1. **Démarrer en PHP**

PHP a été inventé à l'origine pour le développement d'applications **web dynamiques** qui constituent encore le cas d'utilisation le plus courant et son point fort.

PHP est tout d'abord un **langage** de script **interprêté** (en réalité [précompilé en Opcode](http://www.cyruss.com/blog/index.php?2005/09/13/39-php-comment-ca-marche-dedans)), gratuit, **OpenSource** et **distribué sous licence** autorisant la modification et la redistribution.

PHP est **portable** : supporté sur plusieurs systèmes d'exploitation : Windows™, MAC OS X™, Linux ou Sun Solaris.

* très facile de déplacer une application écrite en PHP d'un serveur Windows d'origine vers un serveur Linux sans avoir à la modifier (ou très peu).

Exécution : PHP a besoin d'un **serveur Web** pour fonctionner. Toutes les pages demandées par un client seront construites par le serveur Web, en fonction des paramètres transmis, avant d'être retournées au client.

**Exemple de PHP et HTML imbriqués**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">

<head>

<title>Premier programme PHP !</title>

</head>

<body>

**<?php**

[**echo**](http://www.php.net/echo) **'<strong>** Hello World ! **</strong> ' ;**

**?>**

</body>

</html>

1. **Les commentaires :** documentation technique de 1er niveau – Obligatoire dans nos programmes !

Il existe deux sortes de commentaire. Le commentaire sur une seule ligne …

<?php

***//*** *Ceci est un commentaire sur une ligne*

[echo](http://www.php.net/echo) 'Bonjour le monde !';

?>

… et le commentaire multilignes.

<?php

**/\*** Ce programme a été écrit par Emacs

Il affiche la chaîne 'Hello Word !' à l'écran **\*/**

[echo](http://www.php.net/echo) 'Bonjour le monde !';

?>

1. **Les variables**

Toutes les **variables PHP** sont préfixées **$**

Une variable est une structure de données de ***type primitif*** (entier, réel, caractère, chaîne de caractères, booléen ou null) ou bien de ***type structuré*** (tableau ou objet) qui permet de stocker une ou plusieurs valeurs. Chaque valeur d'une variable est susceptible d'être remplacée par une autre au cours de l'exécution du programme. D'où le terme « variable ».

En programmation, une variable est définie par 4 informations essentielles :

* son nom
* son type
* son domaine de valeurs possibles
* sa sémantique ou sa signification (sens) : logique de la valeur de la variable par rapport à son contexte initial

**Attention ! Certaines syntaxes sont interdites pour nommer les variables**

|  |  |  |
| --- | --- | --- |
| $variable | variable | Une variable doit commencer par $ |
| $Variable1 | $Variable 1 | Les espaces sont interdits |
| $variable\_suite | $variable-suite | Le tiret est interdit |
| $\_variable | $-variable | Le tiret est interdit |
| $variable2 | $2variable | Il ne peut y avoir de chiffre après le $ |

**Exemples**

<?php

$prenom = 'Hugo'; *// Type string (chaine de caractères)*

$nom = "Hamon"; *// Type string (chaine de caractères)*

$age = 19; *// Type entier*

$estEtudiant = true; *// Type booléen*

$cours = [array](http://www.php.net/array)('physique','chimie','informatique','philosophie'); *// Type tableau*

$unEtudiant = new Etudiant(); *// Objet de type Etudiant*

?>

**Les variables sont pas typés dynamiquement**

PHP ne porte pas d'importance au typage des variables. Par exemple, pour une même variable, le programmeur est libre de lui affecter une valeur de type entier à un instant T1 puis de lui affecter une chaine de caractères à un instant T2. On dit que PHP est un langage de typage « faible et dynamique ». Cela rend son utilisation plus souple par les développeurs mais pas forcément plus assidue... C'est d'ailleurs quelque chose que l'on eut reproche à PHP.

La **concaténation** des chaines de caractères s’effectue avec le caractère **.** (point)**:**

*// On concatène $prenom un espace puis $nom dans $identite*

$identite = $prenom **.** ' ' **.** $nom;

**Test d’existence d’une variable**

Il est souvent utile de savoir si une variable existe avant de pouvoir l'exploiter. Le langage PHP met à disposition des développeurs la fonction [isset()](http://fr.php.net/isset) qui permet de savoir si oui ou non la variable existe. Cette fonction retourne un booléen TRUE ou FALSE.

