SP1_CP000
Cahier des spécifications

Ensemble de conventions et de règles pour faciliter la compréhension et donc la maintenance du code.

Groupe

Compilateur OPTIML-1

Module

T-ESP_800

Ecole

Epitech

Préambule

Le but de ce chapitre est de proposer un ensemble de conventions et de règles pour faciliter la compréhension et donc la maintenance du code.

Ces règles ne sont pas à suivre explicitement à la lettre : elles sont uniquement présentées pour inciter les développeurs à définir et à utiliser des règles dans la réalisation du code surtout dans le cadre d'un travail en équipe. Les règles proposées sont celles couramment utilisées. Il n'existe cependant pas de règle absolue et chacun pourra utiliser tout ou partie des règles proposées.

La définition de conventions et de règles est importante pour plusieurs raisons :

La majorité du temps passé à coder est consacrée à la maintenance évolutive et corrective d'une application

Ce n'est pas toujours, voire rarement, l'auteur du code qui effectue ces maintenances.

Ces règles facilitent la lisibilité et donc la compréhension du code

Le contenu de ce document est largement inspiré par les conventions de codage historiquement proposées par Sun

Le formatage du code

IF

```php
<?php

//Déclaration d'une variable de test.
$variable_de_test = 999;

// Vérification de la valeur de la variable.
if($variable_de_test == 1){

    // Code à exécuter

}

?>
```

Une conditionnel IF doit être écrit de la manière suivante. C'est un exemple de conditionnelle simple.

Le commentaire doit être associé à la ligne. Ne pas oublier les crochets ouvrants et fermant. Les Ternaires sont prohibés.

ELSE

```php
<?php

//Déclaration d'une variable de test.
$variable_de_test = 999;

// Vérification de la valeur de la variable.
if($variable_de_test == 1){

    // Code à exécuter

}else{

    // Code à exécuter

}

?>
```

Une conditionnel ELSE doit être écrit de la manière suivante.
C'est un exemple de conditionnelle simple.

Le commentaire doit être associé à la ligne. Ne pas oublier les crochets ouvrants et fermant. Les Ternaires sont prohibés.

WHILE

```php
<?php

//Déclaration d'une variable de test.
$variable_de_test = 999;

// Tanrt que la variable de test est inférieur à 10000.
while($variable_de_test < 0){

    // Code à executer.

    // Incrémentation de la valeur de test.
    $variable_de_test ++;

}

?>
```

Un conditionnel WHILE doit être écrit de la manière suivante.
C'est un exemple de double simple et infini.
Une boucle WHILE doit toujours avoir une porte de sortie.

Le commentaire doit être associé à la ligne. Ne pas oublier les crochets ouvrants et fermant.

FOR

```php
<?php

//Déclaration d'une variable array qui comporte des strings.
$values_array = array("test" , "test1", "test2");

// Pour toutes les valeurs dans la variables.
foreach($values_array as $value_array){

    // Code à executer.

}

//Variable d'incrémentation par défaut à 0.
$i = 0;

// Autre exemple :

// Pour toutes les valeurs dans la variables.
for($i = 0 ; $i < count($values_array) ; $i++){

    // Code à executer.

}

?>
```

Un conditionnel FOR doit être écrit d'une des manières suivantes.

Le commentaire doit être associé à la ligne. Ne pas oublier les crochets ouvrants et fermant.

Les commentaires

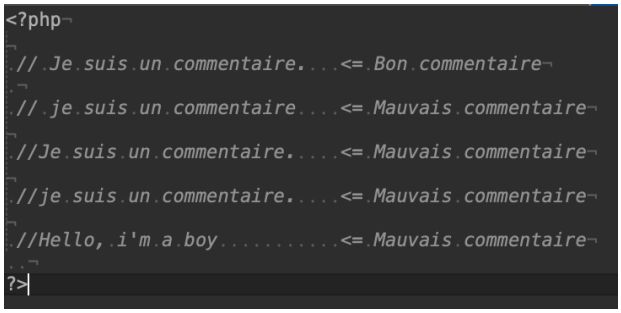
La langue

La langue des commentaires dans l'ensemble du code est le Français.

La typographie du commentaire.

Le commentaire d'écrit la ligne de code associé. Il n'explique rien de plus ou rien de moins. Toutes les lignes de code sans aucune exception doivent être commenter.

Un commentaire comme par une majuscule et se termine par un point comme une phrase basique. Il faut mettre un espace entre le caractère qui déclare le commentaire et le premier caractère de votre commentaire afin d'améliorer la visibilité.



Bonnes pratiques

Si vos commentaires sont correctement faits, une personne n'ayant aucune connaissance de code peut lire et comprendre votre code.

Une bonne pratique est si possible de commenter son code avant de l'avoir écrit.

