

L'organisation des fichiers PHP

Table des matières

I. Contexte	3
II. Découper son HTML	3
III. Exercice : Appliquez la notion	5
IV. Découper le code PHP	5
V. Exercice : Appliquez la notion	7
VI. Architecture d'une application web en PHP	7
VII. Exercice : Appliquez la notion	9
VIII. Auto-évaluation	12
A. Exercice final.....	12
B. Exercice : Défi.....	13
Solutions des exercices	16

I. Contexte

Durée : 1 h

Environnement de travail : Repl.it

Pré-requis : Bases de PHP et d'HTML

Contexte

L'architecture d'une application PHP n'est jamais constituée d'un seul fichier de code contenant toute la logique nécessaire : il s'agit plutôt d'un ensemble de fichiers plus petits ayant chacun leur propre logique et responsabilité. Ce découpage permet en effet de réutiliser des fonctionnalités et de les maintenir plus facilement.

Ce paradigme, c'est-à-dire cette manière d'architecturer son code, peut être illustré par exemple avec la partie `header` des pages d'une application web. En effet, la plupart du temps, celui-ci est toujours le même d'une page à l'autre. Il est donc plus simple d'écrire le code lui correspondant une seule fois dans un fichier spécifique et de réutiliser ce fichier au début de chaque nouvelle page créée. De plus, si une modification devait intervenir, un seul fichier devrait être modifié pour répercuter celle-ci dans toutes les pages de l'application.

Nous allons voir ici comment PHP nous permet de séparer notre code en différents fichiers et de regrouper le contenu de plusieurs fichiers dans un autre, tout d'abord pour le contenu HTML, puis pour le code PHP. Nous verrons enfin une approche d'architecture considérée comme étant une bonne pratique de développement.

II. Découper son HTML

Objectif

- Apprendre à intégrer le contenu d'un fichier HTML dans un fichier PHP

Mise en situation

Tout au long des développements d'une application web PHP, il est recommandé de séparer les différents fichiers HTML les uns des autres et de ne les appeler que lorsque c'est nécessaire. Cela permet de n'écrire que des fichiers HTML plus petits, rangés dans une arborescence facilement maintenable. Nous allons voir comment se fait l'intégration du contenu de ces fichiers au sein d'un seul fichier PHP.

Include et require

L'inclusion d'un fichier dans un script PHP peut se faire avec deux mots-clés : `include` et `require`.

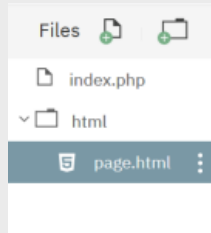
- L'utilisation d'`include` va ajouter le contenu d'un fichier ciblé dans le fichier PHP courant et n'affichera pas d'erreur si celui-ci n'est pas disponible. Cela signifie que la page sera servie normalement, mais que l'espace normalement rempli par le fichier sera vide.
- Similairement, `require` va ajouter le contenu d'un fichier dans le script PHP en cours, mais son exécution s'interrompt et une erreur sera retournée si le fichier n'est pas disponible.

Syntaxe

`include` et `require` ont une syntaxe identique qui consiste à faire suivre le mot-clé par le chemin d'accès du fichier à inclure entre double quotes `" "`.

Exemple

Dans repl.it, il est possible d'ajouter des dossiers et des fichiers grâce au menu de gauche :



Exemple d'un fichier HTML nommé `page.html` et stocké dans un répertoire `html`.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Ma page</title>
5   </head>
6   <body>
7     <div>
8       Hello world !
9     </div>
10  </body>
11 </html>
```

Inclusion du fichier dans un script PHP :

```
1 <?php
2   include "html/page.html"; //Affiche le HTML contenant Hello world"
```

Exemple

Illustration des différences de traitement des erreurs avec include et require

Exemple d'un fichier HTML nommé `page.html` et stocké dans un répertoire `html`.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Ma page</title>
5   </head>
6   <body>
7     <div>
8       Hello word !
9     </div>
10  </body>
11 </html>
```

