

Bases de Données 2

#4 - PHP

Matthieu Nicolas
Polytech S5 - II

Plan

- Principes des applications web
- Présentation de PHP
- Bases de PHP
- Interaction avec une BD en PHP

Principe des applications web

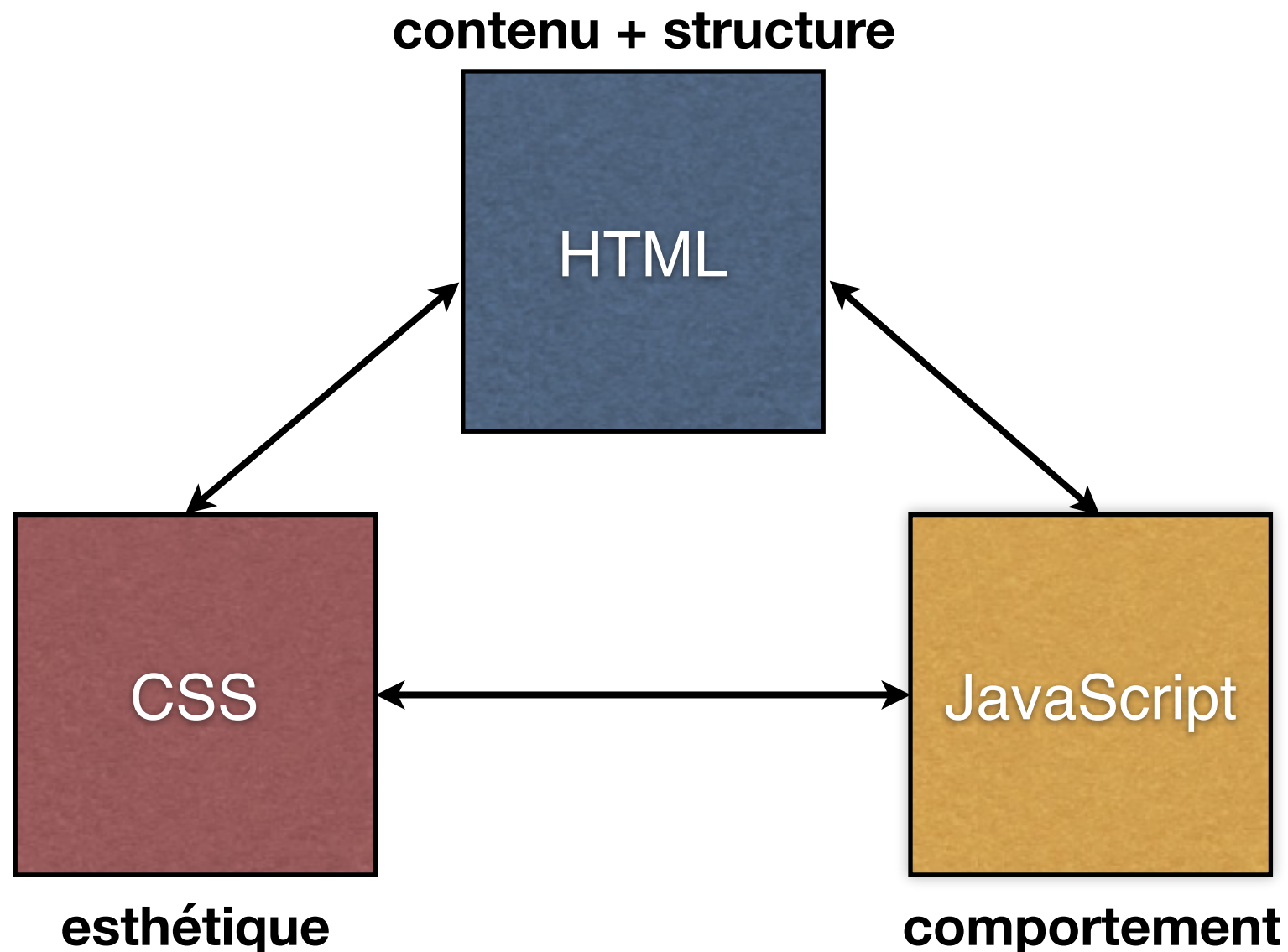
Base de Données 2
#4

Applications web

- **Applications** s'exécutant au sein **d'un navigateur web** (Firefox, Chrome, Edge...)
- Proposées par tous les services
 - Google, Facebook, Netflix, Microsoft Office...
- Utilisées par tous les appareils
 - Ordinateurs, Smartphones, Fusées...

Technologies dans un navigateur web - 1

- 3 technologies :



Technologies dans un navigateur web - 2

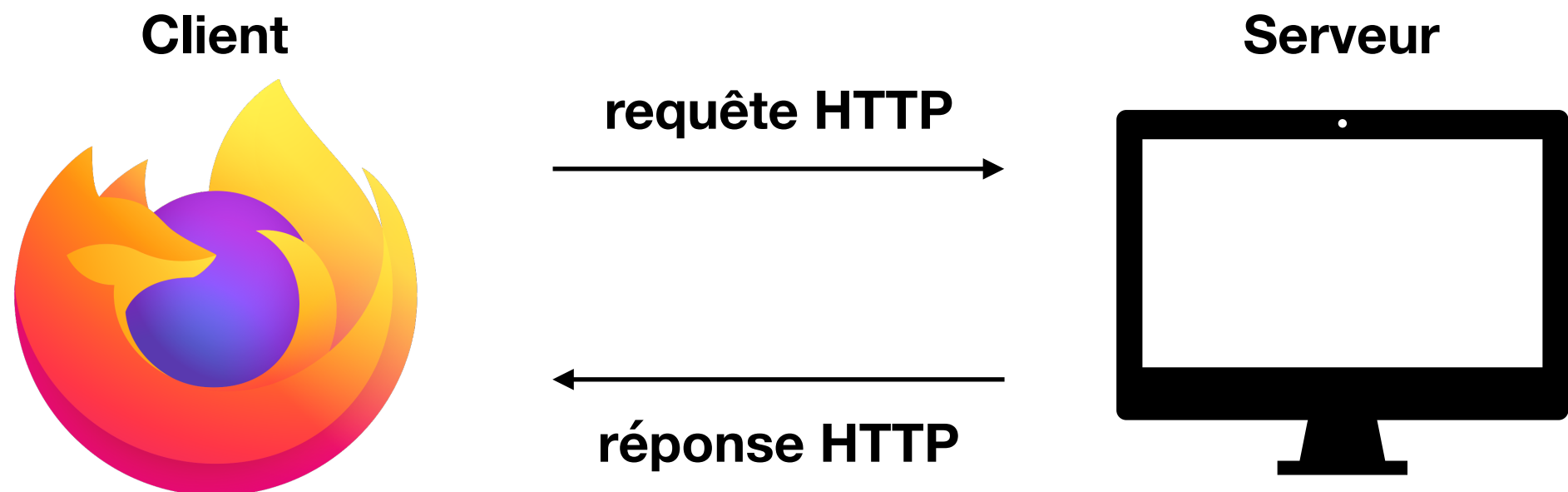
- **Problème :**
 - HTML et CSS ne sont pas des langages de programmation
- **Produisent un contenu dit statique**
 - Comment personnaliser la page en fonction de l'identité de l'utilisateur ?
 - Comment mettre à jour le contenu de la page en fonction des derniers posts/messages ?

