peut attraper plusieurs types d'exceptions différents avec le mot-clé catch : il suffit de séparer les types d'exceptions par un |.

```
try {
switch (mt_rand(0, 2)) {
case 0: throw new Plop();
case 1: throw new Blah();
case 2: throw new Another(); }
 }
catch (Plop | Blah $e) {
printf("1er catch : %s\n", get_class($e));
 }
catch (Another $e) {
printf("2nd catch : %s\n",get_class($e)); }
```

PHP 7 erreurs

PHP 7

Intégration des exceptions au moteur

Une refonte énorme a été débutée dans la façon de signaler des erreurs spécifiques : son moteur ne se contente(ra) plus d'émettre systématiquement une simple erreur mais lèvera en lieu et place une Exception de type adapté

```
function division($x, $y){
  if(!$y){
    throw new Exception('division par 0 impossible');
     }
   return "Résultat : ".$x/$y .'<br />';
                                                    }
try {echo division(4,0);}
catch(Exception $e){

echo 'Message de l\'exception : '.$e->getMessage().'<br />';
echo 'Code de l\'exception : '.$e->getCode().'<br />';
echo 'Ligne de l\'exception : '.$e->getLine().'<br />';
echo 'Trace de l\'exception : '.print_r($e->getTrace()).'<br />';
echo 'Erreur précédente : '.$e->getPrevious().'<br />'; echo
'representation de l\'objet : '.$e->__toString().'<br />';
                                }
```

PHP 7 erreurs

PHP 7

Intégration des exceptions au moteur

Une refonte énorme a été débutée dans la façon de signaler des erreurs spécifiques : son moteur ne se contente(ra) plus d'émettre systématiquement une simple erreur mais lèvera en lieu et place une Exception de type adapté

Error
ParseError : levée quand le code interprété lors d'un `eval` ou une inclusion ((include|require)(_once)) comporte une erreur de syntaxe
TypeError : lancée quand le type ne correspond pas (notamment par rapport au typehinting ou les fonctions de rappel)
ArgumentCountError (PHP 7.1) : levée quand la fonction/méthode reçoit trop ou pas assez d'arguments
ArithmeticError : lancée sur les opérations considérées comme "insensées" telles `intdiv(PHP_INT_MIN, -1)` ou encore `<<(=)`/`>>(=)` avec un décalage de bits négatif
DivisionByZeroError : émise par un modulo (`%`) comme par `intdiv` lors d'une division par zéro

PHP 7

Le bloc **throw** va servir de déclencheur : dans le cas où une exception a été rencontrée, on va la « lancer » c'est-à-dire instancier la classe PHP prête à l'emploi Exception.
Le second bloc, **catch**, va nous permettre de capturer notre exception et de créer un objet qui va contenir les informations relatives à l'erreur levée.
On utilisera également dans notre code un bloc try qui va nous permettre de faciliter la capture d'exceptions potentielles.
La fonction qui peut poser problème doit se trouver à l'intérieur de ce bloc **try**.
Dans le cas où aucune exception n'est rencontrée, le code s'exécutera normalement.
Dans le cas contraire (si une exception se déclenche), notre script lancera une exception.
Dès qu'une exception est lancée, le PHP va chercher dans notre script le premier bloc catch pour l'exécuter.
Si aucun bloc catch n'est trouvé, une erreur fatale sera renvoyée à moins qu'un gestionnaire d'erreur personnalisé ait été défini avec la fonction **set_exception_handler().**

Throwable

PHP 7

PHP 7

Passage d'une fonction de rappel

Throwable

```
interface Throwable
  |- Error implements Throwable
      |- ArithmeticError extends Error
          |- DivisionByZeroError extends ArithmeticError
      |- AssertionError extends Error
      |- ParseError extends Error
      |- TypeError extends Error
          |- ArgumentCountError extends TypeError
  |- Exception implements Throwable
      |- ClosedGeneratorException extends Exception
      |- DOMException extends Exception
      |- ErrorException extends Exception
      |- IntlException extends Exception
      |- LogicException extends Exception
          |- BadFunctionCallException extends LogicException
              |- BadMethodCallException extends BadFunctionCallException
          |- DomainException extends LogicException
          |- InvalidArgumentException extends LogicException
          |- LengthException extends LogicException
          |- OutOfRangeException extends LogicException
      |- PharException extends Exception
      |- ReflectionException extends Exception
      |- RuntimeException extends Exception
          |- OutOfBoundsException extends RuntimeException
          |- OverflowException extends RuntimeException
          |- PDOException extends RuntimeException
          |- RangeException extends RuntimeException
          |- UnderflowException extends RuntimeException
          |- UnexpectedValueException extends RuntimeException
```

PHP 7

Expectations

assert() est une construction de langage en PHP 7, autorisé pour la définition des expectations : les assertions qui prennent effet dans les environnements de développement et de test, mais qui sont optimisées de sorte à avoir un coût zéro en production.
Il est recommandé de n'utiliser les assertions que comme outil de débogage.
Vous pouvez les utiliser pour les vérifications d'usage : ces conditions doivent normalement être vraies, et indiquer une erreur de programmation si ce n'est pas le cas.
Vous pouvez aussi vérifier la présence de certaines extensions ou limitations du système.

```
<?php
class CustomError extends AssertionError {}

assert(true == false, new CustomError('True is not false!'));
echo 'Hi!';
?>
```