Inclusion du fichier dans un script PHP :

```
1 <?php
2   include "/html/page.html"; // L'inclusion d'un chemin de fichier erroné génère un warning
3   non bloquant
4   require "/html/page.html"; // L'inclusion d'un chemin de fichier erroné génère une erreur
5   bloquante
```

```
PHP 7.2.17-0ubuntu0.18.04.1 (cli) (built: Apr 18 2019 14:12:08) (NTS)
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.2.0, Copyright (c) 1998-2018 Zend Technologies
[PHP 7.2.17-0ubuntu0.18.04.3 Development Server started at Mon Mar 23 14:48:54 2020]
Listening on http://0.0.0.0:8080
Document root is /home/runner/UnselfishTerribleCodeWarrior
Press Ctrl-C to quit.
[Mon Mar 23 14:48:54 2020] PHP Warning: include(/html/page.html): failed to open stream: No such file or director
y in /home/runner/UnselfishTerribleCodeWarrior/index.php on line 6
[Mon Mar 23 14:48:54 2020] PHP Warning: include(): Failed opening '/html/page.html' for inclusion (include_path=
'./usr/share/php') in /home/runner/UnselfishTerribleCodeWarrior/index.php on line 6
[Mon Mar 23 14:48:54 2020] PHP Warning: require(/html/page.html): failed to open stream: No such file or director
y in /home/runner/UnselfishTerribleCodeWarrior/index.php on line 7
[Mon Mar 23 14:48:54 2020] PHP Fatal error: require(): Failed opening required '/html/page.html' (include_path=
'./usr/share/php') in /home/runner/UnselfishTerribleCodeWarrior/index.php on line 7
[Mon Mar 23 14:48:54 2020] PHP Fatal error: require(): Failed opening required '/html/page.html' (include
_path=
```

Comme le montre l'exécution du code présenté, l'utilisation d'`include` génère un warning qui ne bloque pas l'exécution du code PHP, tandis que l'utilisation de `require` remonte une erreur qui arrête le traitement du script.

Syntaxe À retenir

- L'inclusion de fichiers dans un script PHP peut se faire par l'utilisation des mots-clés `include` et `require`.

III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



Question 1

[solution n°1 p.17]

En tant que développeur, il n'est pas rare d'avoir son propre blog sur lequel on poste des articles sur ce que l'on a appris. Un blog possède au moins une page d'accueil et une page permettant de lire un article. Le `header`, c'est-à-dire le bandeau en haut du site, sera le même sur toutes les pages.

Initialisez un repl.it et créez un fichier "header.html" contenant un titre de niveau 1 de votre choix dans une balise `<header>`.

Indice :

Un titre de niveau 1 est une balise `<h1>`.

Question 2

[solution n°2 p.17]

Dans `index.php`, incluez votre `header`, puis ajoutez ensuite un titre de niveau 2 pour accueillir vos visiteurs : cette page sera la page d'accueil de votre blog.

Question 3

[solution n°3 p.17]

Créez un fichier "article.php" qui sera la page affichant votre article de blog. Incluez votre `header` dans cette page, puis rajoutez le titre de niveau 2 de votre choix, suivi d'un petit paragraphe sur les `include`.

Dans `index.php`, ajoutez un lien vers votre article.

Indice :

Pour rajouter un lien, il faut utiliser la balise `<a>`. Son attribut "href" permet d'indiquer vers quel fichier on veut notre lien.

IV. Découper le code PHP

Objectif

- Apprendre à intégrer des fichiers PHP dans un même script PHP

1 <https://repl.it/>

Mise en situation

Une bonne organisation des fichiers PHP est garante d'une grande souplesse et d'une meilleure maintenabilité d'une application web PHP en cours de développement. En effet, elle permet de comprendre rapidement où doivent se faire les ajouts et correctifs qui peuvent intervenir tout au long de sa durée de vie. Nous allons voir ici comment PHP nous permet d'inclure des fichiers contenant du code PHP dans un seul fichier PHP.

Include_once et require_once

Lors de l'inclusion de fichiers PHP, il est nécessaire de s'assurer que les fichiers à inclure ne l'ont pas déjà été. Pour cela, PHP propose les mots-clés `include_once` et `require_once`.

Les instructions `include_once` et `require_once` fonctionnent exactement de la même manière que `include` et `require`, mais elles vérifient en plus que les fichiers n'ont pas déjà été inclus lors d'instructions précédentes, ce qui évite d'inclure plusieurs fois le même fichier, et donc le même code.

Inclure un fichier PHP permet d'exécuter tout le code qu'il contient, mais aussi d'utiliser toutes les variables définies à l'intérieur de ce fichier.

Exemple

Voici un fichier PHP nommé `constants.php` dans le répertoire `constants` définissant une constante et affichant le texte `'loaded'` à la fin de son chargement :

```
1 <?php
2 define('CONSTANT_VALUE', 'Ma constante.');
```

Fichier PHP affichant le contenu du fichier `constants.php` :

```
1 <?php
2 require_once "constants/constants.php";
3 echo '<br>';
4 echo CONSTANT_VALUE;
5 require_once "constants/constants.php";
6
```

L'exemple présenté récupère le contenu du fichier PHP `constants.php` et affiche une constante qu'il définit. On peut constater que l'appel à `require_once` une seconde fois ne déclenche pas de chargement du fichier, car le texte `'loaded'` n'est affiché qu'une seule fois.

```
1 loaded
2 Ma constante.
```

Remarque

Il est considéré comme une bonne pratique de déclarer les constantes d'une application dans un fichier séparé.

Syntaxe À retenir

- L'inclusion de codes PHP issus de différents fichiers PHP nécessite une vérification afin qu'ils ne soient pas chargés deux fois.
- Cela peut être fait par l'utilisation des mots-clés `include_once` et `require_once`.

