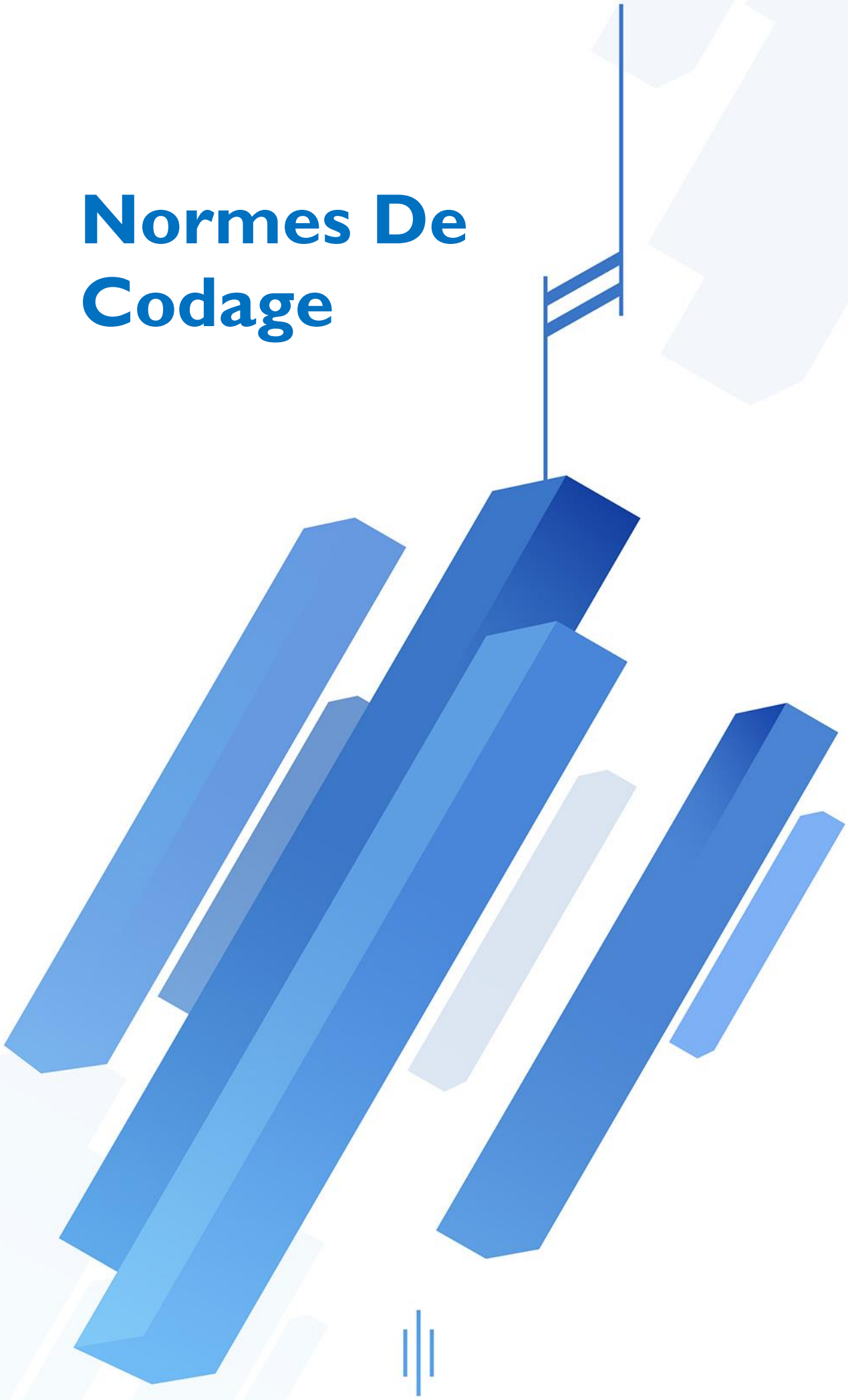
Normes De Codage



1- INTRODUCTION

L'écosystème PHP est vaste et complet : beaucoup d'outils, frameworks et CSM sont disponibles, la communauté est importante et active, les entreprises recrutent, les clients sont nombreux…. Dans un monde faisant intervenir autant d'acteurs, il est important de suivre certaines "normes" afin d'assurer la pérennité des projets (et du langage). Ces conventions ont pour but de fournir des bonnes pratiques afin d'homogénéiser les développements et de simplifier la maintenabilité du code. En effet, un projet PHP qui respecte les normes est plus facile à prendre en main par un nouveau développeur et ainsi plus simple à maintenir qu'un projet ne les respectant pas.

2- Fichiers.

Ce paragraphe a pour but de décrire l'organisation et la présentation des fichiers mis en jeu dans un site Web dynamique écrit en PHP

2.1 Extension des fichiers

Les fichiers PHP doivent obligatoirement se terminer par l'extension **.php** pour une question de sécurité. En procédant ainsi, il n'est pas possible de visualiser le source des fichiers PHP (qui contiennent peut-être des mots de passe), le serveur web les fait interpréter par PHP.

Les fichiers qui ne constituent pas des pages autonomes (des fichiers destinés à être inclus dans d'autres pages web) se terminent Aussi par l'extension **.php**.
Les fichiers contenant des pages statiques (sans code PHP) doivent porter l'extension **.html**.

2.2 Inclusion de scripts

L'inclusion de scripts peut être réalisée par plusieurs instructions prédéfinies en PHP : include, require, include_once, require_once. Toutes ont pour objectif de provoquer leur propre remplacement par le fichier spécifié, un peu comme les commandes de préprocesseur C #include.