PHP 7 assert

PHP 7

Expectations Assert

PHP 7 exception pour la compatibilité descendante et amélioré la fonction ancienne assert().
Il peut atteindre un coût nul affirmation dans un environnement de production, et de fournir une exception est levée et la possibilité de personnaliser l'erreur.
Les anciennes versions de l'API à des fins de compatibilité continueront d'être maintenus, assert() est maintenant une structure linguistique qui permet le premier argument est une expression, pas seulement une chaîne à être calculée à tester ou un booléen.

```
 <? php
ini_set ( 'zend.assertions', 0) ;

affirmer (true == false);
echo 'Salut!';
?>
```

PHP 7 Nouveautés et migration

Sommaire

- Evolution de base
- Le langage
- Erreurs et Assertions
- **Closure et générateur**
- POO
- Migrer

- Closure : liaison à l'invocation.
- Générateur : valeurs de retour et délégation

PHP7

PHP 7

Closure

Les closures, représentant ce qu'on appelle des fonctions anonymes, sont un moyen de créer des fonctions à la volée

```
<?php
function saigon($who) {
        return function() use ($who) {
                echo "$who la Nantaise.<br />";
            };
    }
$greeter = saigon("Lulu");
$greeter();
?>
```

PHP 7 closure

PHP 7

Closure

Les closures sont principalement utilisées en tant que fonctions de rappel. Les fonctions de rappel sont des fonctions demandées par d'autres fonctions pour effectuer des tâches spécifiques.

```
<?php
$var = function() {
return 'I am a ' . func_get_arg(0);
};
print_r($var('Closure'));
```

```
<?php
$additionneur = function($nbr)
{
return $nbr + 5;
};
$listeNbr = [1,2,3,4,5] ;
$listeNbr = array_map ($additionneur, $listNbr);
```

PHP 7 closure

PHP 7

Closure

Importer une variable dans une closure

```
<?php
$quantite = 5
$additionneur = function($nbr) use($quantite)
{
return $nbr + $quantite;
};
$listeNbr = [1,2,3,4,5] ;
$listeNbr = array_map {$additionneur, $listNbr};
```

La quantité fixée à 5 ne peut être changée car la variable a été importée dans la closure dès sa création.

PHP 7 closure

PHP 7

Closure

Nous allons passer par une fonction chargée de renvoyer une closure
Cette fonction prendra en argument la quantité à ajouter.
Nous importerons ainsi cette quantité dans notre closure, puis nous la retournerons

```
<?php
$function creerAdditionneur($quantite)
{
    return function($nbr) use($quantite)
    {
        return $nbr + $quantite;
    };
}
$listeNbr = [1,2,3,4,5] ;
$listeNbr = array_map {$creerAdditionneur(5), $listNbr};
var_dump($listNbr);
```

PHP 7 closure

PHP 7

Closure::call()

La méthode Closure::call()
Lier temporairement une closure à un objet
Pour de petites opérations, la solution précédente peut s'avérer lourde. Depuis la version 7 de PHP, il est possible de lier la closure à un objet **le temps d'un appel**

```
<?php
class Value {
    protected $value;
    public function __construct($value) {
        $this->value = $value;
    }
    public function getValue() {
        return $this->value;
    }
}
$three = new Value(3);
$four = new Value(4);

$closure = function ($delta) { var_dump($this->getValue() + $delta); };
$closure->call($three, 4);
$closure->call($four, 4);
?>
```

PHP 7 closure

PHP 7

Closure::call()

La méthode Closure::call()

5.6

```php
<?php
class NameRegister {
    private $name = "Prosper";
}
// Closure
$getName = function() {
  return $this->name;
  };
$getTheName = $getName->bindTo(new NameRegister, 'NameRegister');
echo $getTheName();
?>
```

7

```php
<?php
class NameRegister {
    private $name = "Prosper";
}
$getName = function() {
   echo $this->name;
};
$getName->call(new NameRegister());
?>
```

PHP 7 closure

PHP 7

Closure::call()

La méthode Closure::call() est devenue plus performante.
Une façon plus courte de lier temporairement une fermeture à la portée d'un objet et l'invoquer.

```
<?php
  class A {
    private $x = 1;
   }
// PHP 7+ code, Define
 $value = function() {
    return $this->x;
    };
 print($value->call(new A));
?>
```

PHP 7 closure

PHP 7

Closure

Dans l'exemple ci dessus, on obtient bien 3 car même si $a n'existe pas dans le scope de "executer_closure", la closure se "souvient" de la référence originale.

Si en JavaScript n'importe quelle variable du scope parent peut être référencée dans le scope de la closure, en PHP il faut explicitement définir avec le mot clé **use** quelles variables seront importées.

Malgré cette limitation, c'est extrêment pratique car on peut promener la closure dans n'importe quel scope, sans se soucier de la visibilité des variables référencées

```
<?php // version 5.3+
 function fabrique_closure () {
      $a = 1;
      return function ($b) use ($a) {
           return $a + $b;
            };
       }
function executer_closure ($f, $b) {
   return $f($b); // additionner
   }
$f = fabrique_closure();
echo executer_closure($f, 2); // 3
```

PHP 7

PHP 7

unserialize

Cette fonctionnalité vise à garantir une meilleure sécurité lorsque la désérialisation d'objets est effectuée avec des données non fiables.