V. Exercice : Appliquez la notion

Nous allons moderniser notre blog en y incluant du PHP pour lui ajouter une fonctionnalité utile : la centralisation des textes.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



Question 1

[solution n°4 p.17]

Le code de notre blog est le suivant :

index.php

```
1 <?php include 'header.html'; ?>
2 <h2>Bienvenue !</h2>
3 <a href="article.php">Mon article</a>
```

article.php

```
1 <?php include 'header.html'; ?>
2 <h2>Mon article</h2>
3 <p>Les includes, c'est super pratique !</p>
```

header.html

```
1 <header>
2   <h1>Mon blog</h1>
3 </header>
```

Créez un fichier "constants.php" qui va définir des constantes PHP contenant le texte des fichiers "index.php" et "article.php". Définissez une constante par texte, sans inclure de HTML.

Indice :

Le titre de l'article placé dans le lien du fichier "index.php" est le même que celui du titre de "article.php". Une seule constante suffit !

Question 2

[solution n°5 p.17]

Remplacez les textes des fichiers index.php et article.php par leurs constantes et incluez le fichier "constants.php" dans chaque page.

VI. Architecture d'une application web en PHP

Objectif

- Comprendre comment architecturer une application web en PHP

Mise en situation

L'architecture, et donc l'organisation, d'une application web en PHP doit suivre des règles considérées comme des bonnes pratiques de développement. Celles-ci garantissent la réalisation d'une application web maintenable et évolutive sur le long terme. Nous allons ici en présenter les grands principes, en montrant l'avantage à séparer la logique algorithmique de la logique d'affichage, c'est-à-dire en séparant les fichiers PHP des fichiers HTML.

1 <https://repl.it/>

Séparer l'affichage des calculs

Un des principes des bonnes pratiques de développement en termes d'architecture est un découpage le plus important possible entre la partie présentation d'une application et sa partie logique.

Ce principe est appelé **couplage faible** entre l'interface et la logique des fonctionnalités de l'application. Son principe repose sur le fait que l'affichage d'une information peut se faire sans que l'interface ait **connaissance** de la logique métier appliquée pour l'obtenir.

Concrètement, cela signifie que des modifications intervenant sur l'interface n'impacteront pas l'affichage, et inversement. Cela assure de plus une grande réutilisabilité des briques de code métier et en facilite la maintenabilité.

Exemple

Un premier fichier PHP nommé **lineBreak.php** placé dans un répertoire **display**.

```
1 <?php
2 /**
3  * Return a HTML line break
4  *
5  * return string
6  */
7 function displayHTMLLineBreak() {
8     return '<br>';
9 }
```

Un second fichier PHP nommé **list.php** incluant le fichier **lineBreak.php** placé dans un répertoire **display**.

```
1 <?php
2 require_once "lineBreak.php";
3 /**
4  * Display a list of data from an array
5  *
6  * @param [] $arrayData
7  * return string
8  */
9 function displayList($arrayData) {
10     $displayedValue = '';
11     foreach($arrayData as $data) {
12         $displayedValue .= $data;
13         $displayedValue .= displayHTMLLineBreak();
14     }
15
16     return $displayedValue;
17 }
```

Un fichier HTML nommé **header.html** placé dans un répertoire **html**.

```
1 <div>
2 <p>
3     Cet élément sera présent dans toutes les pages de mon application
4 </p>
5 </div>
```

Un fichier contenant du HTML et du PHP nommé **page.php** incluant le fichier **header.html** placé dans un répertoire **html**.


```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Ma page</title>
5   </head>
6   <body>
7     <?php include 'html/header.html'; ?>
8     <div>
9       <?php echo $dataToDisplay; ?>
10    </div>
11  </body>
12 </html>

```

Le fichier PHP principal affichant la page et incluant le fichier **list.php**.

```

1 <?php
2   require_once "display/list.php";
3
4   $data = ['Chaque élément', 's\'affiche', 'sur une ligne'];
5   $dataToDisplay = displayList($data);
6   include 'html/page.php';

```

Cet élément sera présent dans toutes les pages de mon application

Chaque élément
s'affiche
sur une ligne

Dans cet exemple, on remarque que le code est réparti dans plusieurs fichiers séparant la logique de l'affichage. Ainsi, le répertoire **display** contient deux fichiers permettant l'organisation des données à afficher, et le répertoire **html** contient deux fichiers gérant uniquement l'affichage. De cette manière, on peut facilement intervenir sur n'importe quel aspect de notre application en limitant les effets de bord.

Syntaxe À retenir

- Un des éléments fondamentaux des bonnes pratiques d'architecture d'une application web est le découpage du code en différents fichiers, permettant un couplage faible entre l'interface et la logique métier afin de permettre une grande souplesse et limiter les impacts des évolutions et corrections qui peuvent intervenir dans la vie de l'application.