Technologies dans un navigateur web - 3

- Javascript **sert** justement à **rendre dynamique** les pages
 - Permet de modifier l'HTML et le CSS de la page
- Mais à partir de **quelles données** ?

Principes des applications web - 1

- Repose sur une **architecture client-serveur**
 - Le client, un navigateur web, **envoie** une **requête** (HTTP) au serveur...
 - ... qui **génère** et **envoie** une **réponse** au client



Principes des applications web - 2

- **Globalement**
 - Le **serveur** fournit les **ressources initiales** (fichiers HTML et CSS, code JS)
 - Le **client s'occupe** de la **vue** (présentation des données) **et transmet** les **actions** de l'**utilisateur** au serveur
 - Le serveur **traite l'action utilisateur** (récupération ou mise à jour des données...) et **envoie la réponse**

Principes des applications web - 3

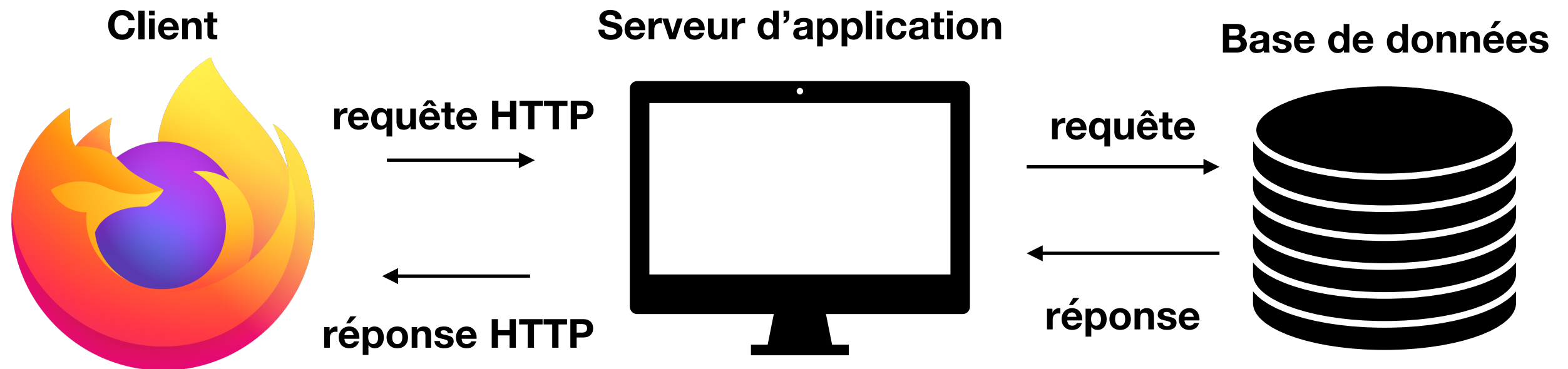
- De moins en moins vrai (IMHO)
 - Javascript s'occupe de plus en plus de tâches avec l'apparition de **AJAX**, des **Single-Page Applications**, des **Progressive Web Apps...**
 - Le serveur sert alors juste à partager et récupérer les données les plus récentes

Architecture application web - 1

- Repose sur une **architecture client-serveur...**
- ... mais généralement, distingue 2 types de serveur :
 - Le **serveur d'application**, qui se charge du **déroulé de l'application** et d'**effectuer les différents traitements**
 - Le **serveur de base de données**, qui **stocke les données** de l'application et qui **les fournit au serveur d'application**

Architecture application web - 2

- Parle aussi d'**architecture 3-tier**



- Permet de facilement **supporter plusieurs clients**, ou **décomposer son application en plusieurs services...**

Et PHP dans tout ça ?

- Langage pour le **développement de serveurs d'application**
- **S'utilise avec un serveur HTTP** (*Apache, NGINX...*)
 - Qui se charge d'exécuter le PHP pour générer la réponse aux requêtes qu'il reçoit

Présentation de PHP

Base de Données 2
#4

PHP - 1

- **PHP : Hypertext Preprocessor**
- Langage de script interprété
- Libre et multi-plateformes
- Créé en 1994

PHP - 2

- Principalement **utilisé** dans le cadre de l'implémentation de **serveurs web**
- 79,1% des sites web utilisent PHP (d'après <https://w3techs.com/technologies/details/pl-php>)
 - Dont **Facebook, Wikipedia...**
 - Très populaire notamment grâce à **WordPress**

Good parts

- **Simplicité** d'utilisation
- **Interactions avec les SGBDs** bien supportées
- Popularité
 - Permet de **trouver facilement des ressources** (tutos, frameworks, librairies...)
 - Beaucoup de ressources dépensées sur le langage (notamment par FB)

Bad parts

- Typage dynamique
 - **Variables non-typées**, valeurs typées
 - Conversions automatiques de type
- Langage interprété
 - **Performances moindres** qu'un langage compilé

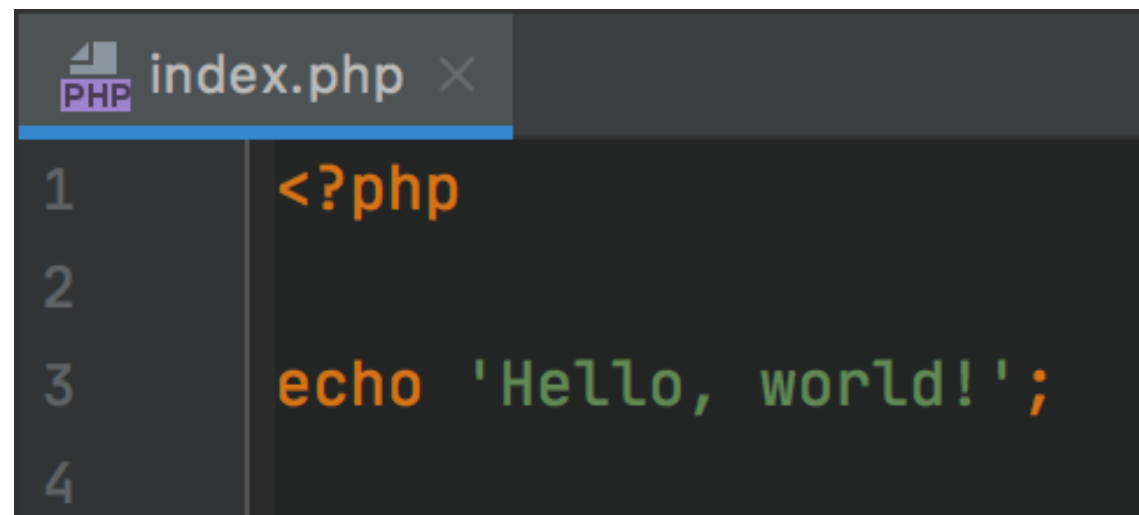
En évolution

- **Fonctionnalités ajoutées** au fur et à mesure des versions
 - Programmation Orientée Objet (PHP 5.0),
Espaces de noms (PHP 5.3)...
- Version 8.0.0 sortie fin novembre 2020
 - Amélioration des performances grâce à la **compilation à la volée** (Just-In-Time Compilation)