Les instructions include et require sont identiques, hormis dans leur gestion des erreurs. include produit une alerte (warning) tandis que require génère une erreur fatale . En d'autres termes, lorsque le fichier à inclure n'existe pas, le script est interrompu. include ne se comporte pas de cette façon, et le script continuera son exécution.

La différence entre require et require_once (idem entre include et include_once) est qu'avec **require_once()**, on est assuré que le code du fichier inclus ne sera ajouté qu'une seule fois, évitant de ce fait les redéfinitions de variables ou de fonctions, génératrices d'alertes.

3- Presentation du code

3.1- Tag PHP

Toujours utiliser <?php ?> pour délimiter du code PHP, et non la version abrégée <? ?>. C'est la méthode la plus portable pour inclure du code PHP sur les différents systèmes d'exploitation et les différentes configurations

3.2- Séparation PHP/HTML

Les balises HTML doivent se situer au maximum dans les sections HTML et non incluses à l'intérieur du texte des messages de l'instruction d'affichage echo.

Exemples

Ne pas écrire

```
<?php
    echo "<select id=\"lstAnnee\" name=\"lstAnnee\">";
    $anCours = date("Y");
    for ( $an = $anCours – 5 ; $an <= $anCours + 5 ; $an++ ) {
        echo "<option value=\"" . $an ."\">" . $an . "</option>";
    }
    echo "</select>";
?>
</select>
```

Mais écrire

```
<select id="lstAnnee" name="lstAnnee">
<?php
    $anCours = date("Y");
    for ( $an = $anCours – 5 ; $an <= $anCours + 5 ; $an++ ) {
?>
        <option value="<?php echo $an ; ?>">echo $an; ?></option>
<?php
    }
?>
</select>
```

Dans les sections HTML, l'éditeur de l'outil de développement applique la coloration syntaxique sur les balises et attributs HTML. Ceci accroît donc la lisibilité et la localisation des erreurs de syntaxe au niveau du langage HTML. Il est aussi plus aisé d'intervenir uniquement sur la présentation, sans effet de bord sur la partie dynamique.