VII. Exercice : Appliquez la notion

Dans cet exercice, nous allons découper du code en différents fichiers et fonctions afin de le rendre plus maintenable.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



1 <https://repl.it/>

Question 1

[solution n°6 p.18]

Nous sommes en charge d'une vieille application multilingue permettant d'afficher les sports préférés d'un utilisateur :

```

1 <?php
2 $user = [
3     'name' => 'John',
4     'language' => 'fr',
5     'hobbies' => [
6         [
7             'name' => 'piano',
8             'type' => 'music'
9         ],
10        [
11            'name' => 'volley-ball',
12            'type' => 'sport'
13        ],
14        [
15            'name' => 'tennis',
16            'type' => 'sport'
17        ],
18        [
19            'name' => 'saxophone',
20            'type' => 'music'
21        ],
22    ]
23 ];
24 ?>
25 <html>
26 <head></head>
27 <body>
28 <h1>
29     <?php
30         if ($user['language'] === 'fr') {
31             echo 'Bonjour ' . $user['name'] . '!';
32         } else {
33             echo 'Hello ' . $user['name'] . '!';
34         }
35     ?>
36 </h1>
37 <div>
38     <?php
39         if ($user['language'] === 'fr') {
40             echo 'Sports préférés :';
41         } else {
42             echo 'Favorites sports :';
43         }
44     ?>
45 <ul>
46     <?php
47         foreach ($user['hobbies'] as $hobby) {
48             if ($hobby['type'] == 'sport') {
49                 echo '<li>' . $hobby['name'] . '</li>';
50             }
51         }
52     ?>
53 </ul>
54 </div>

```

```
55 </body>
56 </html>
```

Cette page aurait besoin d'être nettoyée : le code est complexe à comprendre et à maintenir. Tout d'abord, lancez ce code dans un repl.it et assurez-vous qu'il soit fonctionnel.

Question 2

[solution n°7 p.18]

Il y a quatre points de complexité dans ce code : les données de l'utilisateur, le support des langues multiples, l'affichage de la liste des sports et le fait que le tout soit entremêlé dans le HTML.

Commençons simplement par séparer la partie "visible" de la partie "données" : créez un fichier "user.php" qui contiendra tout ce qui concerne l'utilisateur. Commencez par créer une fonction `getUser()` qui retourne les données de l'utilisateur.

Créez ensuite un fichier "template.php" et mettez-y toute la partie HTML.

Enfin, modifiez index.php pour appeler la fonction `getUser()`, stockez le résultat dans une variable `$user` et incluez template.php (qui utilise la variable `$user`).

Indice :

N'oubliez pas d'inclure le fichier contenant la fonction `getUser` !

Question 3

[solution n°8 p.19]

Gérons maintenant plus simplement l'aspect multilingue du site. Créez un dossier "languages" et deux fichiers "language_fr.php" et "language_en.php". Ces deux fichiers doivent définir des constantes pour chaque texte multilingue du site, mais le nom des constantes d'un même texte doit être le même pour tous les langages : nous allons ensuite charger le bon fichier selon la langue de l'utilisateur. Pour chaque constante créée, remplacez le texte correspondant dans template.php.

Par exemple, le texte de bienvenue aura pour nom 'WELCOME_TEXT' dans les deux fichiers, mais la valeur 'Bonjour' dans language_fr.php et 'Hello' dans language_en.php. Ainsi, dans template.php, le code pour dire 'Bonjour' sera :

```
1 <h1>
2 <?php
3     echo WELCOME_TEXT . ' ' . $user['name'] . ' !';
4 ?>
5 </h1>
```

Une fois les modifications effectuées, créez un fichier language.php à la racine de votre site. Ce fichier doit contenir une fonction `loadLanguage` qui prend en paramètre une langue et qui inclut le fichier de langue correspondant. Appelez ensuite cette fonction dans index.php, et modifiez la langue de l'utilisateur retournée par `getUser()` pour vous assurer que tout fonctionne bien.

Indice :

Vous devriez avoir deux constantes dans chaque fichier.

Indice :

Vous pouvez réutiliser la condition de template.php pour gérer la langue.

Question 4

[solution n°9 p.20]

Le template est déjà beaucoup plus clair, mais il reste une dernière étape.

Dans user.php, créez une fonction `getUserSports` qui prend un utilisateur en paramètre et qui retourne un tableau contenant le nom de tous ses hobbies de type "sport".

Ensuite, appelez-la dans index.php, stockez le résultat dans une variable `$sports` et utilisez-la dans template.php.

Pour complètement séparer les données du template, stockez également le nom de l'utilisateur dans une variable `$name` et utilisez-la dans template.php. Ainsi, la partie template n'a même pas à connaître la structure d'un utilisateur.

Indice :

Là encore, vous pouvez récupérer la logique déjà présente dans `template.php`. Tout ce que l'on fait, au final, c'est déplacer du code et adapter en conséquence.