<?php

$prenom = 'Hugo'; *// Déclaration de la variable $prenom*

[echo](http://www.php.net/echo) [isset](http://www.php.net/isset)($prenom); *// Retourne TRUE -> affiche 1 car $prenom existe et est valorisé*

[echo](http://www.php.net/echo) '<br/>';

[echo](http://www.php.net/echo) [isset](http://www.php.net/isset)($nom); *// Retourne FALSE -> n'affiche rien car $nom n’existe pas*

?>

1. **Les tableaux array()**

La déclaration d'un tableau se fait en *$nom*. Pour déclarer un nouveau tableau, il suffit d'utiliser la structure de langage [array()](http://fr.php.net/array). Cette fonction prend en paramètres facultatifs (séparés par une virgule), les valeurs que l'on souhaite insérer dans le tableau pour l'initialiser. Si rien n'est précisé en paramètre, le tableau créé est vide.

**Déclaration d’un tableau**

<?php

$fruits = [array](http://www.php.net/array)(); *// Déclaration d'un tableau vide*

$legumes = [array](http://www.php.net/array)('carotte','poivron','aubergine','chou'); *// Déclaration d'un tableau indexé numériquement*

// Déclaration d'un tableau associatif

$identite = [array](http://www.php.net/array)(

'nom' => 'Hamon', 'prenom' => 'Hugo', 'age' => 19, 'estEtudiant' => true

);

?>

**Ajout d’éléments dans un tableau**

<?php

$legumes[] = 'salade'; *// Ajout d'un légume à la suite au tableau indexé numériquement*

$legumes[20] = 'piment'; *// Ajout d'un légume au rang 20 dans le tableau indexé*

$identite['taille'] = 180; *// Ajout de la taille de la personne dans le tableau associatif*

?>

A noter : PHP agrandit dynamiquement le tableau $legumes et ajoute la valeur 'piment' à l'index 20.

**Affichage des valeurs d’un tableau associatif**

<?php

*// Affichage des valeurs du tableau associatif - Attention aux apostrophes !*

[echo](http://www.php.net/echo) 'Nom : ', $identite['nom'] ,'<br/>';

[echo](http://www.php.net/echo) 'Prénom : ', $identite['prenom'] ,'<br/>';

[echo](http://www.php.net/echo) 'Age : ', $identite['age'] ,' ans<br/>';

[echo](http://www.php.net/echo) 'Taille : ', $identite['taille'] ,' cm';

?>

**Parcours d’un tableau**

<?php

**foreach($leTableau as $valeur)** *// Affichage des valeurs d'un tableau*

{

[echo](http://www.php.net/echo) $valeur ,'<br/>';

}

**foreach($leTableau as $cle => $valeur)** *// Affichage des couples clé / valeur : la clé d’un tableau associatif = le nom du champ*

*{*

[echo](http://www.php.net/echo) $cle ,' : ', $valeur ,'<br/>';

}

Résultat obtenu :

Array

(

[nom] => Hamon

[prenom] => Hugo

[age] => 19

[estEtudiant] => 1

[taille] => 180

)

1

Affichage du contenu d’un tableau associatif : exemple

<?php

[echo](http://www.php.net/echo) '<pre>';

[print\_r](http://www.php.net/print_r)($identite);

[echo](http://www.php.net/echo) '</pre>';

?>

La fonction ‘<pre>’ permet l’indentation des éléments du tableau

**Opérations sur les tableaux**

L'intérêt d'utiliser des tableaux pour structurer ses applications, c'est que cela permet ensuite d'opérer sur ces derniers. En effet, PHP propose une série de fonctions natives capables de manipuler ces structures. En voici quelques unes fréquemment employées et particulièrement utiles :