Bases de PHP

Base de Données 2
#4

Un premier programme

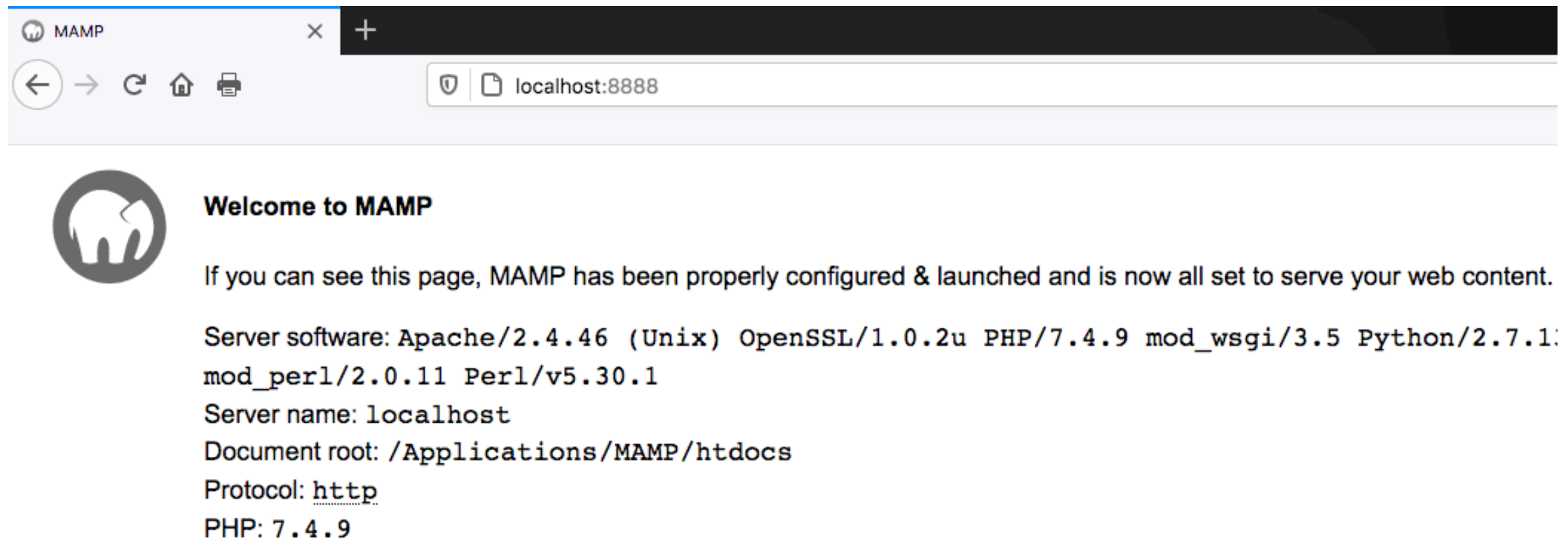


```
index.php x
1 <?php
2
3 echo 'Hello, world!';
4
```

- Permet d'afficher la chaîne “*Hello, world!*”
- Le code PHP s'écrit entre des balises
 - **<?php ... ?>** (?> pas obligatoire si fin du fichier)
- Programme dans un fichier **.php**
- Nécessite un ; à la fin de chaque instruction

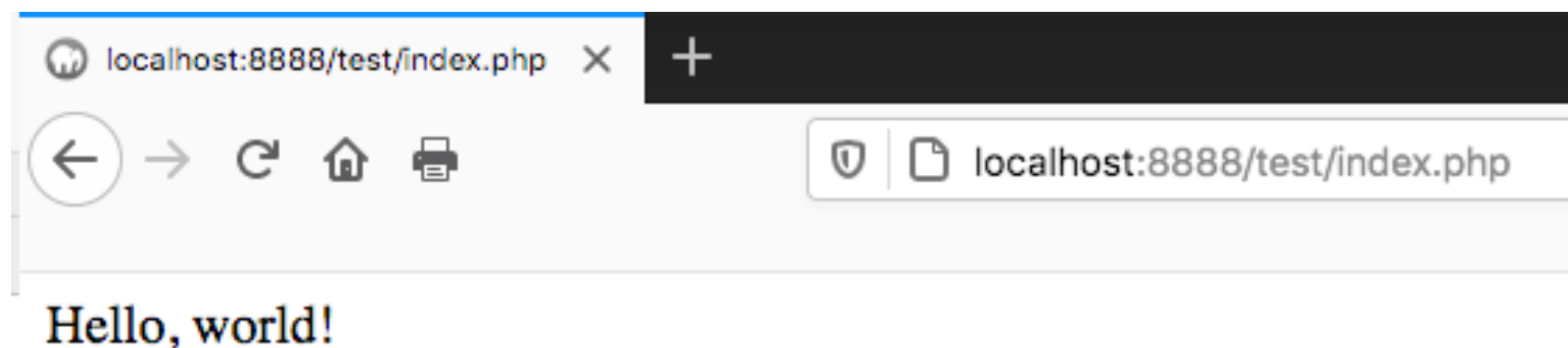
Exécuter ce programme - 1

- Le **programme doit être interprété par PHP**
 - Ne peut pas simplement l'ouvrir dans le navigateur
- Doit **demande au serveur web** fourni par MAMP **de l'exécuter et de nous fournir le résultat**



Exécuter ce programme - 2

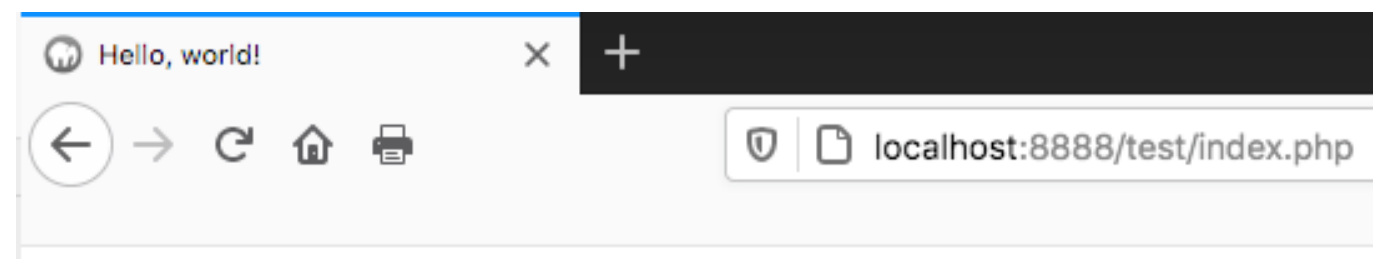
- Doit **placer le code dans un sous-répertoire du serveur web**
 - Dans mon cas, */Applications/MAMP/htdocs*
- Peut ensuite y accéder via **localhost:8888**