Elle empêche les injections de code possible en permettant au développeur de whitelister les classes qui peuvent être désérialisées.

```
<?php

// Convertit tous les objets vers un objet __PHP_Incomplete_Class
$data = unserialize($foo, ["allowed_classes" => false]);

// Convertit tous les objets vers un objet __PHP_Incomplete_Class
// excepté ceux de MyClass et MyClass2
$data = unserialize($foo, ["allowed_classes" => ["MyClass", "MyClass2
"]]);

// Comportement par défaut (le même que lorsqu'on omet le deuxième
argument)
// qui accepte toutes les classes
$data = unserialize($foo, ["allowed_classes" => true]);
```

PHP 7

PHP 7

Unserialize

Elle proposait donc de modifier la fonction unserialize() de PHP pour permettre d'interdire la dé-sérialisation d'objets ou de limiter celle-ci à un ensemble de classes listées.

```
<?php

$data = unserialize($foo);
// Ceci va convertir tous les objets en objets __PHP_Incomplete_Class
$data = unserialize($foo, ["allowed_classes" => false]);

// Ceci va convertir tous les objets à l'exception des instances de
MyClass et de MyClass2 en objets __PHP_Incomplete_Class

$data = unserialize($foo, ["allowed_classes" => ["MyClass",
"MyClass2"]);
// Ceci va accepter toutes les classes comme étant par défaut

$data = unserialize($foo, ["allowed_classes" => true]);
```

PHP 7 générateur

PHP 7

Générateurs

l'instruction **yield**
Cela permet de mettre en place les générateurs.

Une fonction générateur ressemble à une fonction normale, sauf qu'au lieu de retourner une valeur, un générateur **yield** retourne autant de valeurs que nécessaire.

```
<?php
function generateAnimal() {
  echo "Je suis dans le générateur\n";
  yield "Panda";
  echo "Je suis retourné dans le générateur\n";
  yield "Lama";
  echo "je suis de retour\n";
  yield "Alpaga";
  echo "plus de d'animaux\n";
}

$generator = generateAnimal();
foreach ($generator as $value) {
                echo "j'ai reçu $value \n";
 }
```

PHP 7 générateur

PHP 7

Générateurs intérêt

Admettons que je veux faire un foreach sur un tableau d'un millions de lignes.
Pour faire un array de 1 Million de valeurs
Mais cela prend un peu de mémoire.
Utilisons notre générateur

```
<?php
function xrange($min, $max) {
    for ($i = $min; $i < $max; $i++) yield $i;
    }
foreach (xrange(1,1000000) as $value) {
    echo $value;
}
```

PHP 7 générateur

PHP 7

Générateurs intérêt

Pour lire un fichier:

```
<?php
function getLinesFromFile($fileName) {
  $fileHandle = fopen($fileName, 'r');
   while (false !== $line = fget($fileHandle)) {
   yield $line;
}
  fclose($fileHandle);
}
 $lines = getLinesFromFile($fileName);
 foreach ($lines as $line) {
// do something with $line
 }
```

PHP 7 générateur

PHP 7

Générateurs

Avec PHP 7.0, les générateurs ont gagné deux améliorations.
Generators et **return**
Il n'était précédemment pas possible de retourner une valeur depuis un générateur. PHP 7.0 le permet désormais
Le generator peut contenir la fonction return comme une autre fonction
Mais elle n'apparaîtra pas dans une boucle foreach. C'est la méthode getReturn qui l'affichera

```
<?php

function plop() {
        yield 100;
        yield 200;
        return 42;
}
foreach (($generator = plop()) as $val) {
        var_dump($val);
}
var_dump( $generator->getReturn() );
```

PHP 7 générateur

PHP 7

Générateur délégation

le mot-clef yield a été enrichi et il est désormais possible d'utiliser **yield from** pour générer des valeurs depuis un traversable

```
<?php

function test() {
                yield from [10, 20, 30];
               }
foreach (test() as $val) {
                var_dump($val);
                }
```

PHP 7 générateur

PHP 7

Générateur délégation

le mot-clef yield a été enrichi et il est désormais possible d'utiliser **yield from** pour générer des valeurs depuis un traversable

```
<?php

function sub_generator_1() {
                yield 10;
                yield 20; }
function sub_generator_2() {
                yield from [ 'aa', 'bb', ];
                }
function delegating_generator() {
                yield from sub_generator_1();
                yield from sub_generator_2();
                }
foreach (delegating_generator() as $val) { var_dump($val); }
```

PHP 7 Nouveautés et migration

Sommaire

- Evolution de base
- Le langage
- Erreurs et Assertions
- Closure et générateur
- **POO**
- Migrer

- Analyse lexicale contextuelle.
- Dépréciation des constructeurs PHP 4.
- Déclarations groupées.
- Les classes anonymes.
- Les traits

PHP 7

Rappel dur l'objet

La programmation par objets ou par procédures classiques est une affaire de goût, et on trouve des gens compétents quelle que soit la méthode utilisée .
La programmation orientée objet comporte toutefois les avantages suivants :

- un code compréhensible
- chaque objet regroupe ses propriétés et ses méthodes. On comprend tout de suite à quoi s'applique une fonction et à quoi correspond une variable.
- un code réutilisable
- les types d'objets peuvent servir de bases pour d'autres types d'objets, en ne réimplémentant que ce qui change entre les deux.
- un code modulaire
- chaque type contient son propre contexte et n'interfère avec les autres types que via des interfaces bien définies.

Il est facile d'isoler un module ou de développer des modules séparément.

PHP 7
Rappel dur l'objet

Une Classe

Une classe est un modèle de données.
On peut la voir comme une famille d'objets.
Tous les objets d'une même classe sont similaires.
Ils partagent les mêmes attributs et méthodes.
On peut ainsi imaginer une classe représentant les voitures.
Toutes les voitures (tous les objets de la classe voiture) ont des plaques d'immatriculation, un moteur avec une certaine puissance et un nombre de portières identifiables (ils ont des attributs communs).

Tous les objets de cette classe ont aussi des méthodes pour démarrer, freiner, accélérer, tourner, etc.