VIII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°10 p.22]

Exercice

Quelle est la différence entre `include` et `include_once` ?

- ☐ `include_once` provoque une erreur si le fichier est inclus plus d'une fois
- ☐ `include_once` n'inclut pas le fichier s'il est déjà inclus
- ☐ `include_once` coupe l'exécution du programme si le fichier demandé n'existe pas

Exercice

Quelle est la différence entre `include` et `require` ?

- ☐ `require` provoque une erreur si le fichier est inclus plus d'une fois
- ☐ `require` n'inclut pas le fichier s'il est déjà inclus
- ☐ `require` coupe l'exécution du programme si le fichier demandé n'existe pas

Exercice

Qu'affiche ce code ?

```
1 <?php
2 //fichier index.php
3
4 $user = 'admin';
5 if ($user === 'admin') {
6     require_once 'admin.php';
7 } else {
8     require_once 'user.php';
9 }
10
11 echo GREETINGS;
```

```
1 <?php
2 // fichier admin.php
3
4 define('GREETINGS', 'Salutations');
```

```
1 <?php
2 // fichier user.php
3
4 define('GREETINGS', 'Bonjour');
```

- ☐ Bonjour
- ☐ Salutations
- ☐ GREETINGS

Exercice

Qu'affiche le code suivant ?

```
1 <?php
2 //index.php
3
4 require_once 'displayUsername.php';
5 $username = 'John';

1 <?php
2 //displayUsername.php
3
4 echo $username;
```

- ☐ John
- ☐ Une erreur

Exercice

Qu'affiche le code suivant ?

```
1 <?php
2 //index.php
3
4 echo 'Bonjour';
5 require_once 'index.php';
```

- ☐ Bonjour
- ☐ BonjourBonjour
- ☐ Bonjour écrit à l'infini

Exercice

Qu'affiche le code suivant ?

```
1 <?php
2 //index.php
3
4 echo 'Bonjour';
5 require 'index.php';
```

- ☐ Bonjour
- ☐ BonjourBonjour
- ☐ Bonjour écrit à l'infini

B. Exercice : Défi

Dans cet exercice, nous allons réaliser un site permettant d'afficher des recettes de cuisine. Pour cela, nous allons successivement voir le site sous trois angles différents : celui du développeur front-end, celui du développeur back-end, et celui de l'intégrateur. Puis nous allons découper notre code de manière à ce que les trois puissent travailler ensemble.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :



1 <https://repl.it/>

Question 1

[solution n°11 p.24]

Pour cette première étape, mettez-vous dans la peau d'un développeur front-end uniquement : vous savez que vous allez recevoir des données, mais vous n'avez aucune idée de la manière dont elles sont chargées.

Vous travaillez sur le template de la page permettant de lister les recettes de cuisine, qui doit ressembler à ça :

Flan patissier

Temps de préparation : 60 minutes

Difficulté : **

Flan patissier

Temps de préparation : 60 minutes

Difficulté : **

Flan patissier

Votre rôle est de travailler sur le bloc qui constitue une recette : ce bloc sera répété plusieurs fois avec des informations différentes.

Votre développeur back-end vous indique qu'il va vous retourner :

- Un titre, qui est une chaîne de caractères
- Un temps de préparation, qui est un entier représentant le nombre de minutes
- Une difficulté, qui est un entier

Lancez un repl.it, créez un dossier "front" puis un fichier "recipe.php", qui contiendra le code nécessaire pour afficher le bloc d'une recette.

Ce bloc est constitué d'une balise titre de niveau 3 pour le titre, d'une div pour le temps de préparation et d'une div pour la difficulté. Attention : la difficulté est affichée sous forme d'étoiles. Une difficulté de 2 devra donc afficher deux étoiles.

Une fois le bloc terminé, listez le nom des variables dont vous avez besoin dans votre template (pour votre intégrateur) et testez l'affichage de votre bloc en l'incluant dans le fichier index.php. Pensez à renseigner les variables nécessaires avant l'inclusion.

Question 2

[solution n°12 p.24]

Mettons nous maintenant à la place du développeur back-end : votre but est de charger les données des différentes recettes.

Nous récupérons les recettes via une fonction fournie par notre client, que vous pouvez déjà copier/coller dans un fichier "recipesLoader.php" placé dans un dossier "back" :

```
1 <?php
2
3 function loadRecipes(): array
4 {
5     return
6     json_decode(base64_decode('WyJHYXRlYXUgYXUgY2hvY29sYXQiLDQwLDEsIkZsYW4gcGF0aXNzaWVyIiw2MCwyXQ'),
7     true);
```

Cette fonction, inutilement compliquée, va servir à simuler la récupération des recettes : vous ne pouvez pas modifier cette fonction, simplement l'utiliser telle quelle, comme si vous appeliez un service externe. Cette fonction retourne un tableau indicé qui contient, séquentiellement, le titre, le temps de préparation et la difficulté de chaque recette. Ainsi, ce tableau est sous la forme : [titre1, temps1, difficulté1, titre2, temps2, difficulté2, titre3, temps3, difficulté3 ...].