PHP et HTML - 1

- **Souhaite** généralement **fournir du HTML**
- PHP est simple d'utilisation
 - **Du code PHP** peut être directement inclus **dans le code HTML**
 - Permet de **rendre dynamique son contenu**

```
index.php x
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Hello, world!</title>
5      <meta charset="utf-8" />
6  </head>
7  <body>
8  <p>
9      Une page HTML simple.<br />
10     <?php echo "Soudain, du PHP :o"; ?>
11 </p>
12 </body>
13 </html>
```



Une page HTML simple.
Soudain, du PHP :o

Variables

- Spécifiées à l'aide de \$
- Variété de types disponibles : *string, int, float, boolean, array, null*
- *array* correspond plutôt au type *map* (tableau associatif) d'autres langages de programmation

```
<?php

$prenom = 'Matthieu';
$nom = "Nicolas";

$entier = 7;
$flottant = 9.99;

$vrai = true;
$faux = false;

$tableau = array(1, 2, 3, 4);
$map = array(
    "cle1" => "val1",
    "cle2" => "val2"
);

$rien = NULL;
```

Chaînes de caractères

- La concaténation de chaînes se fait à l'aide de l'opérateur .
- Les variables contenues dans les chaînes avec guillemets doubles sont évaluées
- Recommandé d'utiliser les apostrophes

```
<?php

$prenom = 'Matthieu';
$nom = "Nicolas";

$nomComplet1 = $prenom . ' ' . $nom; // Matthieu Nicolas
$nomComplet2 = "$prenom $nom"; // Matthieu Nicolas
```

Structures de contrôle

- Retrouve les structures de contrôle habituelles
 - *if / elseif / else*
 - *for*
 - *while*
- Syntaxe proche du Java

```
<?php
$a = 1;
$b = 2;

if ($a < $b) {
    echo $a . ' < ' . $b;
} elseif ($a === $b) {
    echo $a . ' = ' . $b;
} else {
    echo $a . ' > ' . $b;
}

for ($i = 0; $i < 10; $i++) {
    echo $i;
}

$j = 0;
while ($j < 10) {
    echo $j;
    $j++;
}
```

Fonctions

- Rien de particulier
- Peut **spécifier le type des différents paramètres et de retour**

```
<?php

function findMin (int $x, int $y): int {
    if ($x < $y) {
        return $x;
    } else {
        return $y;
    }
}

function computeSum (array $tab): int {
    $sum = 0;
    for ($i = 0; $i < sizeof($tab); $i++) {
        $sum += $tab[$i];
    }
    return $sum;
}

echo findMin(x: 3, y: 30);
echo computeSum(array(1, 2, 3, 4));
```

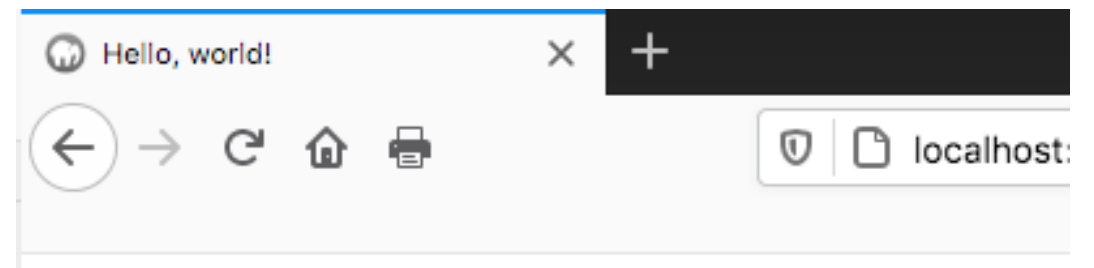
PHP et HTML - 2

- Peut **entremêler** les structures de contrôle de **PHP** et du **code HTML**
- Permet de générer dynamiquement le contenu de la page

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <title>Hello, world!</title>
  <meta charset="utf-8" />
</head>
<body>
  <?php
    for ($i = 0; $i < 5; $i++) {
      ?>
      <p>
        <?php
          if ($i % 2 == 0) {
            ?>
            Cette ligne a été écrite entièrement en HTML.<br />
            <?php
          }
          else {
            echo "Celle-ci a été écrite entièrement en PHP.";
          }
        ?>
      </p>
      <?php
    }
  ?>
</body>
</html>
```

PHP et HTML - 3

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <title>Hello, world!</title>
  <meta charset="utf-8" />
</head>
<body>
<?php
  for ($i = 0; $i < 5; $i++) {
    ?>
    <p>
      <?php
        if ($i % 2 == 0) {
          ?>
          Cette ligne a été écrite entièrement en HTML.<br />
          <?php
        }
        else {
          echo "Celle-ci a été écrite entièrement en PHP.";
        }
      ?>
    </p>
    <?php
  }
  ?>
</body>
</html>
```



Cette ligne a été écrite entièrement en HTML.

Celle-ci a été écrite entièrement en PHP.

Cette ligne a été écrite entièrement en HTML.

Celle-ci a été écrite entièrement en PHP.

Cette ligne a été écrite entièrement en HTML.

Interaction avec une BD en PHP

Base de Données 2
#4

APIs pour BD

- Existe **plusieurs APIs/extensions pour interagir** avec une base de données **en fonction du SGBD**
 - *mysql_* pour **MySQL** (obsolète)
 - *mysql_* pour **MySQL**
 - *pg_* pour **PostgreSQL**
 - ...

PDO

- Nous on va utiliser **PHP Data Object** (PDO)
- **Interface unique** pour interagir avec une BD, **indépendamment du SGBD**
 - Gère en interne les différences entre les SGBDs
- **Propose des méthodes** pour se **prémunir de failles de sécurité** (Injection SQL)
- Disponible de base depuis PHP 5.1

Connexion à une BD

- Avant de pouvoir interroger une BD, **besoin de créer une connexion**
- Pour cela, créer un objet PDO
 - *\$dsn*, qui répertorie les infos de la BD
 - les identifiants de connexion

```
<?php

$host = 'localhost';
$dbname = 'gestion_stock';
$dsn = 'mysql:host=' . $host
      . ';dbname=' . $dbname
      . ';charset=utf8';
$username = 'root';
$password = 'root';

try {
    $pdo = new PDO($dsn, $username, $password);
    $pdo->setAttribute( attribute: PDO::ATTR_ERRMODE,
                       value: PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die($e->getMessage());
}
```