PHP 7
Rappel dur l'objet

Un objet

Chaque objet a des attributs qui lui sont propres.
Mon compte en banque a un attribut qui définit le numéro de compte, un autre qui définit le solde actuel, un troisième qui est une liste des différentes opérations, et ainsi de suite.
Les attributs peuvent .être vus comme les caractéristiques propres de l'objet.
Les objets peuvent avoir des méthodes.
Il s'agit des actions qu'on peut appliquer à un objet.
Toujours en prenant mon objet de compte en banque, il existe une méthode pour le solder, une pour ajouter une écriture, une pour le déplacer dans une autre banque, etc

PHP 7
Rappel dur l'objet

Utiliser une méthode

L'appel d'une méthode se fait de façon similaire à celui d'une fonction.
Comme pour les attributs, il suffit de séparer l'objet et la méthode par l'opérateur **- >**.

```
<?php
class voiture {
    function klaxonne()  {
    echo "Vous klaxonnez fort !";
    }
  }
  $ma_voiture = new voiture() ;
  $ma_voiture->klaxonne() ;// affiche Vous klaxonnez fort
?>
```

PHP 7

Utiliser une constante

Les syntaxes suivantes vous permettront d'accéder aux constantes

```
<?php
class TondeuseGazon {
const POUSSEE = 4 ;
 public $type ; }

$maTondeuse = new TondeuseGazon();
$maTondeuse->type = TondeuseGazon::POUSSEE
echo $maTondeuse->type ;
//affiche le chiffre 4
?>
```

A l'intérieur d'une méthode, il est aussi possible d'accéder aux constantes de l'objet en cours grâce à l'opérateur self ::

```
class TondeuseGazon {
const POUSSEE = 4 ;
 public $type ;
        function setTondeusePoussee ()
        {$this->type = self::POUSSEE
        }
}
```

PHP 7

Héritage

Pour définir un héritage, on utilise le mot-clé **extends** après le nom de la classe dans la déclaration.
L'héritage est probablement le concept le plus important de la programmation orientée objet.
C'est ce qui lui donne toute sa puissance.
Cela permet de réutiliser des classes pour en construire de nouvelle

```
<?php
 class Membre{
    public $statut='adhérent';
  };
  class Admin extends Membre
    { private $niveauAcces='administrateur';
      public function pseudo()
      {echo 'super admin';}
      public function contact()
      { }
    }?>
```

PHP 7

Autoloader

Avant d'être utilisés, ces fichiers doivent être inclus dans toutes les fonctions PHP include telles que require_once ( )
Au fur et à mesure que notre projet se développera, nous aurons de plus en plus de classes et le nombre d'instructions d'inclusion ne cessera d'augmenter
nous pouvons créer une classe de chargeur qui charge toute classe à la demande **spl_autoload_register ( )** ,
une fonction PHP spéciale qui nous permet de créer une série de fonctions pouvant être appelées pour charger n'importe quelle classe.
La fonction **function __autoload** est deprecateted

```
<?php
 if(!function_exists('classAutoLoader')){
function classAutoLoader($class){
                    $class=strtolower($class);
$classFile=$_SERVER['DOCUMENT_ROOT'].'/include/class/'.$class.'php';

if(is_file($classFile)&&!class_exists($classFile)) include $classFile;
                                        }
}
spl_autoload_register('classAutoLoader');
?>
```

PHP 7

Interface

Il existe un moyen d'imposer une **structure** à nos classes, c'est-à-dire d'obliger certaines classes à implémenter certaines méthodes.
Pour y arriver, nous allons nous servir des **interfaces**.

```
interface Movable
{ public function move($dest);
  const DEPART = 'Ma maison !';
                                    }
class Personnage implements Movable
  { public function move($dest){ }
              }
class Voiture implements Movable
   {public function move($dest) { }
   }
echo Personnage::DEPART;
echo Voiture::DEPART;
```

PHP 7

Droits d'accès

Nous avons utilisés devant nos noms de variables et fonctions
public : tout le monde peut accéder à l'élément ;
private : Seule la classe a le droit d'accéder à l'élément ;
protected : identique à **private**, sauf qu'un élément ayant ce droit d'accès dans une classe mère sera accessible aussi dans les classes filles

```php
<?php
class Membre
{
    private $pseudo;
    private $email;
    private $signature;
    private $actif;

    public function getPseudo()
    {//…. }

    public function setPseudo($nouveauPseudo)
    {//…. }
}
?>
```

PHP 7

Typage des paramètres

On peut maintenant spécifier le type que l'on attend dans la signature d'une fonction ou d'une méthode. Cela fonctionne en 2 modes.

Dans le premier mode les paramètres sont convertis dans le type attendu. Par exemple si l'on attend un Int et qu'on passe un Float(1,5), celui-ci sera arrondi et convertie en Int (1).

Il y a un deuxième mode qui pour être utilisé doit être déclaré au tout début du fichier. Dans ce mode, les paramètres ne sont pas convertis et un "Fatal Error" est déclenché en cas de passage d'un mauvais type.

```php
function sendHttpStatus(int $statusCode, string $message) {
     header('HTTP/1.0 ' .$statusCode. ' ' .$message);
}

sendHttpStatus(404, "File Not Found"); // integer and string passed
sendHttpStatus("403", "OK"); // string "403" coerced to int(403)
```