Écrivez une fonction `getRecipes` qui appelle la fonction `loadRecipes` et qui retourne un tableau de recettes réorganisées sous la forme suivante :

```
1 <?php
2 [
3     [
4         'title' => 'titre1',
5         'time' => 'temps1',
6         'difficulty' => 'difficulte1',
7     ],
8     [
9         'title' => 'titre2',
10        'time' => 'temps2',
11        'difficulty' => 'difficulte2',
12    ], // etc...
13 ]
```

Une fois la fonction prête, incluez-la dans index.php et assurez-vous du contenu en utilisant `var_dump`.

Notez également la structure de ce tableau pour votre intégrateur.

Indice :

Lorsque vous parcourez un tableau, vous n'êtes pas obligé de le parcourir de 1 en 1 !

Indice :

Parcourez le tableau de 3 en 3. La clef actuelle est le titre, la clef +1 est le temps, et la clef +2 est la difficulté.

Question 3

[solution n°13 p.25]

Il est maintenant temps de se mettre dans la peau de notre intégrateur ! Nous avons reçu le message suivant de la part de l'équipe back :

La fonction `getRecipes()` du fichier back/recipesLoader.php retourne un tableau indexé contenant un tableau associatif possédant les champs "title", "time" et "difficulty".

Et le message suivant de l'équipe front :

Le fichier front/recipe.php contient le bloc d'une recette. Avant chaque inclusion, il faut renseigner les variables `$title`, `$preparationTime` et `$difficulty`.

Modifiez le fichier index.php pour mettre en commun le travail des deux équipes et afficher la liste des recettes.

Indice :

Attention, les champs retournés par le back et ceux attendus par le front n'ont pas forcément les mêmes noms !

Question 4

[solution n°14 p.25]

Quel est l'avantage d'avoir découpé son code ainsi ?

Solutions des exercices

p. 5 Solution n°1**header.html :**

```
1 <header>
2   <h1>Mon blog</h1>
3 </header>
```

p. 5 Solution n°2**index.php :**

```
1 <?php include 'header.html'; ?>
2 <h2>Bienvenue !</h2>
```

p. 5 Solution n°3**index.php :**

```
1 <?php include 'header.html'; ?>
2 <h2>Bienvenue !</h2>
3 <a href="article.php">Mon article</a>
```

article.php :

```
1 <?php include 'header.html'; ?>
2 <h2>Mon article</h2>
3 <p>Les includes, c'est super pratique !</p>
```

p. 7 Solution n°4

```
1 <?php
2
3 define('WELCOME_TITLE', 'Bienvenue !');
4 define('ARTICLE_TITLE', 'Mon article');
5 define('ARTICLE_CONTENT', 'Les includes, c\'est super pratique !');
```

p. 7 Solution n°5**index.php :**

```
1 <?php
2 include 'header.html';
3 include_once 'constants.php';
4 ?>
5
6 <h2><?php echo WELCOME_TITLE; ?></h2>
7 <a href="article.php"><?php echo ARTICLE_TITLE; ?></a>
```

article.php :

```
1 <?php
2 include 'header.html';
3 include_once 'constants.php';
4 ?>
5
6 <h2><?php echo ARTICLE_TITLE; ?></h2>
7 <p><?php echo ARTICLE_CONTENT; ?></p>
```

On remarque que `ARTICLE_TITLE` est utilisé dans les deux pages : si on doit le modifier, il suffira de modifier la valeur de la constante, et le texte sera mis à jour sur les deux pages !

p. 10 Solution n°6

Le code est fonctionnel et affiche :

Bonjour John!

Sports préférés :

- volley-ball
- tennis

p. 11 Solution n°7

index.php :

```
1 <?php
2 require_once 'user.php';
3 $user = getUser();
4 include 'template.php';
5
```

user.php :

```
1 <?php
2
3 function getUser(): array {
4     return [
5         'name' => 'John',
6         'language' => 'fr',
7         'hobbies' => [
8             [
9                 'name' => 'piano',
10                'type' => 'music'
```

```

11     ],
12     [
13         'name' => 'volley-ball',
14         'type' => 'sport'
15     ],
16     [
17         'name' => 'tennis',
18         'type' => 'sport'
19     ],
20     [
21         'name' => 'saxophone',
22         'type' => 'music'
23     ],
24 ]
25 ];
26 }