* [count()](http://fr.php.net/count) et [sizeof()](http://fr.php.net/sizeof) retournent toutes les deux la taille du tableau passé en paramètre.
* [sort()](http://fr.php.net/sort) trie les éléments d'un tableau du plus petit au plus grand.
* [rsort()](http://fr.php.net/rsort) trie les éléments d'un tableau du plus grand au plus petit.
* [in\_array()](http://fr.php.net/in-array) permet de vérifier qu'une valeur est présente dans un tableau.
* [array\_rand()](http://fr.php.net/array-rand) extrait une ou plusieurs valeurs du tableau au hasard.
* [current()](http://fr.php.net/current) retourne la valeur de l'élément courant du tableau (où se trouve le pointeur)

*Pour connaître toutes les fonctions relatives aux tableaux, voir la documentation officielle de PHP :* [*les tableaux / arrays*](http://fr.php.net/array)

**Pour les opérateurs voir la documentation en ligne ici :**

[**http://www.apprendre-php.com/tutoriels/tutoriel-8-les-oprateurs.html**](http://www.apprendre-php.com/tutoriels/tutoriel-8-les-oprateurs.html)

1. **Les structures**

**De test : si alors sinon if … elseif …. else**

**Exemple**

<?php

$vitesse = 82; // Déclaration d'une variable

if($vitesse > 50 && $vitesse < 70) // On teste la valeur de la variable

[echo](http://www.php.net/echo) 'Je perds 1 point pour excès de vitesse en ville!';

elseif($vitesse > 70 && $vitesse < 80)

[echo](http://www.php.net/echo) 'Je perds 2 points pour excès de vitesse !';

elseif($vitesse > 80 && $vitesse < 90)

[echo](http://www.php.net/echo) 'Je perds 3 points pour excès de vitesse !';

elseif($vitesse > 90)

[echo](http://www.php.net/echo) 'Je perds 4 points pour excès de vitesse !';

else

[echo](http://www.php.net/echo) 'On me retire immédiatement mon permis !’ ;

?>

**De cas – selon : switch(**expression**)** { **case** valeurn : instructions ; **break** ; **default** : instructions ; **break** ; }

switch(expression)

{ case valeur1 : instructions;

break;

case valeur2 : instructions;

break;

default : instructions;

break;

}

**Boucle : for**

<?php

*// Boucle générant la table de multiplication du 9*

for($i=0; $i<=10; $i++) //*La conditon* ***$i<=$10*** *est toujours vraie tant que $i n'atteint pas la valeur 11.*

{

[echo](http://www.php.net/echo) '9 x ', $i ,' = ', (9\*$i) ,'<br/>';

}

?>

**Boucle : while**

<?php

*// Déclaration et initialisation du compteur*

$i = 0;

*// Boucle générant la table de multiplication du 9*

do

{ [echo](http://www.php.net/echo) '9 x ', $i ,' = ', (9\*$i) ,'<br/>'; *// Affichage de la nouvelle ligne*

$i++; *// Incrémentation du compteur*

}

while($i<=10);

?>

**Boucle : foreach**

<?php

**foreach($leTableau as $valeur)** *// Affichage des valeurs d'un tableau*

{ [echo](http://www.php.net/echo) $valeur ,'<br/>'; }

**foreach($leTableau as $cle => $valeur)** *// Affichage des couples clé / valeur*

{ [echo](http://www.php.net/echo) $cle ,' : ', $valeur ,'<br/>'; }

?>

**Exemple**

<?php

// Déclaration des deux tableaux

$legumes = [array](http://www.php.net/array)('salade','oignon','piment','carotte','aubergine');

$couleurs = [array](http://www.php.net/array)('rouge'=>'#ff0000', 'vert' => '#00ff00', 'bleu' => '#0000ff');

// Affichage des légumes

foreach($legumes as $legume)

{ [echo](http://www.php.net/echo) $legume ,'<br/>'; }

// Affichage des couleurs et leur code hexadécimal

foreach($couleurs as $couleur => $codeHexadecimal)

{ [echo](http://www.php.net/echo) $couleur ,' : ', $codeHexadecimal ,'<br/>'; }

?>

1. **Les fonctions**

**Exemple 2**

<?php

function estConnecte($connecte)

{

// Test du paramètre d'entrée

if(1 === $connecte)

{

return true;

}

return false;