PHP 7

Typage des paramètres

Pour combattre efficacement les personnes qui prétendent que PHP est dangereux à cause de son typage dynamique.
Car, pour la première fois de son histoire, PHP intègre le typage statique des variables.
il s'agit de fonctionnalités facultatives, accessibles uniquement lorsque le typage strict est explicitement requis.

```php
<?php

function divide( int a, int b)
{
    return $a / $b;
}

var_dump( divide( 10 , 2) );
var_dump( divide( 10 , '2') );
var_dump( divide( 10, 'a') );
```

```
int(5)
int(5)
PHP Fatal error:  Uncaught TypeError: Argument 2 passed to divide() must be of the type integer, string given, called in /home/admin/test on line 10 and defined in /home/admin/test:3
Stack trace:
#0 /home/admin/test(10): divide(10, 'a')
#1 {main}
  thrown in /home/admin/test on line 3
```

PHP 7

Typage des paramètres

PHP 7 le généralise enfin par rapport aux types, les scalaires sont supportés, comme à la valeur retournée par une fonction/méthode.
Ces types sont int (entier), float (flottant tout en incluant les entiers), bool (booléen) et string (chaîne de caractères).
Pour les paramètres, la syntaxe est inchangée, l'éventuelle précision de type précède son nom

exemple : function charAt(string $string, int $offset) { /* ... */ }

La fonctionnalité « return type déclarations » permet de déclarer le format attendu en sortie de votre fonction.
Comme précédemment, vous obtiendrez une exception si une mauvaise valeur est retournée par votre fonction.

```
<?php
declare(strict_types=1);
function charAt(string $string, int $offset): string {
  return $string[$offset];
}
echo charAt('abc', 1); #
echo charAt(123, 1); #
```

PHP 7

Nullable Types

Pouvoir passer null, généralement pour indiquer « voici une valeur non spécifiée », est un besoin exprimé de nombreuses fois lors du développement de PHP 7.0 et après sa sortie.
Pour répondre à cette demande, PHP **7.1** introduit les types nullables : en positionnant un ? au début du nom d'un type, comme **?**int, on indique que la valeur null est acceptée

```
<?php
function fonc01(?int $a) {
var_dump($a);
}
fonc01(100);
fonc01(null);
```

Constructeurs PHP 4

Les constructeurs de style PHP 4
(méthodes ayant le même nom que la classe dans laquelle elles sont définies) sont obsolètes et seront supprimés à l'avenir.

PHP 7 émet E_DEPRECATED si le constructeur de PHP 4 est le seul constructeur défini dans la classe.

```
<?php
class foo {
    function foo() {
        echo 'Je suis le constructeur';
    }
}
?>
```

PHP 7 namespace

PHP 7

Namespaces en PHP

on importe facilement une classe d'un namespace donné avec l'utilisation de l'opérateur **use**

```
<?php
 namespace Users\formation
 {
   class Goms
   {
     const name = 'MBUNGU NGOMA';
   }
 }
 namespace Users\Production
  {
   class Goms
  {
 const name = 'MBUNGU';
  }
 }
 namespace {
   use Users\formation\Goms as formation;
   use Users\Graphcet\Goms as Production;
   echo formation::name; //MBUNGU NGOMA
   echo Production::name; //MBUNGU
 }
?>
```

PHP 7

Namespaces en PHP

Il peut arriver de devoir faire un paquet d'appels d'autres classes, fonctions et/ou constantes pour nos classes.
On peut utiliser les accolades dans le **use** et les inclusions se feront pour chaque classe dans l'accolade !
On peut ainsi grouper en une seule ligne tous nos appels, c'est plus clair et plus efficace.

```
// Pre PHP 7 code
use foo\bar\ClassA;
use foo\bar\ClassB;
use foo\bar\ClassC as C;

use function foo\bar\FunctionA;
use function foo\bar\FunctionB;

use const foo\bar\ConstA;
use const foo\bar\ConstB;

// PHP 7+ code
use foo\bar\{ClassA, ClassB, ClassC as C};
use function foo\bar\{fn_a, fn_b, fn_c};
use const foo\bar\{ConstA, ConstB, ConstC};
```

PHP 7

Classes anonymes

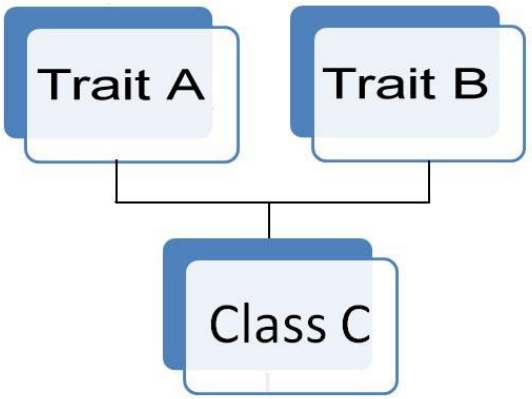
la possibilité de travailler avec des *classes anonymes*.
Cela vient étendre la notion de *fonctions anonymes* qui existait depuis PHP 5.3, en ajoutant tout ce qui est *objet* : héritage, implémentation d'interfaces…

```
<?php
$obj = new class ("au Grisbi") {
                protected $qui;
                public function __construct($qui) {
                $this->qui = $qui;
                                }
                public function hello() {
                   printf("Touche pas, %s !\n", $this->qui);
                    }
                 };
var_dump($obj);
$obj->hello();
?>
```

PHP 7

Traits

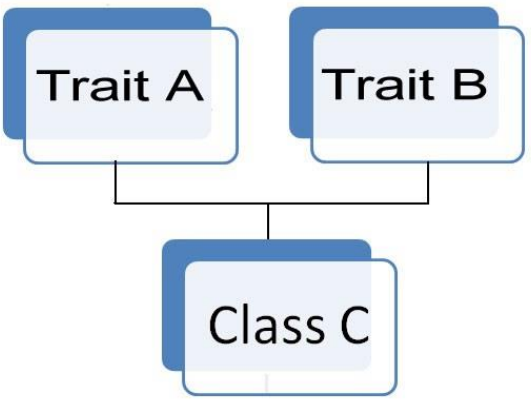
PHP intègre un moyen de réutiliser le code d'une méthode dans
Deux classes indépendantes.
Cette fonctionnalité permet ainsi de repousser les limites
héritage simple



PHP 7

Traits

Un trait caractérise en quelque sorte un héritage **horizontal**, indépendant de la hiérarchie des types.
Pour utiliser un trait, il n'est pas nécessaire que celui-ci partage une nature commune avec la classe qui l'utilise.
On peut le voir en fait comme un "morceau" de classe fournissant des fonctionnalités indépendantes de tout typage.