Maintenabilité : lisibilité

Portabilité : indépendance

3.3 Indentation et longueur des lignes

Le pas d’indentation doit être **fixe** et correspondre à **4 caractères.** Ce pas d'indentation doit être paramétré dans l'éditeur de l'environnement de développement. L’indentation est stockée dans le fichier sous forme de 4 caractères espace (sans tabulation réelle).

Il est recommandé que la longueur des lignes ne dépasse pas 75 à 85 caractères.
Lorsqu’une ligne d’instruction est trop longue, elle doit être coupée après une virgule ou avant un opérateur. On alignera la nouvelle ligne avec le début de l’expression de même niveau de la ligne précédente.

3.4 Presentation des blocs logiques

1. Chaque bloc logique doit être délimité par des accolades, même s’il ne comporte qu’ une seule instruction,

2. Dans une instruction avec bloc, l'accolade ouvrante doit se trouver sur la fin de la ligne de l'instruction ; l'accolade fermante doit débuter une ligne, et se situer au même niveau d'indentation que l'instruction dont elle ferme le bloc,

3. Les instructions contenues dans un bloc ont un niveau supérieur d'indentation.

Exemple :

```
Structures de contrôle conditionnelles
if (...) {
    ...
} elseif (...) {
    ...
} else {
    ...
}
switch (...) {
    case ... :
        ...
    case ... :
            ...
    default :
            ...
}
Définition de fonction
function uneFonction() {
    ...
}
Structures de contrôle itératives
for (...; ...; ...) {
    ...
}
while (...) {
    ...
```

```
}
do {
    …
     }while (...);
```

3.5 Commentaires des instructions du code

Description :

Il existe deux types de commentaires :
1. les commentaires mono ligne qui inactivent tout ce qui apparaît à la suite, sur la même ligne : //
2. les commentaires multi-lignes qui inactivent tout ce qui se trouve entre les deux délimiteurs, que ce soit sur une seule ligne ou sur plusieurs /* */

Il est important de ne réserver les commentaires multi-lignes qu'aux blocs utiles à PHPDocumentor et à l'inactivation de portions de code. Les commentaires mono-ligne permettant de commenter le reste, à savoir, toute information de documentation interne relative aux lignes de code. Ceci afin d'éviter des erreurs de compilation dues aux imbrications des commentaires multi-lignes

Exemples

```
// page inaccessible si visiteur non connecté
if (!estVisiteurConnecte()) {
    header("Location: cSeConnecter.php");
}

// acquisition des données reçues par la méthode post
$mois = lireDonneePost("lstMois", "");
$etape = lireDonneePost("etape","");

2.      Inactivation d'une portion de code pour débogage
/*
```

```
// page inaccessible si visiteur non connecté
if (!estVisiteurConnecte()) {
    header("Location: cSeConnecter.php");
}
*/
```

4- Nommage des identificateurs

4.1 Nommage des fonctions

L'identificateur d'une fonction est un verbe, ou groupe verbal.
Les noms de fonctions ne peuvent contenir que des caractères alphanumériques. Les tirets bas ("_") ne sont pas permis. Les nombres sont autorisés mais déconseillés.
Les noms de fonctions doivent toujours commencer avec une lettre en minuscule. Quand un nom de fonction est composé de plus d'un seul mot, la première lettre de chaque mot doit être mise en majuscule. C'est ce que l'on appelle communément la "notationCamel".

Exemples :
filtrerChaineBD(), verifierInfosConnexion(),estEntier()

4.2 Nommage des variables et paramètres

L'identificateur d'une variable ou paramètre indique le rôle joué dans le code ; c'est en général un nom, ou groupe nominal. Il faut éviter de faire jouer à une variable plusieurs rôles.
Les noms de variables et paramètres ne peuvent contenir que des caractères alphanumériques. Les tirets bas sont autorisés uniquement pour les membres privés d'une classe. Les nombres sont autorisés mais déconseillés.

Comme les identificateurs de fonctions, les noms de variables et paramètres adoptent la notation Camel.

Exemples :
$nomEtud, $login.