}

?>

**Exemple 1**

<?php

**function calculerSomme($a, $b)**

{

$somme = $a + $b; // Déclaration de la variable locale $somme

**return** $somme; // Renvoi de la somme au programme principal

}

?>

**Appel de la fonction calculerSomme : exemple 1**

<?php

$resultatsSomme = [array](http://www.php.net/array)(); *// Déclaration du tableau*

*// Appel de la fonction*

$resultatsSomme[] = calculerSomme(10,1);

$resultatsSomme[] = calculerSomme(7, 23);

$resultatsSomme[] = calculerSomme(130, 231);

$resultatsSomme[] = calculerSomme(98, 62);

*// Affichage du contenu du tableau*

[echo](http://www.php.net/echo) '<pre>';

[print\_r](http://www.php.net/print_r)($resultatsSomme);

[echo](http://www.php.net/echo) '</pre>';

?>

1. **Les formulaires $\_Get et $\_Post**

**Voir l’excellent cours en ligne :**

[**http://www.apprendre-php.com/tutoriels/tutoriel-12-traitement-des-formulaires-avec-get-et-post.html**](http://www.apprendre-php.com/tutoriels/tutoriel-12-traitement-des-formulaires-avec-get-et-post.html)

**A noter :** Les informations saisies sur un formulaire en ligne proviennent de personnes anonymes, nous ne pouvons garantir la véracité et la dangerosité de ces dernières. C'est pourquoi tout doit être vérifié.

* **Le principe numéro 1 lorsque l'on interagit avec un utilisateur est de ne jamais lui faire confiance**.

1. **Require et Include**

Les applications ont besoin de réutiliser des parties de code : c'est le cas par exemple des librairies de fonctions utilisateurs ou bien des fichiers de configuration. Plutôt que de réécrire le même code utilisé à plusieurs endroits du programme on utilise des fonctions capables d'importer et exécuter le code à réutiliser : [include()](http://fr.php.net/include) et [require()](http://fr.php.net/require).

Ces deux fonctions remplissent la même fonction. Leur différence réside dans la gestion des erreurs.

* La fonction include() renverra une erreur de type WARNING si elle n'arrive pas à ouvrir le fichier en question. De ce fait l'exécution du code qui suit dans la page sera exécuté.
* En revanche, la fonction require() affichera une erreur de type FATAL qui interrompt l'exécution du script.

Laquelle préférer? Tout dépend du contexte. Si le fichier doit obligatoirement être présent pour le reste du programme, alors require() est à préférer sinon un include() fera l'affaire.

Lorsqu'un fichier est importé, le code se trouvant dans ce fichier est exécuté. Les variables, constantes, objets, tableaux... du fichier importé peuvent donc être réutilisés dans la suite du programme.

**Exemple**

<?php

require('../dossier/fichier.php'); *// Importations avec require()*

require 'fichier2.php';

include('../dossier/fichier.php'); *// Importations avec include()*

include 'fichier2.php';

?>

### Require\_once() et include\_once(), c'est quoi ?

Les fonctions [require\_once()](http://fr.php.net/require-once) et [require\_once()](http://fr.php.net/include-once) permettent d'importer **une fois seulement** un fichier même s'il y'a plusieurs tentatives d'importation du fichier dans la page.

Néanmoins l'utilisation de ces deux fonctions est dépréciée pour des raisons d'optimisation et de sécurité.

1. **Programmation PHP objet**

**Construction d’une classe :** Exemple de langage objet PHP version 5

<?php

*// Classe PHP 5*

class ClassA

{

private $nom; *// propriété privée de la classe*

**function \_\_construct($var)** *// constructeur de la classe*

{ $this->nom = $var;}

function changeNom($var)

{ $this->nom = $var; }

}

-----

function fctActionObjet($obj) *// Fonction prenant un objet en paramètre et changeant une valeur.*

{

$obj->changeNom('Pierre');

}

$b = new ClassA('Cyril');

fctActionObjet($b);

[print\_r](http://www.php.net/print_r)($b);

?>

**Soit la classe ANIMAL**

**Description en PHP de la classe ANIMAL**

***// ANIMAL.class.php***

**<?php**

**class ANIMAL** *// Par principe en PHP une classe est publique*

{

*// Déclaration des attributs et méthodes.*

private **$couleur** = "gris";  *// Pas de déclaration de type, type implicite*

private **$poids** = 10;

public **function** **manger()**  *// Pas de void, tout est «****function****» même sans retour !*