Exécution d'une requête

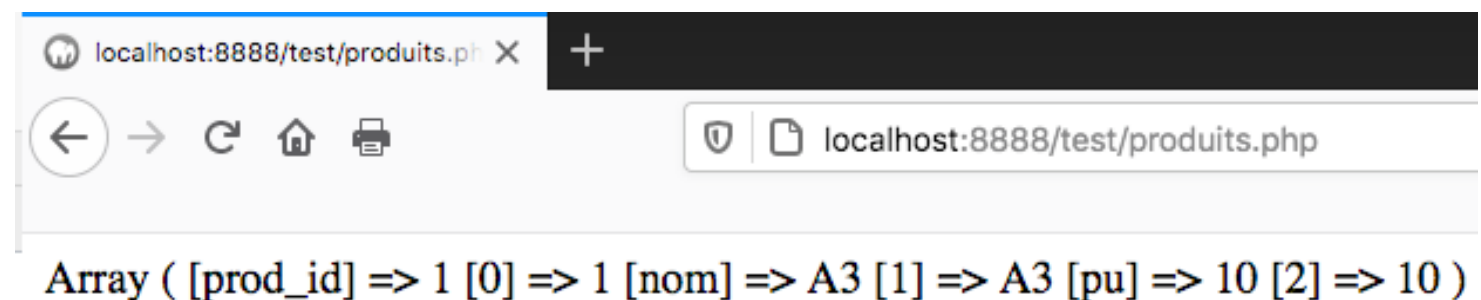
- Peut ensuite exécuter des requêtes SQL sur la BD à partir l'objet **PDO** obtenu
- À l'aide la méthode *query()*
 - *\$pdo* étant un objet, accède à ses méthodes via la notation ->

```
$queryProduits = 'SELECT * FROM produits';  
$results = $pdo->query($queryProduits);
```

Traitement du résultat - 1

- *\$results* contient le **résultat de la requête SQL** sous la forme d'un **PDOStatement**
- S'agit d'un ensemble de tuples
- Peut en récupérer le 1er à l'aide de la méthode *fetch()*

```
$results = $pdo->query($queryProduits);  
$row = $results->fetch();  
print_r($row);
```



localhost:8888/test/produits.php X +

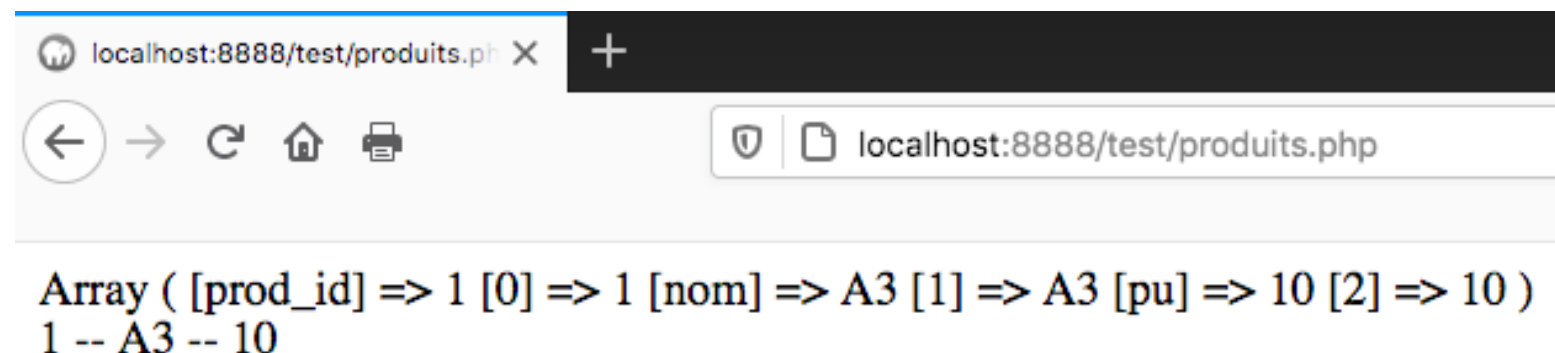
localhost:8888/test/produits.php

Array ([prod_id] => 1 [0] => 1 [nom] => A3 [1] => A3 [pu] => 10 [2] => 10)

Traitement du résultat - 2

- *\$row* est un tableau associatif
- Peut récupérer chaque attribut du tuple **via son index** ou **son nom**

```
$results = $pdo->query($queryProduits);  
$row = $results->fetch();  
print_r($row);  
echo '<br>' . $row[0] . ' -- ' . $row["nom"] . ' -- ' . $row["pu"];
```



localhost:8888/test/produits.php X +

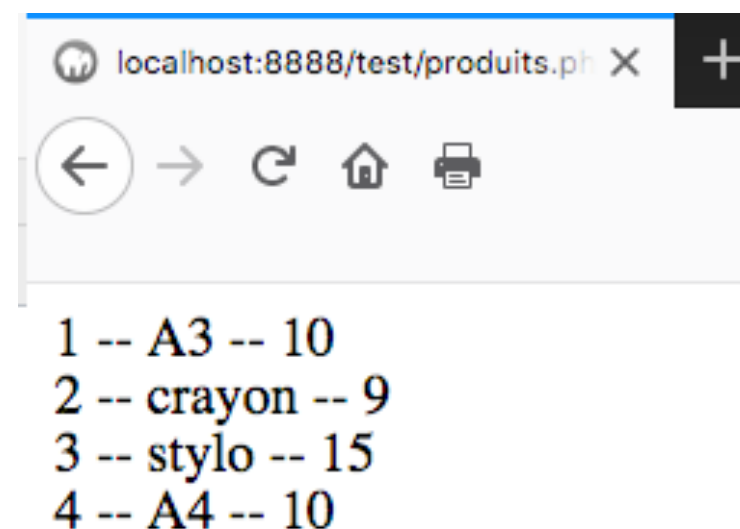
localhost:8888/test/produits.php

Array ([prod_id] => 1 [0] => 1 [nom] => A3 [1] => A3 [pu] => 10 [2] => 10)
1 -- A3 -- 10

Traitement du résultat - 3

- Peut **boucler** sur le **résultat** de *fetch()* pour **parcourir** l'ensemble des **tuples**

```
$results = $pdo->query($queryProduits);  
while ($row = $results->fetch()) {  
    echo $row["prod_id"] . ' -- ' . $row["nom"] .  
        ' -- ' . $row["pu"] . '<br>';  
}
```