PHP 7

Traits

Soit deux classes, Writer et Mailer. La première est chargée d'écrire du texte dans un fichier, tandis que la seconde envoie un texte par e-mail.
Cependant il est agréable de mettre en forme le texte.
Pour ce faire, vous décidez de formater le texte en HTML.
Or, un problème se pose : vous allez devoir effectuer la même opération (celle de formater en HTML) dans deux classes complètement différentes et indépendantes

PHP 7

Traits

Soit deux classes, Writer et Mailer. La première est chargée d'écrire du texte dans un fichier, tandis que la seconde envoie un texte par e-mail.

```
<?php
class Writer
{
    public function write($text)
    {
        $text ='<p>date : '.Date ('d/m/Y').'</p>' ."\n".
                '<p>'.nl2br($text).'</p>';
         file_put_contents('fichier.txt' , $text);
     }
}
class Mailer
{
public function send($text)
{
$text ='<p>date : '.Date ('d/m/Y').'</p>' ."\n".
           '<p>'.nl2br($text).'</p>';
             mail( 'login@fai.tld' Test avec les traits' $text);
 }
}
?>
```

PHP 7

Traits

les traits sont un moyen d'externaliser du code
les traits définissent des méthodes que les classes peuvent utiliser

```
<?php
trait MonTrait
{
   public function hello()
   {
     echo 'Hello world ! ';
    }
}
class A
{
   use MonTrait;
}
class B
{
    use MonTrait;
}
$a = new A;
$a->hello();

$b = new B;
$ba->hello();
?>
```

PHP 7

Traits

Déclaration des traits …

```
<?php
Trait HTMLFormater
{
    public Function format ($text)
    {
   return '<p>Date: '.date('d/m/Y').'</p>'."\n".
    '<p>'. nl2br ($text). '</p>';
    }
}
```

PHP 7

Traits

nous allons modifier nos classes Writer et Mailer, afin qu'elles utilisent ce trait pour formater le texte qu'elles exploiteront
il vous faut utiliser le mot-clé **use** pour importer toutes les méthodes du trait

```
<?php
class Writer
{
   use HTMLFormater;
   public function write($text)
   {
      file_put_contents ('fichier.txt', $this->format ($text) ) ;
    }
}
class Mailer
{
   use HTMLFormater;
    public function send($text)
    {
       mail ( 'login@fai. tld', 'Test avec les traits', $this->format ($text));
    }
}
```

PHP 7

Plusieurs traits

Pour utiliser plusieurs traits il vous suffit de lister tous les traits à utiliser sépares par des virgules.

```
<?php
class Writer
{
 use HTMLFormater, TextFormater;
 public function write($text)
  {
   file_put__contents ( 'fichier.txt', $this->formatHTML ($text) ) ;
  }
 }
```

PHP 7 Nouveautés et migration

Sommaire

- Evolution de base
- Le langage
- Erreurs et Assertions
- Closure et générateur
- POO
- **Migrer**

- Etablir la checklist des points à vérifier pour une bonne migration.
- Identification des familles de problèmes potentiels en fonction de la version à migrer.
- Contournement des problèmes.

PHP 7

Avant de migrer entre PHP 5.6 et PHP 7.0, il existe plusieurs différences clés à prendre en compte

les nouvelles fonctionnalités , fonctions , constantes globales , classes et interfaces .

Mais répertorie également les fonctions modifiées les plus importantes , les modifications incompatibles avec les versions antérieures et les fonctionnalités obsolètes .

PHP 7

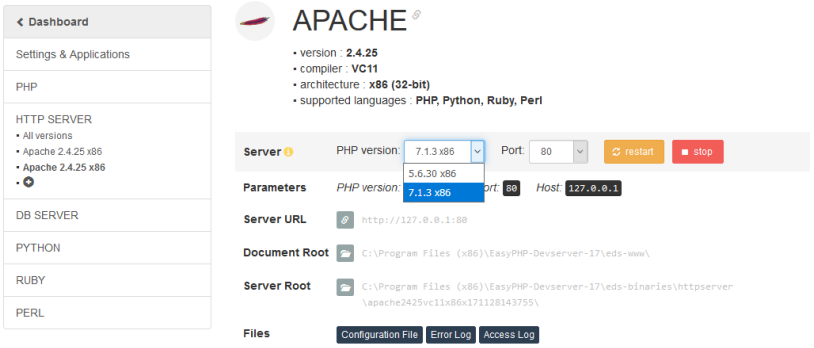
Test de compatibilités

Alors, comment vérifier que votre application est compatible PHP 7 ?
Il existe des outils pour cela.
En fonction de vos environnements de dev
L'un des plus connus est **Phan**, mais cela fait partie d'une de ses nombreuses fonctionnalités.
Il vaut mieux en prendre un qui est dédié pour cela et facile à utiliser et à installer.

