PHP : inclure d'autres pages

Lorsque tu débutes en HTML, il est tentant de copier-coller des sections HTML communes (comme les en-têtes, les menus ou les pieds de page) dans chaque fichier de ton site. Mais ce n'est **pas une bonne pratique**. PHP propose des solutions pour éviter ce travail répétitif et faciliter la maintenance grâce aux fonctions `include`, `require`, et `require_once`.

PHP — `include` & `require` : un site plus facile à maintenir

🎯 Objectifs du cours

À la fin de ce cours, tu seras capable de :

1. **Expliquer** pourquoi répéter du code HTML dans chaque page est une mauvaise pratique
2. **Utiliser** `include` et `require` pour insérer des fichiers PHP dans une page
3. **Centraliser** le header, le menu et le footer d'un site dans des fichiers séparés
4. **Distinguer** `include`, `require`, `include_once` et `require_once`
5. **Structurer** un projet web en dossiers pour une meilleure lisibilité

🔑 Les 5 notions-clés

#	Notion	En une phrase
1	**DRY** (Don't Repeat Yourself)	Ne jamais écrire deux fois le même code
2	`include`	Insère le contenu d'un fichier, continue si introuvable
3	`require`	Insère le contenu d'un fichier, **plante** si introuvable
4	`_once`	Garantit qu'un fichier n'est inséré qu'une seule fois
5	**Fichier partiel**	Un fichier PHP qui ne s'affiche pas seul, mais est conçu pour être inclus

🧩 Partie 1 — Le problème : le copier-coller, ennemi de la maintenance

Imagine un site web avec 5 pages : `index.php`, `agenda.php`, `contact.php`, `artistes.php`, `billetterie.php`.

Chaque page contient ce menu de navigation :

```
<nav>
  <a href="index.php">Accueil</a>
  <a href="agenda.php">Agenda</a>
  <a href="artistes.php">Artistes</a>
  <a href="billetterie.php">Billetterie</a>
```

```
    <a href="contact.php">Contact</a>
</nav>
```

Scénario catastrophe : ton client veut ajouter un lien "Galerie" dans le menu.

➡️ Tu dois ouvrir **5 fichiers** et modifier **5 fois** le même code HTML. ➡️ Tu oublies un fichier → le site est **incohérent**. ➡️ Tu fais une faute de frappe → le site est **cassé sur une page**.

💡 **Principe DRY :** si tu copies-colles du code, c'est le signe qu'il faut le **centraliser** dans un seul fichier.

🏗️ Partie 2 — La solution : structurer son projet

Structure de dossiers recommandée

```
mon-site/
│
├── includes/           ← les fichiers partiels (jamais appelés directement)
│   ├── header.php
│   ├── menu.php
│   └── footer.php
│
├── index.php
├── agenda.php
├── artistes.php
├── billetterie.php
└── contact.php
```

📁 Le dossier `includes/` contient des **fichiers partiels** : des morceaux de page, pas des pages complètes.

🔧 Partie 3 — `include` et `require` en pratique

Syntaxe de base

```php
<?php include 'includes/header.php'; ?>
<?php include 'includes/menu.php'; ?>

<!-- Contenu spécifique à cette page -->

<?php include 'includes/footer.php'; ?>
```

Créons nos fichiers partiels

`includes/header.php`

```
<!DOCTYPE html>
<html lang="fr">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mon Site</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel='
</head>
<body>
```

includes/menu.php

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container">
        <a class="navbar-brand" href="index.php">MonSite</a>
        <ul class="navbar-nav">
            <li class="nav-item"><a class="nav-link" href="index.php">Accueil</a></li>
            <li class="nav-item"><a class="nav-link" href="agenda.php">Agenda</a></li>
            <li class="nav-item"><a class="nav-link" href="artistes.php">Artistes</a></li>
            <li class="nav-item"><a class="nav-link" href="contact.php">Contact</a></li>
        </ul>
    </div>
</nav>
```

includes/footer.php

```
<footer class="bg-dark text-white text-center py-3 mt-5">
    <p>&copy; <?php echo date('Y'); ?> MonSite – Tous droits réservés</p>
</footer>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></s
</body>
</html>
```

Une page complète avec include

index.php

```
<?php include 'includes/header.php'; ?>
<?php include 'includes/menu.php'; ?>

<main class="container mt-4">
    <h1>Bienvenue sur MonSite</h1>
    <p>Voici la page d'accueil.</p>
</main>

<?php include 'includes/footer.php'; ?>
```

agenda.php

```
<?php include 'includes/header.php'; ?>
<?php include 'includes/menu.php'; ?>

<main class="container mt-4">
    <h1>Agenda</h1>
    <p>Les prochains événements arrivent bientôt !</p>
```

```
7 | </main>
8 |
9 | <?php include 'includes/footer.php'; ?>
```

✅ **Résultat :** pour ajouter un lien dans le menu, tu modifies **un seul fichier** (`menu.php`), et **toutes les pages** sont automatiquement mises à jour.

⚡ Partie 4 — `include` vs `require` : quelle différence ?

	`include`	`require`
Fichier introuvable	⚠️ Avertissement, **le script continue**	❌ Erreur fatale, **le script s'arrête**
Quand l'utiliser ?	Éléments facultatifs (widget, bandeau promo...)	Éléments **indispensables** (header, connexion DB...)

Exemple concret