{  *//méthode décrivant comment l’animal mange* }

       public **function** **se\_deplacer()**

       {  *//méthode décrivant comment l’animal se déplace*         }

       public **function** **ajouter\_un\_kilo()**

       {

           $this**->**poids = $this->poids + 1;  *// Ajoute la valeur 1 au poids de l’animal en cours de traitement*

       }

}

**?>**

**Utilisation de la classe ANIMAL**

Dans un premier temps, il faut créer un fichier **ANIMAL.class.php** contenant le code PHP précédent.

Pour utiliser la classe Animal, il faut l’inclure dans la page où on souhaite l’appeler.

Code PHP permettant d’appeler la classe et d’instancier un nouvel objet.

<?php

**include(’Animal.class.php’);**

**$chien = new Animal();**

?>

Maintenant que la classe est chargée, il est possible d’instancier la classe, on appel la classe et on instancie un ANIMAL = nouvel objet associé à $chien.

La variable $chien est une instance de la classe Animal avec les attributs couleur, poids et méthodes manger, se\_deplacer, ajouter\_un\_kilo spécifiques à lui-même.

**Mettre à jour et lire les attributs de l’instance : encapsulation et accesseurs**

Le principe de l’encapsulation veut que tous les attributs doivent être privés. Il faut donc créer des méthodes publiques appelées **accesseurs**, permettant de lire ou d’écrire dans ces attributs.

Leurs noms sont généralement précédés du préfixe **get** pour **lire** la valeur de l’attribut et **set** pour **écrire** la valeur de l’attribut. Les accesseurs sont publics donc ils permettent de lire ou d’écrire dans les attributs depuis n’importe quelle autre classe ou page PHP.

**Nouvelle description en PHP de la classe ANIMAL**

*// Nom du fichier php concerné :* ***ANIMAL.class.php***

**<?php**

**class ANIMAL**

{  *// Déclaration des attributs et méthodes.*

private **$couleur** = "gris";

private **$poids** = 10;

*// Les accesseurs  et getters des attributs privés*

  public function **getCouleur**()

        {  **return $this->couleur;**         }  *//retourne la couleur*

        public function **setCouleur**($couleur)

        {  **$this->couleur = $couleur;**  }  *//écrit dans l’attribut couleur*

        public function **getPoids**()

       {       **return $this->poids;** } *//retourne le poids*

        public function **setPoids**($poids)

        {  **$this->poids = $poids;**         }  *//écrit dans l’attribut poids*

*// Les méthodes autres qu’accesseurs aux attributs*

public function **manger**()

{  *//méthode décrivant comment l’animal mange*  }

        public **function** **se\_deplacer**()

        {  *//méthode décrivant comment l’animal se déplace*  }

        public **function** **ajouter\_un\_kilo**()

        {             $this**->**poids = $this->poids + 1;          }

}

**?>**

**Exemple d’utilisation des accesseurs**

Par exemple, pour créer un chat blanc de 5 kg et un chien noir de 18 kg :

<?php

**include(’ANIMAL.class.php’);**  *//chargement de la classe*

**$chien = new ANIMAL();**  *//instanciation de la classe Animal*

**$chat = new ANIMAL();**  *//instanciation de la classe Animal*

$chien->**setPoids(18)**;  *//mise à jour du poids du chien*

$chat->**setPoids(5)**;  *//mise à jour du poids du chat*

$chien->**setCouleur("noir");**  *//mise à jour de la couleur du chien*

$chat->**setCouleur("blanc")**;  */ /mise à jour de la couleur du chat*

echo "Le poids du chien est : ".$chien->**getPoids()**." kg<br />";  *//lire le poids* *du chien*

echo "La couleur du chien est : ".$chien->**getCouleur()**."<br />";  *//lire la couleur du chien*

echo "Le poids du chat est : ".$chat->**getPoids()**." kg<br />";  *//lire le poids du chat*

echo "La couleur du chat est : ".$chat->**getCouleur()**."<br />";  *//lire la couleur du chat*

?>

**Le constructeur**

Le constructeur sert à construire un objet du type de la classe. Lorsque vous écrivez new ANIMAL(), cela appelle le constructeur par défaut de la classe ANIMAL.