Les entête de fonctions

Bonnes pratiques

Les fonctions doivent avoir des commentaires afin de pouvoir connaître son fonctionnement, ses objectifs et ce qu'elle nous renvoi sans même devoir lire le code de cette dernière.

Afin d'uniformiser l'ensemble du code, toutes les fonctions doivent respecter l'entête suivante :

```php
<?php

/* NOM DE LA FONCTION (Respect du formatage du nom)
 * @description : Description en quelques mots de la fonction
 * @param $var1 : Explication du type de variable et a quoi elle fait référence
 * @param $var2 : Explication du type de variable et a quoi elle fait référence
 * @param $var3 (optionnal): Explications [Default = 0]
 * @return (Type de variable retourner) (Dans quels cas ?).
 */
public function test($var1, $var2, $var3 = 0){

  //On retourne la somme.
  return ($var1 + $var2 + $var3);

}

// ! Exemple concret.

/* testDeFonction
 * @description : Effectue la somme des variables en paramètres.
 * @param $var1 : Entier a sommer.
 * @param $var2 : Entier a sommer.
 * @param $var3 (optionnal): Entier a sommer. [Default = 0]
 * @return Int : Somme des variables en paramètres..
 */
public function testDeFonction($var1, $var2, $var3 = 0){

  //On retourne la somme.
  return ($var1 + $var2 + $var3);

}


?>
```

Nommage des fonctions

Une fonction possède un nom qui correspond à la tâche qu'elle effectue. Elle doit respecter quelques normes.

Le langage utilisé pour le nommage des fonctions est l'ANGLAIS.

Préfixer par le mot clé IS si elle est un getter sur un booléen (Exemple : isAvaible())

Préfixer par le mot clé GET si elle est un getter sur un autre chose qu'un booléen (Exemple : getDate())

Préfixer par le mot clé SET si elle est un setter (Exemple : setDate())

Si le nom de la fonction possède plusieurs mots, il faut l'organiser avec des majuscules.

Exemple : Fonction qui effectue un traitement sur un date de création.

Nom : processDateCreation()

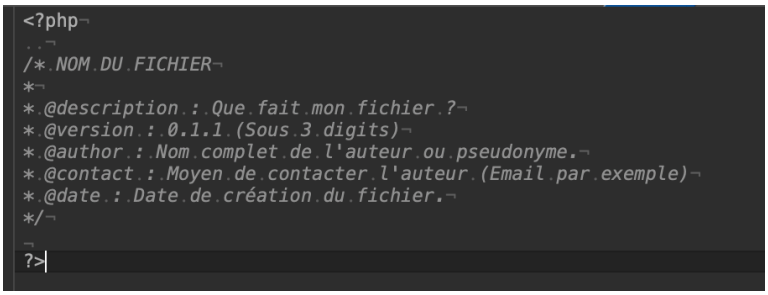
A bannir : process_date_creation()

Si cela concerne un **état** qui doit contenir ou renvoyer un **booléen**
- préfixe avec **is** ou **has** : isLoading, hasPlayed, isNotDone, hasNotPosted…

Si c'est une fonction ou méthode de **callback** pour un **événement**
- préfixe avec **on** : onUpdate, onUserPlayed, onEnded, onSlapped…

Si c'est une variable qui contient un élément du **DOM (JS)**
- préfixe avec toujours la même chose : **ui**Button, **$**button, ui_title…

Entête de fichier



```php
<?php
..
/* NOM DU FICHIER
*
* @description : Que fait mon fichier ?
* @version : 0.1.1 (Sous 3 digits)
* @author : Nom complet de l'auteur ou pseudonyme.
* @contact : Moyen de contacter l'auteur (Email par exemple)
* @date : Date de création du fichier.
*/

?>
```

Chaque fichier doit posséder en faut en tant que première chose à lire un petit entête.

Nommage des variables

Le nom des variables doit être en Français.

Le séparateur entre plusieurs mots dans la variable est le tiret du 8 (_).

La variable a un sens, en lisant le nom de la variable on doit comprendre ce qu'elle contient.

Par exemple une variable qui contient le nombre de mois doit s'appeler : $noms_des_mois

Si la variables contient plusieurs éléments, elle doit être au pluriel. `

Par exemple une variable qui contient le nom de Paul = $nom_de_paul , à l'inverse la variable qui contient le nom des élèves du groupes sera $ $noms_eleves.

Si on ne connait pas le contenue de la variable ou bien qu'elle soit utilisé pour contenir plusieurs éléments différents, il faut utiliser un nom générique. Par exemple une variable qui va contenir des objets de type utilisateur ou groupe s'appellera $objects (Utilisation du pluriel car il peut y avoir plusieurs utilisateurs ou groupe)

Les variables d'incrémentations de nombres, utilisez de préférence le $i ;

Nommage des constantes

Le nom des constantes doit être en Français.

Le séparateur entre plusieurs mots dans la constante est le tiret du 8 (_).

Elles s’écrivent uniquement en majuscules.