```
<?php
1 | // Si ce fichier est absent → erreur fatale, on ne peut pas afficher la page sans connexion DB
2 | require 'includes/connexion.php';
3 |
4 | // Si ce fichier est absent → simple avertissement, la page s'affiche quand même
5 | include 'includes/bandeau-promo.php';
?>
```

💡 **Règle pratique :** utilise `require` pour tout ce qui est **vital** (connexion base de données, authentification, header...). Utilise `include` pour tout ce qui est **optionnel**.

🔒 Partie 5 — `include_once` et `require_once`

Le problème sans `_once`

```
<?php
1 | include 'includes/connexion.php'; // Connexion à la BD → OK
2 | // ... du code ...
3 | include 'includes/connexion.php'; // Connexion une 2e fois → erreur ou doublon !
?>
```

Si `connexion.php` définit une variable `$conn` ou une fonction, l'inclure deux fois peut provoquer des **erreurs** (variable redéfinie, fonction déjà déclarée...).

La solution : `_once`

```
<?php
1 | require_once 'includes/connexion.php'; // Inclus une seule fois, même si appelé plusieurs fois
```

```php
require_once 'includes/connexion.php'; // PHP ignore cet appel silencieusement
?>
```

Quand utiliser `_once` ?

Fichier	Recommandation
`connexion.php` (connexion BD)	✅ `require_once`
`fonctions.php` (tes fonctions)	✅ `require_once`
`header.php`	`require` suffit (pas de risque de doublon en pratique)
`menu.php`	`include` suffit

💡 **Bonne habitude :** utilise systématiquement `require_once` pour les fichiers qui définissent des **fonctions** ou des **variables de configuration**.

🚀 Partie 6 — Aller plus loin : titre dynamique par page

Le header contient `<title>Mon Site</title>` pour toutes les pages... ce n'est pas idéal pour le référencement ni pour l'utilisateur.

Solution : une variable avant l'include

```php
<?php
$titrePage = "Agenda – MonSite"; // ← défini AVANT d'inclure le header
require 'includes/header.php';
?>
```

`includes/header.php` (modifié)

```php
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title><?php echo $titrePage ?? 'MonSite'; ?></title>
    ...
```

💡 `$titrePage ?? 'MonSite'` : si `$titrePage` n'est pas défini, affiche `'MonSite'` par défaut (opérateur null-coalescing).

Avantages de cette approche

1. **Maintenance simplifiée** :
    - Tu modifies une seule fois le contenu commun dans `header.php` ou `footer.php`, et les changements s'appliquent automatiquement partout.
2. **Réduction des erreurs** :

- Avec `require_once`, tu évites les erreurs causées par des inclusions multiples accidentelles.

3. **Code plus propre** :
   - Les pages deviennent plus lisibles et se concentrent sur leur contenu spécifique.
4. **Meilleure organisation** :
   - Les fichiers dédiés (`header.php`, `footer.php`, etc.) permettent de structurer ton projet.

Points d'attention

1. **Chemins relatifs** :
   - Si tes fichiers sont organisés dans des dossiers, fais attention aux chemins.
   - Exemple : si `header.php` est dans un dossier `includes`, tu dois inclure le fichier ainsi :

```php
include 'includes/header.php';
```

2. **Erreurs silencieuses avec `include`** :
   - Si un fichier `include` manque, PHP affichera un avertissement, mais continuera d'exécuter le script. Cela peut entraîner des erreurs invisibles. Utilise `require` si le fichier est indispensable.
3. **Sécurité** :
   - Assure-toi que les fichiers inclus ne contiennent pas d'informations sensibles accessibles depuis le web.

📋 Récapitulatif

```
include 'fichier.php'       → insère le fichier (continue si absent)
require 'fichier.php'       → insère le fichier (STOP si absent)
include_once 'fichier.php'  → idem include, mais une seule fois
require_once 'fichier.php'  → idem require, mais une seule fois
```

Quand utiliser quoi ?

- Header / footer / menu → `include` ou `require`
- Connexion BD, fonctions → `require_once`
- Bandeau optionnel → `include`

💪 Exercices

Exercice 1 — Structure de base ⭐

1. Crée un dossier `mon-site/` avec un sous-dossier `includes/`
2. Crée les fichiers `header.php`, `menu.php`, `footer.php` dans `includes/`
3. Crée 3 pages : `index.php`, `about.php`, `contact.php`
4. Chaque page doit inclure les 3 fichiers partiels
5. Le menu doit contenir des liens vers les 3 pages

Critère de réussite : modifier le menu en un seul endroit doit changer les 3 pages.

Exercice 2 — Titre dynamique ⭐ ⭐

Reprends l'exercice 1 et fais en sorte que chaque page affiche un **titre différent** dans l'onglet du navigateur (`<title>`), sans modifier `header.php` sur chaque page.

Exercice 3 — Connexion centralisée ⭐ ⭐ ⭐

1. Crée un fichier `includes/connexion.php` qui établit une connexion PDO à une base de données MySQL
2. Crée une table `articles` avec quelques entrées (id, titre, contenu)
3. Dans `index.php`, utilise `require_once 'includes/connexion.php'` puis affiche la liste des articles

⚠️ Que se passe-t-il si tu remplaces `require_once` par `include` et que tu effaces temporairement le fichier `connexion.php` ? Teste et observe.

Les fonctions `include`, `require`, et `require_once` en PHP simplifient grandement la gestion des sections HTML répétées dans un site. En centralisant les fichiers communs, tu gagnes du temps, réduis les erreurs et rends ton code plus lisible. Apprends à les utiliser dès maintenant pour structurer proprement tes projets PHP ! 🚀