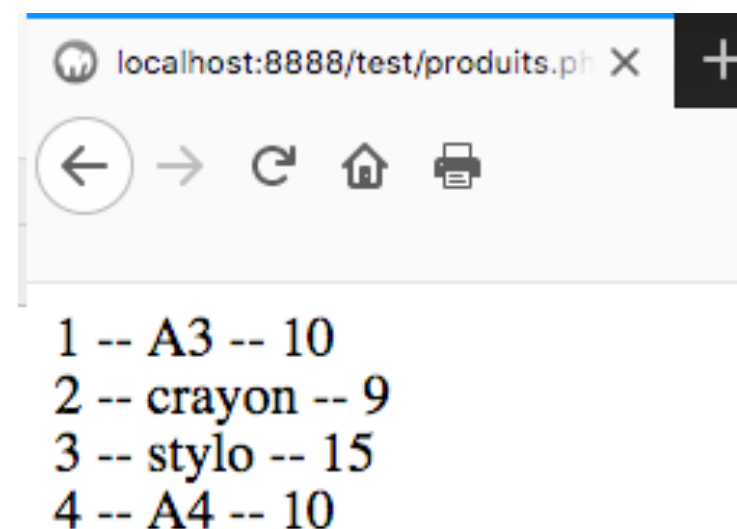
localhost:8888/test/produits.php X +

1 -- A3 -- 10
2 -- crayon -- 9
3 -- stylo -- 15
4 -- A4 -- 10

Traitement du résultat - 4

- **Peut aussi itérer sur *\$results* directement avec *foreach()***
- **Exactement le même résultat qu'avec le *while()* précédent**

```
$results = $pdo->query($queryProduits);  
foreach ($results as $row) {  
    echo $row["prod_id"] . ' -- ' . $row["nom"] .  
        ' -- ' . $row["pu"] . '<br>';  
}
```



localhost:8888/test/produits.php X +

1 -- A3 -- 10
2 -- crayon -- 9
3 -- stylo -- 15
4 -- A4 -- 10

C'est fini ! (pour le moment)

- **Toute à l'heure**
 - Utilisation de formulaires en PHP
 - Utilisation de requêtes préparées avec PDO
 - Factorisation du code commun entre fichiers

Rappel - IDE

- Plusieurs IDEs disponibles
- Installer l'IDE de votre choix
 - Visual Studio Code, PhpStorm (version éducation disponible via votre compte UL), Sublim Text, Vim...

Références

- **Documentation PHP**
 - <https://www.php.net/manual/fr/>
- Le cours de **OpenClassrooms** sur PHP
 - <https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>
- **Mozilla Developer Network (MDN)**
 - <https://developer.mozilla.org/fr/docs/Apprendre>

Des questions ?

Bases de Données 2

#5 - PHP (suite)

Matthieu Nicolas
Polytech S5 - II

Plan

- Utilisation de formulaires en PHP
- Utilisation de requêtes préparées avec PDO
- Factorisation du code commun entre fichiers
- Projet

Utilisation de formulaires en PHP

Base de Données 2
#5

Pour quoi faire ?

- On a appris comment **transmettre notre contenu à l'utilisateur**
 - HTML (statique ou généré dynamiquement par PHP)
- **Ce qui nous manque**
 - **Permettre à l'utilisateur de nous partager son contenu**
- Les formulaires servent justement à ça

Formulaires HTML - 1

```
<form action="creer_produit.php" method="POST">
  <input type="text" name="nom" placeholder="Nom du produit">
  <input type="number" name="pu" placeholder="Prix unitaire du produit">
  <button type="submit">Valider</button>
</form>
```

- `<form>`
 - `action` permet de **préciser le script** auquel on envoie les données
 - `method` la **manière de passer** les données (**GET** ou **POST**)

Choix de method - 1

- La **valeur** de `method` **indique** le **verbe HTTP** utilisé par la requête
 - GET, POST, PUT, DELETE...
- Ce **verbe** donne la **sémantique** de la requête
 - **GET** demande à **récupérer** une **ressource**
 - **POST** à en **créer** une
 - **PUT** à en **modifier** une
 - **DELETE** à en **supprimer** une

Choix de method - 2

- Notamment **utilisé dans les APIs REST**
 - https://fr.wikipedia.org/wiki/Representational_state_transfer
- En pratique
 - PHP **gère** “bien” nativement **que** GET et POST...
 - ... les autres sont un peu compliqués à utiliser
 - Frameworks/librairies aident pour cela
- On **utilisera POST dans nos formulaires**

Formulaires HTML - 2

```
<form action="creer_produit.php" method="POST">
  <input type="text" name="nom" placeholder="Nom du produit">
  <input type="number" name="pu" placeholder="Prix unitaire du produit">
  <button type="submit">Valider</button>
</form>
```

- `<input>`
 - Champs de notre formulaire
 - `type` correspond au **type de données** (texte, date, caché...)
 - `hidden` pratique pour transmettre une donnée qu'on souhaite pas afficher à l'utilisateur (un `id` par exemple)
 - `name` au **nom** de la **variable** dans le **script** d'arrivé

Récupération du contenu en PHP

- PHP génère et met à disposition des tableaux associatifs `$_GET` et `$_POST` en fonction de la méthode HTTP utilisée
- Permet de **récupérer la valeur de chaque champ** du formulaire à partir de son attribut "name"

```
$nom = $_POST['nom'];  
$pu = $_POST['pu'];  
  
echo $nom . ' --- ' . $pu;
```

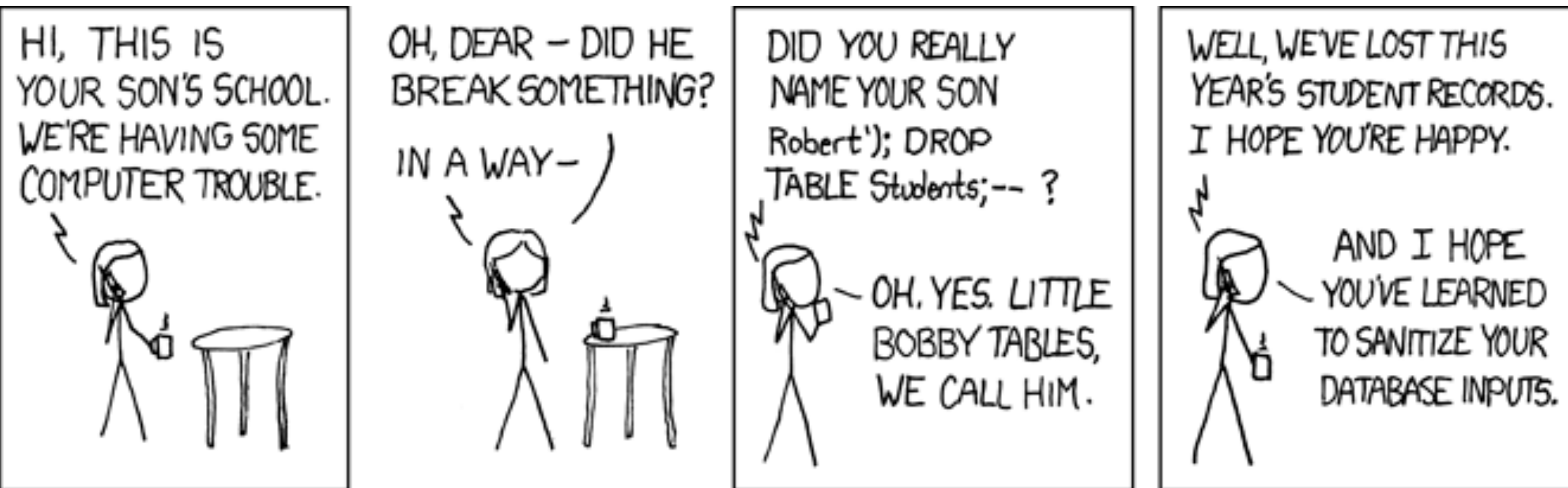
Utilisation de requêtes préparées avec PDO