```

template.php :

```

1 <html>
2 <head></head>
3 <body>
4 <h1>
5     <?php
6         if ($user['language'] === 'fr') {
7             echo 'Bonjour ' . $user['name'] . '!';
8         } else {
9             echo 'Hello ' . $user['name'] . '!';
10        }
11    ?>
12 </h1>
13 <div>
14     <?php
15         if ($user['language'] === 'fr') {
16             echo 'Sports préférés :';
17         } else {
18             echo 'Favorites sports :';
19         }
20    ?>
21 <ul>
22     <?php
23         foreach ($user['hobbies'] as $hobby) {
24             if ($hobby['type'] == 'sport') {
25                 echo '<li>' . $hobby['name'] . '</li>';
26             }
27         }
28    ?>
29 </ul>
30 </div>
31 </body>
32 </html>

```

index.php :

```
1 <?php
2     require_once 'user.php';
3     require_once 'language.php';
4
5     $user = getUser();
6     loadLanguage($user['language']);
7
8     include 'template.php';
9
```

languages/language_en.php :

```
1 <?php
2 define('WELCOME_TEXT', 'Hello');
3 define('SPORTS_TEXT', 'Favorites sports');
```

languages/language_fr.php :

```
1 <?php
2 define('WELCOME_TEXT', 'Bonjour');
3 define('SPORTS_TEXT', 'Sports préférés');
```

language.php :

```
1 <?php
2 function loadLanguage(string $language)
3 {
4     if ($language == 'fr') {
5         require_once('languages/language_fr.php');
6     } else {
7         require_once('languages/language_en.php');
8     }
9 }
```

template.php :

```
1 <html>
2 <head></head>
3 <body>
4 <h1>
5     <?php echo WELCOME_TEXT.' '. $user['name'].' !'; ?>
6 </h1>
7 <div>
8     <?php echo SPORTS_TEXT.' :'; ?>
9     <ul>
10         <?php
11             foreach ($user['hobbies'] as $hobby) {
12                 if ($hobby['type'] == 'sport') {
13                     echo '<li>'. $hobby['name'].'</li>';
14                 }
15             }
16         ?>
17     </ul>
18 </div>
19 </body>
20 </html>
```

template.php :

```
1 <html>
2 <head></head>
3 <body>
4 <h1>
5     <?php echo WELCOME_TEXT.' '.$name.' !'; ?>
6 </h1>
7 <div>
8     <?php echo SPORTS_TEXT.' :'; ?>
9     <ul>
10         <?php
11             foreach ($sports as $sport) {
12                 echo '<li>'.$sport.'</li>';
13             }
14         ?>
15     </ul>
16 </div>
17 </body>
18 </html>
```

index.php :

```
1 <?php
2 require_once 'user.php';
3 require_once 'language.php';
4
5 $user = getUser();
6 loadLanguage($user['language']);
7 $name = $user['name'];
8 $sports = getUserSports($user);
9
10 include 'template.php';
11
```

user.php :

```
1 <?php
2
3 function getUser(): array {
4     return [
5         'name' => 'John',
6         'language' => 'en',
7         'hobbies' => [
8             [
9                 'name' => 'piano',
10                'type' => 'music'
11            ],
12            [
13                'name' => 'volley-ball',
14                'type' => 'sport'
15            ],
16            [
17                'name' => 'tennis',
18                'type' => 'sport'
19            ],
20            [
21                'name' => 'saxophone',
22                'type' => 'music'
23            ],
24        ]
25    ]
26 }
```

```

25 ];
26 }
27
28 function getUserSports(array $user)
29 {
30     $sports = [];
31     foreach ($user['hobbies'] as $hobby) {
32         if ($hobby['type'] == 'sport') {
33             $sports[] = $hobby['name'];
34         }
35     }
36
37     return $sports;
38 }

```

Exercice p. 12 Solution n°10

Exercice

Quelle est la différence entre `include` et `include_once` ?

- ☐ `include_once` provoque une erreur si le fichier est inclus plus d'une fois
- ☒ `include_once` n'inclut pas le fichier s'il est déjà inclus
- ☐ `include_once` coupe l'exécution du programme si le fichier demandé n'existe pas
- ☒ `include_once` vérifie d'abord que le fichier n'a pas déjà été inclus. Si c'est le cas, alors il ne l'inclut pas.

Exercice

Quelle est la différence entre `include` et `require` ?

- ☐ `require` provoque une erreur si le fichier est inclus plus d'une fois
- ☐ `require` n'inclut pas le fichier s'il est déjà inclus
- ☒ `require` coupe l'exécution du programme si le fichier demandé n'existe pas
- ☒ `require` permet de lever une erreur si le fichier requis n'existe pas. `include` ne fait que lancer un warning, mais le programme continue son exécution.