Il est possible de créer plusieurs constructeurs et ainsi de passer en paramètre la valeur des attributs nécessaires à l’instanciation de l’objet.

Le constructeur est toujours nommé **\_\_construct** et ne possède pas de return.

<?php

**class ANIMAL**  *// Par principe en PHP une classe est publique*

{  *// Déclaration des attributs*

        private $couleur = "gris";  *// Pas de déclaration de type, type implicite*

       private $poids = 10;

*// Déclaration des constructeurs avec surcharge*

        public **function** **\_\_construct ()** *//Constructeur générique, sans paramètre*

public **function** **\_\_construct ($couleur)** *//Constructeur demandant 1 paramètre*

     {

           $this->couleur = $couleur; *// Initialisation de la couleur.*

       }

public **function** **\_\_construct ($couleur, $poids)** *//Constructeur demandant 2 paramètres*

     {

           $this->couleur = $couleur; *// Initialisation de la couleur.*

           $this->poids = $poids; *// Initialisation du poids.*

       }

….

?>

**Appel des constructeurs**

Appel du constructeur dans la page utilisation.php :

<?php

include(’ANIMAL.class.php’);  *//chargement de la classe*

$chien1 = new ANIMAL();  *//instanciation de la classe Animal avec constructeur1*

$chien1->setCouleur("noir");

$chien1->setPoids(10);

$chien2 = new ANIMAL(("beige"); *//instanciation de la classe Animal avec constructeur2*

$chien1->setPoids(20);

$chien3 = new ANIMAL(("beige", 30); *//instanciation de la classe Animal avec constructeur3*

?>

**Les méthodes « magiques » : \_\_**

Depuis la version 5, PHP orientée objet implémente des méthodes au comportement prédéfini. Ces méthodes sont appelées « *méthodes magiques* », les méthodes **\_\_set()** et **\_\_get()** en font partie.

Une méthode magique est une méthode appelée automatiquement lorsqu’un évènement se produit.

Par exemple, **\_\_construct**() est une méthode magique : elle s’exécute automatiquement lorsqu’est instanciée la classe contenant \_\_construct().

Les méthodes magiques **\_\_get() et \_\_set()** permettent de lire ou de modifier des attributs dont l’accès est interdit.

<?php

class ANIMAL

   {

       private $couleur = "gris";

       public $poids = 10;

       private $tab\_attributs = array();

       public **function** \_\_**construct**($couleur, $poids) *// Constructeur  demandant 2 paramètres.*

       {

           $this->couleur = $couleur; *// Initialisation de la couleur.*

           $this->poids = $poids;  *// Initialisation du poids.*

       }

       //méthodes magiques

       public **function \_\_get($nom)**

       {

        if (isset ($this->tab\_attributs[$nom]))

return $this->tab\_attributs[$nom];

       }

       public **function \_\_set($nom, $valeur)**

       {

            $this->tab\_attributs[$nom] = $valeur;

       }

       public **function \_\_isset($nom)**

       {

            return isset ($this->tab\_attributs[$nom]);

       }

       public **function** manger ()         {             ...         }

…

}

?>

Ces méthodes se déclenchent automatiquement même si l’attribut est privé ou s’il n’existe pas.

Elles stockent alors le résultat dans le tableau $tab\_attributs avec pour index le nom de l’attribut. La méthode \_\_isset($attribut) permet de savoir si l’attribut existe.

<?php

include(’ANIMAL.class.php’);   *//chargement de la classe*

$chien = new Animal("gris",8);  *//instanciation de la classe Animal*

if (isset($chien->couleur)) {

    echo "L’attribut couleur existe.<br />";

}

else {      echo "L’attribut couleur n’existe pas.<br />";

}

$chien->couleur = "noir";

echo $chien->couleur."<br />";

if (isset($chien->poids)) {

    echo "L’attribut poids existe.<br />";

}

else {   echo "L’attribut poids n’existe pas.<br />";  }

$chien->poids = 25;

echo $chien->poids."<br />";

?>