PHP 7

Sur les serveurs locaux

Il est possible de changer de moteurs php pour voir des différences de Comportement et les messages d'alerte

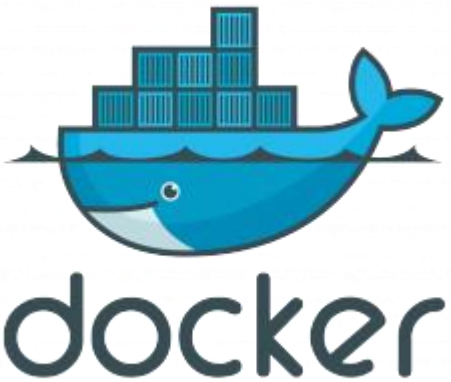


PHP 7

Test de compatibilités

On peut utiliser un conteneur Docker et notamment celui-là :
https://hub.docker.com/r/ypereirareis/php7cc/

Si votre projet fonctionne avec Docker



PHP 7

Test de compatibilités

Voici donc ma recette avec Atom pour retrouver les 3 fonctionnalités essentielles à la programmation PHP orientée objet : la navigation entre classes, l'auto-complétion et la correction syntaxique :
Installer les 4 packages suivant via le menu "Atom > Preferences" puis "+ Install" :
php-integrator-base, indexe votre code PHP et sert de base aux autres packages
php-integrator-linter, détecte des erreurs de syntaxe et fait des suggestions
php-integrator-navigation, transforme les noms de classes en liens qui pointent vers le fichier décrivant la classe
php-integrator-autocomplete-plus, propose des listes de bouts de code pertinents pendant la saisie

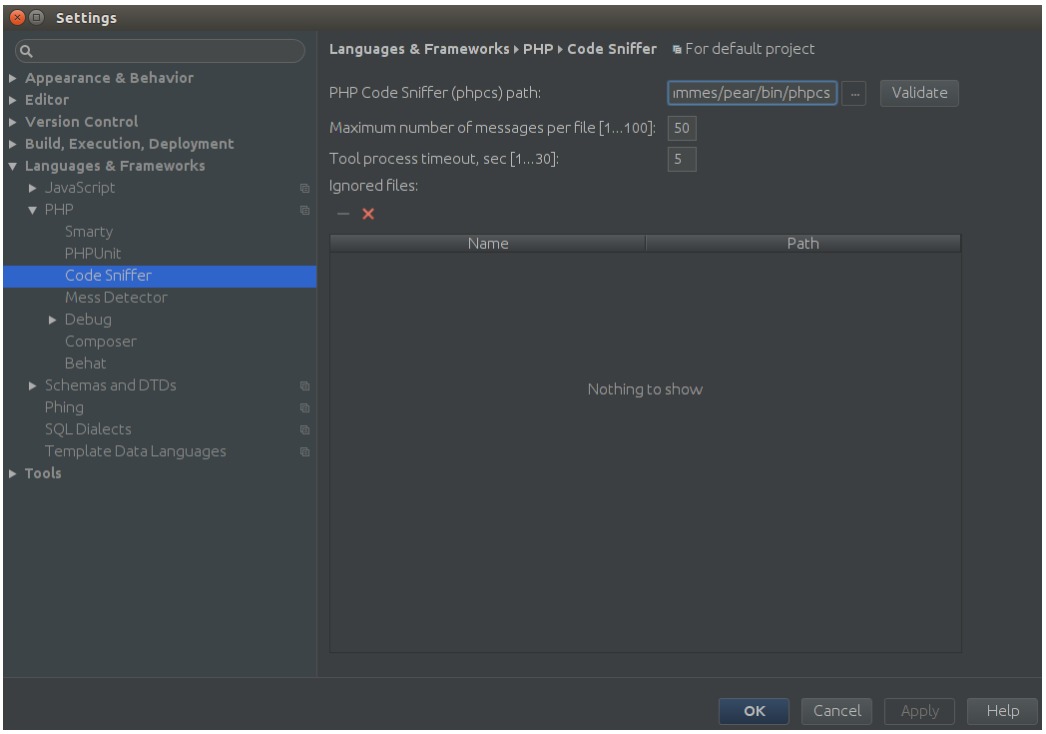
PHP 7

PHP cs (code sniffer)

Intégration de PHP Code Sniffer dans PhpStorm

Dans PhpStorm, allez dans *File* > *Default Settings*.
Dans la fenêtre qui s'ouvre, allez dans *Language & Frameworks* > *PHP* > *Code Sniffer*. Vous n'avez plus qu'à indiquer l'emplacement de **phpcs**. Vous pouvez vérifier si c'est OK en cliquant sur le bouton *Validate*.

http://www.christophe-meneses.fr/article/integrer-php-code-sniffer-dans-phpstorm



PHP 7

Une fois identifies tous les problèmes liés à mon application,
il est temps de corriger les erreurs une à une et manuellement.
Ca prend du temps, mais on n'a pas trop le choix dans ce cas présent non ?
Le plus simple c'est de se préoccuper des errors avant les warnings,
car les errors sont prioritaires.
Mais comment les résoudre efficacement ?
La première chose à faire c'est de mettre à jour toutes vos librairies et dépendances pour bénéficier de leur compatibilité avec PHP 7.
Ensuite quand vous avez une erreur du style « Removed fonction 'Nom de la fonction' called », il faut rechercher avec le nom de la fonction en question sur la documentation de PHP (php.net) et de trouver son remplaçant

PHP 7

Vous allez devoir tester chacun de vos changements manuellement et l'un après l'autre.

Enfin, vous devrez tester l'ensemble de votre application sur votre nouvel environnement PHP 7 comme test final pour valider définitivement vos changements.