Exercice

Qu'affiche ce code ?

```


1 <?php
2 //fichier index.php
3
4 $user = 'admin';
5 if ($user === 'admin') {
6     require_once 'admin.php';
7 } else {
8     require_once 'user.php';
9 }
10
11 echo GREETINGS;

```

```
1 <?php
2 // fichier admin.php
3
4 define('GREETINGS', 'Salutations');
```

```
1 <?php
2 // fichier user.php
3
4 define('GREETINGS', 'Bonjour');
```

- ☐ Bonjour
- ☒ Salutations
- ☐ GREETINGS

 L'utilisateur est un administrateur, donc on inclut le fichier admin.php. Ce dernier déclare une constante "GREETINGS" qui vaut "Salutations", qui est affichée ensuite.


Exercice

Qu'affiche le code suivant ?

```
1 <?php
2 //index.php
3
4 require_once 'displayUsername.php';
5 $username = 'John';
```

```
1 <?php
2 //displayUsername.php
3
4 echo $username;
```

- ☐ John
- ☒ Une erreur


 La variable `$username` est définie après l'inclusion du fichier qui est censé l'afficher. La variable est donc introuvable au moment de l'affichage, ce qui provoque une erreur.

Exercice

Qu'affiche le code suivant ?

```
1 <?php
2 //index.php
3
4 echo 'Bonjour';
5 require_once 'index.php';
```

- ☒ Bonjour
- ☐ BonjourBonjour
- ☐ Bonjour écrit à l'infini


 `require_once` n'importe que les fichiers qui n'ont pas déjà été inclus. Étant donné que nous sommes déjà sur la page `index.php`, alors le fichier est déjà inclus d'office lorsque l'on arrive sur la page, donc le `require_once` ne l'importe pas. Un seul "Bonjour" est affiché.

Exercice

Qu'affiche le code suivant ?

```
1 <?php
2 //index.php
3
4 echo 'Bonjour';
5 require 'index.php';
```

- ☐ Bonjour
- ☐ BonjourBonjour
- ☒ Bonjour écrit à l'infini

 Cette fois, on ne vérifie pas si le fichier est déjà inclus ou non, donc on inclut le fichier actuel, qui inclut le fichier actuel, qui inclut le fichier actuel... C'est une boucle infinie, sans les boucles !

p. 14 Solution n°11

Fichier front/recipe.php :

```
1 <div>
2 <h3><?php echo $title; ?></h3>
3 <div>Temps de préparation : <?php echo $preparationTime; ?> minutes</div>
4 <div>
5     Difficulté : <?php
6         for ($starNumber = 0; $starNumber < $difficulty; $starNumber++) {
7             echo '*';
8         }
9     ?>
10 </div>
11 </div>
```

index.php, juste le test de 3 affichages avec des valeurs de test :

```
1 <?php
2
3 $title = 'Flan patissier';
4 $preparationTime = '60';
5 $difficulty = 2;
6
7 for ($i = 0; $i < 3; $i++) {
8     include 'front/recipe.php';
9 }
```

Et on laisse un petit mot à notre intégrateur :

"Pour utiliser recipe.php, il faut renseigner les variables \$title, \$preparationTime et \$difficulty."

p. 15 Solution n°12

back/recipesLoader.php :

```
1 <?php
2
3 function loadRecipes(): array
4 {
5     return
6     json_decode(base64_decode('WyJHYXRlYXUgY2hvY29sYXQiLDQwLDEsIkZsYW4gcGF0aXNzaWVyIiw2MCwyXQ'),
7         true);
8 }
```



```

8 function getRecipes(): array
9 {
10     $recipes = loadRecipes();
11     $recipesCount = count($recipes);
12     $formattedRecipes = [];
13
14     for ($i = 0; $i < $recipesCount; $i += 3) {
15         $formattedRecipes[] = [
16             'title' => $recipes[$i],
17             'time' => $recipes[$i + 1],
18             'difficulty' => $recipes[$i + 2]
19         ];
20     }
21
22     return $formattedRecipes;
23 }

```

index.php :

```

1 <?php
2
3 require_once 'back/recipesLoader.php';
4 var_dump(getRecipes());

```

Pour l'intégrateur : la fonction `getRecipes()` retourne un tableau indexé contenant un tableau associatif possédant les champs "title", "time" et "difficulty".

p. 15 Solution n°13

```

1 <?php
2
3 require_once 'back/recipesLoader.php';
4
5 foreach (getRecipes() as $recipe) {
6     $title = $recipe['title'];
7     $preparationTime = $recipe['time'];
8     $difficulty = $recipe['difficulty'];
9
10    include 'front/recipe.php';
11 }

```

p. 16 Solution n°14

Même si, dans cet exercice, vous étiez la seule personne à faire les trois éléments, en entreprise, les trois équipes (back, front et intégration) sont souvent composées de personnes différentes. Découper son code ainsi permet de faire en sorte que les trois équipes puissent travailler en parallèle : l'important est de s'être mis d'accord au début.

Le code est également plus clair : si un bloc est mal positionné, on sait que cela vient du fichier `recipe.php`. En revanche, si une donnée n'apparaît pas, on sait que c'est plutôt du côté du back. Ainsi, notre code est bien plus facile à déboguer.