Base de Données 2
#5

Traitement des entrées utilisateurs - 1

- **Arrive à récupérer les entrées de l'utilisateur via les formulaires + PHP**
- **Maintenant, reste plus qu'à les insérer dans la BD, non ?**

Non



<https://xkcd.com/327/>

Traitement des entrées utilisateurs - 2

- **Ne peut pas avoir confiance en les saisies de l'utilisateur**
 - Peut être simplement **invalid**
 - Peut être **malicious**
- **Même lorsque les valeurs possibles sont censées être limitées**
 - Un champ `<select>` d'un formulaire

Traitement des entrées utilisateurs - 3

- **Existe des techniques/outils** pour générer les requêtes “à la main”
 - Modification du code source de la page depuis le navigateur
 - Outil en ligne de commande (cURL)
 - Application dédiée (Postman)

Traitement des entrées utilisateurs - 4

- **Plusieurs types de failles de sécurité existantes**
 - Injection SQL
 - Faille XSS
- **Des façons différentes pour s'en prémunir**

Requêtes préparées - 1

- **Principe**

- Fournir la requête SQL au SGBD en avance...
- ... en précisant quels paramètres devront être fournis pour son exécution

- **Avantages**

- Permet au SGBD d'analyser et optimiser la requête en amont
- Permet d'exécuter plusieurs fois la requête préparée avec différents paramètres, efficacement

Requêtes préparées - 2

- Dans notre cas, **nous prémunit des injections SQL**
 - Code SQL déjà parsé
 - Les paramètres donnés sont uniquement considérés comme des valeurs...
 - ... non plus comme potentiellement des commandes SQL

Requêtes préparées en PDO - 1

- En PDO, se fait à l'aide de la méthode `prepare()`

```
$sql = 'INSERT INTO produits(nom, pu) VALUES (:nom, :pu)';  
$statement = $pdo->prepare($sql);
```

- Paramètres de la requête sont indiqués à l'aide de :
- Nous renvoie un **PDOStatement**

Requêtes préparées en PDO - 2

- Peut ensuite effectuer la requête à l'aide de `execute()`
- Prend en paramètre un **tableau associatif**
- Doit fournir une valeur pour chaque paramètre

```
$nom = $_POST['nom'];  
$pu = $_POST['pu'];  
$statement->execute(array(  
    ":nom" => $nom,  
    ":pu" => $pu  
));
```

Requêtes préparées en PDO - 3

- Par défaut, requêtes préparées sont juste simulées par PDO
- Besoin de paramétrer le bon attribut pour utiliser réellement la fonctionnalité

```
$pdo = new PDO($dsn, $username, $passwd);  
$pdo->setAttribute(attribute: PDO::ATTR_ERRMODE, value: PDO::ERRMODE_EXCEPTION);  
$pdo->setAttribute(attribute: PDO::ATTR_EMULATE_PREPARES, value: false);
```

Impact sur performances - 1

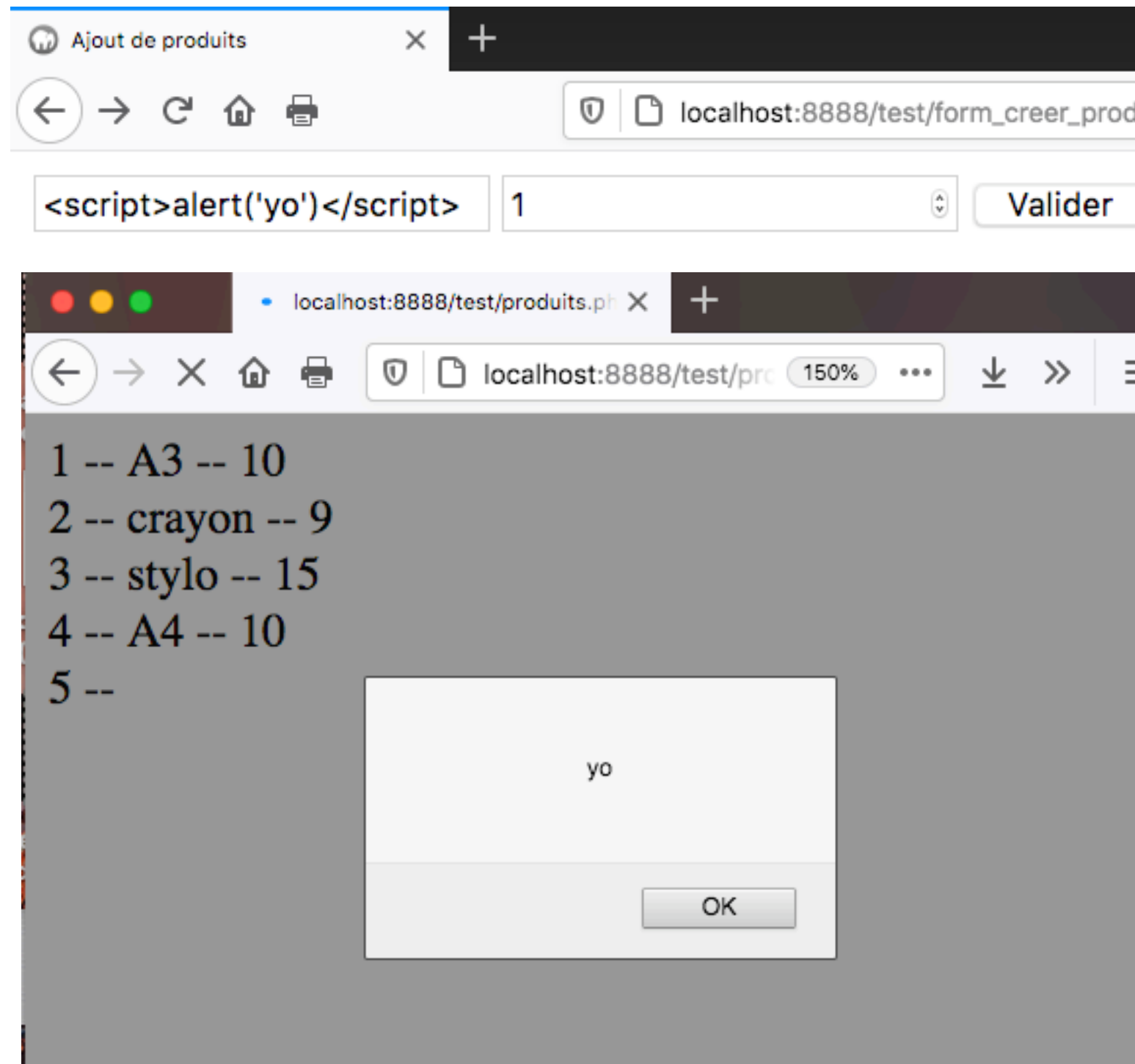
- Quid de l'**impact** des requêtes préparées **sur les performances** ?
- `prepare()` induit une communication supplémentaire
 - **Une pour requête, une autre pour paramètres**
- Dans le cas où on **exécute la requête plusieurs fois**, voit l'intérêt
- Mais **si on exécute la requête qu'une seule fois** ?