Conclusion

Sur le principe passé à PHP 7 n'est pas compliqué mais cela peut être long, très pénible et fastidieux.

PHP 7

Du changement dans les fonctions

debug_zval_dump() maintenant affiche "int" au lieu de "long" et "float" au lieu de "double"

La fonction **dirname()** prend maintenant un deuxième paramètre, depth, pour indiquer le nombre de niveaux plus haut (par rapport au dossier courant) pour atteindre le nom du dossier dans l'arborescence.

getrusage() est désormais supporté sur Windows.

Les fonctions **mktime()** et **gmmktime()** n'acceptent plus le paramètre is_dst.

la fonction **preg_replace()** ne supporte plus "\e" (PREG_REPLACE_EVAL). preg_replace_callback() doit être utilisé à la place.

La fonction **setlocale()** n'accepte plus que le paramètre category soit passé comme chaîne de caractères. Les constantes LC_* doivent être utilisées à la place.

Les fonctions **exec()**, **system()** et **passthru()** ont désormais l'octet NULL de protection.

La fonction **shmop_open()** retourne désormais une ressource à la place d'un entier qui doit être passé aux fonctions **shmop_size(), shmop_write(), shmop_read()**, **shmop_close()** et **shmop_delete()**.

Les fonctions **substr()** et **iconv_substr()** retourne maintenant une chaîne vide, si la longueur de la chaîne est égale à $start.

La fonction **xml_set_object()** exige maintenant de détruire $parser à la fin pour éviter les fuites de mémoire.

PHP 7

Du changement dans les fonctions

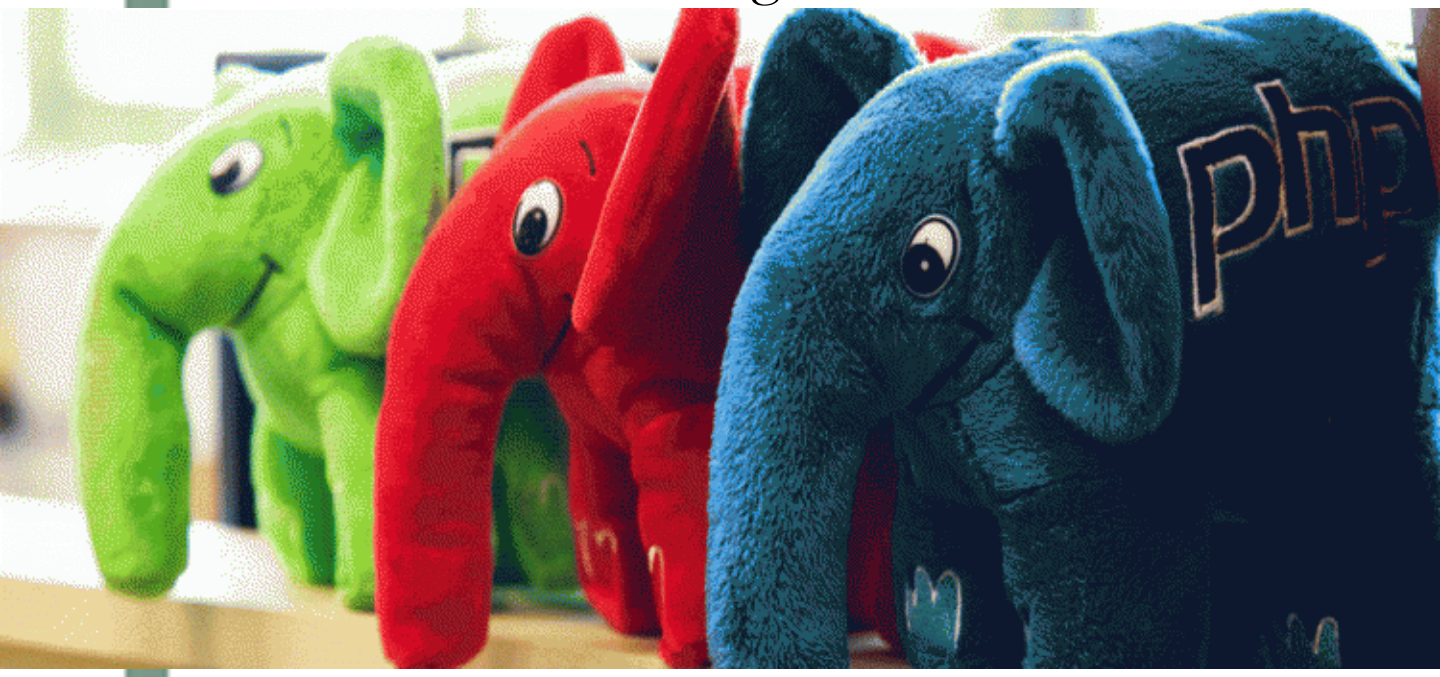
http://forum.wampserver.com/read.php?2,133667

https://github.com/wimg/PHPCompatibility
https://github.com/phan/phan
https://github.com/sstalle/php7cc

PHP 7

Interprétation des variables

Expression	PHP 5 interpretation	PHP 7 interpretation
$$foo['bar']['baz']	${$foo['bar']['baz']}	($$foo)['bar']['baz']
$foo->$bar['baz']	$foo->{$bar['baz']}	($foo->$bar)['baz']
$foo->$bar['baz']()	$foo->{$bar['baz']}()	($foo->$bar)['baz']()
Foo::$bar['baz']()	Foo::{$bar['baz']}()	(Foo::$bar)['baz']()



Annexes

Sommaire

Merci

Si vous avez la moindre question
s.brunet@leserveur.com