Impact sur performances - 2

- Préparer une requête pour l'effectuer qu'une seule fois est moins efficace...
 - ... mais de peu
 - ... et le **gain en sécurité** en échange est **inestimable**
- **Utiliser des requêtes préparées dès que vous avez des entrées utilisateurs**
- **Vous optimiserez le jour où vous aurez un problème de performances**

Failles XSS - 1

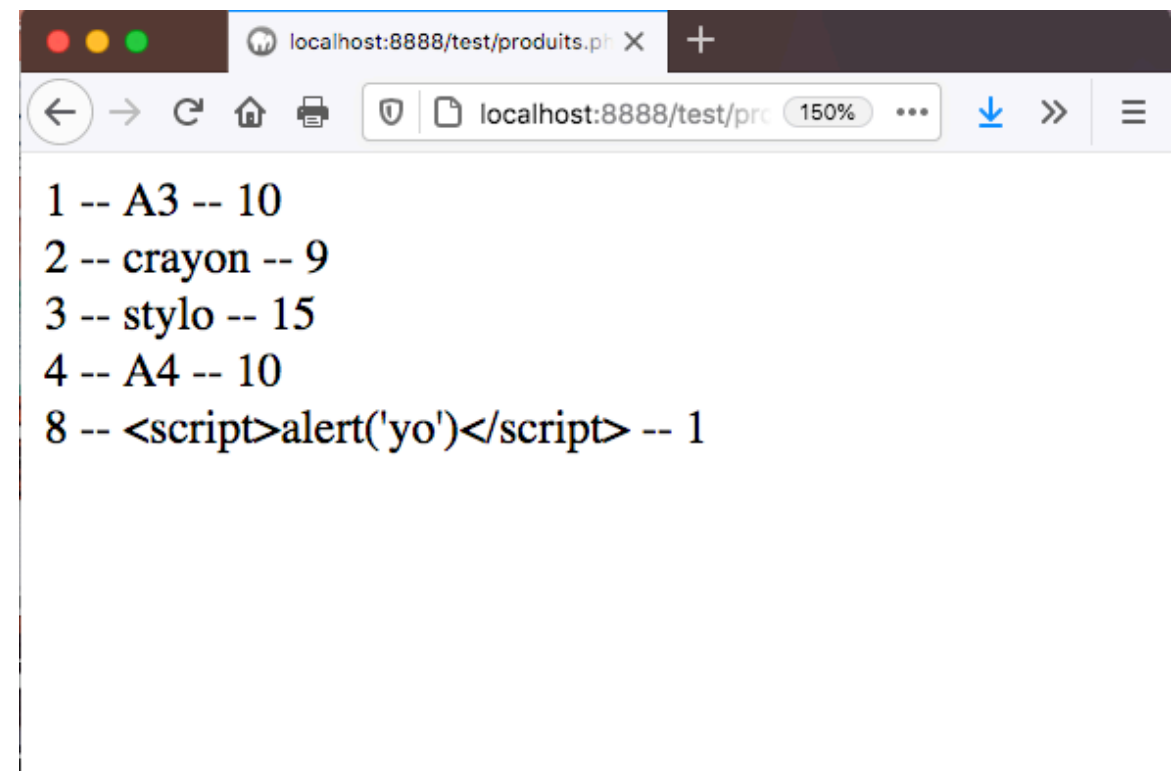
- Consiste à **saisir du JS dans les formulaires**
- De façon à ce que les **autres utilisateurs exécutent notre code** en accédant au contenu du site



Faibles XSS - 2

- `prepare()` nous **protège pas** de ce type de faille
- Doit utiliser une fonction : `htmlspecialchars()`
- Convertir les caractères spéciaux en code HTML pour les “échapper”

```
$statement->execute(array(  
    ':nom' => htmlspecialchars($nom),  
    ':pu' => htmlspecialchars($pu)  
));
```



Redirection

- On a exécuté notre script d'insertion/modification/suppression d'une ressource
- Reste plus qu'à renvoyer l'utilisateur sur la liste des ressources/la page de la ressource concernée

```
header( string: 'Location: ./');  
exit();
```

Factorisation du code commun entre fichiers

Base de Données 2
#5

Problématique

- Va avoir besoin de **partager du code entre scripts** au fur et à mesure que notre application va grossir
 - **instanciation de PDO**
 - **récupération des données d'une ressource**
 - ...
- Comment faire en PHP ?

require et include - 1

- Instructions permettant d'importer le contenu d'un autre script PHP dans le script courant
- Nous permet de **regrouper les fonctions d'une ressource dans un même fichier** (lister, créer, modifier, supprimer)
- Et de les **importer dans les scripts les utilisant**

require et include - 2

- **Exemple**

```
<?php  
require 'connection.php';
```

- **Différences**

- `include` **émet** un seulement **warning** si fichier introuvable
- `require` lui une **erreur** (à privilégier)

require et include - 3

- Existe aussi les variantes `require_once` et `include_once`
- Permettent de ne **pas ré-importer** le contenu du **script si déjà fait** au préalable

Références

- **Documentation PHP**
 - <https://www.php.net/manual/fr/>
- Le cours de **OpenClassrooms** sur PHP
 - <https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>
- **Mozilla Developer Network (MDN)**
 - <https://developer.mozilla.org/fr/docs/Apprendre>

Projet

Base de Données 2
#5

MyMDB

- Consiste à réaliser une version simplifiée de IMDB (<https://www.imdb.com/>)
 - Base de données sur le cinéma
- Application web permettant de consulter et d'interagir avec une BD

Ressources

- **3 ressources principales :**
 - Acteur-rice-s, films, séries
- **1 secondaire :**
 - Rôles

Fonctionnalités attendues

- Doit permettre de
 - **consulter** les différentes **ressources** (liste et page dédiée)
 - les **gérer** (créer, modifier, supprimer)
- Les **rôles** sont à **afficher** conjointement **avec** une **ressource principale**
 - Exemple : afficher les rôles d'un-e acteur-ice sur sa page de profil

Travail à réaliser

- Conception du schéma relationnel
- Écriture de requêtes SQL
- Implém des fonctionnalités de base de MyMDB
- ???

Consignes

- Travail en binôme
- Application web PHP
 - Frameworks/librairies autorisées pour back-end et front-end
- Utilisation d'une BD MySQL

Évaluation

- Portera sur
 - Conception et utilisation de la BD
 - Requêtes SQL
 - Qualité du code (variables claires, utilisation de fonctions)
 - Ergonomie de l'interface

Rendu

- Archive *.zip* à rendre sur Arche d'ici le **18 janvier 2021**
- Doit contenir (détails dans sujet) :
 - *README.md*
 - *schema-relationnel.pdf*
 - *bdd.sql*
 - *requetes.sql*
 - *mymdb/*

Des questions ?